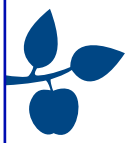

SYDDANSK UNIVERSITET
UNIVERSITY OF SOUTHERN DENMARK

Algorithms For Quantum Computers

Bachelor Thesis by

Magnus Gausdal Find
magnus@gausdalfind.dk

Advisor: Joan Boyar

Department of Mathematics and Computer Science
University of Southern Denmark, Odense

Contents

List of Figures

List of Algorithms

Problem Statement

Danish version

En kvantecomputer er en form for computer, der adskiller sig fra en traditionel computer ved at benytte kvantemekaniske effekter såsom superposition til at udføre beregninger. I en traditionel computer vil hver bit have enten værdien 0 eller 1, men ved at bruge superposition kan værdien 0 og værdien 1 repræsenteres samtidig i een "qubit".

Der skal gives en introduktion til emnet "Kvantecomputer", begreber som qubit, superposition og kvante-parallelisme skal i den forbindelse introduceres. Ved at præsentere vigtige resultater inden for emnet, herunder særligt Shors faktoriseringsalgoritme og Grovers søgningsalgoritme, belyses det hvad en kvantecomputer kan, som en traditionel computer ikke kan. For at illustrere hvordan en kvantecomputer udfører beregninger skal der implementeres en simulation af en af de to algoritmer. Der kan evt. inddrages nyere litteratur om emnet.

English translation

A quantum computer is a kind of computer that differs from traditional computer by making use of quantum mechanical effects, such as superposition, to carry out computations. In a traditional computer, bits has either the value 0 or 1, but by exploiting superposition, a "qubit" can at one time represent the value 0 and 1.

An introduction to the subject "quantum computing" shall be given . This includes introductions of concepts such as qubits, superposition, quantum parallelism. By presenting important results from the field, especially the factorization algorithm of Shor and the searching algorithm of Grover, the strengths of quantum computers shall be shown. To illustrate how a quantum computer works a quantum computer simulator shall be implemented, being able to simulate at least one of the two algorithms. If possible newer literature from the field can be studied.

Preface

This bachelor project in computer science was made during the period from February to August 2009 by Magnus Find with advisory from Joan Boyar. The subject is on algorithms for quantum computers. My own interest in this subject comes from the last year of high school, where every student had to write an assignment on a topic related to subjects that the student had chosen. For technical reasons I couldn't write purely in computer science, so I went to my physics and computer science teacher (luckily the same person) Martin Lund and asked him for suggestions to a co-disciplinary subject. And he suggested quantum computation. At that time the subject was interesting to me, but it was obviously at a quite superficial level since I had little or no knowledge of either algorithmics, complexity theory, linear algebra, etc. Now, having studied computer science for three years, I have decided to return to the topic with an improved foundation in computer science and mathematics. Since my background in physics is as limited as it was in high school, I will focus as little as possible on the physical constitution on the field, restraining it to be a model of computation. This is of course not to belittle the role of physics - the realization of a quantum computer will ultimately rely on the work of physicists.

The report is structured as follows: In chapter 1 I will give a brief summary of the framework of quantum computation, using this framework to present two of the milestone algorithms namely those of Grover and Shor. After that an introduction to two newer algorithms will be given, one for finding minimum spanning trees and one for finding single source shortest paths. Both of these are based on Grover's algorithm. All of these algorithms have been implemented (on a simulator of course), and chapter 2 is about how this has been done, including actual running time analysis of the algorithms (if they were run on a quantum computer).

In keeping with tradition I will here use a few lines to say thanks to some of the people who has helped me during the process. Joan Boyar, for being a very committed, and patient advisor. Martin Lund, for introducing me to the topic. Julie Hansen, for helping me getting rid of some of the worst spelling errors. All inhabitants at "Balkonen" for helpful discussions.

Chapter 1

Quantum Computing

Since quantum computing might not be a well known topic to the reader, a short (and necessarily quite superficial) introduction to the subject will be given here. This is the purpose of the section 1.1, which mainly serves as an explanatory part, but it also explains and introduces notation used later. In Sections 1.2 and 1.3 the algorithms of Shor and Grover are explained and analyzed. Section 1.4 introduces some newer research based on Grovers algorithm.

1.1. Short introduction to quantum computing

1.1.1. Quantum Mechanics

The theory of quantum mechanics has some astonishing consequences. One of the most surprising postulates is the fact that it is possible for a particle to be in more than one state at the same time. This means that it makes sense to talk about a particle being at two *different* locations at *the same time*[1], or for an atom to be in two *different* energy levels *at the same time*. In general this phenomenon is called quantum superposition. One of the postulates of quantum mechanics is[2] that if a particle is able to be in several different states (ie. energy levels, places etc.), then any linear combination of these states is also a feasible state. Mathematically spoken: if $|1\rangle, |2\rangle, \ldots, |n\rangle$ are possible states of our particle, then any state

$$c_1 |1\rangle + c_2 |2\rangle + \ldots + c_n |n\rangle, c_k \in \mathbb{C},$$

is a possible state for our particle (with some restrictions on the norm of the vector). A particle in a superposition can be measured; when measured, one of the base states

[1] This formulation might be a little exaggerating. Whether or not the particle actually is at two locations at the same time or its position just hasn't been defined is a question, I will leave for philosophers and metaphysicists.

[2] Here I am referring to the Copenhagen interpretation of quantum mechanics, which is described in [Nak08]

is obtained according to probabilities. The probability for state $|k\rangle$ to be measured is $|c_k|^2$. We of course now realize that $\sum_{k=1}^{n} |c_k|^2 = 1$, since the probabilities must add up to one, so the state vector is a unit vector (with respect to the Euclidean norm). A measurement causes the superposition to "collapse", meaning that if $|k\rangle$ is measured, c_k becomes 1 and all others become 0 [Nak08].

1.1.2. Qubits

Attention is often paid to particles having two "base states", say $|0\rangle$ and $|1\rangle$, since they can be considered as a generalization of a "classical" bit - being able to be in state $|0\rangle$ and $|1\rangle$ and "*anywhere between*". Such a particle is called a quantum bit, or short a *qubit*. In classic computer science gate operations such as `AND` and `OR` constitute the core of data manipulation. Similar operations are possible on qubits. Consider the base states $|0\rangle$ and $|1\rangle$ as an orthonormal basis of a two dimensional complex vector space. Thus a possible state for a qubit is of the form $a|0\rangle + b|1\rangle$, where $|a|^2 + |b|^2 = 1$ or shortly written as $(a, b)^{\mathrm{T}}$. The gate operations possible to perform are exactly all unitary linear operations. This constraint is due to physical limitations, which I will not justify.

An important example is the Hadamard transformation. This is defined as

$$a|0\rangle + b|1\rangle \to \frac{a+b}{\sqrt{2}}|0\rangle + \frac{a-b}{\sqrt{2}}|1\rangle$$

Or if we put $|0\rangle = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$ and $|1\rangle = \begin{pmatrix} 0 \\ 1 \end{pmatrix}$, it can be represented as the matrix:

$$\frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}$$

Observe especially that the Hadamard gate applied to $|0\rangle$ yields:

$$\frac{1}{\sqrt{2}}|0\rangle + \frac{1}{\sqrt{2}}|1\rangle$$

This is called an equally weighted superposition, since a measurement yield the values 0 and 1 with equal probability. If there are n qubits in a register, and Hadamard gates are applied the operation is called Walsh-Hadamard transformation. This gate is of course obtained by tensor multiplication $H \otimes H \otimes \ldots \otimes H = H^{\otimes n}$. For example

$H^{\otimes 3} = H \otimes H \otimes H$ is:

$$\begin{aligned} H \otimes (H \otimes H) &= \frac{1}{\sqrt{2}}\begin{pmatrix}1 & 1\\ 1 & -1\end{pmatrix} \otimes \left(\frac{1}{\sqrt{2}}\begin{pmatrix}1 & 1\\ 1 & -1\end{pmatrix} \otimes \frac{1}{\sqrt{2}}\begin{pmatrix}1 & 1\\ 1 & -1\end{pmatrix}\right) \\ &= \frac{1}{2\sqrt{2}}\begin{pmatrix}1 & 1\\ 1 & -1\end{pmatrix} \otimes \begin{pmatrix}1 & 1 & 1 & 1\\ 1 & -1 & 1 & -1\\ 1 & 1 & -1 & -1\\ 1 & -1 & -1 & 1\end{pmatrix} \\ &= \frac{1}{2\sqrt{2}}\begin{pmatrix}1 & 1 & 1 & 1 & 1 & 1 & 1 & 1\\ 1 & -1 & 1 & -1 & 1 & -1 & 1 & -1\\ 1 & 1 & -1 & -1 & 1 & 1 & -1 & -1\\ 1 & -1 & -1 & 1 & 1 & -1 & -1 & 1\\ 1 & 1 & 1 & 1 & -1 & -1 & -1 & -1\\ 1 & -1 & 1 & -1 & -1 & 1 & -1 & 1\\ 1 & 1 & -1 & -1 & -1 & -1 & 1 & 1\\ 1 & -1 & -1 & 1 & -1 & 1 & 1 & -1\end{pmatrix} \end{aligned}$$

1.1.3. Composite states and entanglement

The situation becomes even more interesting when the quantum system consists of two or more qubits. Consider n qubits in a system each having state space ψ_i. The composite state space is $\psi_1 \otimes \ldots \otimes \psi_n$. For example let $\left\{\begin{pmatrix}1\\0\end{pmatrix}, \begin{pmatrix}0\\1\end{pmatrix}\right\}$ be basis vectors for each of two qubits. Then basis vectors for the system consisting of both qubits are:

$$\left\{\begin{pmatrix}1\\0\\0\\0\end{pmatrix}, \begin{pmatrix}0\\1\\0\\0\end{pmatrix}, \begin{pmatrix}0\\0\\1\\0\end{pmatrix}, \begin{pmatrix}0\\0\\0\\1\end{pmatrix}\right\}$$

Corresponding to $|0\rangle \otimes |0\rangle, |0\rangle \otimes |1\rangle, |1\rangle \otimes |0\rangle, |1\rangle \otimes |1\rangle$ respectively. In the literature the symbol "$\otimes$" between basis vectors are often omitted, so $|0\rangle \otimes |0\rangle$ and $|00\rangle$ are used interchangeably. Composite states can be quite remarkable. Observe that for two qubits in a composite state, the state $\frac{1}{\sqrt{2}}|00\rangle + \frac{1}{\sqrt{2}}|11\rangle$ is a valid state, called a Bell State. Thus performing a measurement on the first qubit has an instantaneous consequence for the second qubit. Imagine for instance that the first bit is measured to $|0\rangle$, then there is only one possibility for the second qubit - namely to be in position $|0\rangle$. This phenomenon is called quantum entanglement, this is one of the core elements of quantum computing. Notice that this phenomenon applies even when the two

particles are at great distance.[3]

As for the example of one qubit, there exists gates acting on two and more qubits. An often used example of a quantum gate acting upon 2 bits is the *Controlled-not* gate (or CNOT). The classical interpretation is as follows: if the first qubit is 1, flip the value of the second bit. The quantum interpretation is a little more subtle. The gate can be expressed as:

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}$$

which exactly maps $|00\rangle$ to $|00\rangle$, $|01\rangle$ to $|01\rangle$, $|10\rangle$ to $|11\rangle$ and $|11\rangle$ to $|10\rangle$. Note that if the qubits are in a superposition of states, this gate (since it is a linear mapping) acts on all the basis vectors. For the sake of brevity, I will from now on use $|k\rangle$ as short notation for the state obtained by the binary representation of k.

1.1.4. At least as good as classic circuits

Another important gate is the *Controlled-controlled-not* gate, also known as the *Toffoli* gate. This works similarly to the *controlled not* gate. If the two first input qubits are $|1\rangle$ the third is flipped. This is of interest because this gate can be used to perform all types of classical operations.

It is a well known result that the two classical gates `NOT` and `AND` together form a functionally complete set of logical operators. Thus showing that the Toffoli gate can perform these two operations is sufficient to show that any operation that can be carried out using a classical circuit, can also be efficiently carried out on a quantum computer.

Not: Suppose we want negate the boolean value of some value x. From the definition of the Toffoli gate, if the two first inputs are $|1\rangle$ and the third qubit is x, then the third output is exactly $\neg x$.

And: Suppose x and y are qubits and we want to compute $x \wedge y$. Let the inputs to the Toffoli gate be x, y and $|0\rangle$ respectively then the third output is $|1\rangle$ if and only if both x and y are $|1\rangle$.

Thus any circuit of n classical gates can be implemented using $\Theta(n)$ Toffoli gates.

[3] This is sometimes referred to as the Einstein-Podolsky-Rosen paradox since this effect apparently makes information travel with speed greater than light. [Ved06]

1.1.5. Quantum Parallelism

Consider a function $f : \{0,1\}^n \to \{0,1\}^n$ computable using a circuit consisting of k classical gates. This circuit can be implemented with $\Theta(k)$ Toffoli gates. Let us think of this circuit as one gate and call it U. Consider a quantum system in an equally weighted superposition of all states of an n-qubit register:

$$\frac{1}{\sqrt{2^n}}\left(|0\rangle + |1\rangle + |2\rangle + \ldots + |2^n - 1\rangle\right)$$

Applying the gate U yields the state:

$$\frac{1}{\sqrt{2^n}}\left(|f(0)\rangle + |f(1)\rangle + |f(2)\rangle + \ldots + |f(2^n - 1)\rangle\right)$$

Thus computation of the function f is done exponentially many times (in the number of bits) using the same time as used for computing one value. This phenomenon is called quantum parallelism, and this is obviously a huge advantage of quantum computing, though it takes some non-trivial methods to truly exploit it. This is due to the fact that simply measuring the outcome yields just one of the results, which is no better than just computing $f(i)$ for some randomly chosen $i \in \{0, \ldots, 2^n - 1\}$. Smarter ways of exploiting quantum parallelism are demonstrated throughout the rest of this chapter.

1.2. Shor's Algorithm

Ever since the ancient Greeks, prime factorization has been a topic of great interest, and after cryptographic protocols such as RSA have become standard, this interest has only increased. Therefore it was of great interest when Peter Shor in 1994 [Sho94] published a polynomial time quantum algorithm for solving the problem of finding non-trivial factors of a composite number.

The algorithm reduces the problem of finding divisors of a composite number to the problem of order finding.

Definition 1. Given a number a and a modulus m with $gcd(a, m) = 1$ the order is the least positive integer such that $a^r \equiv 1 \mod m$.

The reduction is not very complicated and completely non-quantum, as shown on Algorithm 1 (this is taken from [NC00]). The conclusion in line 12 might not seem obvious, and I will here prove its correctness.

Algorithm 1: Reducing factoring to order-finding
Input: integers a and N.
1: Let a be a random number in $\{2, \ldots, N-1\}$
2: **if** $gcd(a, N) \neq 1$ **then**
3: **return** a
4: **else**
5: Use quantum algorithm to find the order r of a modulo N
6: **if** r odd **then**
7: try again with another value of a
8: **else**
9: **if** $a^{\frac{r}{2}} \equiv -1 \mod N$ **then**
10: try again with another value of a
11: **else**
12: $gcd(a^{\frac{r}{2}} - 1, N)$ and $gcd(a^{\frac{r}{2}} + 1, N)$ are nontrivial divisors.
13: **end if**
14: **end if**
15: **end if**

Suppose that the even integer r is the order of a modulo N and that $a^{\frac{r}{2}} \not\equiv -1 \mod N$. Then by definition of order:

$$a^r \equiv 1 \mod N$$

And since r is even:

$$\begin{aligned} \left(a^{\frac{r}{2}}\right)^2 - 1 &\equiv 0 \mod N \\ \Updownarrow & \\ \left(a^{\frac{r}{2}} - 1\right)\left(a^{\frac{r}{2}} + 1\right) &\equiv 0 \mod N \end{aligned}$$

Assume for the sake of contradiction that $gcd(a^{\frac{r}{2}} - 1, N) = 1$, ie. is a trivial divisor to N. This implies that $a^{\frac{r}{2}} - 1$ has en inverse element in $\mathbb{Z}_N$, which means that

$$a^{\frac{r}{2}} + 1 \equiv 0 \mod N$$

Violating the assumption that $a^{\frac{r}{2}} \not\equiv -1 \mod N$. The same argument shows that if $gcd(a^{\frac{r}{2}} + 1, N) = 1$, then $a^{\frac{r}{2}} \equiv 1 \mod N$ violating the assumption that r was the order of a.

1.2.1. Finding the order

From algorithm 1 it is (almost) clear, that a polynomial algorithm to find the order immediately gives a polynomial algorithm for factoring. This is done using a mixture of classical and quantum computing.

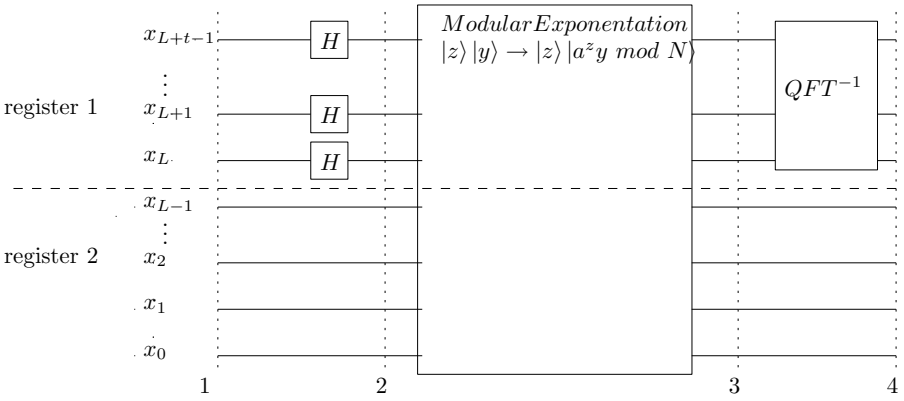

Figure 1.2.1: Circuit demonstrating how the order finding algorithm works

Quantum Part

The qubits are divided into two registers. Let L be the number of bits necessary to represent the modulo N and let ϵ be the probability for the algorithm to fail. Register 1 consists of $t = 2L+1+\lceil \log\left(2+\frac{1}{2\epsilon}\right)\rceil$ qubits (this number will be justified in step 4), and register 2 consists of L qubits. Figure 1.2.1 illustrates the steps of the algorithm. Vertical dotted lines indicate timestamps.

Step 1 Initially (at time 1) register 1 is set to $|0\rangle$ and register 2 to $|1\rangle$. Thus the initial states of the registers are:

$$|0\rangle |1\rangle = |00\ldots01\rangle$$

(I will continue throughout this section to use the notation $|a\rangle |b\rangle$ meaning register 1 is in state a and register 2 is in state b).

Step 2 Then Hadamard gates are applied to all qubits in register 1. This yields the state (corresponding to time 2):

$$\frac{1}{\sqrt{2^t}} \sum_{k=0}^{2^t-1} |k\rangle |1\rangle$$

Step 3 Now modular exponentiation is applied. Ie. the operation $|z\rangle |y\rangle \rightarrow |z\rangle |a^z y \mod N\rangle$ is carried out on the superposition. Notice that this operation can be done efficiently using a purely classical strategy (For instance an $O(\log(m)^3)$ algorithm is given in

[CLRS01]). Thus it can (cf. Section 1.1.4) be implemented using a polynomial number of quantum gates. The state at time 3 is then:

$$\frac{1}{\sqrt{2^t}} \sum_{k=0}^{2^t-1} |k\rangle \left| x^k \mod N \right\rangle$$

This can be rewritten in a quite convenient way. Let us introduce the state $|u_s\rangle$ defined as[4]:

$$|u_s\rangle = \frac{1}{\sqrt{r}} \sum_{j=0}^{r-1} \exp\left(\frac{-2\pi i s j}{r}\right) |x^j \mod N\rangle$$

In [NC00] it is claimed, but not proved that:

$$\left| x^k \mod N \right\rangle = \frac{1}{\sqrt{r}} \sum_{s=0}^{r-1} \exp\left(\frac{2\pi i s k}{r}\right) |u_s\rangle \tag{1.2.1}$$

Proof: Consider the right hand side (RHS), inserting the definition of u_s:

$$\begin{aligned} & \frac{1}{\sqrt{r}} \sum_{s=0}^{r-1} \exp\left(\frac{2\pi i s k}{r}\right) \frac{1}{\sqrt{r}} \sum_{j=0}^{r-1} \exp\left(\frac{-2\pi i s j}{r}\right) |x^j \mod N\rangle \\ = \; & \frac{1}{r} \sum_{j=0}^{r-1} \sum_{s=0}^{r-1} \exp\left(\frac{2\pi i s k}{r} + \frac{-2\pi i s j}{r}\right) |x^j \mod N\rangle \end{aligned}$$

Consider now the "inner sum", for $j \neq k$:

$$\begin{aligned} & \sum_{s=0}^{r-1} \exp\left(\frac{2\pi i s k}{r} + \frac{-2\pi i s j}{r}\right) \\ = \; & \sum_{s=0}^{r-1} \left(\exp\left(\frac{2\pi i}{r}(k-j)\right)\right)^s \\ = \; & \frac{\exp\left(\frac{2\pi i (k-j)}{r}\right)^r - 1}{r-1} \\ = \; & 0 \end{aligned}$$

And if $j = k$, $\exp\left(\frac{2\pi i}{r}(k-j)\right) = 1$, and the sum is therefore r. Thus the only term in the sum over j contributing to the sum is for $j = k$, and the RHS of 1.2.1 reduces to the LHS, thus proving the claim. ∎

The state of the system at time 3 can thus be written as:

$$\frac{1}{\sqrt{r 2^t}} \sum_{s=0}^{r-1} \sum_{k=0}^{2^t-1} \exp\left(\frac{2\pi i s k}{r}\right) |k\rangle |u_s\rangle \tag{1.2.2}$$

[4]This state is defined in the book of [NC00] and is of special interest since it is an eigenvector of the gate performing the operation $|y\rangle \rightarrow |xy \mod N\rangle$

Step 4 In [NC00] it is shown how to implement the "Quantum Fourier Transformation" (QFT) using a polynomial number of gates. The QFT performs the mapping:

$$|j\rangle \rightarrow \frac{1}{\sqrt{N}} \sum_{k=0}^{N-1} \exp\left(\frac{2\pi ijk}{N}\right) |k\rangle$$

Thus the inverse QFT performs the mapping

$$\frac{1}{\sqrt{N}} \sum_{k=0}^{N-1} \exp\left(\frac{2\pi ijk}{N}\right) |k\rangle \rightarrow |j\rangle$$

This operation can be implemented using the exact same number of gates. This is due to the fact that all gates correspond to unitary matrices combined with the fact from linear algebra that every unitary matrix has an inverse given by the complex conjugation of the transposed matrix. Concluding that given a circuit to compute the QFT, the circuit for computing the inverse QFT can be obtained by reversing the order of the gates an complex conjugate and transpose all the gates.

Consider the "inner" part of 1.2.2:

$$\begin{aligned} & \sum_{k=0}^{2^t-1} \exp\left(\frac{2\pi isk}{r}\right) |k\rangle \\ = & \sum_{k=0}^{2^t-1} \exp\left(\frac{2\pi ki\frac{s2^t}{r}}{2^t}\right) |k\rangle \end{aligned}$$

Applying the inverse Quantum Fourier Transformation gives:

$$\sqrt{2^t} \left| 2^t \frac{s}{r} \right\rangle$$

This holds if $\frac{s2^t}{r}$ can be written with the number of bits available in the particular register, that is if the fraction $\frac{s}{r}$ has at most t digits in its binary expansion. But this is not necessarily the case. If not the case, it is shown in [NC00] pp. 224 that using $n + \left\lceil \log\left(2 + \frac{1}{2\epsilon}\right) \right\rceil$ qubits gives a result correct with n bits with a probability of at least $1 - \epsilon$.

Thus applying the inverse Fourier transformation on register 1, the state with high probability (that is at least $1 - \epsilon$) becomes:

$$\frac{1}{\sqrt{r}} \sum_{s=0}^{r-1} \left| \widetilde{2^t \frac{s}{r}} \right\rangle |u_s\rangle$$

where $\widetilde{2^t \frac{s}{r}}$ denotes a number close to $2^t \frac{s}{r}$. In this setup it means that at least the $2L + 1$ most significant bits are correct.

Thus a measurement of register 1 yields $2L + 1$ correct bits of the fractional part of $\frac{s}{r}$ for some random (an at the time of measurement unknown) number $s \in \{0, \ldots, r-1\}$.

Classical part

Say the value x is measured in register 1. The difference between x and the true value $\frac{s}{r}$ is at most (since the t high order bits are correct:

$$\left|x - \frac{s}{r}\right| < \frac{1}{2^{2L+1}} = \frac{1}{2 \cdot (2^L)^2} \leq \frac{1}{2N^2} \leq \frac{1}{2r^2}$$

Since N can be expressed using L bits and $r < N$. Thus the necessary conditions for using the continued fractions expansion to find the numbers s and r are satisfied [NC00] pp. 637. This might give rise to a problem, namely the possibility that r and s have a common factor, and since the continued fractions expansions finds irreducible fractions, no order will be found. This can be overcome in two ways.

The naive way The obvious way would be to hope that s and r are coprime. In this case the correct value would be obtained (since nothing could factor out). The prime number theorem [Pri03] states that the number of primes below r (roughly) equals $\frac{r}{\log r}$. So the probability that a uniformly chosen element in $\{2, 3, \ldots, r-1\}$ would be prime is approximately $\frac{1}{\log r}$ which is at least $\frac{1}{\log N}$. And since any prime number would be coprime to r this is of course a lower bound on the probability of having a number coprime to r. Thus running the procedure described above $\Theta(\log N)$ times would expectedly give a correct solution.

The (maybe) more sophisticated way A more subtle way of doing this is described in [NC00] pp. 231. The order finding procedure is executed twice, and for each pair of fractions (s_1, r_1) and (s_2, r_2) in the sequence of convergents, $\gcd(s_1, s_2)$ is computed. If this is 1, it is claimed, the order can be obtained as the least common multiple of r_1 and r_2. It is shown in [NC00] that the occurrence of having two sequences of continued fractions expansions such that there are s_1 and s_2 with $gcd(s_1, s_2) = 1$ happens with a probability at least 0.25. The good thing about this is that it requires only a constant number of applications of the order finding procedure. There is just one problem: According to my simulation (described in the next chapter) it does not seem to work. In Section A.6 I have printed the output of my simulator factoring the number 21. The two lists resulting from the continued fractions expansions are $(numerator, denominator)$:

$$convs1 : [(1,1), (1,2), (3,5), (4,7), (15,26), (19,33), (34,59), (87,151), (295,512)]$$
$$convs2 : [(1,1), (127,128)]$$

Notice that $gcd(3, 127) = 1$ and the least common multiple of 5 and 128, 640 is not even a multiple of the order! I will emphasize, that I do *not* claim that the trick is not an improvement over the naive approach, it might even be that it still only requires a constant number of applications of the order finding function. What I am claiming is that s_1 and s_2 being coprime *is not* a sufficiently strong condition to conclude that the order is $lcm(r_1, r_2)$.

1.2.2. Running Time

Order finding

The first part (application of the hadamard gates) can of course be done in time $O(1)$. [CLRS01] describes a (classic) algorithm for computing the modular exponentiation in $O(t^3)$ time, thus the second part (modular exponentiation) can be done using $O(t^3)$ Toffoli gates, (according to Section 1.1.4). The third part (applying inverse Quantum Fourier Transformation) can be done using $O(t^2)$ gates (as previously argued). Computing the continued fractions expansion can be done in time $O(t^3)$ ([NC00] pp. 637). If the "trick" described above is used, an additional running time of $O(t^2)$ is introduced. This is due to the fact that the lengths of the convergent sequences are $O(t)$, which is clearly dominated by the $O(t^3)$. If the trick is not used, an extra factor of $O(\log(N))$ is introduced

The total expected running time of the order finding algorithm with bounded error probability is therefore $O(\log(N)^4)$.

Shor's algorithm

It can be shown ([NC00]) that the probability for the order to be even and $x^{\frac{1}{2}} \not\equiv -1 \mod N$, can be bounded from below, giving that the probability of actually finding an order (and one that works with reduction) is at least a constant. Thus Shor's algorithm for factoring composite numbers finds a nontrivial divisor to a number N in time $O(\log(N)^4)$ with a bounded probability of error.

1.3. Grover's Algorithm

A standard problem is the one of searching for elements in an unstructured database. More formally: given indicies $0, 1, \ldots, N-1$ and a boolean function f mapping each index to either "yes" or "no", find a good element. (Ie. an index i such that $f(i) =$ "yes"). Using simple adversary argumentation, it is easy to prove that the classical complexity of this algorithm is $\Omega(N)$. Therefore it was a significant discovery when Lov Grover in 1996 published an $O(\sqrt{N})$ algorithm for finding such an element ([Gro96]).

1.3.1. Description of the algorithm

(This description and analysis follows those given in [Nak08] and [NC00].) Let A denote the set of good elements. Let $d = |A|$. To model the problem in the framework of quantum computation, it is assumed that we have a so called "Grover Oracle",

performing the mapping:

$$|x\rangle \rightarrow \begin{cases} -|x\rangle & x \in A \\ |x\rangle & otherwise \end{cases}$$

For example, applied to the state

$$|\Psi\rangle = \frac{1}{\sqrt{2^n}} \sum_{j=0}^{2^n-1} |j\rangle$$

the result would be

$$\frac{1}{\sqrt{2^n}} \sum_{j=0}^{2^n-1} |j\rangle - \frac{2}{\sqrt{2^n}} \sum_{z \in A} |z\rangle$$

The algorithm works by repeating the so called Grover Iteration (see below) a number of times. A pseudo code for the algorithm is shown as "Algorithm 2".

Algorithm 2: Grover's Algorithm
Input: A Grover Oracle O, and size of search space.
1: Create a new quantum register with sufficiently many qubits to index the search space
2: Put the register in state $|0\rangle$
3: Apply the Walsh-Hadamard transformation to the register
4: Apply the Grover Iteration an appropriate number of times
5: Measure the register

The Grover iteration is done in four steps:

1. Apply the Grover Oracle O
2. Apply Hadamard gates to all bits
3. Apply the selective phase shift (described below)
4. Apply Hadamard gates to all bits

Thus in terms of matrices this is $H^{\otimes n} P_0 H^{\otimes n} O$. A diagram showing this iteration is given in [Nak08], and In Figure 1.3.1. The "Selective phase shift" changes the sign of all coefficients, except for the coefficient of vector $|0\rangle$, ie. it performs the mapping:

$$|x\rangle \rightarrow \begin{cases} |x\rangle & x = 0 \\ -|x\rangle & otherwise \end{cases}$$

In terms of matrices, it can be expressed as

$$P_0 = 2\,|0\rangle\,\langle 0| - I$$

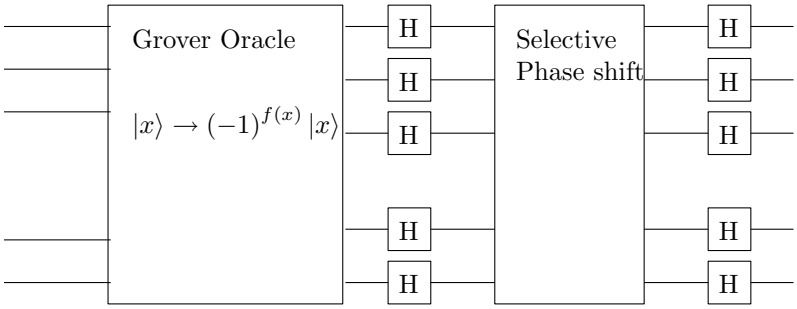

Figure 1.3.1: The Grover iteration, the basic block of the Grover Algorithm. "H" denotes a Hadamard gate.

The total effect of steps 2, 3 and 4 can be simplified:

$$\begin{aligned} H^{\otimes n} P_0 H^{\otimes n} &= H^{\otimes n} \left(2 |0\rangle \langle 0| - I\right) H^{\otimes n} \\ &= 2H^{\otimes n} |0\rangle \langle 0| H^{\otimes n} - H^{\otimes n} I H^{\otimes n} \\ &= 2 |\phi_0\rangle \langle \phi_0| - I \end{aligned}$$

where $\phi_0 = \frac{1}{\sqrt{2^n}} \sum_{j=0}^{2^n-1} |j\rangle$. I use that the Walsh-Hadamard gate is its own inverse. Let $|\Psi\rangle = \sum_{j=0}^{2^n-1} w_j |j\rangle$. Observe that the effect of applying the operator $2 |\phi_0\rangle \langle \phi_0| - I$ to $|\Psi\rangle$ is:

$$\begin{aligned} \left(2 |\phi_0\rangle \langle \phi_0| - I\right) |\Psi\rangle &= 2 |\phi_0\rangle \langle \phi_0| |\Psi\rangle - |\Psi\rangle \\ &= 2 \begin{pmatrix} \frac{1}{2^n} & \cdots & \frac{1}{2^n} \\ \vdots & & \vdots \\ \frac{1}{2^n} & \cdots & \frac{1}{2^n} \end{pmatrix} |\Psi\rangle - |\Psi\rangle \\ &= 2 \begin{pmatrix} \frac{1}{2^n} & \cdots & \frac{1}{2^n} \\ \vdots & & \vdots \\ \frac{1}{2^n} & \cdots & \frac{1}{2^n} \end{pmatrix} \sum_{k=0}^{2^n-1} w_k |k\rangle - \sum_{k=0}^{2^n-1} w_k |k\rangle \\ &= 2 \sum_{k=0}^{2^n-1} \sum_{j=0}^{2^n-1} \frac{w_j}{2^n} |k\rangle - \sum_{k=0}^{2^n-1} w_k |k\rangle \end{aligned}$$

If $\bar{w} = \sum_{j=0}^{2^n-1} \frac{w_j}{2^n}$ (the mean value of the w_j's), the result equals:

$$\sum_{k=0}^{2^n-1} \left(2\bar{w} - w_k\right) |k\rangle = \sum_{k=0}^{2^n-1} \left(\bar{w} + (\bar{w} - w_k)\right) |k\rangle \tag{1.3.1}$$

Therefore this operation is said to be a "reflection about the mean". So loosely speaking if one element out of many has an amplitude much smaller than the mean

amplitude then this amplitude will be greatly increased (remember that these amplitudes are reals thus it makes sense to talk about ordering). And this is (informally) what happens when the Grover Oracle is applied. I will now elaborate on this in a more formal way.

Claim 1. Let d be the number of good elements, and N be the total number of elements. Let a_k and b_k respectively be the amplitudes of the good elements and the bad elements after Grover iteration k. Then they satisfy the following recursion equations:

$$\begin{aligned} a_k &= \tfrac{N-2d}{N}a_{k-1} + \tfrac{2(N-d)}{N}b_{k-1} \\ b_k &= -\tfrac{2d}{N}a_{k-1} + \tfrac{N-2d}{N}b_{k-1} \end{aligned} \tag{1.3.2}$$

Proof: Let the system be in a state

$$\phi = a_k \sum_{z\in A} |z\rangle + b_k \sum_{z\notin A} |z\rangle$$

After applying the Grover Oracle the state becomes:

$$-a_k \sum_{z\in A} |z\rangle + b_k \sum_{z\notin A} |z\rangle$$

The mean value is

$$\bar{w} = \frac{-da_k + (N-d)b_k}{N}$$

Using 1.3.1:

$$a_{k+1} = 2\bar{w} - (-a_k) = \frac{-2da_k + 2(N-d)b_k}{N} + a_k = \frac{N-2d}{N}a_k + \frac{2(N-d)}{N}b_k$$

and similarly

$$b_{k+1} = 2\bar{w} - (b_k) = \frac{-2da_k + 2(N-d)b_k}{N} - b_k = \frac{-2d}{N}a_k + \frac{N-2d}{N}b_k$$

∎

Note that this is not consistent with what is claimed in [Nak08], which apparently has a small error in that part. It is however consistent with [Hir04], although the proof I gave is quite different. Mika Hirvensalo has, by the way, also a minor error (but more obvious than the one in [Nak08]) in his outline of the proof (see section A.5).

This system of recursion equations can be solved explicitly.

Claim 2. The system 1.3.2 has a solution:

$$\begin{aligned} a_k &= \tfrac{1}{\sqrt{d}}\sin((2k+1)\theta) \\ b_k &= \tfrac{1}{\sqrt{N-d}}\cos((2k+1)\theta) \end{aligned}$$

Where θ satisfies that $\sin\theta = \sqrt{\frac{d}{N}}$. This means that $\cos\theta = \sqrt{1-\frac{d}{N}}$

Proof: By induction on k. Basis step omitted since it is trivial. Suppose that the relations holds for some integer k. We want to show that

$$\begin{aligned} a_{k+1} &= \tfrac{1}{\sqrt{d}}\sin((2(k+1)+1)\theta) \\ b_{k+1} &= \tfrac{1}{\sqrt{N-d}}\cos((2(k+1)+1)\theta) \end{aligned}$$

Part a: The right hand side:

$$\frac{1}{\sqrt{d}}\sin((2(k+1)+1)\theta) = \frac{1}{\sqrt{d}}\sin((2k+1)\theta+2\theta)$$

Using trigonometric additions formula this becomes:

$$\frac{1}{\sqrt{d}}\sin((2k+1)\theta)\cos(2\theta) + \frac{1}{\sqrt{d}}\cos((2k+1)\theta)\sin(2\theta)$$

The induction hypothesis gives that $\frac{1}{\sqrt{d}}\sin((2k+1)\theta) = a_k$ and that $\cos((2k+1)\theta) = \sqrt{N-d}b_k$.

$$a_k\cos(2\theta) + \frac{1}{\sqrt{d}}\sqrt{N-d}b_k\sin(2\theta)$$

Using trigonometric identities:

$$a_k\left(1-2\sin^2\theta\right) + \frac{1}{\sqrt{d}}\sqrt{N-d}b_k 2\cos(\theta)\sin(\theta)$$

Using the assumption on θ:

$$\begin{aligned} & a_k\left(1-\frac{2d}{N}\right) + \frac{1}{\sqrt{d}}\sqrt{N-d}b_k 2\sqrt{1-\frac{d}{N}}\sqrt{\frac{d}{N}} \\ = \quad & \left(1-\frac{2d}{N}\right)a_k + \frac{2(N-d)}{N}b_k \end{aligned}$$

Which exactly corresponds to the left hand side of equation 1.3.2.

Part b: (a little less detailed - the strategy is exactly the same) The right hand side:

$$\begin{aligned} \frac{1}{\sqrt{N-d}}\cos((2k+1)\theta+2\theta) &= b_k\cos(2\theta) - \frac{\sqrt{d}}{\sqrt{N-d}}\sin(2\theta)a_k \\ &= -2\sqrt{\frac{d}{N-d}}\sin(\theta)\cos(\theta)a_k + \left(2\cos^2\theta-1\right)b_k \\ &= -2\sqrt{\frac{d}{N-d}}\sqrt{\frac{d}{N}}\sqrt{1-\frac{d}{N}}a_k + \left(2\left(1-\frac{d}{N}\right)-1\right)b_k \\ &= \frac{-2d}{N}a_k + \frac{N-2d}{N}b_k \end{aligned}$$

Which again equals the left hand side of the equation. ∎

1.3.2. Running Time and Correctness

Correctness (This analysis is based on the one given in [Nak08]. So how many times must one use the Grover iteration? Of course a_k should be as large as possible. So we want to maximize $\frac{1}{\sqrt{d}}\sin((2k+1)\theta$, which of course happens when $(2k+1)\theta) = \frac{\pi}{2}$ thus the number of applications of the Grover Iteration is

$$\tilde{m} = \frac{\pi}{4\theta} - \frac{1}{2}$$

which is not necessarily an integer. So let m denote the integer closest to $\tilde{m}$. This of course implies that

$$|m - \tilde{m}| \leq \frac{1}{2}$$

This implies that

$$\left|(2m+1)\theta - \frac{\pi}{2}\right| = |(2m+1)\theta - ((2\tilde{m}+1)\theta| = |((2m+1) - 2\tilde{m} - 1)\theta| \leq \theta \quad (1.3.3)$$

There are d good elements, and after Grover iteration k they are each observed with probability:

$$\left(\frac{1}{\sqrt{d}}\sin((2k+1)\theta)\right)^2 = \frac{1}{d}\sin^2((2k+1)\theta)$$

So the probability that one out of the d elements is observed is $\sin^2((2k+1)\theta)$. Which can be written as:

$$1 - \cos^2((2m+1)\theta) = 1 - \sin^2\left((2m+1)\,\theta - \frac{\pi}{2}\right)$$

And notice that since $0 < \theta < \frac{\pi}{2}$, according to 1.3.3:

$$\sin^2\left((2m+1)\,\theta - \frac{\pi}{2}\right) \leq \sin^2\theta$$

So

$$1 - \sin^2\left((2m+1)\,\theta - \frac{\pi}{2}\right) \geq 1 - \sin^2\theta = 1 - \frac{d}{N}$$

This is by the way very counterintuitive since one would not expect this probability to decrease when d is increasing! But when $d < \frac{N}{2}$ (otherwise no algorithm is needed) the probability of error is certainly at most $\frac{1}{2}$.

Running Time Notice that the number of Grover iterations m is at most $\frac{\pi}{4\theta} + 1$. And since θ is positive, $\sin\theta \leq \theta$, thus:

$$m \leq \frac{\pi}{4\theta} + 1 \leq \frac{\pi}{4\sin\theta} + 1 = \frac{\pi}{4}\sqrt{\frac{N}{d}} + 1 = O\left(\sqrt{\frac{N}{d}}\right)$$

This concludes, that after $O\left(\sqrt{\frac{N}{d}}\right)$ Grover iterations a good solution is measured with a probability at least $1 - \frac{d}{N}$. It has, by the way, been proved ([BBHT96]) that Grover's algorithm is asymptotically optimal.

1.4. Extensions of Grovers algorithm

The purpose of this section is to describe two quantum algorithms, one for the minimum spanning tree problem, and one for the single source shortest paths problem. Both of the algorithms use the "find d minimum of different kind" subroutine which is also described. This subroutine uses a generalization of Grover's algorithm. All of the algorithms, except for the generalization of Grover's algorithm, are from [DHHM06].

1.4.1. G-BBHT, A generalization of the Grover algorithm

A downside to Grover's Algorithm is obviously that it seldom occurs that one actually knows the number of solutions a priori. So one has actually no chance of using the Grover Oracle an appropriate number of times. An algorithm for finding a good solution amongst an unknown number of good solutions is described in [BBHT96]. The expected running time of the algorithm is $O\left(\sqrt{\frac{N}{d}}\right)$, where d is the number of good solutions and N is the size of the search space. I will, as it is done several times in literature (e.g. [Gru99]), call this algorithm G-BBHT, G is for Grover, and the other letters are for Boyer,Brassard, Høyer and Tapp - the four credited inventors of the algorithm

1.4.2. Find d minimum of different kind

The problem is about finding d minima of a function where all the minima must be different in some sense. More precisely:

Given a set $\{0, 1, \ldots, N-1\}$ and two functions f and g, find the minimum d elements (according to f) such that for all chosen indices i, j, if $i \neq j$ then $g(i) \neq g(j)$. The function f is said to define the *value* of an element, and the function g is said to define the *kind* or *type*. The terms kind and type are used interchangeably in the article and I shall do the same.

The algorithm works by maintaining a working set I, and using the G-BBHT algorithm to find an improving element. Notice that an element j is *improving* if one of the two requirements is satisfied:

- There is no element in $i \in I$ with $g(i) = g(j)$ and there exists some $k \in I$ such that $f(j) < f(k)$
- There exists an element $k \in I$ with $g(k) = g(j)$ and $f(j) < f(k)$.

A pseudo code sketch of the algorithm is given on Algorithm 3.

Algorithm 3: Algorithm for finding the d minimum of different kind

Input: d, functions f and g

1: Let I initially be a set of d elements such that $f(i) = \infty$ for all $i \in I$, and let $g(i)$ attain distinct values for all $i \in I$.
2: **repeat**
3: Find an improving element using the G-BBHT algorithm.
4: Insert the new improving element, and remove the appropriate element. (i.e. either the element of same type or the element with highest value)
5: **until** stopped

Expected running time I will now prove, that after an expected number of $O(\sqrt{dN})$ applications of the function f the set I is optimal. This is a more detailed version of the proof given in [DHHM06]. In the following let T_k denote the set of improving elements after iteration k, and let $t = |T_0|$. Furthermore let y_{mid} denote the lower median of T_0 and let $low(M)$ denote the number of elements in M less than or equal to y_{mid}.

Fact 1. If $low(T_k) < \frac{t}{4}$ at least a fraction of $\frac{1}{4}$ of the initial t elements has been ruled out.

Proof: Suppose $low(T_k) < \frac{t}{4}$, then at least half of the values with value less than or equal to y_{mid} cannot be improving elements. An illustration is given in figure 1.4.1. ∎

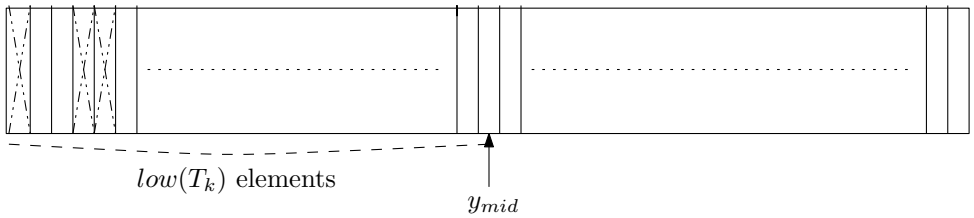

Figure 1.4.1: Figure showing the elements ordered by the ordering induced by f. Marked elements are "nonimproving" elements. At least half of the elements left to y_{mid} should be marked if $low(T_k) < \frac{t}{4}$

Fact 2. If $low(I_k) = d$ then at least half of the initial t elements are ruled out.

Proof: Suppose $low(I_k) = d$, then all elements in the index set are less than or equal to the median. This gives that with certainty no element larger than the median can be an improving element. Thus the "upper half" (according to the ordering induced by f) are ruled out. ∎

With these two facts in mind the next lemma is crucial:

Lemma 1.4.1. *At each iteration in the for-loop with probability at least $\frac{1}{32}$ one of the following two things happens:*

- $low(T_{k+1}) \leq low(T_k)\left(1 - \frac{1}{32d}\right)$
- $low(I_{k+1}) = low(T_k) + 1$

Thus after $O(d)$ iterations, a fraction of at least $\frac{1}{4}$ of the initial good solutions have been ruled out.

Proof: Firstly assume that $low(T_k) \geq \frac{t}{4}$. If not, no proof is necessary since the number of improving indicies would be at most $\frac{3t}{4}$. Since a new element is chosen uniformly among all the improving elements, the probability for choosing an element with value less than or equal to y_{mid} is at least $\frac{1}{4}$.

Case 1 Assume that the majority of the elements in T_k with value less than or equal to y_{mid} is of unknown type ie. having a type not represented in I. That is the probability for choosing a new improving element of unknown kind with value less than or equal to y_{mid} is at least $\frac{1}{8}$. In this case $low(I_{k+1}) = low(I_k) + 1$.

Case 2 Assume now that the majority of the elements in T_k are of known kind. Then the probability of choosing a new improving element of known kind with value less or equal to y_{mid} is at least $\frac{1}{8}$. I now claim that with a probability of at least $\frac{1}{2}$ there are $\frac{low(T_k)}{4d}$ or more elements of the same kind as the chosen element. Note that since $\frac{low(T_k)}{4}$ is a lower bound on the number of elements with value less than or equal to y_{mid}, $\frac{low(T_k)}{4d}$ is a lower bound on the average number of elements pr. known kind with value no greater than y_{mid}.

Let B denote the set of known types with strictly less than $\frac{low(T_k)}{4d}$ elements less than or equal to y_{mid}. Let similarly G denote the set of known types with at least $\frac{low(T_k)}{4d}$ elements less than y_{mid}. So what I want to show is that it is at least as likely to draw an element with type in G as in B. Let $n(s)$ denote the number of elements with type s and value less than or equal to y_{mid}. So the claim can be rephrased (since new elements are chosen uniformly):

$$\sum_{s \in B} n(s) \leq \sum_{s \in G} n(s)$$

And since at least half of the elements in T_k are known, the following inequality is valid:

$$\sum_{s \in G} n(s) \geq \frac{low(T_k)}{2} - \sum_{s \in B} n(s)$$

so it suffices to show:

$$2\sum_{s\in B} n(s) \leq \frac{low(T_k)}{2} \tag{1.4.1}$$

Assume for the sake of contradiction that 1.4.1 does not hold. Since $\frac{low(T_k)}{4d}$ is an upper bound of the size of each element in B this implies that:

$$\begin{aligned} 4|B|\frac{low(T_k)}{4d} &> low(T_k) \\ &\Downarrow \\ |B| &> d \end{aligned}$$

Recall that d is the size of the set I and all elements in I have different kinds. This of course implies that there can be at most d known types, but all types in B were by definition known, contradicting that $|B| > d$.

So with a probability of at least $\frac{1}{16}$ the element chosen is an element of which there are at least $\frac{low(T_k)}{4d}$ elements with value at most y_{mid}. Conditioned this happens with probability at least $\frac{1}{2}$ the element chosen is smaller than half of the other elements of that type, and thus $\frac{low(T_k)}{2\cdot 4d} \geq \frac{t}{4\cdot 2\cdot 4d} = \frac{t}{32d}$ elements have been eliminated, proving the lemma. ∎

Running Time So after $O(d)$ iterations of the for-loop at least $\frac{1}{4}$ of the elements have been ruled out. Thus the running time for this part is $O(d\sqrt{\frac{N}{t}})$ where t is the number of improving solutions. So the number of times i we have to do $O(d)$ iterations of the main loop before $t \leq 2d$ is obtained by the equation:

$$\left(\frac{3}{4}\right)^i \cdot N = 2d$$

Having the solution:

$$i = \frac{\log\left(\frac{2d}{N}\right)}{\log(\frac{3}{4})}$$

So the number of applications of the functions f and g for this is at most:

$$d \sum_{i=0}^{\frac{\log\left(\frac{2d}{N}\right)}{log(\frac{3}{4})}} \sqrt{\frac{N}{\left(\frac{3}{4}\right)^i N}} = \sum_{i=0}^{\frac{\log\left(\frac{2d}{N}\right)}{log(\frac{3}{4})}} \sqrt{\left(\frac{4}{3}\right)^i}$$

And with a little help from Maple, this sum can be written explicitly:

$$\begin{aligned} &d\cdot 3\sqrt{3^{-\left(\ln(2)-\ln\left(\frac{d}{N}\right)-\ln(3)\right)(2\ln(2)-\ln(3))^{-1}}4^{\left(\ln(2)-\ln\left(\frac{d}{N}\right)-\ln(3)\right)(2\ln(2)-\ln(3))^{-1}}} \\ +d\cdot 2&\sqrt{3^{-\left(\ln(2)-\ln\left(\frac{d}{N}\right)-\ln(3)\right)(2\ln(2)-\ln(3))^{-1}}4^{\left(\ln(2)-\ln\left(\frac{d}{N}\right)-\ln(3)\right)(2\ln(2)-\ln(3))^{-1}}}\sqrt{3}-3-2\sqrt{3} \end{aligned}$$

Which is

$$\begin{aligned} & O\left(d\sqrt{4^{\left(\ln(2)-\ln\left(\frac{d}{N}\right)-\ln(3)\right)(2\ln(2)-\ln(3))^{-1}}}\right) \\ = \; & O\left(d\sqrt{4^{\ln\frac{N}{d}}}\right) \\ = \; & O(\sqrt{dN}) \end{aligned}$$

When $t \leq 2d$, an upper bound on the work done is as follows (at least one element will be removed in each iteration):

$$\sum_{i=1}^{2d} \sqrt{\frac{N}{i}} = \sqrt{N} \sum_{i=1}^{2d} \frac{1}{\sqrt{i}} = O(\sqrt{N}\sqrt{d}) = O(\sqrt{dN})$$

Thus the total expected number of applications of the function f is $O(\sqrt{dN})$.

1.4.3. Quantum MST algorithm

The quantum algorithm for finding the minimum spanning tree of a graph is a modified version of Borůvka's algorithm ([NMN01]). It maintains a spanning forest, and merges some of the trees until just one tree is left. Quantum algorithmics is used for the part of finding which trees to merge. Adjacency list representation of the graph is used. The pseudo code of the algorithm is on Algorithm 4. It uses the invariant that $T_1, T_2, \ldots, T_k$ is a spanning forest. Ie. the algorithm finishes exactly when $k = 1$.

Algorithm 4: Quantum algorithm for finding the minimum spanning tree of a graph.
Input: a graph G, with n vertices and m edges, and a weight function w mapping each edge to a weight.

Initially $k = n$, all trees in the spanning forest are singletons
$l \leftarrow 0$
while $k \neq 1$ **do**
  $MST \leftarrow \varnothing$
  $l \leftarrow l + 1$
  Use quantum algorithm *find d minimum of different kind*, where $d = k$ where the value of each edge is given by $w(u, v)$ if u and v are not already connected and ∞ if they are. Let the kind of edge (u, v) be the source vertex u. Stop after $(l + 2)\sqrt{km}$ queries. In this way k edges between trees not already connected is found.
  for each edge (u, v) found **do**
    if u and v are in two different trees **then**
      Merge the trees containing u and v
      $MST \leftarrow MST \cup \{(u, v)\}$
      $k \leftarrow k - 1$
    end if
  end for
end while

Correctness The probability that the "find d minimum of different kind" algorithm produces a wrong answer in iteration l is at most $\frac{1}{2^{l+2}}$ because of the $l + 2$ factor. This is justified in [DHHM06]. So the probability of error is at most:

$$\sum_{l=1}^{\infty} \frac{1}{2^{l+2}} = \frac{1}{4}$$

Query Complexity Analysis Note that in iteration l there are at most $\frac{n}{2^{l-1}}$ trees, since there are initially n trees, and the number of trees is at least reduced by a factor $\frac{1}{2}$. The number of applications of the function f in iteration l is therefore $O((l+2)\sqrt{\frac{nm}{2^{l-1}}})$. Thus an upper bound of the total number of queries to the function f is:

$$\sum_{l=1}^{\infty}(l + 2)\sqrt{\frac{nm}{2^{l-1}}} = 2(5 + 3\sqrt{2})\sqrt{nm} = O(\sqrt{nm})$$

For an analysis of the actual running time of this algorithm, see Section 2.6.

1.4.4. Quantum SSSP algorithm

The quantum algorithm for the single source shortest paths problem is variant of Dijkstras algorithm [CLRS01]. As in Dijkstras algorithm a single source shortest

paths tree is maintained, and for all vertices in the tree the shortest path from the source is completely within the tree. Instead of maintaining a priority queue, quantum algorithmics is used to determine which vertices to add to the tree. Initially the single source shortest paths tree contains no edges and only covers the source vertex. There appears to be a minor flaw in the algorithm given in [DHHM06], since they use "weight" instead of the distance from the source to the target of that edge. So as in Dijkstra's algorithm, the distance from the source to any vertex in the SSSP-tree must be maintained - in my case I use the array d. Algorithm 5 sketches my interpretation of what is meant in the article.

Algorithm 5: Single source shortest path algorithm
Input: Adjacency list represented Graph $G = (V, E)$, $|V| = n$ and $|E| = m$, with a weight function w, a source vertex v_0.

1: $d[v] \leftarrow \infty$ for all $v \in V$
2: $d[v_0] \leftarrow 0$
3: $P_1 \leftarrow \{v_0\}$
4: $SSSPTree \leftarrow \varnothing$
5: $l \leftarrow 1$
6: **while** $SSSPTree$ does not cover all vertices in G **do**
7: (A) Use the *Find d minimum of different kind* subroutine to find the $|P_l|$ minimal edges with different target vertices. To do this use $f(u, v) = d[u] + w(u, v)$ if u and v are not connected and ∞ if they are. Use $g(u, v) = v$. Let this set of edges be denoted as A_l.
8: (B) Find the cheapest edge $e = (u, v)$ (as of the value specified before) in $A_1 \cup \ldots \cup A_l$ such that $v \notin P_1 \cup \ldots \cup P_l$
9: $SSSP \leftarrow SSSP \cup \{(u, v)\}$
10: $d[v] \leftarrow d[u] + w(u, v)$
11: $P_{l+1} = \{v\}$
12: $l \leftarrow l + 1$
13: **while** $l \geq 2$ and $|P_{l-1}| = |P_l|$ **do**
14: Merge P_l and P_{l-1} to P_{l-1}
15: $l \leftarrow l - 1$
16: **end while**
17: **end while**

So roughly speaking what the algorithm does is that it partitions the SSSPTree into sets $P_1, \ldots, P_l$ all having sizes of powers of two and $|P_1| > |P_2| > \ldots > |P_l|$. In each iteration it finds the cheapest edges going out from P_l and after that selects the minimal edge of all edges going out any of the P_i-sets and adds the target vertex of that edge to P_{l+1} and adds the particular edge to the SSSP tree.

Query Complexity Analysis and Correctness Firstly I will analyze the query complexity of part A, and the I will show that the running time of part B is not greater than that of part A.

Part A Observe that during the whole execution of the algorithm P_l will have size s at most $\frac{n}{s}$ times. This can be clarified by recalling that all of the P_i have sizes of powers of two. Thus the claim is actually stating that if all numbers from 1 to n are listed in binary representation the digit i will be the least significant nonzero digit at most $\frac{n}{2^i}$ times, which is certainly true. Thus for a fixed s the number of queries of function f (in the subroutine) is

$$\sum_{i=1}^{\frac{n}{s}} \sqrt{sm_i} \tag{1.4.2}$$

Where m_i is the number of edges exiting the vertex set i. During the entire execution of the algorithm any vertex will be in a set P_i with size s exactly once. Therefore the following equality is valid:

$$\sum_{i=1}^{\frac{n}{s}} m_i = m$$

And since 1.4.2 is a concave function, the sum were maximized if all m_i's were equally large, $m_i = \frac{m}{\frac{n}{s}} = \frac{sm}{n}$. Thus:

$$\begin{aligned} \sum_{i=1}^{\frac{n}{s}} \sqrt{sm_i} &\leq \sum_{i=1}^{\frac{n}{s}} \sqrt{s \cdot \frac{sm}{n}} \\ &= \frac{n}{s}\sqrt{s \cdot \frac{sm}{n}} \\ &= \sqrt{nm} \end{aligned}$$

And since all sizes are powers of two, at most $\log n$ different sizes can occur giving a running of $O(\log n\sqrt{nm})$. But this is not it. We must take account of the probability for failing. As described in [DHHM06] the method *Find d minimum of different kind* yields correct result (with constant probability) after an expected number of $c\sqrt{dN}$ queries, for some constant c. But if one instead uses $kc\sqrt{dN}$ queries, the probability of failure will be at most $1/2^k$.

Note that there are at most n sets of each size, and at most $\log n$ different sizes. Thus the number of calls to the subroutine is at most $n \log n$. Let p be the probability that a call to the subroutine succeeds. The probability that all the calls succeed are then

$p^{n \log n}$. And we want this to succeed with some constant probability, say $\frac{1}{2}$.

$$\begin{aligned} p^{n \log n} &\geq \frac{1}{2} \\ &\Downarrow \\ n \log n \log p &\geq \log \frac{1}{2} = -1 \\ &\Downarrow \\ \log p &\geq \frac{-1}{n \log n} \\ &\Downarrow \\ p &\geq \left(\frac{1}{2}\right)^{\frac{1}{n \log n}} \end{aligned}$$

Thus the probability of succeeding in each call to the *Find d minimum of different kind* must be at least $\left(\frac{1}{2}\right)^{\frac{1}{n \log n}}$. I now claim that it suffices to put $p \geq 1 - \frac{1}{2n \log n}$.

Proof: Observe that

$$1 - \frac{1}{2n \log n} \geq \left(\frac{1}{2}\right)^{\frac{1}{n \log n}}$$

If and only if

$$1 \geq \frac{1}{2n \log n} + \left(\frac{1}{2}\right)^{\frac{1}{n \log n}}$$

Call the right hand side of the inequality $f(n)$. Observe that the derivate of the f is:

$$f'(n) = \frac{1}{2} \frac{\ln(2) \left(-\ln(n) - 1 + 2\,\mathrm{e}^{-\frac{(\ln(2))^2}{n \ln(n)}} \ln(2) \ln(n) + 2\,\mathrm{e}^{-\frac{(\ln(2))^2}{n \ln(n)}} \ln(2) \right)}{n^2 (\ln(n))^2}$$

Which is strictly positive for $n > 1$. Observe similarly that

$$\lim_{n \to \infty} f(n) = 1$$

Thus $f(n) < 1$, proving the claim. ∎

Thus the number k can be chosen to be $\log(2n \log n) = O(\log n)$, introducing another factor of $O(\log n)$. Thus the algorithm has a query complexity of $O(\sqrt{nm} \log^2 n)$, and succeeds with a bounded error probability. For a running time analysis see Section 2.7.

Part B All the sets A_i can be stored as balanced search trees (using lexicographical ordering by value, weight, target and source vertex). Thus finding the lightest edge can be done in time $O(\log n)$ for each set, and since there is at most $\log n$ sets, the total work of finding the minimum is $O\left(\log^2 n\right)$. And since this shall be done n times, the total work for this step is $O\left(n \log^2 n\right)$.

Note however, that adding an edge (u, v) can make another edge (t, v) infeasible in the sense that the two edges have the same target vertex. Thus after adding a vertex, at most one edge from each A_i must be removed. The removal (not finding) of such an edge can be done in time $O\left(\log n\right)$ (for each A_i). Thus by using "lazy removal" (removing such an edge if it encountered) doesn't change the asymptotical running time, since it will be done at most n times, the work due to this will be $O(n \log^2 n)$, and thus dominated by the query complexity, since $m = \Omega(n)$, $O(n \log^2 n) = O(\sqrt{nm} \log^2 n)$. For at more practically oriented approach see 2.7.

Chapter 2

Simulation

As always it would be interesting to have a more concrete version of the algorithms, making one able to run them and observe that they actually work and are capable of performing their respective tasks. Unfortunately, due to technical and financial shortcomings, it was not possible for me to get acces to a real quantum computer. So therefore I have chosen the second best solution, namely to implement a quantum computer simulator, and then implement quantum algorithms using this simulator. This of course prevents me from actually gaining the running time benefits from the quantum algorithms (otherwise no quantum computer was needed!).

All the algorithms from the previous chapter have been implemented using the java programming language. In this chapter I will give a brief description of how I have chosen to represent a quantum system, and how this is used to simulate the algorithms. Data structures will be briefly presented, and a description will be given on how gate operations acting upon my quantum state are simulated. This is followed by explanation of how the algorithms can be simulated using a classical model of computation. Focus will be on the underlying ideas and not on the implementational details (since there are quite a lot!). For a brief sketch on the hierarchical structure of my program, see figure 2.0.1. The program should be available at www.gausdalfind.dk/Qsim.jar, and a small guide on how to download and run the program is given in Section A.1. In Section A.2 I have made a dump of a run of the program intending to show how it should be used.

The verb "implement" deserves a little warning, since it is a homonym. It can mean both the process of actually writing program code, but in the field of object oriented oriented programming the word (unfortunately) has another meaning, namely in the sense that given an interface a class can *implement* an interface by supporting the required operations and declaring the required variables. Since there really are no other adequate synonyms for either I will use the same word having two different meanings - I believe, however, that it is clear from the context what is meant.

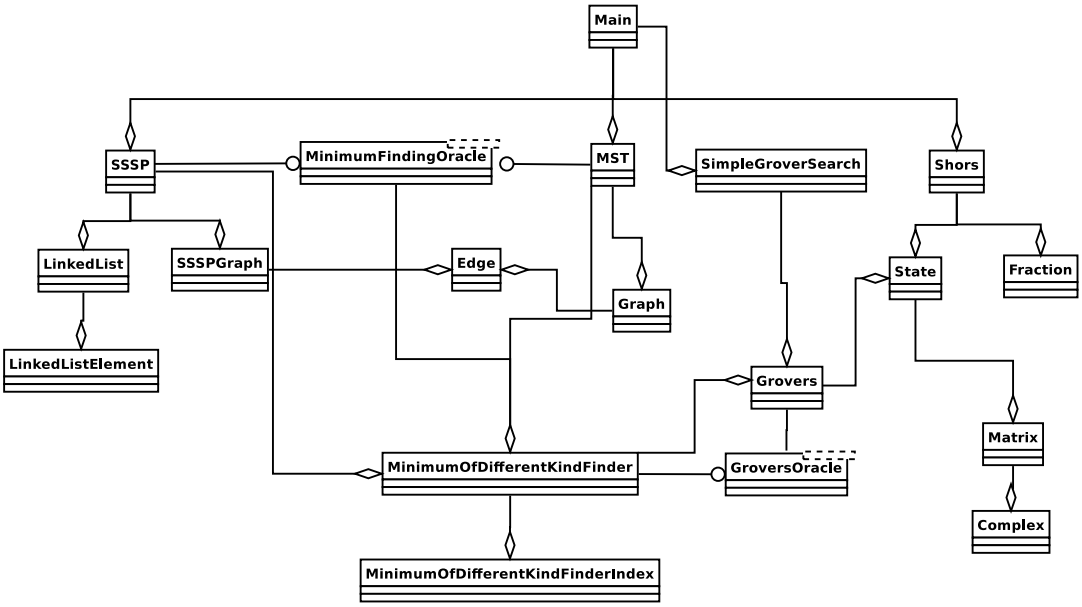

Figure 2.0.1: A UML style overview of my program. Some of the test and driver classes have been omitted for simplicity. Lines with circle denotes interface implementaion. A line with a diamond from A to B denotes that A "has a" B. Normal lines denote that the classes in some other way are in relationship to each other.

2.1. Simulating the quantum framework

2.1.1. Underlying data structures

A quantum system of n bits is really a vector in the complex vector space $\mathbb{C}^{2^n}$, and quantum gates are really unitary complex matrices. Both of these can then be modelled using matrices over complex numbers. The state vector is just a matrix with one single coloumn, and 2^n rows. Therefore a datastructure modeling complex numbers and their arithmetic and a data structure modeling matrices over complex numbers have been necessary. This data structure can of course handle standard matrix arithmetic (including matrix multiplication, addition, tensor multiplication). Complex numbers and matrices are implemented in a perfectly straightforward manner, and will not be given any further attention. Notice that since exponentially many rows in my state vector are maintained, my simulation will be exponentially slower, and use exponentially more space, than it would using a quantum computer.

2.1.2. Simulating gates operating on the entire state

Let $|v\rangle$ be the 2^n dimensional state vector of a quantum system with n qubits. A gate U operating on all bits is really a $2^n \times 2^n$ matrix. The resulting state is simply obtained by the matrix product $U|v\rangle$.

2.1.3. Simulating operations on one qubit

Assume that we want to operate a 2×2 matrix U on bit i in a quantum state consisting of $n > 1$ qubits. The naive method would be to obtain the tensor product $U' = I^{\otimes(i-1)} \otimes U \otimes I^{\otimes(n-i)}$, and the resulting vector would be the matrix product $U' |v\rangle$. Unfortunately this method is a very time- and memory consuming way since the matrix U' would have size $2^n \times 2^n$. This means that for $n = 10$, $2^{30} \approx 10^9$ complex additions would be necessary.

This gives motivation for a smarter approach. Remember that the state $|v\rangle$ can be represented as a linear combination of the computational basis as:

$$|v\rangle = \sum_{j=0}^{2^n-1} a_j |j\rangle$$

Let f be a function that operates matrix U on bit i on any of the basis states. Due to the fact that gates are linear operators, the result of applying U on bit i on the superposition is:

$$\sum_{j=0}^{2^n-1} a_j f(|j\rangle)$$

A simple algorithm computing f is shown on Algorithm 6. This is clearly much better than the naive approach since a constant amount of time is used for each basis vector - this gives a running time of $O(2^n)$, giving an exponential speedup.

Algorithm 6: Computes the result of applying operator U on bit i in basis vector n. $\oplus$ denotes bitwise XOR
Input: Matrix U, basis vector $|n\rangle$, bitindex i

1: **if** bit i in number n is 1 **then**
2: **return** $U_{10} \cdot \left|n \oplus 2^i\right\rangle + U_{11} \cdot |n\rangle$
3: **else**
4: **return** $U_{00} \cdot |n\rangle + U_{01} \cdot \left|n \oplus 2^i\right\rangle$
5: **end if**

Controlled version: Applying a controlled bit operation instead is done in mostly the same way. Firstly it should be checked whether the control bit is 1, if so Algorithm 6 is applied, otherwise the basis vector $|n\rangle$ is returned.

2.1.4. Measurement

When performing a measurement, a basisvector is chosen according to the magnitude of their amplitudes. If a basis vector has the amplitude a the propability of choosing that vector should be $|a|^2$. After the measurement the superposition should "collapse" - meaning that the amplitude of the selected vector should be 1 and all other amplitudes should be 0. An algorithmic sketch is given (algorithm 7).

Algorithm 7: Performs a measurement in the computational basis
Input: State vector a
1: $r \leftarrow$ random number chosen uniformly from $(0, 1]$
2: $s \leftarrow 0$
3: $i \leftarrow -1$
4: **while** $s < r$ **do**
5: $\quad i \leftarrow i + 1$
6: $\quad s \leftarrow s + |a_i|^2$
7: **end while**
8: $a_i \leftarrow 1$
9: $a_j \leftarrow 0$ for $j \neq i$

2.2. Simulation of Shors algorithm

The simulation of Shors algorithm is divided into several subtasks. The main factoring algorithm is done in quite the same way as described in [NC00]. This really is a non-quantum algorithm, so I will not give it further attention here.

2.2.1. Order-finding

The order finding algorithm makes use of two registers. Register 1 with $t = 2L + 1 + \left\lceil \log\left(2 + \frac{1}{2\epsilon}\right) \right\rceil$ bits, and register 2 with L bits. Since these two registers in general are entangled they can not be simulated as two separate "states", instead one larger register will be used handling both "subregisters", such that the L low order bits correspond to register 2 and the t high order bits correspond to register 1 (cf. figure 1.2.1). An algorithmic sketch of the order-finding algorithm is available (algorithm 8).

Algorithm 8: Order-finding algorithm, not using the "trick" from Section 1.2.1
Input: An L-bit integer N, probability of error ϵ, integer x
Output: the least integer r such that $x^r \equiv 1 \mod N$

1: $t = 2L + 1 + \left\lceil \log\left(2 + \frac{1}{2\epsilon}\right)\right\rceil$
2: Make a quantum state s with $t + L$ qubits
3: put s in the state $|1\rangle$
4: **for** $i \leftarrow 1$ to t **do**
5: apply hadamard gate on bit $t + L - i$
6: **end for**
7: Perform modular exponentiation, i.e. $|x\rangle|1\rangle \rightarrow |x\rangle|x^z \mod N\rangle$ using classical algorithm
8: Apply Inverse Fourier Transformation on register 1
9: Perform measurement
10: $C \leftarrow$ Convergents from continued fractions expansion of value in register 1
11: **for** each $\frac{s}{r} \in C$ **do**
12: **if** $x^r \equiv 1 \mod N$ **then**
13: **return** r
14: **end if**
15: **end for**
16: **return** failure (no order found)

2.2.2. (Inverse) Fourier Transformation

At page 219 in [NC00] a quantum circuit for the Quantum Fourier transformation is shown. The inverse Fourier transformation can then be obtained by reversing the order of the gates and complex conjugating all gates, as mentioned in 1.2.1.

This gives rise to algorithm 9. It uses the `apply P-gate` procedure explained later in this section. The matrix $\bar{R}_j$ equals:

$$\begin{pmatrix} 1 & 0 \\ 0 & e^{-2\pi i/2^j} \end{pmatrix}$$

Algorithm 9: Algorithm performing Inverse Fourier Transformation on the k last bits (that is, high order) of a quantum register with n bits.

1: apply P-gate on the last k bits
2: **for** $i \leftarrow 0$ to $k - 1$ **do**
3: **for** $j \leftarrow i + 1$ down to 2 **do**
4: apply controlled $\bar{R}_j$ gate, target bit $n - i - 1$, control bit $n - i - j$
5: **end for**
6: Apply Hadamard gate on bit i
7: **end for**

2.2.3. P-gate

The P-gate reverses the order of the last k bits, ie. applied on the bit sequence $b_{n-1}b_{n-2}\dots b_{n-k}b_{n-k-1}\dots b_1b_0$ should be $b_{n-k}\dots b_{n-2}b_{n-1}b_{n-k-1}\dots b_1b_0$.

The gate can be implemented in several ways. In a real quantum computer, swap gates can be realized using controlled-not operations. Since I am merely simulating, I have chosen a slightly different approach, algorithmic sketch is given (algorithm 10). This procedure is applied on all basis vectors. I first developed a very simple algorithm to reverse the order of the *first* (low order) k bits, and my algorithm works by first reversing the order of all bits (line 1 to 6), then reversing the first k bits (line 7 to 12), and then reversing all bits (line 13 to 18).

Algorithm 10: Algorithm reversing the order of the last k bits in the bit representation of an n bit basis state.
Input: an $n-$bit number a, and the number of bits k to reverse.

1: $a_{rev} \leftarrow 0$
2: **for** $i \leftarrow 1$ to n **do**
3: $\quad a_{rev} \leftarrow a_{rev} * 2$
4: $\quad a_{rev} \leftarrow a_{rev} + (a \mod 2)$
5: $\quad a \leftarrow \lfloor \frac{a}{2} \rfloor$
6: **end for**
7: $o \leftarrow \lfloor \frac{a_{rev}}{2^k} \rfloor$
8: **for** $i \leftarrow 0$ to $k-1$ **do**
9: $\quad o \leftarrow o \cdot 2$
10: $\quad o \leftarrow o + (a_{rev} \mod 2)$
11: $\quad a_{rev} \leftarrow \lfloor \frac{a_{rev}}{2} \rfloor$
12: **end for**
13: $res \leftarrow 0$
14: **for** $i \leftarrow 1$ to n **do**
15: $\quad res \leftarrow res * 2$
16: $\quad res \leftarrow res + (o \mod 2)$
17: $\quad o \leftarrow \lfloor \frac{o}{2} \rfloor$
18: **end for**
19: **return** res

2.2.4. Modular exponentiation

Modular exponentiation is done using the algorithm from page 176 in [Ros95].

2.2.5. Continued fractions algorithm

The algorithm for performing continued fractions is directly taken from page 636 in [NC00]. The procedure computes all the convergents and returns them as a list of tuples.

2.3. Simulation of Grovers Algorithm

Simulating Grovers algorithm is slightly simpler than simulating Shor's. A data structure simulating the "Grover oracle" is given as a parameter, ie. a method performing the operation:[1]

$$|x\rangle \rightarrow \begin{cases} -|x\rangle & \text{If x is a good/seeked element} \\ |x\rangle & \text{Otherwise} \end{cases}$$

In practice I have represented the "Grover Oracle" as an Interface, that can be implemented by several different classes. This is practical since Grover's algorithm can be used for several different purposes.

My simulation takes three algorithm parameters: the Grover oracle, an integer specifying the number of bits used to index the search space and the number of Grover iterations. The pseudo code for my Grover simulation is given on Algorithm 11. The algorithm uses the subroutine "selective phase shift" this is done by multiplying the amplitude of all basis vectors except $|0\rangle$ with -1 (as described in Section 1.3).

Algorithm 11: Grovers Algorithm
input: Grover oracle *go*, number of qubits n, number of Grover iterations $nrIts$
1: Make a new quantum state with n qubits, and put it in state $|0\rangle$
2: **for** $j \leftarrow 0$ to $n-1$ **do**
3: apply hadamard gate on bit j
4: **end for**
5: **for** $i \leftarrow 0$ to $nrIts$ **do**
6: apply *go* on state
7: **for** $j \leftarrow 0$ to $n-1$ **do**
8: apply Hadamard gate on bit j
9: **end for**
10: perform selective phase-shift
11: **for** $j \leftarrow 0$ to $n-1$ **do**
12: apply Hadamard gate on bit j
13: **end for**
14: **end for**
15: **return** Measurement value

2.4. G-BBHT

The extended version of Grovers algorithm has been implemented exactly as described in [BBHT96]. It takes a Grover Oracle as parameter, and the code can be found in

[1] Technically it would suffice just to send as parameter a function determining whether elements are improving would suffice, that is however not the way I chose.

the class `Grovers`, with the name `extendedGroverSearchUnknownSolNumber`. Since the implementation is done "out of the box", I will not give it any further attention.

2.5. Find d of different kinds

In this section I will describe how I have chosen to implement the *Find d of different kinds* algorithm. My focus will be on how I have made the algorithm interact with the `G-BBHT` algorithm, and how the set I is maintained. It should be noted, that in the theoretical treatment of this algorithm (1.4.2 and [DHHM06]) it is proved that after an expected number of $c\sqrt{dN}$ iterations, the algorithm will yield the optimal values of I. Though this is indeed true, it is quite impractical since it is not easy to determine the correct value of c. Becaues of that I have made it a user parameter, so the user of my simulator has the possibility of choosing an appropriate "constant".

2.5.1. Implementing the Grover Oracle Interface

Since the algorithm in nature is quite generic I have made an interface, `minimum finding oracle` having the methods `kind` and `value` to support the two functions g and f respectively. Given this oracle and a set I, the `Grover Oracle` interface can be implemented. To be able to this, the class should implement a function to determine whether an element is "good" - in this case an improving element to the set I.

To do this in a reasonable time, two datastructures are maintained: `indexOfType` and `IndsInSetTree`. The former is an array of length e (number of types) where `indexOfType[j]` contains the index of the element with type j if such an element is in I, and otherwise -1. The `IndsInSetTree` is a balanced tree containing all indicies in I ordered lexicographically by their *value* and *index*. The only reason why *index* is used as a second key is to ensure uniqueness of comparing.

The `isGood` method needed to implement the Grover Oracle can thus be done as shown on Algorithm 12.

Algorithm 12: Algorithm determining whether an index is improving
input: index i, and a `minimumFindingOracle` mfo
1: $newIndKind \leftarrow mfoKind(i)$
2: $newIndValue \leftarrow mfoValue(i)$
3: **if** $indexOfType[newValueKind] \neq -1$ **then**
4: **return** $newIndValue < mfoValue(indexSet[indexOfType[newIndValue]])$
5: **else**
6: **return** $newIndValue < mfoValue(Largest(indsInTreeSet))$
7: **end if**

Assuming that the largest value in $indsInTreeSet$ can be retrieved in constant time (by maintaining a reference to the element), this whole procedure takes time proportional to the running time of `mfoKind` and `mfoValue`. In my implementation I use

the `TreeSet` class provided by the Java Runtime Environment, where this operation is supported in constant time.

2.5.2. Inserting a new element

Inserting an improving element is done in a quite straightforward manner: If the new element is of known kind, it replaces the element with the same kind (both in the index set I and the tree ordering of the index set). Otherwise if the new element is unknown, the element to be replaced is the largest element in I.

2.5.3. Running Time

Thus inserting a new element can be done in time $O(\log d)$ where d is the size of the set I. It is not a priori clear that this term will not be asymptotically more time consuming than what is done by the quantum computer. Observe however that for each call to the quantum subroutine at most one element can be inserted. Thus an estimate of the running time including the time for insertion is:

$$d \sum_{i=0}^{\frac{\log\left(\frac{2d}{N}\right)}{\log\left(\frac{3}{4}\right)}} \left(\sqrt{\frac{1}{\left(\frac{3}{4}\right)^i}} + \log(d) \right)$$

Which approximately is (thanks to Maple):

$$3.47\, d \left(1.85 \sqrt{3^{1.40+3.47 \ln\left(\frac{d}{N}\right)} 4^{-1.40-3.47 \ln\left(\frac{d}{N}\right)}} - 0.4 \ln(d) - 1 \ln(d) \ln\left(\frac{d}{N}\right) - 1.85 \right)$$

Which is clearly $O\left(\sqrt{dN}\right)$. (I intentionally use an approximate representation since the exact solution would take the space of several lines!).

2.6. Quantum MST algorithm

The main algorithm is done just as described in 1.4.3, thus I will here focus on how the `minimum finding oracle` interface is implemented in my actual simulation.

The spanning forest consists of one or more disjoint sets, so an obvious choice is the tree representation of disjoint sets using path compression and union by rank, as described in [CLRS01]. The Graph is represented as an array of edges each having a source and target vertices and a weight.

2.6.1. Implementing the Minimum Finding Oracle-Interface

Having the disjoint tree representation the two methods `kind` and `value` follows immediate. The kind of an edge is simply the representative of the set containing the

source vertex of that edge. Similarly the value of an edge is the weight of the edge if the source and target vertices are in disjoint sets (having different representatives), otherwise I define it to be the maximal integer value, making it unlikely that this edge will ever be the minimal edge.

2.6.2. Merging

When the k candidate edges are found by the `G-BBHT` algorithm, the trees in the spanning forest are merged two by two, if the two sets to merge are not already connected. A sketch is given on Algorithm 13

Algorithm 13: Pseudo code sketching how the sets in the spanning forest are merged

```
1: for each candidate edge (u, v) returned from the G-BBHT algorithm do
2:   if Representative of v and u are different then
3:     union u and v
4:     MST ← MST ∪ {(u, v)}
5:   end if
6: end for
```

2.6.3. Running Time

My simulation Notice that in the beginning of each iteration in the for loop of *find d minimum of different kinds*, the computation of the methods `value` and `kind` actually takes time $O(\alpha(n))$ (amortized, see [CLRS01]), where $\alpha(\cdot)$ denotes the inverse Ackermann function. Though after one Grover iteration, all `find-set` operations will take time $O(1)$ (since all children in the disjoint set trees will be children of their respective representatives). Thus in one of these iterations an amount of at most $O(\alpha(n))$ work will be done. In 2.5.3 it is shown that an extra amount of $O(\log d)$ work does not change the asymptotic running time, and exactly the same can be shown with $O(\log n)$ extra amount of work. And since $\alpha(n) = o(\log n)$ an extra term of $O(\alpha(n))$ does not increase the asymptotic running time. This is of course unimportant since I am merely simulating, and thus a factor of $\alpha(n)$ or not is peanuts compared to the huge extra amount of work done by simulating the quantum system.

A real quantum computer The reason why the argumentation in the preceding paragraph about the height of the disjoint set trees makes sense is that my simulator (of course) is not a real quantum computer. Thus my implementation of the disjoint set trees is purely classical, and the simulation of superposition is done sequentially. To be honest I have no qualified clue as to what would happen if such a datastructure were implemented in quantum circuits, since all of the "find" operations would have side effects, and the entire data structure would be in a superposition, and thus the argumentation about the heights of the trees does not really make sense. To avoid this, one might want to consider omitting the "path compression" heuristic. This has

the benefit that all operations are now well defined (i.e. no superposition of the data structure is created), but it also has the disadvantage that every application of the function f takes time $O(\log n)$. This causes the total running time of the algorithm to be $O(\sqrt{nm}\log n)$, which (sadly) is not an improvement over the algorithm of Kruskal if $n = \Theta(m)$, though it is faster for dense graphs ($m = \Theta(n^2)$), giving a running time of $O(n^{\frac{3}{2}}\log n)$ and thereby outperforming Kruskal's.

2.7. Quantum SSSP algorithm

As in the Minimum Spanning Tree algorithm, the implementation of this method is done by creating a class that implements the `Minimum Finding Oracle` interface, the rest is done as described on Algorithm 5 on page 24. In this section I will focus on what data structures I have used to represent the neccesary data and how I have implemented the interface. The main function is merely classical so I will not give it attention. The P-sets are represented using double linked lists giving the ability to make constant time unions. The A-sets are represented as balanced search trees, where the edges are equipped with a lexicographic ordering of the d-value, *weight*, *target*-vertex and the *source*-vertex.

For each call to the quantum subroutine the search space consists of a new set of edges. The naive way to handle this would be, prior to every call to the quantum sub routine to add the relevant edges to a random access data structure, obtaining an extra $O(m)$ running time. Since this would not be very convenient, I use an alternative way of evaluating the functions *value* and *kind*. Every time a P_l has been updated, I assign all edges in the search space to their respective source vertex. To do this I use a balanced Tree-map (in java the `TreeMap`-class) in the following way: The first vertex in the set P_l (having degree say d_{v1}) is assigned to the value 0, the second vertex in the set is assigned to the value d_{v1}, and so on. A pseudo code sketch is shown on Algorithm 14.

Algorithm 14: Pseudo code sketch of how edges are associated with their respective vertices

```
1: count ← 0
2: for each vertex v ∈ P_l do
3:     Map count to deg[v]
4:     count ← count + deg[v]
5: end for
```

In this way, given an index of an edge, the corresponding edge can be found by first looking up the appropriate source vertex (finding the largest element in the key set with value less than or equal to the index), and finding the appropriate edge in the edge incidence list of that vertex. This is shown on Algorithm 15

Algorithm 15: Pseudo code sketch of how to find an edge given the index
input: index i
1: Look up the largest element with key k less than or equal to i
2: Let v be the vertex associated to key k
3: **return** Edge $v - k$ in the edge incidence list of vertex v

Given these procedures the "kind" and "value" functions can easily be derived.

2.7.1. Running Time

Consider now the work done in Algorithm 14 during the whole execution of the SSSP algorithm. Using the same argumentation as in 1.4.4, there are at most $\frac{n}{s}$ sets of size s. Since inserting an element in a balanced search tree takes logarithmic time, inserting s elements takes not more than $O(n \log n)$ basic operations. And there are at most $\log n$ of these sets, thus the total work is $O(n \log^2 n)$ which is obviously not dominating the overall running time.

As indicated before, the functions *kind* and *value* are not executed in constant time, but looking at Algorithm 15 they can obviously be done in logarithmic time, thus the total running time of the algorithm is $O(\sqrt{nm} \log^3 n)$, which unfortunately is worse than the algorithm of Dijkstra for sparse graphs ($m = \Theta(n)$). Notice that this would be even if the functions f and g could be computed in constant time! Although for dense graphs ($m = \Theta(n^2)$) the running time becomes $O(n^{\frac{3}{2}} \log^3 n)$ which is clearly asymptotically better than Dijkstra's algorithm.

Chapter 3

Conclusion

The general framework of quantum computing has briefly been presented. Since whole books about this sole topic exists, my presentation is necessarily a lot less detailed. This framework is used for the presentation of Shor's and Grover's algorithms. A description of Shor's algorithm has been given, as are a proof of its correctness and running time analysis. It turns out that Shor's algorithm is capable of finding non trivial divisors of a composite number in polynomial time. As written in 1.2.1, I am a little uncertain whether or not that trick described in [NC00] actually works. If it does, Shor's algorithm has running time $O(\log(n)^3)$, and if it does not, Shor's algorithm has a running time of $O(\log(n)^4)$, never the less it is polynomial in the number of bits of the number, which is better than any known classical algorithm.

Similarly Grover's algorithm has been presented, and a proof of correctness and running time has been given. Grover's search algorithm for finding one or more elements out of N in an unstructured search space has an expected running time of $O(\sqrt{N})$ which is provably better than any classical algorithm.

A java program simulating a quantum computer has been implemented, and this is used to simulate the algorithms of Shor and Grover.

With offset in Grover's algorithm, the main results from [DHHM06] have been presented. This includes the "find d minimum of different kind algorithm". There are several theorems in this thesis regarding this algorithm. Some of these theorems are detailed reformulations of theorems in the article, and some theorems I present are theorems, which the authors of [DHHM06] have taken for granted or have omitted. This subroutine turns out to be convenient in the design of the two algorithms for finding respectively the minimum spanning tree of an undirected graph and the single source shortest paths of a directed graph. Description and analysis of these two algorithms have been presented. It turns out, that the quantum query complexity of the minimum spanning tree problem is $O(\sqrt{nm})$. The query complexity of the single source shortest paths problem has query complexity $O(\sqrt{nm}\log^2(n))$. These three algorithms have, as in the case of Shor's and Grover's algorithms, been implemented

on my simulator, but as I reason in the sections 2.6 and 2.7 in both cases, it has been necessary for me to introduce an extra factor of $O(\log n)$ since it has not been possible for me to implement the *kind* and *value* functions as constant time functions, and I am not sure whether this is possible at all. Therefore the running time of the quantum minimum spanning tree algorithm is in my case $O(\sqrt{nm}\log n)$. which is better than Kruskal's algorithm for dense graphs, but there is no improvement for sparse graphs. The running time of the quantum single source shortest paths algorithm is in my case $O(\sqrt{nm}\log^3(n))$, which again is better than Dijkstra's algorithm for dense graphs but worse for sparse graphs.

The theory of quantum algorithms is interesting, but its practical relevance mainly depends on the answer to a certain question. Namely the one of whether or not there will ever be built a working quantum computer with a register large enough to make the quantum algorithms such as Grover's and Shor's outperform the best state of the art classical algorithms.

Despite the importance of the question. I have by no means tried to answer it. What I have argued is merely that if such a quantum is ever built, it would have quite far-reaching consequences. In the continuation of this, I will end this thesis with an, in my opinion, appropriate quotation:

> *I never said it was possible. I only said it was true*
> Charles Richet, Nobel Laureate in Physiology

Appendix A

Appendix

A.1. Instructions on the simulator

It can be downloaded either using the command `wget www.gausdalfind.dk/Qsim.jar` or simply by opening `www.gausdalfind.dk/Qsim.jar` in a browser. It can be executed using the command `java -jar Qsim.jar`, and not simply by double clicking. The degree of output from the program can be regulated by changing the "verbose level" (command 9) this must be an integer from 1 to 6, and the higher the value, the more output. Example input files for the minimum spanning tree and the single source shortest paths can be found in sections A.3 and A.4.

A.2. Dump of a run

```
magnus@magnus-lap[NetBeansProjects/Qsim/dist](0)% java -jar Qsim.jar
Quantum Simulator. Made by Magnus Gausdal Find.
Email: magnus@gausdalfind.dk
Last modified: Mon july 27, 13:14
This application should be available at:
www.gausdalfind.dk/Qsim.jar

Pick your choice:
0) Exit
1) Built in examples
2) Shor's Algorithm (with trick from Nielsen and Chuang)
3) Grover's Algorithm (simple version)
4) The G-BBHT extension of Grover's Algorithm
5) Simple version of "find d minimum of different kind"
6) Quantum Minimum Spanning Tree Algorithm
7) Quantum Single Source Shortest Path Algorithm
8) More Info/Help
```

```
9) Change verbose level
10) Shor's Algorithm (without the trick - just trying until an order is found)
11) Reset Random seed

>>>3
Grover's Algorithm (simple version)
Choose number to find
>>>5
Choose number of bits to use
>>>4
Starting Grover search
nrIts: 1.221441469079183
Initial state:
1.0+0.0i
0.0+0.0i
0.0+0.0i
0.0+0.0i
0.0+0.0i
0.0+0.0i
0.0+0.0i
0.0+0.0i
0.0+0.0i
0.0+0.0i
0.0+0.0i
0.0+0.0i
0.0+0.0i
0.0+0.0i
0.0+0.0i
0.0+0.0i

Applied walsh-hadamard:
0.24999999999999992+0.0i
0.24999999999999992+0.0i
0.24999999999999992+0.0i
0.24999999999999992+0.0i
0.24999999999999992+0.0i
0.24999999999999992+0.0i
0.24999999999999992+0.0i
0.24999999999999992+0.0i
0.24999999999999992+0.0i
0.24999999999999992+0.0i
0.24999999999999992+0.0i
0.24999999999999992+0.0i
0.24999999999999992+0.0i
0.24999999999999992+0.0i
```

```
0.24999999999999992+0.0i
0.24999999999999992+0.0i
0.24999999999999992+0.0i

After Grover iteration nr. 1
0.1874999999999998+0.0i
0.18749999999999983+0.0i
0.1874999999999998+0.0i
0.1874999999999998+0.0i
0.1874999999999998+0.0i
0.6874999999999993+0.0i
0.1874999999999998+0.0i
0.1874999999999998+0.0i
0.1874999999999998+0.0i
0.1874999999999999+0.0i
0.1874999999999998+0.0i
0.1874999999999998+0.0i
0.1874999999999998+0.0i
0.18749999999999983+0.0i
0.1874999999999998+0.0i
0.1874999999999998+0.0i

After Grover iteration nr. 2
0.07812499999999988+0.0i
0.07812499999999983+0.0i
0.07812499999999985+0.0i
0.07812499999999986+0.0i
0.07812499999999988+0.0i
0.9531249999999984+0.0i
0.07812499999999985+0.0i
0.07812499999999986+0.0i
0.07812499999999988+0.0i
0.07812499999999983+0.0i
0.07812499999999985+0.0i
0.07812499999999986+0.0i
0.07812499999999988+0.0i
0.07812499999999983+0.0i
0.07812499999999985+0.0i
0.07812499999999986+0.0i

Measurement yields: 5
result: 5
Pick your choice:
0) Exit
```

```
1) Built in examples
2) Shor's Algorithm (with trick from Nielsen and Chuang)
3) Grover's Algorithm (simple version)
4) The G-BBHT extension of Grover's Algorithm
5) Simple version of "find d minimum of different kind"
6) Quantum Minimum Spanning Tree Algorithm
7) Quantum Single Source Shortest Path Algorithm
8) More Info/Help
9) Change verbose level
10) Shor's Algorithm (without the trick - just trying until an order is found)
11) Reset Random seed

>>>9
Choose new verbose levelInteger from 1 to 6. The higher levelthe more verbose
>>>1
Pick your choice:
0) Exit
1) Built in examples
2) Shor's Algorithm (with trick from Nielsen and Chuang)
3) Grover's Algorithm (simple version)
4) The G-BBHT extension of Grover's Algorithm
5) Simple version of "find d minimum of different kind"
6) Quantum Minimum Spanning Tree Algorithm
7) Quantum Single Source Shortest Path Algorithm
8) More Info/Help
9) Change verbose level
10) Shor's Algorithm (without the trick - just trying until an order is found)
11) Reset Random seed

>>>3
Grover's Algorithm (simple version)
Choose number to find
>>>5
Choose number of bits to use
>>>4
result: 5
Pick your choice:
0) Exit
1) Built in examples
2) Shor's Algorithm (with trick from Nielsen and Chuang)
3) Grover's Algorithm (simple version)
4) The G-BBHT extension of Grover's Algorithm
5) Simple version of "find d minimum of different kind"
6) Quantum Minimum Spanning Tree Algorithm
```

```
7) Quantum Single Source Shortest Path Algorithm
8) More Info/Help
9) Change verbose level
10) Shor's Algorithm (without the trick - just trying until an order is found)
11) Reset Random seed

>>>9
Choose new verbose levelInteger from 1 to 6. The higher levelthe more verbose
>>>3
Pick your choice:
0) Exit
1) Built in examples
2) Shor's Algorithm (with trick from Nielsen and Chuang)
3) Grover's Algorithm (simple version)
4) The G-BBHT extension of Grover's Algorithm
5) Simple version of "find d minimum of different kind"
6) Quantum Minimum Spanning Tree Algorithm
7) Quantum Single Source Shortest Path Algorithm
8) More Info/Help
9) Change verbose level
10) Shor's Algorithm (without the trick - just trying until an order is found)
11) Reset Random seed

>>>6
Quantum Minimum Spanning Tree Algorithm
Choose the constant c (integer)
>>>4
Specify the file containing the graph info
The file must be plain text.
The first line should contain nothing but the number of vertices.
All other lines must be on the form u v w, if there
is an edge between u and v with weight w
>>>/home/magnus/SDU/bachelor/graph562
Start finding Minimum spanning tree
Minimum Of Different Kind Finder started
Index set is now: [-1, -1, -1, -1, -1, -1, -1, -1, -1]
IndexOfType is now: [-1, -1, -1, -1, -1, -1, -1, -1, -1]
Nr of Grover applications (max): 0(126)
new element candidate: 0
New improving element found: 0
New Element: (-1,8,0)
Index set is now: [-1, -1, -1, -1, -1, -1, -1, -1, 0]
IndexOfType is now: [8, -1, -1, -1, -1, -1, -1, -1, -1]
Nr of Grover applications (max): 0(126)
```

```
new element candidate: 23
New improving element found: 23
New Element: (-1,10,8)
Index set is now: [-1, -1, -1, -1, -1, -1, -1, 23, 0]
IndexOfType is now: [8, -1, -1, -1, -1, -1, -1, -1, 7]
Nr of Grover applications (max): 12(126)
new element candidate: 12
New improving element found: 12
New Element: (-1,9,2)
Index set is now: [-1, -1, -1, -1, -1, -1, 12, 23, 0]
IndexOfType is now: [8, -1, 6, -1, -1, -1, -1, -1, 7]
Nr of Grover applications (max): 20(126)
new element candidate: 6
New improving element found: 6
New Element: (-1,7,1)
Index set is now: [-1, -1, -1, -1, -1, 6, 12, 23, 0]
IndexOfType is now: [8, 5, 6, -1, -1, -1, -1, -1, 7]
Nr of Grover applications (max): 25(126)
new element candidate: 20
New improving element found: 20
New Element: (-1,6,4)
Index set is now: [-1, -1, -1, -1, 20, 6, 12, 23, 0]
IndexOfType is now: [8, 5, 6, -1, 4, -1, -1, -1, 7]
Nr of Grover applications (max): 33(126)
new element candidate: 24
New improving element found: 24
New Element: (-1,1,6)
Index set is now: [-1, -1, -1, 24, 20, 6, 12, 23, 0]
IndexOfType is now: [8, 5, 6, -1, 4, -1, 3, -1, 7]
Nr of Grover applications (max): 41(126)
new element candidate: 22
New improving element found: 22
New Element: (-1,10,5)
Index set is now: [-1, -1, 22, 24, 20, 6, 12, 23, 0]
IndexOfType is now: [8, 5, 6, -1, 4, 2, 3, -1, 7]
Nr of Grover applications (max): 44(126)
new element candidate: 26
New improving element found: 26
New Element: (-1,2,7)
Index set is now: [-1, 26, 22, 24, 20, 6, 12, 23, 0]
IndexOfType is now: [8, 5, 6, -1, 4, 2, 3, 1, 7]
Nr of Grover applications (max): 45(126)
new element candidate: 7
New improving element found: 7
```

```
New Element: (-1,7,2)
Index set is now: [-1, 26, 22, 24, 20, 6, 7, 23, 0]
IndexOfType is now: [8, 5, 6, -1, 4, 2, 3, 1, 7]
Nr of Grover applications (max): 46(126)
new element candidate: 16
New improving element found: 16
New Element: (-1,8,3)
Index set is now: [16, 26, 22, 24, 20, 6, 7, 23, 0]
IndexOfType is now: [8, 5, 6, 0, 4, 2, 3, 1, 7]
Nr of Grover applications (max): 47(126)
new element candidate: 3
New improving element found: 3
New Element: (-1,4,3)
Index set is now: [3, 26, 22, 24, 20, 6, 7, 23, 0]
IndexOfType is now: [8, 5, 6, 0, 4, 2, 3, 1, 7]
Nr of Grover applications (max): 48(126)
new element candidate: 13
New improving element found: 13
New Element: (-1,9,5)
Index set is now: [3, 26, 13, 24, 20, 6, 7, 23, 0]
IndexOfType is now: [8, 5, 6, 0, 4, 2, 3, 1, 7]
Nr of Grover applications (max): 49(126)
new element candidate: 27
New improving element found: 27
New Element: (-1,2,8)
Index set is now: [3, 26, 13, 24, 20, 6, 7, 27, 0]
IndexOfType is now: [8, 5, 6, 0, 4, 2, 3, 1, 7]
Nr of Grover applications (max): 50(126)
new element candidate: 8
New improving element found: 8
New Element: (-1,2,1)
Index set is now: [3, 26, 13, 24, 20, 8, 7, 27, 0]
IndexOfType is now: [8, 5, 6, 0, 4, 2, 3, 1, 7]
Nr of Grover applications (max): 51(126)
new element candidate: 9
New improving element found: 9
New Element: (-1,2,4)
Index set is now: [3, 26, 13, 24, 9, 8, 7, 27, 0]
IndexOfType is now: [8, 5, 6, 0, 4, 2, 3, 1, 7]
Nr of Grover applications (max): 52(126)
new element candidate: 25
New improving element found: 25
New Element: (-1,1,7)
Index set is now: [3, 25, 13, 24, 9, 8, 7, 27, 0]
```

```
IndexOfType is now: [8, 5, 6, 0, 4, 2, 3, 1, 7]
Nr of Grover applications (max): 54(126)
new element candidate: 2
New improving element found: 2
New Element: (-1,4,0)
Index set is now: [3, 25, 13, 24, 9, 8, 7, 27, 2]
IndexOfType is now: [8, 5, 6, 0, 4, 2, 3, 1, 7]
Nr of Grover applications (max): 55(126)
new element candidate: 14
Edges found from the "minimum of different kind" algorithm:
(3,0,4), (7,6,1), (5,2,9), (6,7,1), (4,1,2), (1,4,2), (2,1,7), (8,7,2), (0,3,4),
Iteration over. Edges in tree are now:
[(3,0,4), (7,6,1), (5,2,9), (4,1,2), (2,1,7), (8,7,2)]
working on p-set: 1
nr. of tree in the spanning forest: 3
Minimum Of Different Kind Finder started
Index set is now: [-1, -1, -1]
IndexOfType is now: [-1, -1, -1, -1, -1, -1, -1, -1, -1]
Nr of Grover applications (max): 0(109)
new element candidate: 0
New improving element found: 0
New Element: (-1,8,0)
Index set is now: [-1, -1, 0]
IndexOfType is now: [2, -1, -1, -1, -1, -1, -1, -1, -1]
Nr of Grover applications (max): 0(109)
new element candidate: 23
New improving element found: 23
New Element: (-1,10,6)
Index set is now: [-1, 23, 0]
IndexOfType is now: [2, -1, -1, -1, -1, -1, 1, -1, -1]
Nr of Grover applications (max): 8(109)
new element candidate: 11
New improving element found: 11
New Element: (-1,4,6)
Index set is now: [-1, 11, 0]
IndexOfType is now: [2, -1, -1, -1, -1, -1, 1, -1, -1]
Nr of Grover applications (max): 9(109)
new element candidate: 6
New improving element found: 6
New Element: (-1,2147483647,1)
Index set is now: [6, 11, 0]
IndexOfType is now: [2, 0, -1, -1, -1, -1, 1, -1, -1]
Nr of Grover applications (max): 10(109)
new element candidate: 22
```

```
New improving element found: 22
New Element: (-1,10,1)
Index set is now: [22, 11, 0]
IndexOfType is now: [2, 0, -1, -1, -1, -1, 1, -1, -1]
Nr of Grover applications (max): 11(109)
new element candidate: 1
New improving element found: 1
New Element: (-1,8,1)
Index set is now: [1, 11, 0]
IndexOfType is now: [2, 0, -1, -1, -1, -1, 1, -1, -1]
Nr of Grover applications (max): 12(109)
new element candidate: 10
New improving element found: 10
New Element: (-1,4,1)
Index set is now: [10, 11, 0]
IndexOfType is now: [2, 0, -1, -1, -1, -1, 1, -1, -1]
Nr of Grover applications (max): 13(109)
new element candidate: 25
Edges found from the "minimum of different kind" algorithm:
(1,8,4), (8,1,4), (0,1,8), Iteration over.
Edges in tree are now:
[(3,0,4), (7,6,1), (5,2,9), (4,1,2), (2,1,7), (8,7,2), (1,8,4), (0,1,8)]
working on p-set: 2
nr. of tree in the spanning forest: 1
Finished. The MST consists of the following edges:
[(3,0,4), (7,6,1), (5,2,9), (4,1,2), (2,1,7), (8,7,2), (1,8,4), (0,1,8)]
Pick your choice:
0) Exit
1) Built in examples
2) Shor's Algorithm (with trick from Nielsen and Chuang)
3) Grover's Algorithm (simple version)
4) The G-BBHT extension of Grover's Algorithm
5) Simple version of "find d minimum of different kind"
6) Quantum Minimum Spanning Tree Algorithm
7) Quantum Single Source Shortest Path Algorithm
8) More Info/Help
9) Change verbose level
10) Shor's Algorithm (without the trick - just trying until an order is found)
11) Reset Random seed

>>>9
Choose new verbose levelInteger from 1 to 6. The higher levelthe more verbose
>>>2
Pick your choice:
```

```
0) Exit
1) Built in examples
2) Shor's Algorithm (with trick from Nielsen and Chuang)
3) Grover's Algorithm (simple version)
4) The G-BBHT extension of Grover's Algorithm
5) Simple version of "find d minimum of different kind"
6) Quantum Minimum Spanning Tree Algorithm
7) Quantum Single Source Shortest Path Algorithm
8) More Info/Help
9) Change verbose level
10) Shor's Algorithm (without the trick - just trying until an order is found)
11) Reset Random seed

>>>6
Quantum Minimum Spanning Tree Algorithm
Choose the constant c (integer)
>>>4
Specify the file containing the graph info
The file must be plain text.
The first line should contain nothing but the number of vertices.
All other lines must be on the form u v w, if there
is an edge between u and v with weight w
>>>/home/magnus/SDU/bachelor/graph562
Start finding Minimum spanning tree
Iteration over.
Edges in tree are now:
[(7,6,1), (1,4,2), (3,0,4), (5,2,9), (2,1,7), (8,7,2)]
working on p-set: 1
nr. of tree in the spanning forest: 3
Iteration over. Edges in tree are now:
[(7,6,1), (1,4,2), (3,0,4), (5,2,9), (2,1,7), (8,7,2), (1,8,4), (0,1,8)]
working on p-set: 2
nr. of tree in the spanning forest: 1
Finished.
The MST consists of the following edges:
[(7,6,1), (1,4,2), (3,0,4), (5,2,9), (2,1,7), (8,7,2), (1,8,4), (0,1,8)]
Pick your choice:
0) Exit
1) Built in examples
2) Shor's Algorithm (with trick from Nielsen and Chuang)
3) Grover's Algorithm (simple version)
4) The G-BBHT extension of Grover's Algorithm
5) Simple version of "find d minimum of different kind"
6) Quantum Minimum Spanning Tree Algorithm
```

```
7) Quantum Single Source Shortest Path Algorithm
8) More Info/Help
9) Change verbose level
10) Shor's Algorithm (without the trick - just trying until an order is found)
11) Reset Random seed

>>>0
Exit
```

A.3. Example input to the MST algorithm

```
9
0 1 8
0 3 4
0 6 11
1 2 7
1 4 2
1 8 4
2 5 9
2 8 14
3 6 8
4 6 7
4 7 6
5 8 10
6 7 1
7 8 2
```

A.4. Example input to the SSSP algorithm

```
5
0 1 10
0 3 5
1 2 1
1 3 2
2 4 4
3 1 3
3 2 9
3 4 2
4 0 7
4 2 6
```

A.5. Query and Answer from Mika Hirvensalo

from Mika Hirvensalo <mikhirve@utu.fi>
to Magnus Find <magnus@gausdalfind.dk>
date Mon, Jun 1, 2009 at 10:25 AM
subject Re: Typo in "Quantum Computing"

hide details Jun 1 (3 days ago)

Reply

Follow up message
Dear Magnus,

You're right, there is a typo -- to be corrected just as you suggested.
Thanks for pointing this out.

Best regards,
Mika Hirvensalo
- Hide quoted text -

----- Original Message -----
From: Magnus Find <magnus@gausdalfind.dk>
Date: Monday, June 1, 2009 11:02 am
Subject: Typo in "Quantum Computing"
To: mikhirve@utu.fi

> Der Mika
>
> I am writing to you because I encountered a small typo in your book. I don't
> know if you are the person I ought to contact - if not, please ignore
> this mail.
> At page 84 equation 5.24, I am quite certain that the 4th entrance in the
> matrix should be 1-(2k)/(2^n) and not 1-2k/(2^k) (Otherwise it wouldn't
> really make any sense!?).
>
> Cheers
> Magnus Find, Computer Science/Math student

A.6. Failure of Trick by Nielsen and Chuang

```
magnus@magnus-lap[NetBeansProjects/Qsim/dist](0)% java -jar Qsim.jar
Quantum Simulator. Made by Magnus Gausdal Find.
Email: magnus@gausdalfind.dk
Last modified: Mon july 27, 13:14
This application should be available at:
www.gausdalfind.dk/Qsim.jar

Pick your choice:
0) Exit
1) Built in examples
2) Shor's Algorithm (with trick from Nielsen and Chuang)
3) Grover's Algorithm (simple version)
4) The G-BBHT extension of Grover's Algorithm
5) Simple version of "find d minimum of different kind"
6) Quantum Minimum Spanning Tree Algorithm
7) Quantum Single Source Shortest Path Algorithm
8) More Info/Help
9) Change verbose level
10) Shor's Algorithm (without the trick - just trying until an order is found)
11) Reset Random seed

>>>9
Choose new verbose levelInteger from 1 to 6. The higher levelthe more verbose
>>>4
Pick your choice:
0) Exit
1) Built in examples
2) Shor's Algorithm (with trick from Nielsen and Chuang)
3) Grover's Algorithm (simple version)
4) The G-BBHT extension of Grover's Algorithm
5) Simple version of "find d minimum of different kind"
6) Quantum Minimum Spanning Tree Algorithm
7) Quantum Single Source Shortest Path Algorithm
8) More Info/Help
9) Change verbose level
10) Shor's Algorithm (without the trick - just trying until an order is found)
11) Reset Random seed

>>>2


Shor's Algorithm
```

```
What number to you want to factorize?
>>>35
Choose propability of error (epsilon value). Must be strictly greater than 0.
0.1
x chosen to be: 25
No quantum order find necessary.
factor found: 5
Pick your choice:
0) Exit
1) Built in examples
2) Shor's Algorithm (with trick from Nielsen and Chuang)
3) Grover's Algorithm (simple version)
4) The G-BBHT extension of Grover's Algorithm
5) Simple version of "find d minimum of different kind"
6) Quantum Minimum Spanning Tree Algorithm
7) Quantum Single Source Shortest Path Algorithm
8) More Info/Help
9) Change verbose level
10) Shor's Algorithm (without the trick - just trying until an order is found)
11) Reset Random seed

>>>2


Shor's Algorithm
What number to you want to factorize?
>>>35
Choose propability of error (epsilon value). Must be strictly greater than 0.
0.1
x chosen to be: 14
No quantum order find necessary.
factor found: 7
Pick your choice:
0) Exit
1) Built in examples
2) Shor's Algorithm (with trick from Nielsen and Chuang)
3) Grover's Algorithm (simple version)
4) The G-BBHT extension of Grover's Algorithm
5) Simple version of "find d minimum of different kind"
6) Quantum Minimum Spanning Tree Algorithm
7) Quantum Single Source Shortest Path Algorithm
8) More Info/Help
9) Change verbose level
10) Shor's Algorithm (without the trick - just trying until an order is found)
```

```
11) Reset Random seed

>>>2


Shor's Algorithm
What number to you want to factorize?
>>>35
Choose propability of error (epsilon value). Must be strictly greater than 0.
0.1
x chosen to be: 9
Finding order of 9 modulo 35
Getting the convergents first time
nrTimesTrickWentWrong: 0
nrTimesTrickWentWell: 0
Setting up a quantum computer
Computer has 16 bits.Register 1 consisting of 10 bits.Register 2 consisting of 6 bits.
Applying Hadamard gates
modular exponentiating..
Applying inverse fourier transformon the first 10 bits
measurement: 590
Getting the convergents second time
nrTimesTrickWentWrong: 0
nrTimesTrickWentWell: 0
Setting up a quantum computer
Computer has 16 bits.Register 1 consisting of 10 bits.Register 2 consisting of 6 bits.
Applying Hadamard gates
modular exponentiating..
Applying inverse fourier transformon the first 10 bits
measurement: 1016
convs1: [(0,1), (1,1), (1,2), (3,5), (4,7), (15,26), (19,33), (34,59), (87,151), (295,512)]
convs2: [(0,1), (1,1), (127,128)]
numerator1: 0
numerator2: 0
gcd: 0
lcm: 1
numerator1: 0
numerator2: 1
gcd: 0
lcm: 1
numerator1: 0
numerator2: 127
gcd: 0
lcm: 128
```

```
numerator1: 1
numerator2: 0
gcd: 0
lcm: 1
numerator1: 1
numerator2: 1
gcd: 1
lcm: 1
numerator1: 1
numerator2: 127
gcd: 1
lcm: 128
numerator1: 1
numerator2: 0
gcd: 0
lcm: 2
numerator1: 1
numerator2: 1
gcd: 1
lcm: 2
numerator1: 1
numerator2: 127
gcd: 1
lcm: 128
numerator1: 3
numerator2: 0
gcd: 0
lcm: 5
numerator1: 3
numerator2: 1
gcd: 1
lcm: 5
numerator1: 3
numerator2: 127
gcd: 1
lcm: 640
Wrong order(640) found. "trick" from nielsenchuangwent wrong.
numerator1: 4
numerator2: 0
gcd: 0
lcm: 7
numerator1: 4
numerator2: 1
gcd: 1
```

```
lcm: 7
numerator1: 4
numerator2: 127
gcd: 1
lcm: 896
Wrong order(896) found. "trick" from nielsenchuangwent wrong.
numerator1: 15
numerator2: 0
gcd: 0
lcm: 26
numerator1: 15
numerator2: 1
gcd: 1
lcm: 26
numerator1: 15
numerator2: 127
gcd: 1
lcm: 1664
Wrong order(1664) found. "trick" from nielsenchuangwent wrong.
numerator1: 19
numerator2: 0
gcd: 0
lcm: 33
numerator1: 19
numerator2: 1
gcd: 1
lcm: 33
numerator1: 19
numerator2: 127
gcd: 1
lcm: 4224
The order candidate found was: 4224
Trick went well.
order was: 4224
x^(r/2) was 1.. trying r/2
x^(r/2) was 1.. trying r/2
x^(r/2) was 1.. trying r/2
x^(r/2) was 1.. trying r/2
x^(r/2) was 1.. trying r/2
x^(r/2) was 1.. trying r/2
x^(r/2) was 1.. trying r/2
The divisor candidates are:7 and 5
factor found: 7
Pick your choice:
```

```
0) Exit
1) Built in examples
2) Shor's Algorithm (with trick from Nielsen and Chuang)
3) Grover's Algorithm (simple version)
4) The G-BBHT extension of Grover's Algorithm
5) Simple version of "find d minimum of different kind"
6) Quantum Minimum Spanning Tree Algorithm
7) Quantum Single Source Shortest Path Algorithm
8) More Info/Help
9) Change verbose level
10) Shor's Algorithm (without the trick - just trying until an order is found)
11) Reset Random seed

>>>
```

A.7. Source Code

A.7.1. Complex.java

```
1  public class Complex {

       /* Class implementing Complex numbers. Supports the
          standard operations
        * as addition, division, multiplication, subtraction,
          magnitude (modulus)
        * Used as a part of bachelor project of Magnus Gausdal
          Find
6       * parts taken from
        * www.java2s.com/Code/Java/Language-Basics/
          AclasstorepresentComplexNumbers.htm
        */
       //real part
       private double r;
11
       //imaginary part
       private double i;

       Complex(double rr, double ii) {
16         r = rr;
           i = ii;
       }


21     /* Return real part */
```

```
    public double getReal() {
        return r;
    }

    /* Return real part */
    public double getImaginary() {
        return i;
    }

    /* Return magnitude*/
    public double magnitude() {
        return Math.sqrt(r*r + i*i);
    }

    /* Add another Complex to this one     */
    public Complex add(Complex other) {
        return add(this, other);
    }

    /* Add two Complexes
     */
    public static Complex add(Complex c1, Complex c2) {
        return new Complex(c1.r+c2.r, c1.i+c2.i);
    }

    /* Subtract another Complex from this one
     */
    public Complex subtract(Complex other) {
        return subtract(this, other);
    }

    /* Subtract two Complexes
     */
    public static Complex subtract(Complex c1, Complex c2) {
        return new Complex(c1.r-c2.r, c1.i-c2.i);
    }

    /* Multiply this Complex times another one
     */
    public Complex multiply(Complex other) {
        return multiply(this, other);
    }

    /* Multiply two Complexes
```

```
66       */
         public static Complex multiply(Complex c1, Complex c2) {
             return new Complex(c1.r*c2.r - c1.i*c2.i, c1.r*c2.i +
                 c1.i*c2.r);
         }

71       /* Divide c1 by c2.
          */
         public static Complex divide(Complex c1, Complex c2) {
             return new Complex(
                                 (c1.r*c2.r+c1.i*c2.i)/(c2.r*c2.r+
                                    c2.i*c2.i),
76                               (c1.i*c2.r-c1.r*c2.i)/(c2.r*c2.r+
                                    c2.i*c2.i));
         }

         /* Compare this Complex number with another
          */
81       @Override
         public boolean equals(Object o) {
             if (!(o instanceof Complex))
                 throw new IllegalArgumentException(
                                                      "Complex.
                                                         equals
                                                         argument
                                                         must be a
                                                         Complex");
86           Complex other = (Complex)o;
             return r == other.r && i == other.i;
         }


91       @Override
         public String toString() {
             if (i<0)
                 return r+""+i + "i";
             else
96               return r+"+"+i + "i";
         }
     }
```

A.7.2. Edge.java

```
     /*
2     * Class implementing an Edge
```

```
 * For use in the Graph class
 *
 */

/**
 *
 * @author Magnus Gausdal Find.
 * Used as a part of a bachelor project.
 * Contact info: magnus@gausdalfind.dk
 */
public class Edge implements Comparable<Edge>{

    int from;
    int to;
    int weight;
    int relaxedValue; //used in SSSP. The least currently
       known
    // distance from the source
    // vertex to the "to" vertex using this edge

    public Edge(int u, int v, int w){
        from=u;
        to=v;
        weight = w;
        relaxedValue=Integer.MAX_VALUE;
    }

    public String toString(){
        return "("+ from + "," + to +"," + weight + ")";
    }

    /*
     * The "natural" ordering is lexicographic ordering by
     * relaxedvalue, weight, from to
     * the last three are quite arbitrary, it is only to
        ensure that
     * the ordering induces a relation between any two edges
     */
    public int compareTo(Edge e){
        if(e.relaxedValue!=this.relaxedValue){
            return (new Integer(this.relaxedValue)).compareTo
               (e.relaxedValue);
        }
        else if(e.weight!=this.weight){
```

```
                return (new Integer(this.weight)).compareTo(e.
                    weight);
            }
            else if(this.from!=e.from){
47              return (new Integer(this.from)).compareTo(e.from)
                    ;
            }
            else{
                return (new Integer(this.to)).compareTo(e.to);
            }
52      }

    }
```

A.7.3. FindMinimumOfDifferentKind.java

```
 1  import java.util.ArrayList;
    import java.util.Arrays;
    import java.util.Random;
    import java.util.TreeSet;

 6  /*
     * Used in the "MinimumOfDifferentKindFinder" class.
     * Implements the GroversOracle interface such that
     * It can be given as a parameter to a Grover algorithm
     * (Grovers algorithm or G-BBTH)
11   * This class should only be used as part of the
         MinimumOfDifferentKindFinder
     */

    /*
     * @author Magnus Gausdal Find
16   * Used as part of bachelor project.
     * Contact info: Magnus@gausdalfind.dk
     */
    public class FindMinimumOfDifferentKind implements
        GroversOracle {

21      MinimumFindingOracle mfo = null;
        int nrOfApplications;
        int[] indexSet;
        int[] indexOfType;
        ArrayList<Integer> indexOfm1;
26      TreeSet<MinimumOfDifferentKindIndex> indsInSetTree;
```

```
    public FindMinimumOfDifferentKind(
            MinimumFindingOracle mfo,
            int d,
31          int nrOfTypes,
            Random r) {
        this.mfo = mfo;
        nrOfApplications = 0;
        indexSet = new int[d];
36      indexOfType = new int[nrOfTypes];
        Arrays.fill(indexOfType, -1);
        Arrays.fill(indexSet,-1);
        indexOfm1 = new ArrayList<Integer>();
        for (int i = 0; i < indexSet.length; i++) {
41          indexOfm1.add(i);
        }

        indsInSetTree = new TreeSet<
            MinimumOfDifferentKindIndex>();
    }
46
    public void setMinimumFindingOracle(MinimumFindingOracle
        mfo) {
        this.mfo = mfo;
    }

51  public int getNrOfOracleApplications() {
        return nrOfApplications;
    }

    //look up whether or not an element is an improving
56  //element with respect to the current index set
    public boolean isGoodElement(int n) {
        int valueOfElem = mfo.value(n);
        int kindOfElem = mfo.kind(n);

61      if (kindOfElem == -1) {
            return false;
        }
        //check if improving of known kind
        if (indexOfType[kindOfElem] != -1) {
66          //check if already contained
            if (indexSet[indexOfType[kindOfElem]] == n) {
                return false;
            }
```

```
                //an element of same type already contained
71              if (valueOfElem < mfo.value(indexSet[indexOfType[
                    kindOfElem]])) {
                    //new element is less than the current
                        included element
                    //of the same type
                    return true;
                } else {
76                  return false;
                }
            } else { //no element of same type contained
                if (indexOfm1.isEmpty()) {
                    if(valueOfElem < indsInSetTree.last().value){
81                      return true;
                    } else{
                        return false;
                    }
                } else { //there are still "unset" elements
86                  return true;
                }
            }
        }

91      /** applies the Grover Oracle to the state s
         * ie. all goot elements gets their amplitued multiplied
             with
         * the complex number -1
         */
        public void applyGroverOracle(State s) {
96          nrOfApplications++;
            Complex minus1 = new Complex(-1, 0);
            for (int i = 0; i < s.nrStates; i++) {
                if (this.isGoodElement(i)) {
                    s.m.setEntrance(i, 0, (s.m.getEntrance(i, 0))
                        .multiply(minus1));
101             }
            }
        }

    }
```

A.7.4. Fraction.java

```
    /*
     * Class implementing a fraction of integers
```

```
 * Used in the order finding algorithm (which is a subroutine
     to Shor's)
 * in the part of continued fractions.
 * Supports standard operations as addition, subtraction,
     multiplication
 * reduction, reciproce
 *
 * Used as part of my bachelor project.
 * Author: Magnus Gausdal Find
 * Contact information: magnus@gausdalfind.dk
 */

public class Fraction {

    int num;
    int denom;

    public Fraction(int num, int denom){
        this.num=num;
        this.denom=denom;
    }

    public Fraction add(Fraction b){
        return new Fraction(this.num*b.denom + b.num*this.
            denom,
                b.denom*this.denom);
    }

    public Fraction subtract(Fraction b){
        return this.add(new Fraction(-1*b.num, b.denom));
    }

    public Fraction add(int b){
        return this.add(new Fraction(b,1));
    }

    public Fraction subtract(int b){
        return this.add(new Fraction(-b, 1));
    }

    public Fraction mult(int a){
        return new Fraction(a*num, denom);
    }
```

```
    public void reduce(){
45      int gcd = ToolBox.gcd(num,denom);
        num/=gcd;
        denom/=gcd;

    }
50
    public Fraction inverse(){
        return new Fraction(denom, num);
    }


55
    public String toString(){
        return "(" + num + "," + denom + ")";
    }

60 }
```

A.7.5. Graph.java

```
import java.util.ArrayList;


 5 /**
   * A Class to represent directed graphs
   * Very simple representation
   * vertices are number as 0..(nrVertices-1), and directed
     edges kept in an
   * ArrayList
10 * Used in the quantum algorithm for finding a minimum
     spanning tree
   * of a graph
   * @author Magnus Gausdal Find
   * contact info: Magnus@gausdalfind.dk
   */
15 public class Graph {

    int nrVertices;
    ArrayList<Edge> edges;

20  ArrayList<Integer>[] incidenceList;

    //default constructor. Initializes the vertices with 0..
       nrVertices-1
```

```
        //and no edges
        public Graph(){
        }

        public Graph(int nrVertices){
            this.nrVertices = nrVertices;
            edges = new ArrayList<Edge>();
            incidenceList = new ArrayList[nrVertices];
            for(int i = 0;i<incidenceList.length;i++){
                incidenceList[i]=new ArrayList<Integer>();
            }
        }

        public void addDirectedEdge(int u,int v, int w){
            edges.add(new Edge(u,v,w));
        }

        public void addUndirectedEdge(int u, int v, int w){
            Edge e = new Edge(u,v,w);
            Edge ep = new Edge(v,u,w);
            edges.add(e);
            edges.add(ep);
        }

        @Override
        public String toString(){
            String s = "Nr. of vertices: " + nrVertices + "\n";
            s +=   "Edges: " + edges;
            return s;
        }
    }
```

A.7.6. Grovers.java

```
import java.util.Random;

/*
 * Implementation of the Grover search algorithm
 * as well as the extension of the algorithm
 * due to Boyer, Brassard, Hoeyer and Tapp
 *
 * Used as part of my bachelor project.
 * author: Magnus Gausdal Find
 * email: Magnus@gausdalfind.dk
```

```
12  */

    public class Grovers {


17      /**
         *
         * @param go the grover oracle
         * @param nrBits the number of bits used for indexing
         * @param nrIts the number of grover iterations
22       * @param verbose if true, a lot of status output is
             printed
         * @return a "good" element according to the oracle (with
             some prop)
         */
        public static int groverSearch(
                GroversOracle go,int nrBits, double nrIts,boolean
                    verbose, Random r)
27      {
            if(verbose){
                System.out.println("Starting Grover search");
                System.out.println("nrIts: " + nrIts);
            }
32          State s = new State(nrBits, 0,r);
            if(verbose){
                System.out.println("Initial state:\n"+s);
            }
            Matrix hada = Matrix.hadamard();
37          for(int i = 0;i<nrBits;i++){
                s.apply1bit(hada, i);
            }
            if(verbose){
                System.out.println("Applied walsh-hadamard:\n"+s)
                    ;
42          }
            for(int i = 0;i<nrIts;i++){
                go.applyGroverOracle(s);
                for(int j = 0;j<nrBits;j++){
                    s.apply1bit(hada, j);
47              }
                phaseShift(s);
                for(int j = 0;j<nrBits;j++){
                    s.apply1bit(hada, j);
                }
```

```
52              if(verbose){
                    System.out.println(
                            "After Grover iteration nr. " + (i+1)
                               +"\n"+s);
                }
            }
57          if(verbose){
                System.out.println("Measurement yields: "+s.
                    measure());
            }
            return s.measure();
        }
62

        /**
         * This function is an implementation of the algorithm
         * in section 6 in the article "tight bounds on Quantum
             Searching"
67       * by Boyer, Hoyer, Brassard, and Tapp
         */
        public static int extendedGroverSearchUnknownSolNumber(
                GroversOracle go,
                int nrBits,
72              int maxNrOfOracleCalls,
                Random rand,
                int verbose){
            if(verbose>4){
                System.out.println("Starting the extended Grover
                    search" +
77                      "algorithm");
            }
            double newVal;
            int solIndex=0;
            double m = 1.;
82          int j;
            double lambda = 8./7.;
            int rootN = (int) Math.sqrt(ToolBox.pow(2, nrBits));
            if(verbose>4){
                System.out.println("m: 1");
87          }
            while(!go.isGoodElement(solIndex)
                    &&
                    go.getNrOfOracleApplications()<
                        maxNrOfOracleCalls
```

```
                ){
92              j=rand.nextInt((int)m)+1;
                if(verbose>4){
                    System.out.println("no. of iterations now: "+
                        j);
                }
                boolean gverbose = (verbose>5);
97              solIndex=groverSearch(go, nrBits, j,gverbose,rand
                    );
                newVal =  (lambda*m);
                if(newVal<rootN){
                    m=newVal;
                }
102             else{
                    m=rootN;
                }
                if(verbose>4){
                    System.out.println("New value of m: " + m );
107             }
            }

            return solIndex;
        }
112

        /**
         *
         * multiplies all coefficients with -1 except for base
             state 0
117      */
        public static void phaseShift(State s){
            Complex minus1 = new Complex(-1,0);
            for(int i = 1;i<s.nrStates;i++){
                s.m.setEntrance(i, 0, (s.m.getEntrance(i, 0)).
                    multiply(minus1));
122         }
        }
    }
```

A.7.7. GroversOracle.java

```
1   /**
     * The Grover interface
     * If a class implements this interface, it can use the
         Grover algorithm
```

```
 *
 * @author magnus gausdal find
 * email: magnus@gausdalfind.dk
 */
public interface GroversOracle {

    public int getNrOfOracleApplications();

    public boolean isGoodElement(int n);

    public void applyGroverOracle(State s);

}
```

A.7.8. LinkedListElement.java

```
/**
 * Class implementing elements in a linked list.
 * Contains a value (the element) and a pointer to the next
    element in the list
 *
 * @author Magnus gausdal find
 * email: magnus @ gausdalfind.dk
 */
public class LinkedListElement<E> {

    E value;
    //successor == null => tail
    LinkedListElement successor;

    public LinkedListElement(
            E value,
            LinkedListElement successor) {
        this.value = value;
        this.successor = successor;
    }

    public void addElement(LinkedListElement<E> e){
        this.successor=e;
    }

    public String toString(){
        if(successor!=null){
            return "" +value + "->" + successor.toString();
        }
```

```
29          else{
                return ""+value;
            }

        }
34
}
```

A.7.9. LinkedList.java

```
/**
 * Single Linked List class
 *
 * Support constant time operations add and union
 * can contain any elements.
 * Used as a part of the quantum algorithm for finding single source
 * shortest paths.
 *
 * It maintains a reference to the first and the last element in the list
 * All additions of elements should be done using the linkedlist class
 * and not manually! This might screw up the head/tail pointers.
 *
 * @author Magnus Gausdal Find
 * email: magnus@gausdalfind.dk
 */

public class LinkedList<E> {

    LinkedListElement<E> head;
    LinkedListElement<E> tail;

    public LinkedList() {
        this.head = null;
        this.tail = null;
    }

    public void addElement(E element){
        LinkedListElement<E> lle = new LinkedListElement(element, null);
        if(head==null){
            head=lle;
```

```
            tail=lle;
        } else {
            tail.addElement(lle);
            tail=lle;
        }
    }

    public void union(LinkedList ll2){
        if(ll2.head!=null){
            this.tail.addElement(ll2.head);
            this.tail=ll2.tail;
        }
    }

    public String toString(){
        if(head==null){
            return "[]";
        }
        return head.toString();
    }

}
```

A.7.10. Main.java

```
import java.io.File;
import java.util.Arrays;
import java.util.HashSet;
import java.util.Random;
import java.util.Scanner;

/**
 * Main class of the Qsim project, constituting a part of my
    bachelor project
 * contains a menu, and calls the respective functions
 *
 *
 * @author magnus gausdal find
 * email: magnus @ gausdalfind . dk
 */

public class Main {

    static final String LASTUPDATE = "Mon july 27, 13:14";
```

```
    static Scanner sc;
    static Random r = new Random(101);
    static int verbose;

    public static void main(String [] args){
        sc = new Scanner(System.in);
        verbose=3;
        showMenu();
    }

    public static void mstExample(){
        //example from page 562 in Cormen et al.
        System.out.println("Starting to find the Minimum
            spanning tree" +
                "for the graph available at page 562 in \"
                    Cormen et. al\"");
        Graph g = new Graph(9);
        g.addUndirectedEdge(0, 1, 8);
        g.addUndirectedEdge(0, 3, 4);
        g.addUndirectedEdge(0, 6, 11);
        g.addUndirectedEdge(1, 2, 7);
        g.addUndirectedEdge(1, 4, 2);
        g.addUndirectedEdge(1, 8, 4);
        g.addUndirectedEdge(2, 5, 9);
        g.addUndirectedEdge(2, 8, 14);
        g.addUndirectedEdge(3, 6, 8);
        g.addUndirectedEdge(4, 6, 7);
        g.addUndirectedEdge(4, 7, 6);
        g.addUndirectedEdge(5, 8, 10);
        g.addUndirectedEdge(6, 7, 1);
        g.addUndirectedEdge(7, 8, 2);

        MST mst = new MST(g, 15, r,verbose);
        mst.findMST();
        System.out.println(
                "Finished. The MST consists of the following
                    edges:"
                +mst.edgesInMST);
    }

    public static void finddofdifferentkindexample(){
        System.out.println("Performing very simple" +
                "example of the \"find d minimum of
                    different kind\"");
```

```
        System.out.println("here d=2");
        int[] vals = {1,3,5,7,9,10};
        int[] kinds = {1,1,1,0,0,1};
        System.out.println("Values: " + Arrays.toString(vals)
            );
        System.out.println("Kinds: " + Arrays.toString(vals))
            ;
        simpleMFO smfo = new simpleMFO(kinds, vals);


        int[] res = MinimumOfDifferentKindFinder.
                findDMinimumOfDifferentKind(smfo, 2, 5, 6, r
                    ,2,verbose);
        System.out.println("Result obtained: " +Arrays.
            toString(res));
    }

    public static void shorsExample(){
        System.out.println("Performing Shors factorization
            algorithm" +
                "\nFactoring the number 35, with epsilon=0.1"
                    );
        System.out.println("factor found: "+ShorsWithTrick.
            factor(35, 0.1,verbose,r));
    }

    public static void groversExample(){
        System.out.println("Trying to find the number 3 in
            the range " +
                "0..7");
        HashSet<Integer> targs = new HashSet<Integer>();
        targs.add(3);
        SimpleSearch ss = new SimpleSearch();
        ss.numbersToFind=targs;
        Grovers.groverSearch(ss, 3,
                (Math.PI)/4*Math.sqrt(ToolBox.pow(2, 3))-1,
                    true,r);
    }

    public static void groversExtendedExample(){
        HashSet<Integer> targs = new HashSet<Integer>();
        System.out.println("Elements 1..12 are \"good\"");
        System.out.println("7 bits are used. So the search
            space" +
```

```
                "␣consists␣of␣the␣elements␣0..127");
        targs.add(1);
        targs.add(2);
        targs.add(3);
 98     targs.add(4);
        targs.add(5);
        targs.add(6);
        targs.add(7);
        targs.add(8);
103     targs.add(9);
        targs.add(10);
        targs.add(11);
        targs.add(12);
        SimpleSearch ss = new SimpleSearch();
108     ss.numbersToFind=targs;
        System.out.println(
                "Measurement␣yields␣:" +
                Grovers.extendedGroverSearchUnknownSolNumber(
                    ss, 7,50,r,verbose)
113                 );
    }

    private static void UIGBBTAlgorithm() {
        int nrBits;
118     int maxIts;
        try {
            HashSet<Integer> targs = new HashSet<Integer>();
            int newelem=0;
            while(newelem>=0){
123             System.out.println("Add␣good␣elements␣(add␣-1
                    ␣when␣finished)");
                System.out.print(">>>");
                newelem= Integer.parseInt(sc.nextLine());
                targs.add(newelem);
            }
128         System.out.println("Choose␣number␣of␣bits␣in␣
                search␣space");
            System.out.print(">>>");
            nrBits=Integer.parseInt(sc.nextLine());
            System.out.println("Choose␣max␣number␣of␣Grover␣
                iterations");
            System.out.print(">>>");
133         maxIts=Integer.parseInt(sc.nextLine());
            SimpleSearch ss = new SimpleSearch();
```

```
            ss.numbersToFind = targs;
            System.out.println(
                    "Measurement yields :" +
                    Grovers.
                        extendedGroverSearchUnknownSolNumber(
                    ss, nrBits, maxIts, r, verbose));
        } catch (Exception e) {
            System.out.println("Error in input");
        }
    }

    private static void UIGroversAlgorithm() {
        int numberToFind;
        int nrBits;
        try{
            System.out.println("Choose number to find");
            System.out.print(">>>");
            numberToFind = Integer.parseInt(sc.nextLine());
            System.out.println("Choose number of bits to use"
                );
            System.out.print(">>>");
            nrBits = Integer.parseInt(sc.nextLine());
            HashSet<Integer> targs = new HashSet<Integer>();
            targs.add(numberToFind);
            SimpleSearch ss = new SimpleSearch();
            ss.numbersToFind = targs;
            System.out.println("result: " +
                Grovers.groverSearch(ss,
                    nrBits,
                    (Math.PI) / 4 * Math.sqrt(ToolBox.pow(2,
                        3)) - 1,
                    (verbose>2?true:false),
                    r)
            );

        }
        catch(Exception e){
            System.out.println("Error in input. " +
                    "Got the following error message: "+e);
        }
    }

    private static void UIQMST() {
        int c = -1, u, v, w;
```

```
        try {
            System.out.println("Choose the constant c (
                integer)");
178         System.out.print(">>>");
            c = Integer.parseInt(sc.nextLine());
        } catch (Exception e) {
            System.out.println("Error in reading input." +
                    "Got the following error: " + e);
183     }
        System.out.println("Specify the file containing the
            graph info");
        System.out.println("The file must be plain text." +
                "\nThe first line should" +
                " contain nothing but the number of vertices
                    .\n" +
188             "All other lines" +
                " must be on the form u v w, if there\n" +
                "is an edge between u and" +
                " v with weight w");
        System.out.print(">>>");
193     String path = sc.nextLine();
        int nrVerts;
        try{
            Scanner fileScan = new Scanner(new File(path));
            nrVerts = Integer.parseInt(fileScan.nextLine());
198         Graph g = new Graph(nrVerts);
            while(fileScan.hasNext()){
                u=fileScan.nextInt();
                v=fileScan.nextInt();
                w=fileScan.nextInt();
203             g.addUndirectedEdge(u, v, w);
            }

            MST mst = new MST(g, c, r, verbose);
            mst.findMST();
208         System.out.println(
                    "Finished. The MST consists of the
                        following edges:" + mst.edgesInMST);
        }
        catch(Exception e){
            System.out.println("Error in reading input." +
213                 " Got the following error: " +e);
        }
    }
```

```
    private static void UIQSSSP() {
218         int c = -1, u, v, w, sourceVertex;
            int nrVerts;
            try {

                System.out.println("Choose the constant c (
                    integer)");
223             System.out.print(">>>");
                c = Integer.parseInt(sc.nextLine());
            } catch (Exception e) {
                System.out.println("Error in reading input." +
                        "Got the following error: " + e);
228         }
            try {
                System.out.println("Specify the file containing
                    the graph info");
                System.out.println("The file must be plain text."
                    +
                        "\nThe first line should" +
233                     " contain nothing but the number of
                            vertices.\n" +
                        "All other lines" +
                        " must be on the form u v w, if there\n"
                            +
                        "is an edge from u to" +
                        " v with weight w");
238             System.out.print(">>>");
                String path = sc.nextLine();
                Scanner fileScan = new Scanner(new File(path));
                nrVerts = Integer.parseInt(fileScan.nextLine());
                SSSPGraph g = new SSSPGraph(nrVerts);
243
                while (fileScan.hasNext()) {
                    u = fileScan.nextInt();
                    v = fileScan.nextInt();
                    w = fileScan.nextInt();
248                 //System.out.println(u + " " + v + " " + w);
                    g.addDirectedEdge(u, v, w);
                }
                System.out.println("Choose source vertex");
                System.out.print(">>>");
253             sourceVertex = Integer.parseInt(sc.nextLine());
```

```
            SSSP sssp = new SSSP(g, sourceVertex, c, r,
                verbose);
            System.out.println(
                    "The edges in the single source shortest
                        path tree" +
                    " are: " + sssp.findSSSP());
            System.out.println("The distances are: " + Arrays
                .toString(sssp.d));


        } catch (Exception e) {
            System.out.println("Error in reading input. " +
                    "Got the following error: "+e);
        }
    }

    private static void UIShorsAlgorithmWithTrick() {
        int n;
        double epsilon;
        try{
            System.out.println("What number to you want to
                factorize?");
            System.out.print(">>>");
            n = Integer.parseInt(sc.nextLine());
            if (n<2){
                throw new Exception();
            }
            System.out.println("Choose propability of error (
                epsilon value)." +
                    " Must be strictly greater than 0.");
            epsilon = Double.parseDouble(sc.nextLine());
            if(epsilon<=0){
                throw new Exception();
            }
            System.out.println(
                    "factor found: "+ShorsWithTrick.
                        factor(n, epsilon,verbose,r));
        }
        catch(Exception e){
            System.out.println("Error in input");
        }
    }

    private static void UIShorsNoTrick() {
```

```
293         int n;
            double epsilon;
            try{
                System.out.println("What number to you want to
                    factorize?");
                System.out.print(">>>");
298             n = Integer.parseInt(sc.nextLine());
                if (n<2){
                    throw new Exception();
                }
                System.out.println("Choose propability of error (
                    epsilon value)." +
303                     " Must be strictly greater than 0.");
                System.out.print(">>>");
                epsilon = Double.parseDouble(sc.nextLine());
                if(epsilon<=0){
                    throw new Exception();
308             }
                System.out.println(
                            "factor found: "+ShorsWithoutTrick.
                                factor(n, epsilon,verbose,r));
            }
            catch(Exception e){
313             System.out.println("Error in input");
            }
        }

        private static void UIbuiltInExamples() {
318         int choice=-1;
            boolean exit=false;
            while (!exit) {
                System.out.println("");
                System.out.println("Built in examples are
                    available for the following" +
323                     "algorithms:");
                System.out.println("0) Back to main menu");
                System.out.println("1) Grover's Algorithm");
                System.out.println("2) G-BBHT extension");
                System.out.println("3) Minimum Spanning Tree");
328             System.out.println("4) Single Source shortest
                    Path");
                System.out.println("Choose:");
                try {
                    System.out.print(">>>");
```

```
                choice = Integer.parseInt(sc.nextLine());
333         } catch (Exception e) {
                System.out.println("Error in input");
            }
            switch(choice){
                case 0:
338                 System.out.println("Going back to main
                        menu");
                    exit=true;
                    break;
                case 1:
                    System.out.println("Starting example of
                        Grover's Algorithm");
343                 groversExample();
                    break;
                case 2:
                    System.out.println("Starting example of
                        the" +
                            "G-BBHT extension of Grover's
                                Algorithm");
348                 groversExtendedExample();
                    break;
                case 3:
                    System.out.println("Starting example of Q
                        -MST algorithm");
                    mstExample();
353                 break;
                case 4:
                    System.out.println("Starting example of Q
                        -SSSP algorithm");
                    ssspExample();
                    break;
358             default:
                    System.out.println("Wrong input.");
                    break;
            }
        }
363 }

    private static void UIfindDofDifferentKind() {
        int nrKinds;
        int nrElements;
368     int d;
        int c;
```

```
        try {
            System.out.println("Choose number of different
                kinds");
            System.out.print(">>>");
373         nrKinds = Integer.parseInt(sc.nextLine());
            System.out.println("Choose number of elements");
            System.out.print(">>>");
            nrElements = Integer.parseInt(sc.nextLine());
            System.out.println("Choose the number of elements
                wanted (d)");
378         System.out.print(">>>");
            d=Integer.parseInt(sc.nextLine());
            System.out.println("Choose the number c");
            System.out.print(">>>");
            c=Integer.parseInt(sc.nextLine());
383         int[] vals = new int[nrElements];
            int[] kinds = new int[nrElements];
            for(int i = 0;i<nrElements;i++){
                System.out.println("Choose value of element "
                    +i);
                System.out.print(">>>");
388             vals[i]=Integer.parseInt(sc.nextLine());
                System.out.print(">>>");
                System.out.println("Choose kind of element "
                    +i);
                System.out.print(">>>");
                kinds[i]=Integer.parseInt(sc.nextLine());
393         }

            System.out.println("Values: " + Arrays.toString(
                vals));
            System.out.println("Kinds: " + Arrays.toString(
                vals));
            simpleMFO smfo = new simpleMFO(kinds, vals);
398
            int[] res = MinimumOfDifferentKindFinder.
                    findDMinimumOfDifferentKind(
                    smfo, d, nrKinds, c, r, nrKinds, verbose)
                        ;
            System.out.println("Result obtained: " + Arrays.
                toString(res));
403     }
        catch(Exception e){
            System.out.println("Error in input");
```

```
        }
    }
408
    private static void UIinfo() {
        System.out.println(
                "This program was developed as a part of my
                    bachelor project" +
                "\nwith the title \"Algorithms for quantum
                    computers\"." +
413             "\nAs one might notice the purpose of this
                    program is" +
                " to demonstrate how a quantum computer would
                    process" +
                " quantum algorithms." +
                "\nMore information about the algorithms
                    should (soon)" +
                " be available at www.gausdalfind.dk/report.
                    pdf." +
418             "\nIf you want the source code (for whatever
                    reason) you" +
                " can contact me at magnus@gausdalfind.dk." +
                "\nWhile writing this info it occurs to me,
                    that it is probably" +
                " for no real use since I don't really expect
                    anyone except" +
                " my advisor and the external examiner to use
                    this program." +
423             "\nA little note about the randomness of the
                    program." +
                " Since quantum computers by nature to some
                    extent" +
                " behave randomly randomness i incorporated
                    in this" +
                "application.\n" +
                "But to be able to reconstruct the results a
                    certain seed" +
428             " to the Random object is used every time." +
                " If more surprises are wanted, the menu
                    point" +
                "\n\"reset random seed\" can be used.\n"
                );
    }
433
    private static void showMenu() {
```

```
        int choice;
        String in;
        boolean exit = false;
438     System.out.println("Quantum Simulator. Made by Magnus
            Gausdal Find.");
        System.out.println("Email: magnus@gausdalfind.dk");
        System.out.println("Last modified: " + LASTUPDATE);
        System.out.println("This application should be
            available at:" +
                "\nwww.gausdalfind.dk/Qsim.jar\n");
443     while (!exit) {
            System.out.println("Pick your choice: ");
            System.out.println("0) Exit");
            System.out.println("1) Built in examples");
            System.out.println("2) Shor's Algorithm (with
                trick from Nielsen " +
448                 "and Chuang)");
            System.out.println("3) Grover's Algorithm (simple
                version)");
            System.out.println("4) The G-BBHT extension of
                Grover's Algorithm");
            System.out.println("5) Simple version of " +
                    "\"find d minimum of different kind\"");
453         System.out.println("6) Quantum Minimum Spanning
                Tree Algorithm");
            System.out.println("7) Quantum Single Source
                Shortest Path Algorithm");
            System.out.println("8) More Info/Help");
            System.out.println("9) Change verbose level");
            System.out.println("10) Shor's Algorithm (without
                the trick - " +
458                 "just trying until an order is found)");
            System.out.println("11) Reset Random seed");
            System.out.println("");
            System.out.print(">>>");
            in = sc.nextLine();
463         try{
                choice = Integer.parseInt(in);
                switch (choice) {
                    case 1:
                        System.out.println("\n\nBuilt in
                            Examples.");
468                     UIbuiltInExamples();
                        break;
```

```
                case 2:
                    System.out.println("\n\nShor's
                        Algorithm");
                    UIShorsAlgorithmWithTrick();
473                 break;
                case 3:
                    System.out.println("Grover's
                        Algorithm (simple version)");
                    UIGroversAlgorithm();
                    break;
478             case 4:
                    System.out.println("G-BBHT extension
                        of Grover's Algorithm");
                    UIGBBTAlgorithm();
                    break;
                case 5:
483                 System.out.println("Simple version of
                        " +
                            "\"find d minimum of
                                different kind\"");
                    UIfindDofDifferentKind();
                    break;
                case 6:
488                 System.out.println("Quantum Minimum
                        Spanning Tree Algorithm");
                    UIQMST();
                    break;
                case 7:
                    System.out.println(
493                     "Quantum Single Source Shortest
                            Path Algorithm");
                    UIQSSSP();
                    break;
                case 8:
                    System.out.println("Info/Help:");
498                 UIinfo();
                    break;
                case 0:
                    System.out.println("Exit");
                    exit=true;
503                 break;
                case 9:
                    System.out.println("Choose new
                        verbose level" +
```

```
                                    "Integer from 1 to 6. The
                                        higher level" +
                                    "the more verbose");
508                         try{
                                System.out.print(">>>");
                                verbose=Integer.parseInt(sc.
                                    nextLine());
                            }
                            catch(Exception e){
513                             System.out.println("Wrong input")
                                    ;
                            }
                            break;
                        case 10:
                            System.out.println("Using Shor's
                                algorithm without" +
518                                 " using the trick from
                                        Nielsen and " +
                                    "Chuang");
                            UIShorsNoTrick();
                            break;
                        case 11:
523                         System.out.println("Random object
                                reset");
                            Main.r=new Random(System.
                                currentTimeMillis());
                        default:
                            System.out.println("Error in input");
                            break;
528                 }
                }
                catch(Exception e){
                    System.out.println("Error.\n" +
                            "Got the following error: "+e.
                                getLocalizedMessage()
533                         + e.getStackTrace().toString());
                }
            }
        }

538     private static void ssspExample() {
            //using example from cormen at page 596
```

```
            System.out.println("Performing the quantum single
                source shortest path" +
                    " on a graph. The graph can be found in \"
                        Cormen\" page 596");
543
            SSSPGraph g = new SSSPGraph(5);
            g.addDirectedEdge(0, 1, 10);
            g.addDirectedEdge(0, 3, 5);
            g.addDirectedEdge(1, 2, 1);
548         g.addDirectedEdge(1, 3, 2);
            g.addDirectedEdge(2, 4, 4);
            g.addDirectedEdge(3, 1, 3);
            g.addDirectedEdge(3, 2, 9);
            g.addDirectedEdge(3, 4, 2);
553         g.addDirectedEdge(4, 0, 7);
            g.addDirectedEdge(4, 2, 6);
            SSSP sssp = new SSSP(g, 0, 10, r,verbose);
            System.out.println(
                    "The edges in the single source shortest path
                        tree" +
558                 " are: "+sssp.findSSSP());
            System.out.println("The distances are: " + Arrays.
                toString(sssp.d));
        }

        private static void ssspExample2() {
563         //using example from cormen at page 596

            System.out.println("Performing the quantum single
                source shortest path" +
                    " on a graph. The graph can be found in \"
                        Cormen\" page 596");

568         SSSPGraph g = new SSSPGraph(3);
            g.addDirectedEdge(0, 1, 4);
            g.addDirectedEdge(1, 2, 5);
            g.addDirectedEdge(0, 2, 7);
            SSSP sssp = new SSSP(g, 0, 10, r,verbose);
573         System.out.println(
                    "The edges in the single source shortest path
                        tree" +
                    " are: "+sssp.findSSSP());
            System.out.println("The distances are: " + Arrays.
                toString(sssp.d));
```

```
    }
578 }
```

A.7.11. Matrix.java

```
/*
 * Class implementing matrices over complex numbers
 * internally a matrix is represented as a 2d array
 * of complex numbers.
 *
 * supports some elementary matrix algebra
 * such as addition, ordinary multiplication, tensor
    multiplication
 * to ease the usage of class it hase methods for creating
    the most commonly
 * used matrices, including:
 *   -Hadamard gate
 *   -the r_k matrices (for building the quantum fourier
    transformation cirtuit)
 *     (described in the book of Nielsen and Chuang)
 *   -the r_k matrices transposed and complex conjugated
 *
 * Used as for representation of quantum gates and quantum
    states
 * made as a part of my bachelor project
 * author: magnus gausdal find
 * email: Magnus @ gausdalfind .dk
 */
public class Matrix{

    public int rows;
    public int coloumns;
    private Complex[][] a;

    public Matrix(int n, int m){
        //create new matrix, initialize all entrances to 0
        rows=n;
        coloumns = m;
        a= new Complex[n][m]; // should maybe be n+1 x m+1
        for(int i = 0;i<n;i++){
            for(int j = 0;j<m;j++){
            a[i][j]=new Complex(0,0);
            }
        }
    }
```

```
37
    public Matrix multiply(Matrix m2){
        Matrix newM = new Matrix(rows,m2.coloumns);
        Complex sum;
        if(m2.rows != coloumns)
42          throw new IllegalArgumentException(
                "Dimensions for the two matrices do not match"
                    );
        else{
            for(int i = 0;i<newM.rows;i++){
                for(int j = 0;j<newM.coloumns;j++){
47                  //calculate entrance (i,j)
                    sum=new Complex(0, 0);
                    for(int k=0;k<m2.rows;k++){
                        sum=
                         sum.add(this.getEntrance(i, k).
                             multiply(m2.getEntrance(k, j)));
52                  }
                    newM.a[i][j]=sum;
                }
            }
        }
57
        return newM;
    }

    public Matrix tensor(Matrix m2){
62      Matrix newM=new Matrix(this.rows*m2.rows, this.
            coloumns*m2.coloumns);
        for(int i1=0;i1<this.rows;i1++){
            for(int i2=0;i2<m2.rows;i2++){
                for(int j1 = 0;j1<this.coloumns;j1++){
                    for(int j2=0;j2<m2.coloumns;j2++){
67                      newM.a[i1*m2.rows+i2][j1*m2.coloumns+
                            j2]
                                =a[i1][j1].multiply(m2.a[i2][
                                    j2]);
                    }
                }
            }
72      }
        return newM;
    }
```

```
    public Matrix add(Matrix m2){
77      if(m2.rows!=rows || m2.coloumns!=coloumns)
            throw new IllegalArgumentException(
                    "In add: dimenstions does not match");
        else{
            Matrix newM= new Matrix(rows,coloumns);
82          for(int i = 0;i<rows;i++){
                for(int j = 0;j<coloumns;j++){
                    newM.a[i][j]=this.a[i][j].add(m2.a[i][j])
                        ;
                }
            }
87          return newM;
        }
    }

    public  static Matrix hadamard(){
92      Matrix o = new Matrix(2,2);
        o.setEntrance(0, 0, new Complex(1/Math.sqrt(2), 0));
        o.setEntrance(0, 1, new Complex(1/Math.sqrt(2), 0));
        o.setEntrance(1, 0, new Complex(1/Math.sqrt(2), 0));
        o.setEntrance(1, 1, new Complex(-1/Math.sqrt(2), 0));
97      return o;
    }

    public static Matrix pauliX(){
        Matrix o = new Matrix(2,2);
102     o.setEntrance(0, 1, new Complex(1, 0));
        o.setEntrance(1, 0, new Complex(1, 0));
        return o;
    }

107 public static Matrix rMatrix(int k){
        Matrix rk = new Matrix(2,2);
        rk.setEntrance(0, 0, new Complex(1,0));
        rk.setEntrance(1, 1,
                new Complex(
112             Math.cos(2*Math.PI/ToolBox.pow(2, k))
                ,Math.sin(2*Math.PI/ToolBox.pow(2, k))
                ));

        return rk;
117 }
```

```
    public static Matrix rHerm(int k){
        Matrix rk = new Matrix(2,2);
        rk.setEntrance(0, 0, new Complex(1,0));
122     rk.setEntrance(1, 1,
                new Complex(
                Math.cos(2*Math.PI/ToolBox.pow(2, k))
                ,-Math.sin(2*Math.PI/ToolBox.pow(2, k))
                ));
127
        return rk;
    }

    //should NOT be used.. use bitwise hadamard instead
132 public static Matrix walshHadamard(int n){
        Matrix h = Matrix.hadamard();
        Matrix o = h;
        for(int i = 1;i<n;i++){
            o=o.tensor(h);
137     }
        return o;
    }

    public static Matrix makeIdentity(int size){
142     Matrix out = new Matrix(size,size);
        Complex one = new Complex(1,0);
        for(int i =0;i<size;i++){
            out.a[i][i]=one;
        }
147
        return out;
    }

    @Override
152 public String toString(){
        String out = "";
        for(int i = 0;i<rows;i++){
            for(int j = 0;j<coloumns;j++){
            out += a[i][j] + " ";
157         }
            out += "\n";
        }
        return out;
    }
162
```

```
    public Complex getEntrance(int i, int j){
        return a[i][j];
    }

167 public void setEntrance(int i,int j, Complex v){
        a[i][j]=v;
    }

    public static void main(String[] args){
172     Matrix sx = new Matrix(2,2);
        Matrix sy = new Matrix(2,2);
        Matrix sz = new Matrix(2,2);

        //the three matrices of Pauli:
177     //sigma x
        sx.setEntrance(0, 1, new Complex(1, 0));
        sx.setEntrance(1, 0, new Complex(1, 0));

        //sigma y
182     sy.setEntrance(0, 1, new Complex(0, -1));
        sy.setEntrance(1, 0, new Complex(0, 1));

        //sigma z
        sz.setEntrance(0, 0, new Complex(1, 0));
187     sz.setEntrance(1, 1, new Complex(-1, 0));


        System.out.println(
                "The three matrices of pauli are as follows (
                    x,yz):");
192     System.out.println(sx + "\n" + sy + "\n" + sz +"\n");
        //products
        System.out.println("multiplying... :");
        System.out.println(sx.multiply(sx));
        System.out.println(sy.multiply(sy));
197     System.out.println(sz.multiply(sz));

        Matrix a = sx.tensor(sz);
        System.out.println(a);
        System.out.println("\n\n\n");
202     a=a.tensor(sz);
        System.out.println(a);
        System.out.println("\n\n\n");
        a=a.tensor(sz);
```

```
            System.out.println(a);
207         System.out.println("a rows: " + a.rows + " a cols: "
                + a.coloumns);
            System.out.println("\n\n\n");
            a=a.tensor(sz);
            System.out.println(a);
            System.out.println("a rows: " + a.rows + " a cols: "
                + a.coloumns);
212
            System.out.println(Matrix.hadamard().multiply(Matrix.
                hadamard()));
        }
    }
```

A.7.12. MinimumFindingOracle.java

```
    /**
     * The minimum finding oracle interface
     * implementing this interface (in a meaningful way ..) gives
     * the ability to use the FindMinimumOfDifferentKind
         procedure
 5   *
     * @author magnus gausdal find
     * email: Magnus  @ gausdal find . dk
     */

10  public interface MinimumFindingOracle {

        public int kind(int n);

        //index -1 has pr definition infinity
15      public int value(int n);

    }
```

A.7.13. MinimumOfDifferentKindFinder.java

```
    import java.util.Arrays;
    import java.util.Random;
 3
    /**
     * Implementation of the Find d Minimum of Different kind
         algorithm
     *
     * @author magnus
```

```
 8   * email: Magnus @ gausdalfind. dk
     */
    public class MinimumOfDifferentKindFinder {

            /**
13       *
         * @param mfo minimum finding oracle. Contains value and
             kind
         * for each index
         * @param d number of elements wanted
         * @param c runs the Grover iterations c*sqrt(d*N)
18       * @param N nrIndicies. Legal indicies are 0..N-1
         * @param r random object
         * @param nrTypes nr of possible types
         * @return
         */
23       public static int[] findDMinimumOfDifferentKind(
                MinimumFindingOracle mfo,
                int d,
                int c,
                int N,
28              Random r,
                int nrTypes,
                int verbose) {
            int maxNrOfApplications = (int) (c * Math.sqrt(d * N)
                );
            int newElem;
33
            int nrBits = (int) (Math.log(N) / Math.log(2)) + 1;

            FindMinimumOfDifferentKind fmod;
            fmod = new FindMinimumOfDifferentKind(mfo, d, nrTypes
                , r);
38
            if(verbose>2){
                System.out.println("Minimum Of Different Kind
                    Finder started");
            }
            while (fmod.getNrOfOracleApplications() <
                maxNrOfApplications) {
43              if (verbose > 2) {
                    System.out.println("Index set is now: " +
                        Arrays.toString(fmod.indexSet));
                    System.out.println("IndexOfType is now: " +
```

```
                    Arrays.toString(fmod.indexOfType));
            System.out.println(
                    "Nr of Grover applications (max): " +
                    fmod.getNrOfOracleApplications() +
                    "(" + maxNrOfApplications + ")");
        }
        newElem =
                Grovers.
                    extendedGroverSearchUnknownSolNumber(
                fmod, nrBits, maxNrOfApplications, r,
                    verbose);
        if(verbose>2){
            System.out.println("new element candidate: "
                +newElem);
        }
        if (fmod.isGoodElement(newElem)) {
            //newelem is a new improving element
            MinimumOfDifferentKindIndex modiNEW =
                    new MinimumOfDifferentKindIndex
                    (-1,//not known yet
                        mfo.value(newElem),
                        mfo.kind(newElem)
                    );
            if(verbose>2){
                System.out.println("New improving element
                    found: " + newElem);
                System.out.println("New Element: " +
                    modiNEW);
            }
            //insert newElem in indSet
            if (fmod.indexOfType[modiNEW.kind] == -1) {
                //unknown type
                if (fmod.indexOfm1.isEmpty()) { //no
                    elements are "undefined"
                    //remove the largest element from
                        indexset
                    //replace with largest element in
                        indexSet
                    MinimumOfDifferentKindIndex toRm =
                            fmod.indsInSetTree.pollLast()
                                ;
                    fmod.indexSet[toRm.index] = newElem;
                    //replace element at place "toRm.
                        index" with
```

```
                        //the new element. Ie replace the old
                            element with  the
                        //at index there toRm used to be
                        modiNEW.index = toRm.index;
83 //                      System.out.println("new elem: " +
    modiNEW);
                        fmod.indsInSetTree.add(modiNEW);
                        //update indexOfType
                        fmod.indexOfType[toRm.kind] = -1;
                        fmod.indexOfType[modiNEW.kind] =
                            modiNEW.index;
88                  } else {//at least one element in index
                        set is "undefined"
                        //note that no element must be
                            deleted
                        int toRm =
                                fmod.indexOfm1.remove(fmod.
                                    indexOfm1.size() - 1);
                        fmod.indexSet[toRm] = newElem;
93                      modiNEW.index = toRm;
    //                  System.out.println("new elem: " +
    modiNEW);
                        fmod.indsInSetTree.add(modiNEW);
                        //update indexOfType
                        fmod.indexOfType[modiNEW.kind] =
                            modiNEW.index;
98                  }
                } else {//known type
                    int toRmIndex = fmod.indexOfType[modiNEW.
                        kind];

                    MinimumOfDifferentKindIndex toRm
103                         = new MinimumOfDifferentKindIndex
                                (
                                toRmIndex,
                                mfo.value(fmod.indexSet[toRmIndex
                                    ]),
                                mfo.kind(fmod.indexSet[toRmIndex
                                    ])
                                );
108                 modiNEW.index = toRmIndex;

    //                System.out.println("new elem: " +
    modiNEW);
```

```
                        //remove element at toRmIndex
113                     fmod.indsInSetTree.remove(toRm);
                        //add new element
                        fmod.indsInSetTree.add(modiNEW);

                        fmod.indexSet[toRmIndex]= newElem;
118                 }
                //finished insertion
                }
            }
            return fmod.indexSet;
123     }

}
```

A.7.14. MinimumOfDifferentKindIndex.java

```
/**
 * Class made for storing new found indices by the
 * minimumofdifferentkindfinder method. Since they are
    ordered by value
 * the largest element can easily be retrieved
5 *
 * @author magnus gausdal find
 * email: Magnus @ gausdalfind.dk
 */

10 public class MinimumOfDifferentKindIndex
        implements Comparable<MinimumOfDifferentKindIndex> {

    int index; // the index in the "indSet" array
    int value;
15  int kind;

    public MinimumOfDifferentKindIndex(int i, int v, int k){
        this.index=i;
        this.value=v;
20      this.kind=k;
    }


    public int compareTo(MinimumOfDifferentKindIndex arg0) {
25      if(value!=arg0.value){
```

```
            return (new Integer(value)).compareTo(new Integer
                (arg0.value));
        }
        else{
            return (new Integer(index)).compareTo(new Integer
                (arg0.index));
        }
    }

    @Override
    public boolean equals(Object o){
        if(!(o instanceof MinimumOfDifferentKindIndex)){
            throw new IllegalArgumentException(
                    "error at equals in
                        MinimumOfDifferentKindIndex" +
                    " Argument had wrong type");
        }
        else{
            MinimumOfDifferentKindIndex m = (
                MinimumOfDifferentKindIndex) o;
            return(m.index==this.index && m.value == this.
                value);
        }
    }

    @Override
    public String toString(){
        return "("+index + ","+ value+","+kind+")";
    }
}
```

A.7.15. MST.java

```
import java.util.ArrayList;
import java.util.Arrays;
import java.util.Random;


/**
 * Implementation of the Quantum algorithm for finding a
    minimum spanning tree
 * of a graph, as well as the necessary data structures
 *
 * made as a part of my bachelor project
 * @author Magnus Gausdal Find
```

```
 * email: Magnus@ gausdalfind.dk
 */
public class MST implements MinimumFindingOracle{

    int c;
    Random r;
    Graph g;
    int[] p; //disjoint tree representation, parent pointer
    int[] rank; //rank in disjoint tree forest
    int l;
    int nrEdgesInMST;
    int k; //nr of trees
    ArrayList<Edge> edgesInMST;
    int verbose;

    public MST(Graph g, int c, Random r, int verbose){
        this.c=c;
        this.g=g;
        this.r = r;
        this.verbose=verbose;
        edgesInMST=new ArrayList<Edge>();
        k=g.nrVertices;
        nrEdgesInMST=0;
        p = new int[g.nrVertices];
        rank = new int[g.nrVertices];
        //make all trees singletons with rank 0
        for(int i = 0;i<p.length;i++){
            p[i]=i;
            rank[i]=0;
        }
        l=0;
    }

    /* finding the representative (root) of a tree in the
     * disjoint tree forest
     */
    public int findRepresentative(int n){
        if(p[n]!=n){
            p[n]=findRepresentative(p[n]);
        }
        return p[n];
    }

    //union and link as described in "Cormen et. al"
```

```
    public void union(int x, int y){
        link(findRepresentative(x),findRepresentative(y));
    }

60  public void link(int x, int y) {
        if (rank[x] > rank[y]) {
            p[y] = x;
        } else {
            p[x] = y;
65          if (rank[x] == rank[y]) {
                rank[y]++;
            }
        }
    }
70
    //implementation of the kind
    //if index is -1 or out of bounds -1 is returned as an
       error code
    public int kind(int n) {
        if(n==-1 || n>=g.edges.size())
75          return -1;
        else
            return findRepresentative(g.edges.get(n).from);
    }

80  /* implementation of the value function
     * if index is -1 or out of bounds (meaning that the
        edges actually does
     * not exist, maxinteger is returned, making it quite
        unlikely that this
     * edge will be the cheapest
     */
85  public int value(int n) {
        if(n==-1 || n>=g.edges.size())
            return Integer.MAX_VALUE;
        else{
            int to = g.edges.get(n).to;
90          int from = g.edges.get(n).from;
            if(findRepresentative(to) == findRepresentative(
               from)){
                return Integer.MAX_VALUE;
            }
            else{
95              return g.edges.get(n).weight;
```

```
                }
            }
        }

100
        /*
         * quantum version of boruvkas algorithm
         *
         */
105     public void findMST(){
            if(verbose>0){
                System.out.println("Start finding Minimum
                    spanning tree");
            }
            while(nrEdgesInMST<g.nrVertices-1){
110             mergeLevels();
                l++;
                if(verbose>0){
                    System.out.println(
                            "Iteration over. Edges in tree are
                                now: "
115                         + this.edgesInMST);
                }
                if(verbose>1){
                    System.out.println("working on p-set: " +l);
                    System.out.println("nr. of tree in the
                        spanning forest: " + k);
120             }
            }
        }

        private void mergeLevels(){
125         int u,v;
            int[] edgesToAdd=
            MinimumOfDifferentKindFinder.
                findDMinimumOfDifferentKind(
                    this,k, (l+2)*c, g.edges.size(), r, g.
                        nrVertices,verbose);
            if (verbose > 2) {
130             System.out.println("Edges found from the " +
                        "\"minimum of different kind\" algorithm:
                            ");
                for (int i = 0; i < edgesToAdd.length; i++) {
                    if (edgesToAdd[i] == -1) {
```

```
                    System.out.print("null" + ",");
135                 } else {
                        System.out.print(g.edges.get(edgesToAdd[i
                            ]) + ", ");
                    }
                }
            }
140         if(verbose>3){
                System.out.println("spanning trees currently: " +
                        Arrays.toString(p));
            }
            for(int i=0;i<edgesToAdd.length;i++){
145             if(edgesToAdd[i]==-1)
                    continue;
                u = g.edges.get(edgesToAdd[i]).from;
                v = g.edges.get(edgesToAdd[i]).to;
                if(findRepresentative(u) != findRepresentative(v)
                    ){
150                 union(u,v);
                    if(verbose>3){
                        System.out.println("Adding " + u + " and
                            " + v);
                    }
                    k--;
155                 edgesInMST.add(g.edges.get(edgesToAdd[i]));
                    nrEdgesInMST++;
                }
            }
        }
160
}
```

A.7.16. ShorsWithoutTrick.java

```
import java.util.ArrayList;
import java.util.Random;

 4 /**
   * impelementation of the algorithm of Shor
   * not using the "trick" described in the book of Nielsen and
       Chuang
   *
   * For the ease of implementing, this class is very similar
       to the corresponding
 9 * class incorporating the trick
```

```
 *
 * @author magnus gausdal find
 * email: Magnus @ gausdalfind .dk
 */

public class ShorsWithoutTrick {


    //variables made to do some statistics.. not currently
       used
    public static int errNoQness = 0;
    public static int errOddOrder = 0;
    public static int errTrivialDivisors =0;
    public static int errNoOrderFound =0;
    public static int nrOrderfindcalls =0;


    /**finds an element r st. x^r = 1 (mod N)
    with prop epsilon of failure
    */
    public static ArrayList<Fraction> orderFinding(
            int x, int n, double epsilon,int verbose,Random r
               ){
        if(verbose>3){
            System.out.println("Setting up a quantum computer
               ");
        }
        int L = (int) Math.floor((Math.log(n)/(Math.log(2))))
           +1;
        double insideLog = 2+1/(2*epsilon);

        int p = (int) (Math.ceil(Math.log(insideLog)/Math.log
           (2)));
        int t = L+1+p;
        int nrBits = t+L;

        if(verbose>1){
        System.out.println("Computer has " + nrBits + " bits.
           " +
                "Register 1 consisting of " + t + " bits." +
                "Register 2 consisting of " + L + " bits.");
        }
        State s = new State(nrBits,1,r);
        if(verbose>4){
```

```
            System.out.println("State: "+s);
49          }
            Matrix had = Matrix.hadamard();
            if(verbose>1){
                System.out.println("Applying Hadamard gates");
            }
54          for(int i = 1;i<=t;i++){
                s.apply1bit(had, nrBits-i);
            }
            if(verbose>4){
                System.out.println("State: "+s);
59          }
            if(verbose>1){
                System.out.println("modular exponentiating..");
            }
            s.modularExponentiate(L, n, x);
64
            if(verbose>4){
                System.out.println("State: "+s);
            }
            if(verbose>1){
69              System.out.println("Applying inverse fourier
                    transform" +
                     "on the " + t + " high order bits");
                s.applyInverseFourierTransform(t);
            }

74          if(verbose>4){
                System.out.println("State: "+s);
            }

            int result = s.measure()/ToolBox.pow(2, L);
79          if(verbose>1){
                System.out.println("measurement: "+result);
            }
            if(verbose>4){
                System.out.println("State: "+s);
84          }
            ArrayList<Integer> a =
                    ToolBox.continuedFraction(result, ToolBox.pow
                        (2, t));

            ArrayList<Fraction> convs = ToolBox.convergents(a);
89          return convs;
```

```
    }

    public static int findOrder(int x, int n, double epsilon,
        int verbose,
            Random r) {
94      if (verbose > 1) {
            System.out.println("Finding order of " + x + "
                modulo " + n);
        }
        ArrayList<Fraction> convs1;
        while (true) {
99          if (verbose > 1) {
                System.out.println("Getting the convergents")
                    ;
            }
            convs1 = orderFinding(x, n, epsilon, verbose, r);
            if (verbose > 3) {
104             System.out.println("convs: " + convs1);
            }
            for (Fraction f1 : convs1) {
                if (ToolBox.modExp(x, f1.denom, n) == 1) {
                    if (verbose > 1) {
109                     System.out.println(
                                "The order candidate found
                                    was: " + f1.denom);
                    }
                    return f1.denom;
                }
114         }
        }
    }

    public static int factor(int n, double epsilon, int
        verbose, Random r){
119     //verbose       0   nothing is printed
        //              1   very little detail is printed
        //              2   more detailed, including info
            about the
        //                  "quantum computer"
        //              3   Not used (=2)
124     //              4   much output is printed. Not
            including quantum states
        //              5   A LOT of output is printed.
            Including all states.
```

```
        int x;
        if(n%2==0)
            return 2;
129     int order;
        int d1=-1, d2=-1, g;
        while(true){
            x = (int) (r.nextDouble()*(n-2))+2;
            if(verbose>0){
134             System.out.println("x chosen to be: " + x);
            }
            if(ToolBox.gcd(x,n)!=1){
                if(verbose>0){
                    System.out.println("No quantum order find
                        necessary.");
139             }
                errNoQness++;
                return ToolBox.gcd(x,n);
            }
            else{
144             order=findOrder(x, n, epsilon,verbose,r);
                if(verbose>0){
                    System.out.println("order was: " + order)
                        ;
                }
                if(order%2==1){
149                 if(verbose>0){
                        System.out.println("Odd order, trying
                            again");
                    }
                    errOddOrder++;
                    continue;
154             }
                else{
                    g=ToolBox.modExp(x, order/2,n);
                    while(g==1){
                        if(verbose>0){
159                         System.out.println("x^(r/2) was
                                1.. trying r/2");
                        }
                        order=order/2;
                        g=ToolBox.modExp(x, order,n);
                    }
164                 d1=ToolBox.gcd(g-1, n);
                    d2=ToolBox.gcd(g+1, n);
```

```
                        if(verbose>0){
                            System.out.println("The divisor
                                candidates are:" +
                                 d1 + " and " + d2);
169                     }
                        if (d1!=1 && d1!= n)
                            return d1;
                        else if(d2!=1 && d2 != n)
                            return d2;
174                     if(verbose>0){
                            System.out.println("Divisors were
                                unfortunately trivial.." +
                                 " trying again.");
                        }
                        errTrivialDivisors++;
179                 }
                }
            }
        }
    }
```

A.7.17. ShorsWithTrick.java

```
    import java.util.ArrayList;
 2  import java.util.Random;

    /**
     * impelementation of the algorithm of Shor
     * using the "trick" described in the book of Nielsen and
        Chuang
 7   *
     * For the ease of implementing, this class is very similar
        to the corresponding
     * class not incorporating the trick
     *
     * @author magnus gausdal find
12   * email: Magnus @ gausdalfind .dk
     */


    public class ShorsWithTrick {
17
        //variables enabling statistics.. not currently used

        public static int errNoQness = 0;
```

```
    public static int errOddOrder = 0;
    public static int errTrivialDivisors =0;
    public static int errNoOrderFound =0;
    public static int nrOrderfindcalls =0;

    public static int nrTimesTrickWentWell=0;
    public static int nrTimesTrickWentWrong=0;

    /**finds an element r st. x^r = 1 (mod N)
    with prop epsilon of failure
    */
    public static ArrayList<Fraction> orderFinding(
            int x, int n, double epsilon,int verbose,Random r
                ){
        System.out.println("nrTimesTrickWentWrong: " +
            nrTimesTrickWentWrong);
        System.out.println("nrTimesTrickWentWell: " +
            nrTimesTrickWentWell);
        if(verbose>3){
            System.out.println("Setting up a quantum computer
                ");
        }
        int L = (int) Math.floor((Math.log(n)/(Math.log(2))))
            +1;
        double insideLog = 2+1/(2*epsilon);

        int p = (int) (Math.ceil(Math.log(insideLog)/Math.log
            (2)));
        int t = L+1+p;
        int nrBits = t+L;

        if(verbose>1){
        System.out.println("Computer has " + nrBits + " bits.
            " +
                "Register 1 consisting of " + t + " bits." +
                "Register 2 consisting of " + L + " bits.");
        }
        State s = new State(nrBits,1,r);
        if(verbose>4){
            System.out.println("State: "+s);
        }
        Matrix had = Matrix.hadamard();
        if(verbose>1){
            System.out.println("Applying Hadamard gates");
```

```
            }
            for(int i = 1;i<=t;i++){
                s.apply1bit(had, nrBits-i);
            }
            if(verbose>4){
                System.out.println("State: "+s);
            }
            if(verbose>1){
                System.out.println("modular exponentiating..");
            }
            s.modularExponentiate(L, n, x);

            if(verbose>4){
                System.out.println("State: "+s);
            }
            if(verbose>1){
                System.out.println("Applying inverse fourier
                    transform" +
                     "on the first " + t + " bits");
                s.applyInverseFourierTransform(t);
            }

            if(verbose>4){
                System.out.println("State: "+s);
            }

            int result = s.measure()/ToolBox.pow(2, L);
            if(verbose>1){
                System.out.println("measurement: "+result);
            }
            if(verbose>4){
                System.out.println("State: "+s);
            }
            ArrayList<Integer> a =
                    ToolBox.continuedFraction(result, ToolBox.pow
                        (2, t));

            ArrayList<Fraction> convs = ToolBox.convergents(a);
            return convs;
        }

        public static int findOrder(int x, int n, double epsilon,
            int verbose,
                Random r){
```

```
        if(verbose>1){
            System.out.println("Finding order of " + x + "
                modulo "+n);
        }
102     ArrayList<Fraction> convs1;
        ArrayList<Fraction> convs2;
        int lcm;
        while (true) {
            if(verbose>1){
107         System.out.println("Getting the convergents first
                time");
            }
            convs1=orderFinding(x, n, epsilon,verbose,r);
            if(verbose>1){
                System.out.println("Getting the convergents
                    second time");
112         }
            convs2=orderFinding(x, n, epsilon,verbose,r);
            //            System.out.println("verbose level:
                " + verbose);
            if(verbose>3){
                System.out.println("convs1: " + convs1);
117             System.out.println("convs2: " + convs2);
            }
            for (Fraction f1 : convs1) {
                for (Fraction f2 : convs2) {

122                 lcm = ToolBox.lcm(f1.denom, f2.denom);
                    if (verbose > 3) {
                        System.out.println("numerator1: " +
                            f1.num);
                        System.out.println("numerator2: " +
                            f2.num);
                        System.out.println("gcd: " + ToolBox.
                            gcd(f1.num, f2.num));
127                     System.out.println("lcm: " + lcm);
                    }
                    if (f1.num != f2.num
                            && f1.num!= 1
                            && f2.num != 1
132                         && ToolBox.gcd(f1.num, f2.num) ==
                                1) {
                        if (lcm > 1 && ToolBox.modExp(x, lcm,
                            n) == 1) {
```

```
                                    if (verbose > 1) {
                                        System.out.println(
                                                        "The order
                                                            candidate
                                                            found
                                                            was: "
                                                            + lcm);
137                                     System.out.println("Trick
                                            went well.");
                                        nrTimesTrickWentWell++;
                                    }
                                    return lcm;
                                } else {
142                                 if(verbose>1){
                                        System.out.println("Wrong
                                            order" +
                                                        "(" + lcm
                                                            + ")" +
                                                             "
                                                            found"
                                                            +
                                                        ". \"trick
                                                            \" from
                                                            nielsenchuang
                                                            " +
                                                        "went
                                                            wrong."
                                                            );
147                                     nrTimesTrickWentWrong++;
                                    }
                                }
                            }
                        }
152                 }
                }
            }

            public static int factor(int n, double epsilon, int
                verbose, Random r){
157             //verbose       0   nothing is printed
                //              1   very little detail is printed
```

```
//                2   more detailed, including info
    about the
//                    "quantum computer"
//                3   Not used (=2)
//                4   much output is printed. Not
    including quantum states
//                5   A LOT of output is printed.
    Including all states.
int x;
if(n%2==0)
    return 2;
int order;
int d1=-1, d2=-1, g;
while(true){
    x = (int) (r.nextDouble()*(n-2))+2;
    if(verbose>0){
        System.out.println("x chosen to be: " + x);
    }
    if(ToolBox.gcd(x,n)!=1){
        if(verbose>0){
            System.out.println("No quantum order find
                necessary.");
        }
        errNoQness++;
        return ToolBox.gcd(x,n);
    }
    else{
        order=findOrder(x, n, epsilon,verbose,r);
        if(verbose>0){
            System.out.println("order was: " + order)
                ;
        }
        if(order%2==1){
            if(verbose>0){
                System.out.println("Odd order, trying
                    again");
            }
            errOddOrder++;
            continue;
        }
        else{
            g=ToolBox.modExp(x, order/2,n);
            while(g==1){
                if(verbose>0){
```

```
197                                 System.out.println("x^(r/2) was
                                        1.. trying r/2");
                                }
                                order=order/2;
                                g=ToolBox.modExp(x, order,n);
                            }
202                         d1=ToolBox.gcd(g-1, n);
                            d2=ToolBox.gcd(g+1, n);
                            if(verbose>0){
                                System.out.println("The divisor
                                    candidates are:" +
                                     d1 + " and " + d2);
207                         }
                            if (d1!=1 && d1!= n)
                                return d1;
                            else if(d2!=1 && d2 != n)
                                return d2;
212                         if(verbose>0){
                                System.out.println("Divisors were
                                    unfortunately trivial.." +
                                     " trying again.");
                            }
                            errTrivialDivisors++;
217                     }
                    }
                }
            }
        }
```

A.7.18. simpleMFO.java

```
import java.util.InputMismatchException;

4 /*
  * A very simple example of an implementation of the
  * minimum finding oracle
  * two arrays specifying kind and value for each of the
     elements
  */
9
/**
 *
 * @author Magnus Gausdal Find
 * email: Magnus @ gausdalfind.dk
```

```
14  */
public class simpleMFO implements MinimumFindingOracle{

    int nrIndicies;
    int[] kinds;
19  int[] vals;

    public simpleMFO(int[] kinds, int[] vals){
        this.kinds=kinds;
        this.vals=vals;
24      if(vals.length!=kinds.length)
            throw new InputMismatchException("lists should
                have same length");
        nrIndicies=vals.length;
    }

29  public int kind(int n) {
        if(n>=0 && n < nrIndicies){
            return kinds[n];
        }
        else{
34          return -1;
        }
    }

    public int value(int n) {
39      if(n>=0 && n < nrIndicies){
            return vals[n];
        }
        else{
            return Integer.MAX_VALUE-2;
44      }
    }

}
```

A.7.19. SimpleSearch.java

```
import java.util.HashSet;
3

/**
 * Implementation of the simplest possible setup for the
    Grover algorithm
 * There are a set of numbers, which are "good"
```

```
 *
 * @author magnus gausdal find
 * email: Magnus @ gausdalfind. dk
 */
public class SimpleSearch implements GroversOracle {

    HashSet<Integer> numbersToFind;

    private int nrOfOracleApplications;

    public SimpleSearch(){
        nrOfOracleApplications=0;
    }



    public void applyGroverOracle(State s){
        nrOfOracleApplications++;
        Complex minus1 = new Complex(-1,0);
        for(int i = 0;i<s.nrStates;i++){
            if(numbersToFind.contains(i)){
                s.m.setEntrance(i, 0, (s.m.getEntrance(i, 0))
                    .multiply(minus1));
            }
        }
    }

    public boolean isGoodElement(int n){
        return numbersToFind.contains(n);
    }

    public int getNrOfOracleApplications() {
        return nrOfOracleApplications;
    }

}
```

A.7.20. SSSPGraph.java

```
import java.util.ArrayList;

/**
```

```
 * Graph class used in the quantum single source shortest
    paths algorithm
 * Vertices are indexed 0..(nrvertices-1)
 * edges are stored in an adjecency list, so for each vertex
 * there is a list of edge going out from this vertex
 * @author Magnus Gausdal Find
 * email: Magnus @ Gausdalfind. dk
 */
public class SSSPGraph {

    public int nrVertices;
    public ArrayList<ArrayList<Edge>> edgeAdjecency;
    int nrEdgesIncidentToVertex[];


    public SSSPGraph(int nrVertices){
        this.nrVertices=nrVertices;
        nrEdgesIncidentToVertex = new int[nrVertices];
        edgeAdjecency=new ArrayList<ArrayList<Edge>>();
        for(int i = 0; i<nrVertices;i++){
            nrEdgesIncidentToVertex[i]=0;
            edgeAdjecency.add(new ArrayList<Edge>());
        }
    }


    public void addDirectedEdge(int u, int v, int w){
        Edge edgeToAdd = new Edge(u, v, w);
        edgeAdjecency.get(u).add(edgeToAdd);
        nrEdgesIncidentToVertex[u]++;
    }

}
```

A.7.21. SSSP.java

```
import java.util.ArrayList;
import java.util.Arrays;
import java.util.Random;
import java.util.TreeMap;
import java.util.TreeSet;


/**
 * Class implementing the quantum algorithm for finding the
```

```
 * single source shortest paths in a graph.
 *
 *
 * @author magnus gausdal find
 * email: Magnus@ Gausdalfind.dk
 */

public class SSSP implements MinimumFindingOracle {

    Random r;
    int verboseLevel;

    SSSPGraph g;
    boolean[] inTree;

    int[] sizes;
    int[] nrEdgesIncidentToVerticesInPSet;

    int[] d;

    LinkedList<Integer>[] verticesInPSets;

    TreeSet<Edge>[] aSets;

    ArrayList<Edge> ssspTree;

    int c;
    int sourceVertex;
    int l;

    TreeMap<Integer,Integer> verticesAssToEdges;


    /**
     *
     * @param g the SSSPGraph
     * @param sourceVertex the source vertex
     * @param c constant for the quantum searching algorithm
     * @param r a random-Object
     * @param verbose level of verbosity
     */
    public SSSP(SSSPGraph g, int sourceVertex, int c, Random
       r, int verbose){
        this.r=r;
```

```
53          this.g=g;
            this.sourceVertex=sourceVertex;
            this.c=c;
            this.verboseLevel=verbose;

58
            ssspTree=new ArrayList<Edge>();

            inTree= new boolean[g.nrVertices];
            d = new int[g.nrVertices];
63          d[sourceVertex]=0;

            Arrays.fill(inTree, false);

            int arrLength = (int) (Math.log(g.nrVertices)/Math.
                log(2))+1;
68
            sizes = new int[arrLength];
            nrEdgesIncidentToVerticesInPSet= new int[arrLength];
            verticesInPSets = new LinkedList[arrLength];

73          verticesAssToEdges=new TreeMap<Integer, Integer>();
            verticesAssToEdges.put(0, 0);

            for(int i = 0;i<arrLength;i++){
                sizes[i]=0;
78              nrEdgesIncidentToVerticesInPSet[i]=0;
                verticesInPSets[i] = new LinkedList<Integer>();
            }

            aSets = new TreeSet[arrLength];
83
            inTree[sourceVertex]=true;
            l=0;
            sizes[0]=1;
            verticesInPSets[0].addElement(sourceVertex);
88          nrEdgesIncidentToVerticesInPSet[0]
                    =g.nrEdgesIncidentToVertex[sourceVertex];
        }

        public ArrayList<Edge> findSSSP(){
93          if(verboseLevel>0){
                System.out.println("Starting to find the SSSP-
                    tree");
```

```
        }
        while(ssspTree.size()<g.nrVertices-1){
            if(verboseLevel>3){
98              System.out.println("sssp tree currently: "+
                    ssspTree);
            }
            findAndAddNextEdges();
        }
        return ssspTree;
103     }

        public void findAndAddNextEdges(){
            int tempCount=0;
            if (verboseLevel > 4) {
108             System.out.println("vertsinpsets: " + Arrays.
                    toString(verticesInPSets));
                System.out.println("sizes of psets: " + Arrays.
                    toString(sizes));
                System.out.println(
                        "nrEdgestops: " + Arrays.toString(
                            nrEdgesIncidentToVerticesInPSet));
                System.out.println("sssp so far: " + ssspTree);
113             System.out.println("l: " + l);
            }

            //associates edge indices to vertices
            LinkedListElement<Integer> vertexIterator =
                verticesInPSets[l].head;
118         verticesAssToEdges.clear();

            while(vertexIterator!=null){
                verticesAssToEdges.put(tempCount,vertexIterator.
                    value);
                tempCount+=g.nrEdgesIncidentToVertex[
                    vertexIterator.value];
123             vertexIterator=vertexIterator.successor;
            }

            //use a quantum procedure to find edge candidates to
                add
            int[] newEdges =
128             MinimumOfDifferentKindFinder.
                    findDMinimumOfDifferentKind(
                     this,
```

```
                    sizes[l],
                    c*((int) Math.log(g.nrVertices)+1),
                    nrEdgesIncidentToVerticesInPSet[l],
133                 r,
                    g.nrVertices,
                    verboseLevel
                );

138         //add all found edges to new A-set
            TreeSet<Edge> newIncidentEdges = new TreeSet<Edge>();
            Edge e;
            for(int i = 0;i<newEdges.length;i++){
                e = getEdge(newEdges[i]);
143             e.relaxedValue = d[e.from]+e.weight;
                newIncidentEdges.add(e);
            }
            if(verboseLevel>3){
                System.out.println("aset "+ l +": " +
                    newIncidentEdges);
148         }
            aSets[l]=newIncidentEdges;

            //find minimal edge among the aSets
            //(whose target vertex is not already covered)
153         //and add it to the SSSP tree
            Edge currentBest = null;
            Edge candidateBest=null;
            for(int i = 0;i<=l;i++){
                candidateBest=aSets[i].pollFirst();
158             if(candidateBest!=null && inTree[candidateBest.to
                    ]){
                    i--;
                    continue;
                }
                if (
163                     (currentBest == null
                      ||
                        d[candidateBest.from]+candidateBest.
                            weight
                        < d[currentBest.from] + currentBest.
                            weight)
                    ) {
168                 currentBest = candidateBest;
                }
```

```
        }


173     d[currentBest.to]=d[currentBest.from]+currentBest.
            weight;
        ssspTree.add(currentBest);
        inTree[currentBest.to]=true;

        //part 2c
178     l++;
        LinkedList<Integer> newLinkedList = new LinkedList<
            Integer>();
        newLinkedList.addElement(currentBest.to);
        verticesInPSets[l]=newLinkedList;
        sizes[l]=1;
183     nrEdgesIncidentToVerticesInPSet[l]
                =g.nrEdgesIncidentToVertex[currentBest.to];

        //do merging
        while(l>=1 && sizes[l]==sizes[l-1]){
188         verticesInPSets[l-1].union(verticesInPSets[l]);
            sizes[l-1]+=sizes[l];
            nrEdgesIncidentToVerticesInPSet[l-1]
                    +=nrEdgesIncidentToVerticesInPSet[l];
            l--;
193     }
    }


    public int kind(int n) {
198     if (n < 0 || n >= nrEdgesIncidentToVerticesInPSet[l])
            {
            //if index to small or large return -1 as an
                error
            return -1;
        } else {
            //otherwise return the target vertex
203         return getEdge(n).to;
        }
    }

    public int value(int n) {
208     if(n<0
                ||
```

```
            n>= nrEdgesIncidentToVerticesInPSet[l]
                  ||
            inTree[getEdge(n).to]){
213          //if index to small or large return maxint
             //making int unlikely that this "edge" (becaues
                this actually
             //means that there is no edge) will ever
             //be the cheapest edge
             return Integer.MAX_VALUE;
218      }
         else{
             //otherwise return the cost of this edge
             return getEdge(n).weight+d[getEdge(n).from];
         }
223  }

     public Edge getEdge(int n){
         int vertAss;
         //in the case an edge exactly matches to a vertex.
228      //ie. is the first edge to a vertex
         if (verticesAssToEdges.containsKey(n)) {
                 return (g.edgeAdjecency.get(
                    verticesAssToEdges.get(n)).get(0));
             }//otherwise get the largest integer less than
             else {
233              vertAss = verticesAssToEdges.lowerKey(n);
                 return g.edgeAdjecency.get(
                         verticesAssToEdges.get(
                         vertAss)).get(
                         n - vertAss);
238          }
     }

}
```

A.7.22. State.java

```
import java.util.Random;

4 /**
  * Class implementing a quantum computer register
  * By doing the actual simulation of a quantum computer
  * this class constitutes the core of the Qsim package.
  *
```

```
 9   * Made as a part of my bachelor project
     * @author magnus gausdal find
     * email: magnus @ gausdalfind .dk
     */

14  public class State{

        int nrBits;
        int nrStates;
        Matrix m;
19      Random r;

        //applies the gate u to the entire state
        public void applyGate(Matrix u){
            m = u.multiply(m);
24      }

        //applies the 2x2 matrix u to bit "bit" in the.
        //I.e. applies the matrix u to bit "bit" on all basis
            vectors
        public void apply1bit(Matrix u, int bit){
29          Matrix nyM = new Matrix(m.rows,1);
            int bitVal;
            int bitPwr2 = ToolBox.pow(2, bit);
            //System.out.println(bitPwr2);
            int state0=-1; //i with bit "bit" changed to zero
34          int state1=-1; //i with bit "bit" changed to one
            Complex st0co;
            Complex st1co;
            int valueAtBit;
            for(int i = 0;i<nyM.rows;i++){
39              bitVal = (bitPwr2 & i);
                if(bitVal!=0){ //bit nr "bit" is one
                    valueAtBit=1;
                    state0 = i^bitPwr2;
                    state1 = i;
44              }
                else{//bit nr "bit" is zero
                    valueAtBit=0;
                    state0 = i;
                    state1 = i^bitPwr2;
49              }
```

```
                st0co = (m.getEntrance(i, 0).multiply(u.
                    getEntrance(0,valueAtBit)));
                st1co = (m.getEntrance(i, 0).multiply(u.
                    getEntrance(1,valueAtBit)));
                nyM.setEntrance(state0, 0, nyM.getEntrance(state0
                    , 0).add(st0co));
54              nyM.setEntrance(state1, 0, nyM.getEntrance(state1
                    , 0).add(st1co));
            }
            this.m=nyM;
        }

59      //same as apply1bit, just conditioned that controlBit=1
        public void applyControlled1bit(Matrix u, int targetBit,
            int controlBit){
            Matrix nyM = new Matrix(m.rows,1);
            int bitVal;
            int bitPwr2 = ToolBox.pow(2, targetBit);
64          //System.out.println(bitPwr2);
            int state0=-1; //i with bit "bit" changed to zero
            int state1=-1; //i with bit "bit" changed to one
            Complex st0co;
            Complex st1co;
69          int valueAtBit;
            int controlBitPwr2 = ToolBox.pow(2, controlBit);
            for(int i = 0;i<nyM.rows;i++){
                if((controlBitPwr2 & i) == 0){ //control bit zero
                    . dont apply gate
                    nyM.setEntrance(i, 0, m.getEntrance(i, 0));
74                  continue;
                }
                else{
                    bitVal = (bitPwr2 & i);
                    if(bitVal!=0){ //bit nr "bit" is one
79                      valueAtBit=1;
                        state0 = i-bitPwr2;
                        state1 = i;
                    }
                    else{
84                      valueAtBit=0;
                        state0 = i;
                        state1 = i+bitPwr2;
                    }
    //              System.out.println("i: "+ i);
```

```
89  //                  System.out.println("st0: " + state0);
    //                  System.out.println("st1: " + state1);
                        st0co = (m.getEntrance(i, 0).multiply(u.
                            getEntrance(0,valueAtBit)));
                        st1co = (m.getEntrance(i, 0).multiply(u.
                            getEntrance(1,valueAtBit)));
                        nyM.setEntrance(state0, 0, nyM.getEntrance(
                            state0, 0).add(st0co));
94                      nyM.setEntrance(state1, 0, nyM.getEntrance(
                            state1, 0).add(st1co));
                    }
                }
                this.m=nyM;
            }
99
            //modular exponentiate the l loworder bits
            public void modularExponentiate(int l, int n, int x){
                Matrix nyM = new Matrix(m.rows,1);
                int twoPowL = ToolBox.pow(2, l);
104             int j;
                int lastBits;
                int firstBits;
                for(int i = 0;i<nrStates;i++){
                    if(m.getEntrance(i, 0).magnitude()==0)
109                     continue;
                    j=i;
                    //last l bits
                    lastBits = i % twoPowL;
                    //first bits
114                 firstBits = i / twoPowL;
                    j=i-lastBits; // set last l bits to 0
                    j+=ToolBox.modExp(x,firstBits,n);
                    nyM.setEntrance(j, 0, m.getEntrance(i, 0));
                }
119             m=nyM;
            }

            //perform a measurement in computational basis
            //consider implementing partial measurements
124         public int measure(){
                double max = this.r.nextDouble();
                int i = 0;
                double sum = Math.pow(m.getEntrance(0, 0).magnitude()
                    ,2);
```

```
        while(sum<max){
129         i++;
            sum += Math.pow(m.getEntrance(i, 0).magnitude()
                ,2);
        }
        Complex zero = new Complex(0,0);
        for(int j = 0;j<m.rows;j++){
134         m.setEntrance(j, 0, zero);
        }
        m.setEntrance(i, 0, new Complex(1,0));
        return i;
    }
139

    //initialize a new state with n bits to state initState
    public State(int n, int initState, Random r){
        this.r=r;
144     nrBits=n;
        nrStates=ToolBox.pow(2,n);
        m = new Matrix(nrStates, 1);
        m.setEntrance(initState, 0, new Complex(1,0));
    }
149

    //initialize a new state with n bits.. all coefficients 0
    public State(int n){
        nrBits=n;
        nrStates=ToolBox.pow(2,n);
154     m = new Matrix(nrStates, 1);
    }

    //reverses all qubits. ie. reverse order of all bits in
       all
    //vectors in the basis
159 public void applyPGate(int lastbits){
        Matrix newM = new Matrix(m.rows, 1);
        for(int i = 0;i<m.rows;i++){
            {
                newM.setEntrance(ToolBox.reverseOrder(i,
                    nrBits,lastbits), 0,
164                     m.getEntrance(i, 0));
            }
        }
        m=newM;
    }
```

```
169
        //apply Fourier transformation on entire state.
        //Uses circuit from Nielsen and Chuang
        public void applyFourierTransform(){
            Matrix had = Matrix.hadamard();
174         for(int i = nrBits-1;i>=0;i--){
                apply1bit(had, i);
                for(int j = 2;j<=i+1;j++){
                    applyControlled1bit(Matrix.rMatrix(j), i, i-j
                        +1);
                }
179         }
            applyPGate(nrBits);
        }

        // performs inverse quantum fourier transformation
184     // on the last "lastIbits" bits
        // uses circuit from Nielsen and Chuang but in reversed
            direction
        // and all gates transposed and complex conjugated
        public void applyInverseFourierTransform(int lastIbits){
            applyPGate(lastIbits);
189         Matrix had = Matrix.hadamard();
            for(int i = 0; i<=lastIbits-1;i++){
                for(int j = i+1; j>=2;j--){
                    applyControlled1bit(Matrix.rHerm(j), nrBits-i
                        -1, nrBits-i-j);
                }
194             apply1bit(had, nrBits-i-1);
            }
        }


199     @Override
        public String toString(){
            return m.toString();
        }

204 }
```

A.7.23. ToolBox.java

```
import java.util.ArrayList;
```

```
/*
 * Class containing static methods for use in various
    situations
 * throughout the Qsim program.
 */

/**
 *
 * @author magnus gausdal find
 * email: Magnus @ gausdalfind.dk
 */
public class ToolBox {

    //returns b to the power of e
    public static int pow(int b,int e){
        int out = 1;
        for(int i = 0;i<e;i++){
            out*=b;
        }
        return out;
    }


     /**calculates b^n mod m
     *uses algorithm from K. Rosen: Discrete Mathematics
     */
    public static int modExp(int b, int n, int m){
        int result = 1;
        int power = b%m;
        int ai;
        while(n>0){
            ai=n%2;
            if(ai==1){
                result = (result*power)%m;
            }
            power=(power*power)%m;
            n=n/2;
        }
        return result;
    }

    /**
     * reverses the "lastbits" loworder bits of the nrbits-
        bit nr a
```

```
     */
    public static int reverseOrder(int a, int nrBits, int
        lastbits){
        //first reverse all bits, save value in temp
49      int temp=0;
        for(int i = 0;i<nrBits;i++){
            temp*=2;
            temp+=a%2;
            a/=2;
54      }
        //reverse the "lastbits" loworder bits and save in o
        int o =0;
        o=temp/(ToolBox.pow(2, lastbits));
        for(int i =0;i<lastbits;i++){
59          o*=2;
            o+=temp%2;
            temp/=2;
        }
        //reverse all bits in o to get right result
64      int res=0;
        for(int i = 0;i<nrBits;i++){
            res*=2;
            res+=o%2;
            o/=2;
69      }
        return res;
    }

    //the good old euclidean greatest common divisor
        algorithm
74  public static int gcd(int a, int b){
        if(a==0 || b==0)
            return 0;
        while(a!=b){
            if(a>b){
79              a-=b;
            }
            else{
                b-=a;
            }
84      }
        return a;
    }
```

```
    //the continued fractions algorithm as described in the
        appendix
89  //in the book of Nielsen and Chuang
    public static ArrayList<Integer> continuedFraction(int
        num, int denom){
        ArrayList<Integer> out = new ArrayList<Integer>();
        Fraction f = new Fraction(num,denom);
        out.add(f.num/f.denom);
94      f=f.subtract(f.num/f.denom);
        while(f.num != 0){
            f=f.inverse();
            out.add(f.num/f.denom);
            f=f.subtract(f.num/f.denom);
99      }
        return out;
    }

    //returns all the convergents from a continued fractions
        expansion
104 public static ArrayList<Fraction> convergents(ArrayList<
        Integer> a){
        ArrayList<Fraction> convs = new ArrayList<Fraction>()
            ;
        if(a.size()==0)
            return convs;
        convs.add(new Fraction(a.get(0),1));
109     if(a.size()==1)
            return convs;
        convs.add(new Fraction(1+a.get(0)*a.get(1),a.get(1)))
            ;
        for(int i = 2;i<a.size();i++){
            convs.add(
114                 new Fraction(
                    a.get(i)*convs.get(i-1).num +convs.get(i
                        -2).num,
                    a.get(i)*convs.get(i-1).denom + convs.get
                        (i-2).denom
                    )
            );
119     }

        return convs;

    }
```

```
124
        /**
         *
         * @param a integer
         * @param b integer
129      * @return least common multiple of a and b
         */
        public static int lcm(int a, int b){
            return a*b/gcd(a,b);
        }
134 }
```

References

[BBHT96] M. Boyer, G. Brassard, P. Hoyer, and A. Tapp. Tight bounds on quantum searching, 1996.

[CLRS01] T.H. Cormen, C.E. Leiserson, R.L. Rivest, and C. Stein. Introduction to algorithms, 2001.

[DHHM06] C. Durr, M. Heiligman, P. Hoyer, and M. Mhalla. Quantum query complexity of some graph problems. *SIAM Journal on Computing*, 35(6):1310–1328, 2006.

[Gro96] L.K. Grover. A fast quantum mechanical algorithm for database search. In *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*, pages 212–219. ACM New York, NY, USA, 1996.

[Gru99] Jozef Gruska. *Quantum computing*. McGraw-Hill England, 1999.

[Hir04] M. Hirvensalo. *Quantum computing*. Springer, 2004.

[Nak08] M. Nakahara. *Quantum computing: from linear algebra to physical realizations*. Inst of Physics Pub Inc, 2008.

[NC00] Michael A. Nielsen and Isaac L. Chuang. *Quantum Computation and Quantum Information*. Cambridge University Press, October 2000.

[NMN01] J. Nesetril, E. Milkova, and H. Nesetrilova. Otakar Boruvka on minimum spanning tree problem. *Discrete Math*, 233(1-3):3–36, 2001.

[Pri03] HA Priestley. *Introduction to complex analysis*. Oxford University Press, USA, 2003.

[Ros95] Kenneth H. Rosen. *Discrete mathematics and its applications*. McGraw-Hill New York, 1995.

[Sho94] P.W. Shor. Algorithms for quantum computation: Discrete log and factoring. In *Proceedings of the 35th Annual Symposium on Foundations of Computer Science*, pages 124–134, 1994.

[Ved06] V. Vedral. *Introduction to quantum information science*. Oxford University Press, USA, 2006.