CPSC 4110 Project Marking Sheet
Fall 2018
30 marks

Student names: David Connelly, Julius Moore	Score: 28

Components		Your Score	Remarks
Programming (20)	C-NOT (4 marks)	4	
	Toffoli (6 marks)	6 →	
	Deutch's algorithm (10 marks)	9 →	output is little awkward
Programming style (2 marks)		1	
Summary report (8 marks)		8	

CPSC4110 - Project

David Connolly and Julius Moore

December 5, 2018

GREEN.
BURTON & Co
USB STICK.

Abstract

Description of Components

Matrix ADT Class

To represent the quantum states and operators we created a matrix ADT. The basic arithmetic operations addition, multiplication, scaler multiplication were implemented by overloading the standard operators. Additionally the comparison operator (==) was overloaded to allow boolean comparisons of equality for matrices which is used for testing whether a matrix is a Hermitian. Non-member functions preforming the following operations were implemented:

tensor(Matrix, Matrix) The tensor function calculates the tensor product of two matrices and returns it. If A and B are matrices then *tensor(A,B)* preforms $A \otimes B$ returning a matrix with dimensions $(rows_A \cdot rows_B) \times (cols_A \cdot cols_B)$

trans(Matrix) The transposition returns a matrix that is transposition of the matrix given as an argument so for trans(A) where A is a matrix the transposition calculates: $(A[i,j])^T = A[j,i]$ for all entries in A.

dagger(Matrix) The dagger function returns the adjoint (conjugate transpose) of the matrix given to it as an argument.
A couple Useful Member functions in Matrix:

isUnitary() is member function that tests whether the matrix is unitary and returns a boolean answer. It determines this by calculating itself (M) multiplied by its own adjoint dagger(M) and seeing if the result is an identity matrix. $MM^\dagger = I$

isHerm() is a member function to

Circuit Class

The Circuit class is used to implement te

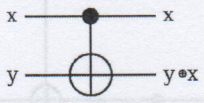
Implementation

Controlled-NOT

We implemented the CNOT gate by coding a matrix

$$CNOT = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

CNOT Circuit The CNOT corresponds to the following circuit diagram and truth table:



input		output	
x	y	x	y+x
|0⟩	|0⟩	|0⟩	|0⟩
|0⟩	|1⟩	|0⟩	|1⟩
|1⟩	|0⟩	|1⟩	|1⟩
|1⟩	|1⟩	|1⟩	|0⟩

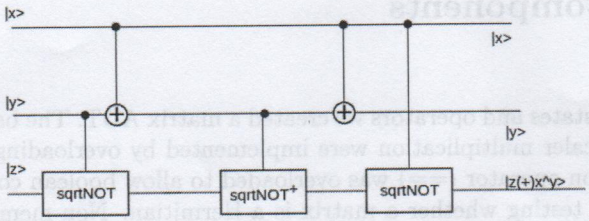


Figure 1

Figure 2

Toffoli Gate

We implemented the Toffoli gate with the following unitary matrix gates: $C\text{-}\sqrt{NOT}$, $C\text{-}\sqrt{NOT}^\dagger$, $C\text{-}NOT$ and $SWAP$.

$$\text{C-}\sqrt{NOT} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & \frac{1}{2}+\frac{1}{2}i & \frac{1}{2}-\frac{1}{2}i \\ 0 & 0 & \frac{1}{2}-\frac{1}{2}i & \frac{1}{2}+\frac{1}{2}i \end{bmatrix} \qquad \text{C-}\sqrt{NOT}^\dagger = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & \frac{1}{2}-\frac{1}{2}i & \frac{1}{2}+\frac{1}{2}i \\ 0 & 0 & \frac{1}{2}+\frac{1}{2}i & \frac{1}{2}-\frac{1}{2}i \end{bmatrix}$$

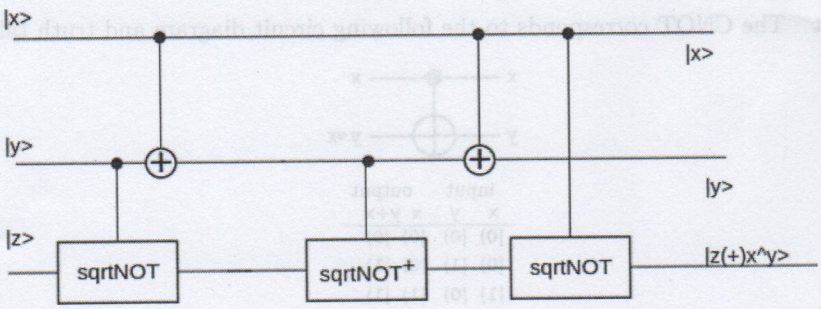
$$\text{SWAP} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad CNOT = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix} \qquad I^{[2\times 2]} = I = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

Putting these matrices together as follows:

$$U_{toffoli} = (SWAP \otimes I)(I \otimes \text{C-}\sqrt{NOT})(SWAP \otimes I)(CNOT \otimes I)(I \otimes \text{C-}\sqrt{NOT}^\dagger)(CNOT \otimes I)(I \otimes \text{C-}\sqrt{NOT})$$

$$= \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$

Circuit Diagram This corresponds to the following circuit diagram:



Note: On the circuit diagram there is not corresponding SWAP. This is because the CNOT and C-$\sqrt{NOT}$can only be applied to adjacent lines so the final application of the controlledC-$\sqrt{NOT}$ with the top and the bottom line requires the top to lines to be swapped and then swapped back after the gate is applied so when converting that type of operation to a matrix representation the swap is implicit.

Deutsch Algorithm

Setup Deutsch's algorithm takes a "blackbox" matrix which implements one of the 4 balanced or constant functions from the set 0, 1 to the set 0, 1. It's purpose is to discover whether the function is constant or balanced, and it acomplishes this by evaluating the function only once. To do this, deutsch's algorithm takes advantage of the superposition states that the Hadamard matrix creates; essentially feeding all of these states into the function; then evaluating that function's output. The top qubit, which is entangled, is read to determine the output: If it evaluates to 0, the blackbox is constant. Alternatively, it can evaluate to 1, which implies that the blackbox is balanced. The bottom qubit is discarded, and has a 50% chance of evaluating to either 1 or 0, and so contains no true information about the function through which it passed.

There are four functions from 0, 1 to 0, 1, and so four unitary matrices are used to represent them in our program. Say x is the top qubit and y is the bottom qubit. x emerges unchanged after passing through these matrices, being the control variable, while y emerges as y XOR f(x). Strangely enough, it is the top qubit x, that is evaluated. Some example matrices, and the functions they correspond to, are shown below, along with an image of the quantum circuit diagram.

we implemented the Deutsch Algorithm using the following circuit:

with this circuit it it is possible to test whether the function is balanced or constant

Functions

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad \begin{bmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix} \qquad \begin{bmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

$f(x) = 0$ $\qquad$ $f(x) = 1$ $\qquad$ $f(x) = x$ $\qquad$ $f(x) = NOT(x)$

Testing To test the algorithms functionality the different possible balanced input output mappings implemented in the black box and