Assignment 2

***Due**: Wednesday, Mar 2nd, 2021 @ 5:00PM*

CPSC 447/547 Introduction to Quantum Computing (Spring 2022)

1 Introduction

Welcome to Assignment 2 for CPSC 447/547 (Introduction to Quantum Computing). As usual, collaboration is encouraged; if you discussed with anyone besides the course staff about the assignment, *please list their names* in your submission.

Getting Started.

This assignment has two parts, a *written portion* and a *programming portion*. Typesetting your solutions to the written portion is not mandatory but highly encouraged. See the instructor's note on Ed for details about Latex for quantum computing. Some basic familiarity with Python and object-oriented programming is required to complete the programming portion of this assignment. To start,

- Create a folder for Assignment 2, e.g., `A2/`
- Download the starter files for this assignment to that folder from the course website:
  - `written.tex`
  - `A2.py`
  - `requirement_A2.py` (Do not modify)
- Write your solutions in `written.tex` and `A2.py`
- Debug and test your solution by running '`python3 A2.py`' on command line. This will check for any violation of the requirements and run correctness tests. Feel free to add more tests in `A2.py`. Do not hardcode your solutions for each test cases.

Submission.

Once you have completed and are ready to submit, upload two files to Canvas: `written.pdf` and `A2.py`. Late submissions (for up to two days) will receive a 50% penalty. Your written solution will be graded manually by our course staff; your programming solution will be graded using our grading script. If your file fails the `requirement_A2.py` check, a **0** score will be assigned, in which case you will have 2 days to fix the issue and re-submit, with a 50% penalty.

WRITTEN PORTION
This portion of the assignment has a total of 66% *points.*

2 Measurements

Recall from lecture, a quantum measurement, represented by an observable O, will probablistically collapse a quantum state to one of the eigen-basis states of O. By Spectral Theorem, we can write $O = \sum_i \lambda_i |e_i\rangle\langle e_i|$. We often associate the measurement outcome of quantum states with a random variable x, where $x = \lambda_i$ if the measurement outcome is its associated eigen-basis $|e_i\rangle$.

Task 2.1 (6%)

For each of the following quantum states, first write it in the form of $|x\rangle = \alpha|+\rangle + \beta|-\rangle$, where α, β are some complex numbers and $|\pm\rangle = \frac{|0\rangle \pm |1\rangle}{\sqrt{2}}$. Then answer what is the probability that the measurement outcome is $|+\rangle$, given that the observable is defined by the Pauli X operator $\sigma_x = (+1)|+\rangle\langle+| + (-1)|-\rangle\langle-|$?

(a) $|x\rangle = \frac{1}{\sqrt{2}}|0\rangle + \frac{i}{\sqrt{2}}|1\rangle$

(b) $|x\rangle = \frac{1}{\sqrt{2}}|0\rangle - \frac{i}{\sqrt{2}}|1\rangle$

(c) $|x\rangle = \frac{1+i}{2}|0\rangle + \frac{1-i}{2}|1\rangle$

Task 2.2 (12%)

Suppose we are given two qubits, A and B, in the following state: $|\psi_{AB}\rangle = \frac{1}{\sqrt{3}}|00\rangle + \frac{1}{\sqrt{3}}|01\rangle + \frac{1}{\sqrt{3}}|10\rangle$. For each of the following measurement scenarios (a-d), first write down the observable O and then calculate the expectation value when measuring $|\psi_{AB}\rangle$.

(a) Only measure qubit A in the x basis, that is, with observable $O = \sigma_x \otimes I$.

(b) Only measure qubit B in the x basis, that is, with observable $O = I \otimes \sigma_x$.

(c) Measure both qubits in the x basis, that is, with observable $O = \sigma_x \otimes \sigma_x$.

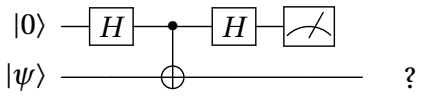
(d) Flip a fair coin; if the coin is Heads, we measure qubit A in the z basis; if it is Tails, we measure qubit A in the x basis. (Hint: observables are *linear* operators.)

(e) Following part (a), if the measurement outcome is $|+\rangle$, what is the state of qubit B after the measurement?

(f) Following part (d), if the coin-flip and measurement outcomes are (Heads, $|0\rangle$), what is the state of qubit B after the measurement?

Task 2.3 (10%)

Suppose we are given a qubit and a promise that the state of the qubit $|\psi\rangle$ is either $|+\rangle$ or $|-\rangle$. In this question, we will learn how to figure out which state the qubit without directly measuring $|\psi\rangle$. The key here is to use an ancillary qubit and two-qubit gates. Let's take a look at the following quantum circuit:



(a) Suppose $|\psi\rangle = |+\rangle$, then what is the probability that the top qubit measures to be $|0\rangle$? If non-zero, what is the state of the bottom qubit, given this measurement outcome?

(b) Suppose $|\psi\rangle = |+\rangle$, then what is the probability that the top qubit measures to be $|1\rangle$? If non-zero, what is the state of the bottom qubit, given this measurement outcome?

(c) Suppose $|\psi\rangle = |-\rangle$, then what is the probability that the top qubit measures to be $|0\rangle$? If non-zero, what is the state of the bottom qubit, given this measurement outcome?

(d) Suppose $|\psi\rangle = |-\rangle$, then what is the probability that the top qubit measures to be $|1\rangle$? If non-zero, what is the state of the bottom qubit, given this measurement outcome?

(e) Suppose $|\psi\rangle = \sqrt{1-\epsilon}\,|+\rangle + \sqrt{\epsilon}\,|-\rangle$ for some small ϵ, i.e., a slight perturbation from $|+\rangle$. Then what is the probability of measuring $|0\rangle$ for the top qubit?

3 Sharing Entanglement

Entanglement is one of the most powerful yet elusive properties in quantum computing. In lectures, we have studied entanglement between two parties, such as Alice and Bob. We saw how to use entanglement to accomplish something like quantum teleportation. The concept of entanglement can be generalized to three parties or more. In this question, we will explore what does it mean for more qubits to be entangled.

Task 3.1 (6%)

Suppose Alice, Bob, and Carol hold on to each of the three qubits in the quantum state $|\psi_{ABC}\rangle$, respectively. For each of the following scenarios of $|\psi_{ABC}\rangle$, first answer whether or not $|\psi_{ABC}\rangle$ is entangled (i.e., is it not possible to write $|\psi_{ABC}\rangle = |\psi_A\rangle \otimes |\psi_B\rangle \otimes |\psi_C\rangle$), then write down what is the joint state of Alice and Bob's qubits after Carol measured her qubit (conditioned on her measurement outcome). Finally, does Carol's measurement breaks the entanglement between Alice and Bob? Answer whether Alice's qubit and Bob's qubit are still entangled, if not, write down the product form $|\psi_A\rangle \otimes |\psi_B\rangle$.

(a) $|\psi_{ABC}\rangle = \frac{1}{\sqrt{2}}|000\rangle + \frac{1}{\sqrt{2}}|111\rangle$: "GHZ state"

(b) $|\psi_{ABC}\rangle = \frac{1}{\sqrt{3}}|001\rangle + \frac{1}{\sqrt{3}}|010\rangle + \frac{1}{\sqrt{3}}|100\rangle$: "W state"

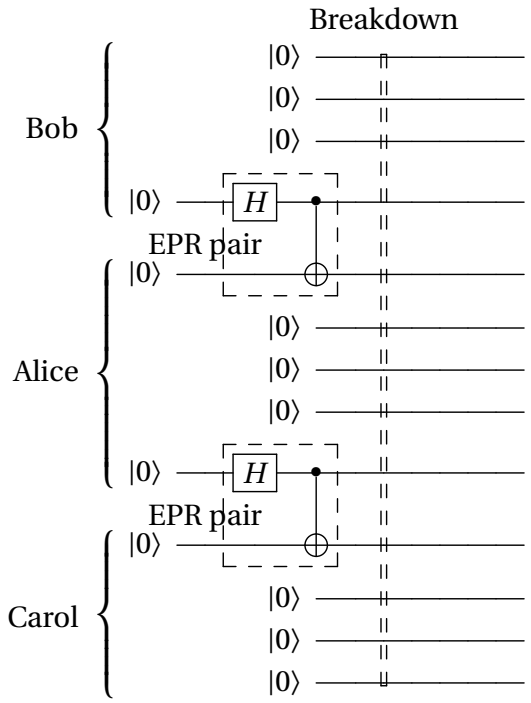
Task 3.2 (10%)

Suppose Alice lives in Chicago, Bob lives in Los Angeles, and Carol lives in New Haven. They can communicate by classical channels (such as sending bits via phone calls or radio signals) as well as quantum channels (such as qubits implemented by photons via optical fibers). One day, all quantum channels broke down... So they have to resort to using quantum teleportation for communicating qubits. Fortunately, right before the breakdown, Alice prepared two EPR pairs; she sent one of the qubit in the first EPR pair to Bob and one of the qubit in the second EPR

pair to Carol. However, Bob and Carol do not share entangled qubits with each other before the breakdown. In this case, can we still create an EPR pair shared between Bob and Carol despite having no quantum channel in between?

(a) Describe how you would accomplish this in fewer than five sentences.

(b) Complete the quantum circuit below to implement your algorithm. Note that no multi-qubit gates are allowed across different people's qubits after the breakdown point. Some qubits are initialized for each person; you do not have to use all of them.



4 Quantum Compiling

Task 4.1 (12%)

For the following sequences of gates, give their single gate equivalents. Write your answers using gates from the following set, with an appropriate phase factor:

$$\{I, X, Y, Z, H, S, S^\dagger, T, T^\dagger\}.$$

For example, $-Z$ or $e^{i\pi/16}Y$ are valid answers. For reference, we list the gate definitions here:

$$I = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, X = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}, Y = \begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix}, Z = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}, H = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix},$$

$$S = \begin{bmatrix} 1 & 0 \\ 0 & i \end{bmatrix}, S^\dagger = \begin{bmatrix} 1 & 0 \\ 0 & -i \end{bmatrix}, T = \begin{bmatrix} 1 & 0 \\ 0 & e^{i\pi/4} \end{bmatrix}, T^\dagger = \begin{bmatrix} 1 & 0 \\ 0 & e^{-i\pi/4} \end{bmatrix}$$

(a) XYZ

(b) $HZHZ$

(c) $TXT^\dagger X$

(d) $TZT^\dagger Z$

(e) $SXS^\dagger$

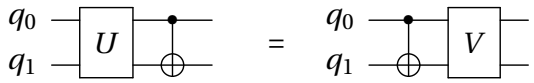
(f) $S^\dagger(TH)Z(TH)^\dagger$

Task 4.2 (10%)

For the following scenarios of U, solve for a V such that the circuit equality holds. Write your answers using gates from the following set, with an appropriate phase factor:

$$\{I, X, Y, Z, H, S, S^\dagger, T, T^\dagger, CNOT_{0,1}, CNOT_{1,0}, SWAP_{0,1}\}.$$

Note that at most one gate per qubit is allowed. For example, $-(Z_0X_1)$ is a valid answer. To avoid ambiguity, we use subscripts to indicate the qubit that the gate acts on. For example, $Z_0(HS)_1 = (Z \otimes H)(I \otimes S)$ and $CNOT_{0,1}$ has a control qubit q_0 and a target qubit q_1.



(a) $U = X_0 I_1$

(b) $U = I_0 X_1$

(c) $U = X_0 Z_1$

(d) $U = (ZX)_0 I_1$

(e) $U = SWAP_{0,1}$

PROGRAMMING PORTION

This portion of the assignment has a total of 34% *points.*

5 Quantum Compiler

In lectures, we discussed one of the topics in quantum compilation, namely circuit synthesis (which deals with decomposing an arbitrary multi-qubit unitary into a sequence of one- or two-qubit quantum gates). The role of a quantum compiler goes beyond circuit synthesis – register/qubit allocation is another topic in quantum compilation, mapping qubits in the quantum register of a program to qubits in the hardware backend.

Let's use an example program to motivate. Suppose we have a quantum computer (backend) with 5 qubits, i.e., $q_0, q_1, \ldots, q_4$. We want to run a quantum circuit with 4 qubits. Now we have a choice: we can run the circuit on $\{q_0, q_1, q_2, q_3\}$ or on $\{q_0, q_1, q_2, q_4\}$, etc. This process of choosing the qubits from backend is called the *register allocation* or *qubit mapping* problem. Typically, we have many constraints (such as quality of qubits, fidelity of gates, connectivity, etc) to force us to prefer one mapping over the other. In this question, we will simplify the problem

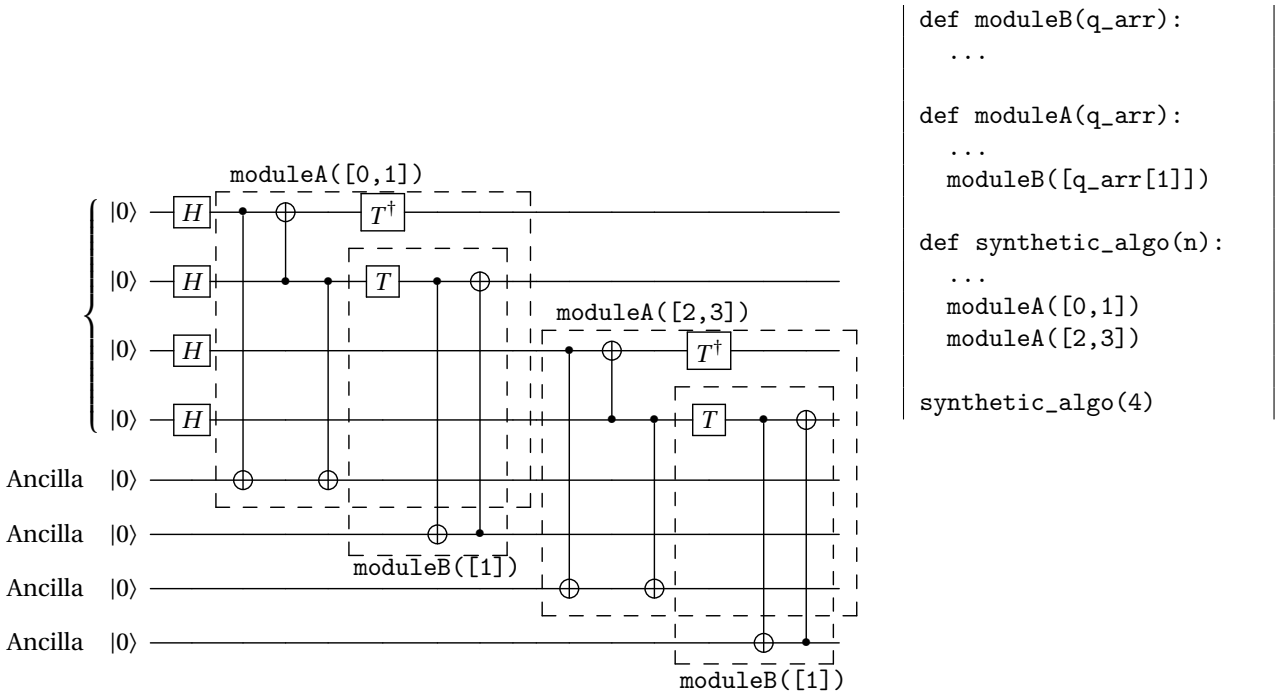


Figure 1: An example hierarchical quantum circuit: `synthetic_algo(4)`. Here dashed lines are boundaries of a module. The call structure (i.e., the relationship between modules) is shown on the right. Each instance of `moduleA` and `moduleB` needs to allocate an ancilla qubit.

by ignoring these constraints – without loss of generality, we assume to always prefer using the lower-indexed qubits. Thus, in the example earlier, the mapping $\{q_0, q_1, q_2, q_3\}$ wins.

How do we map qubits for programs with hierarchical structures? Consider a quantum circuit as shown below.

Because of its modular structure, the circuit can be viewed as multiple calls to a subroutine (boxed in dash lines). As such, a convenient way to write a quantum program as a sequence of quantum gates and sub-circuits. To provide programmer with this flexibility, we will augment the `QuantumCircuit` class in `A2.py` with some new features and implement a mini quantum compiler!

Note: Throughout this section, we assume that all `QuantumCircuits` are initialized with the same size of `QuantumRegister` and `ClassicalRegister`, that is `QuantumCircuit(n,n)` for some n.

Task 5.1 (4%)

A subroutine (sub-circuit) may take any subset of the qubits as input or even requires additional ones as ancillary qubits to perform its computation. In the `QuantumRegister` class (and similarly for the `ClassicalRegister` class), implement the following two functions:

- `select(self, ids)`: it takes a list of indices (`ids`) and return a new `QuantumRegister` which includes only the qubits indexed by `ids`. If `ids` is empty, raise an `Exception('ids must be non-empty!')`. *Hint:* please read carefully the new `__init__` definition for the `QuantumRegister` for information about different ways to initialize a quantum register.

- `__add__(self, other)`: it concatenates two quantum registers. We assume `reg1 + reg2` results in a larger array containing qubits in `reg1` followed by qubits in `reg2`. *Hint:* make sure the returned `QuantumRegister` has appropriate values of `size`, `array`, and `new_q`, inherited from the two inputs.

Note: `new_q` a list of indicator for whether the qubits in this register is newly initialized or copied from other existing registers. This is useful later when we need to keep track of allocations of qubits. For `ClassicalRegister`, you do not need to keep track of an analogous `new_q`.

Task 5.2 (5%)

In the `QuantumCircuit` class, implement the `allocate(self, ids, new)` function, which returns a tuple of `QuantumRegister` and `ClassicalRegister`. `ids` is a list of indices of qubits from `self.qubits` to include in the new register and `new` is a (non-negative) integer for the number of new qubits to include in the new register. That is, the new register includes the subset of qubits selected from `self.qubits` followed by `new` number of new qubits. Similarly, `new` number of classical bits should be allocated for the `ClassicalRegister`.

Task 5.3 (5%)

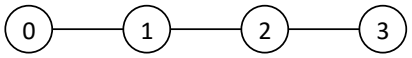
`Backend` is a class containing information about the hardware backend, including number of qubits (`num_q`), number of qubits that have already been allocated (`in_use`), a list of all `Qubits` (`all_qubits`), a name (`label`), and an indicator for whether the backend has a variable, infinite size (`variable==True`) or a fixed, finite size (`variable==False`). Implement the `alloc(self, n)` function, where `n` is a (non-negative) integer for the number of new qubits to allocate from the backend. We assume to allocate lower-indexed qubits first. So, if backend is finite and `(in_use + n)` exceeds `num_q`, raise an `Exception('Allocation error: not enough qubits!')`, otherwise return a list of indices (of length `n`) for those new qubits.

Task 5.4 (5%)

Implement the quantum circuit from Figure 1 in the `synthetic_algo` function in `A2.py`, using two sub-circuits, `moduleA` and `moduleB`.

Task 5.5 (7%)

Implement the `compile_fully_connected(self, backend, mapping)` function as part of the `QuantumCircuit` class, where `backend` is the intended hardware backend (assumed to be fully connected, i.e., a complete graph) to run the circuit and `mapping` is an (optional) existing qubit mapping. A mapping is a `Dictionary` of key-value pairs, where the key is the index of a qubit in the quantum register of the circuit and the value is the index of the qubit in the backend. Because of our assumptions, this same `mapping` is also valid for indexing classical bits. The `compile` function should return the flattened and mapped circuit as a list of gates – "flattened" means that the returned list should contain only quantum gates (i.e., without subcircuits) and "mapped" means that the qubits in the circuit are indices to the backend. Raise an `Exception('Backend too small!')` if the number of qubits needed in the circuit exceeds that of the backend.



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

Figure 2: An example backend of 4 qubits (left) and its adjacency matrix (right). Here `cx(0,3)` cannot be performed directly, because qubit 0 and qubit 3 are not connected.

Task 5.6 (8%)

Implement the `compile(self, backend, mapping)` function as in the previous task, but this time without assuming that the intended hardware backend is fully connected. Note that you can still assume that the backend is connected (that is no disconnected components). This means we need to update our definition of `Backend` to include information, and then implement an algorithm to resolve the connectivity constraints:

- In `Backend`, add an input argument `adj_matrix`, which is the adjacency matrix describing the connectivity graph of qubits in the backend. Thus, for a backend of `num_q` qubits, `adj_matrix` is a size `num_q`×`num_q` matrix with type `np.ndarray`. Lastly, do not forget to initialize a field for `adj_matrix`. Note: an infinite/variable backend can be assumed to be fully connected, in which case `adj_matrix` can be set to `None`.
- In additional to all requirements in Task 5.5, implement an algorithm, in `compile(self, backend, mapping)`, that replaces any two-qubit gates (e.g., `cx(0,3)` on the backend shown in Figure 2) acting on qubits that are not directly connected by the following sequence of gates:
  1. Swapping one of the qubit to be next to the other qubit. For instance, apply `swap(0,1)`, then `swap(1,2)`.
  2. Apply the two qubit gates on the connected qubits. In our example, `cx(2,3)`.
  3. Finally, swapping the qubit back to its original location. For instance, apply `swap(1,2)`, then `swap(0,1)`.

  Therefore, `compile` function will return a list of gates, where every two-qubit gate acts only on a pair of qubits that are connected. Note that, since `swap` gate is not part of the instruction set in `QuantumCircuit`, we need to use the `cx` gates to implement it. You can assume that the original program does not use the three-qubit gate, `toffoli`, from the instruction set.