Algorithm implementation

Description

We implement an algorithm for quantum walk based on a transducer from paper "Taming Quantum Time Complexity", authors of paper are Aleksandrs Belovs, Stacey Jeffery, and Duyal Yolcu. Our implementation is based on the description in Section 6 "Example I: Quantum Walks".

The walk is performed on a bipartite graph, edges are undirected. We denote the set of vertices on the left side as A, and set of vertices on the right side as B ($A \cap B = \emptyset$). The walk starts from probabilistic distribution on some vertices from the set A. Technically, walk is performed on edges, therefore, one extra vertex u' is added, and is connected with new added edges (called dangling edges) to the vertices of set A. The probability distribution is located on the new edges according to the initial probability distribution of the vertices of set A. Those probabilities on edges are normalized to represent a valid quantum state, which represents a superposition of dangling edges. The initial state is denoted as ξ.

The main part of the algorithm is based on a transducer operator $S_M = R_B R_A$. For each vertex from $u \in A \cup B$ that is not marked, we have an operator that is reflection over ψ_u, where ψ_u represents a superposition of all edges (including dangling ones for the vertices from set A) connected to the vertex u. If vertex is marked, i.e., $u \in M$, then we have identity operator. R_A is combined of all operators for each vertex from set A, and R_B is combined of all operators for each vertex from set B.

The operator S_M is applied to the system for several times, so we implement additional register with a counter that allows us the keep track of the state of the system at each step. In the end, we check the state of the system kept before the first application of S_M. State should remain ξ if there are marked vertices. State will be $-\xi$ if there were no marked vertices.

Implementation

We use the following two variables to prepare the system with required number of qubits: qubits_per_side, qubits_for_k. qubits_per_side means how many qubits we use to encode the vertices on each side of the bipartite graph. The edges are encoded with the following basis states: $x_1x_2x_3y_1y_2y_3$, where $x_1x_2x_3$ is basis state denoting vertex from set A (left side of the graph), and $y_1y_2y_3$ is basis state denoting vertex from set B (right side of the graph). Therefore, states that represent edges of the graph use 2*qubits_per_side qubits. We have a counter to perform the necessary number of operations S_M and keep track of the state before the first iteration. We perform k iterations in total, where $k = 2^{\text{qubits\_for\_k}}$. Therefore, we use qubits_for_k qubits for our counter.

We need one more qubit to distinguish between ξ and $-\xi$ in the end, which will serve us as a miniature Phase estimation procedure – translating phase into a basis state of a qubit.

In our encoding of the edges, we have special case for dangling edges – they are represented as vertices from the set A connected to the vertex 11…1 in a set B. Therefore, set B does not contain a vertex 11…1, but it denotes a virtual vertex connected to the vertices in the set A.

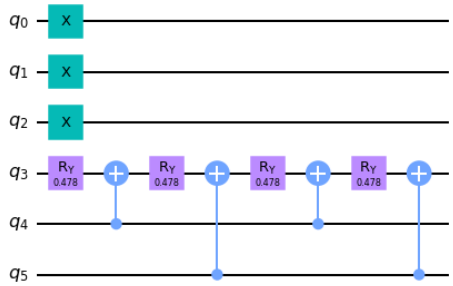
Here is example of the graph given by the following set of edges:

```
edges={
   '000111' : 1/3,
   '001111' : 2/3,
   '000000' : 1/2,
   '000001' : 1/2,
   '001000' : 1/2,
   '001001' : 1/2,
   '010000' : 1/2,
   '010001' : 1/2,
}
```

This means that in the set A we have vertices 000, 001, and 010; in the set B we have vertices 000 and 001. We have dangling edges to the vertex 000 with probability 1/3 and to the vertex 001 with probability 2/3. Here we have a complete bipartite graph, each vertex from set A is connected with each vertex from the set B, and each edge has weight 1/2.

Preparation of state ξ

We initialize qubits_per_side many qubits in state 1 to represent dangling edges, and remaining qubits are initialized in a superposition, which will correspond to the probabilistic distribution of vertices of set A, that contributed to the dangling edges. For our example, the circuit will look like this:



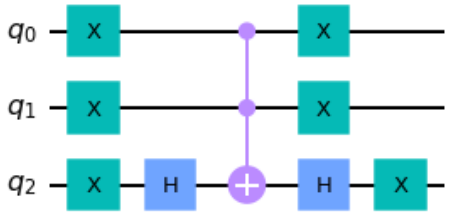
Implementation of operator S_M

For each vertex u that is not marked, we prepare state $|\psi_u\rangle$ is a superposition of edges that are connected to the vertex u. Weights on edges are taken into account, therefore, for vertex u we normalize weights of edges by dividing each weight by sum of absolute values of weights, and then taking square root.

Then, since we need to perform the reflection over state $|\psi_u\rangle$, we implement operator $I - 2|\psi_u\rangle\langle\psi_u|$. To get $I - 2|\psi_u\rangle\langle\psi_u|$, we need to implement the following sequence:

- apply a transformation that takes $|\psi_u\rangle$ to $|0\rangle$;
- apply $I - 2|0\rangle\langle 0|$;
- apply a transformation that takes $|0\rangle$ to $|\psi_u\rangle$.

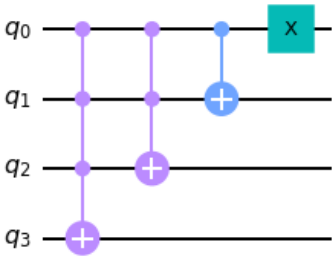
$I - 2|0\rangle\langle 0|$ can be implemented with the following circuit:



Then, we combine $I - 2|\psi_u\rangle\langle\psi_u|$ for each vertex of A into R_A, we combine $I - 2|\psi_u\rangle\langle\psi_u|$ for each vertex of B into R_B, and we concatenate two operations into $S_M = R_B R_A$.

Implementation of a counter

We prepare a counter with qubits_for_k qubits, applying Hadamard to each of qubits to form an equal superposition. We also need an operator that allows us to increase the value of the counter by +1 modulo k. This is done with the following quantum circuit:

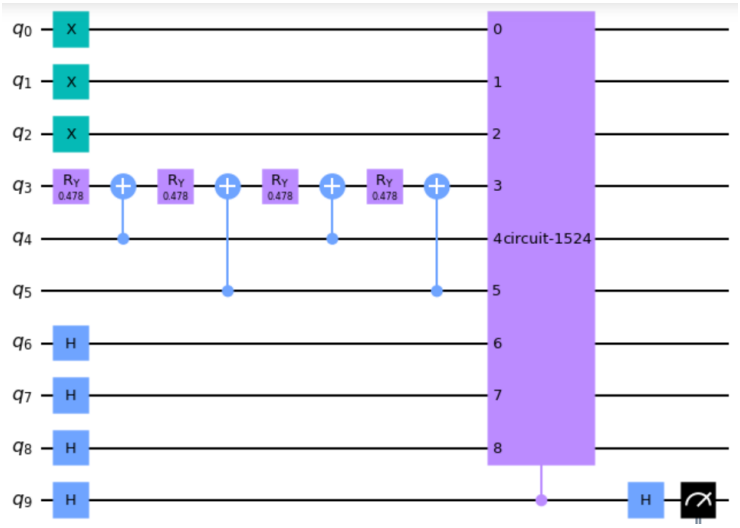


Final implementation

First, we initialize the state of our quantum system. We prepare state ξ and apply Hadamards to qubits of our step counter.

The main operation of our circuit is combination of S_M and counter increase operation. State ξ has qubits_per_side qubits initialized in state 1, therefore, if we want a counter to separate that state from the other system, we apply counter that is controlled by mentioned qubits_per_side, meaning, that counter will only increase together with the state ξ, associated with dangling edges. We perform k iterations, therefore, the final operation consists of k repetitions of S_M + counter increase controlled by mentioned qubits_per_side qubits. We denote this complete operation as U.

Since we perform simplified version of Phase estimation to distinguish between phases +1 and -1, we use one more qubit, that is initialized in a superposition (Hadamard applied), then we use this qubit to control operation U, and then we apply Hadamard and measure the qubit. The example of final circuit:



Top 6 qubits represent space of our edges, where all operations before large purple block initialize state ξ. Qubits q6, q7, and q8 store counter, Hadamards initialize it in a superposition. Qubit q9 is for Phase estimation. It controls large purple block, that represents circuit U. If we measure state 0 in last qubit, it means that the state is ξ, meaning that there were marked vertices. If measurement outcome is state 1, then state is $-\xi$, and there were no marked vertices detected.

Experiments

We try our code for the following parameters

qubits_per_side = 3
qubits_for_k = 3
and graph is like in the description of implementation:

edges={
'000111' : 1/3,
'001111' : 2/3,

'000000' : 1/2,
'000001' : 1/2,
'001000' : 1/2,
'001001' : 1/2,
'010000' : 1/2,
'010001' : 1/2,
}
We repeat experiment for 10000 times.

When there are no marked vertices, we get outcome: `{'1': 3778, '0': 6222}`
With marked vertex ‘000’ in set A, we get outcome: `{'0': 9830, '1': 170}`
With marked vertex ‘001’ in set B, we get outcome: `{'0': 8237, '1': 1763}`
With marked vertex ‘010’ in set A and vertex ‘000’ in set B, we get outcome: `{'1': 2138, '0': 7862}`

We see that results are distinguishable, but still probability of state 1 in case of no marked vertices is still small. Next, we try larger number of steps, by setting qubits_for_k = 4.

When there are no marked vertices, we get outcome: `{'0': 1809, '1': 8191}`
With marked vertex ‘000’ in set A, we get outcome: `{'0': 9447, '1': 553}`
With marked vertex ‘001’ in set B, we get outcome: `{'0': 7248, '1': 2752}`
With marked vertex ‘010’ in set A and vertex ‘000’ in set B, we get outcome: `{'1': 4047, '0': 5953}`

We also tested with different weights and connections between vertices.

edges={
'000111' : 1/3,
'001111' : 2/3,
'000000' : 1/5,
'000001' : 1/4,
'001000' : 1/6,
'001001' : 1/7,
'010000' : 1/4
}
We used the following parameters

qubits_per_side = 3
qubits_for_k = 3
We repeat experiment for 10000 times.

When there are no marked vertices, we get outcome: `{'0': 5832, '1': 4168}`
With marked vertex ‘001’ in set A, we get outcome: `{'0': 9895, '1': 105}`
With marked vertex ‘001’ in set B, we get outcome: `{'0': 9807, '1': 193}`
With marked vertex ‘001’ in set A and vertex ‘001’ in set B, we get outcome: `{'0': 9993, '1': 7}`

Next, we try larger number of steps, by setting qubits_for_k = 4.

When there are no marked vertices, we get outcome: `{'0': 4598, '1': 5402}`
With marked vertex ‘001’ in set A, we get outcome: `{'0': 9601, '1': 399}`
With marked vertex ‘001’ in set B, we get outcome: `{'1': 728, '0': 9272}`
With marked vertex ‘001’ in set A and vertex ‘001’ in set B, we get outcome: `{'0': 9988, '1': 12}`