COMP2123C Programming technologies and tools

Assignment 3

Get lost in HKU? ☺

Due day: 24 April 23:00

Consider the following directed graph with 6 nodes and 12 edges.

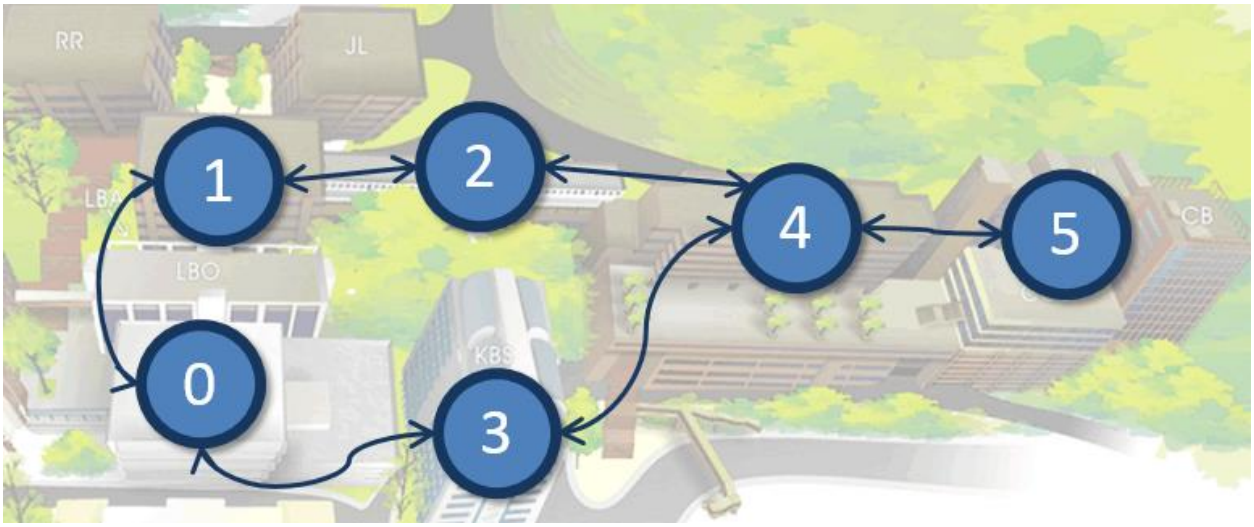

Figure 1. A directed graph with 6 nodes and 12 directed edges (a bidirectional edge is regarded as two directed edges in this question, so there are 12 directed edges in total).

- A `Node` class is defined as follow

```
class Node{
    public:
        Node();
        Node(int,string);
        int id;
        string name;
};
```

- A `Graph` class is defined as follow

```
class Graph{
    public:
         void InsertNode( Node x );
         void InsertEdge( int x, int y);
         void CommonNeighbor( int x, int y );
         void ShortestPath(int source, int destination);
         // You can add more member functions to help your
implementation.
    private:
         /* The data type is decided by you */ nodes;
         /* The data type is decided by you */ edges;
};
```

- We would like to implement a program `graph.cpp` that supports four commands, namely `InsertNode`, `InsertEdge`, `CommonNeighbor` and `ShortestPath`:

The program source code is show below.

```
#include<iostream>
#include<string>
#include<algorithm>
#include<map>
#include<vector>
// You can add more libraries here (if needed)



using namespace std;

// Please define the classes and member functions here




int main(){
   Graph g;
   string command;
   int id1, id2;
   string name;

   while (cin >> command ){
         if (command == "InsertNode"){
            cin >> id1 >> name;
            Node n(id1,name);
            g.InsertNode(n);
         }if (command == "InsertEdge"){
            cin >> id1 >> id2;
            g.InsertEdge(id1,id2);
         }else if (command == "CommonNeighbor"){
            cin >> id1 >> id2;
            g.CommonNeighbor(id1, id2);
         }else if (command == "ShortestPath"){
            cin >> id1 >> id2;
            g.ShortestPath(id1, id2);
         }else if (command == "Exit"){
            return 0;
         }
   }
}
```

1. `InsertNode x y` – Insert the node with `id` as `x` and name as `y` in a graph. For example, we issue the following commands to insert `6` nodes in the graph in Figure 1.

```
InsertNode 0 Library_Building
InsertNode 1 Hui_Oi_Chow_Science_Building
InsertNode 2 University_Street
InsertNode 3 Kadoorie_Biological_Sciences_Building
InsertNode 4 Haking_Wong_Building
InsertNode 5 Chow_Yei_Ching_Building
```

   - You can assume that the `name` of the nodes will not contain any space.
   - The `id` of the nodes may not start from `0` and may not be in ascending order.
   - Decide a suitable STL container to implement the `Graph::nodes` member variable. We need to access to a `Node` given the node `id` in other operations. Some containers may greatly simplify your program☺.
   - Before you proceed to the next part, you may implement `Graph::printAllNodes()` to output the name of all nodes in the `Graph` object and validate the correctness of your program.

2. `InsertEdge x y` – Insert an edge `x`→`y` in the graph. Where `x` and `y` are the `id` of nodes. For example, we issue the following commands to insert the 12 edges in the graph in Figure 1.

```
InsertEdge 0 1
InsertEdge 1 0
InsertEdge 1 2
InsertEdge 2 1
InsertEdge 0 3
InsertEdge 3 0
InsertEdge 2 4
InsertEdge 4 2
InsertEdge 3 4
InsertEdge 4 3
InsertEdge 4 5
InsertEdge 5 4
```

   - If there is no `Nodes` in the `Graph` with `id` equal to `x` or `y`, output "`No such node.`" on screen.
   - Decide a suitable STL container to implement the `Graph::edges` member variable. We would like to access to a collection of neighbor `Node`(s) given the `id` of a node. Some containers may greatly simplify your program☺.

3. `CommonNeighbor x y` – Returns the common neighbor of nodes `x` and `y`. (You can assume that both node `x` and `y` exist in the graph)

```
CommonNeighbor 1 3
0 Library_Building
CommonNeighbor 1 5
No common neighbor.
CommonNeighbor 1 0
No common neighbor.
```

- If there are no common neighbor, output "`No common neighbor.`".
- If there are more than one common neighbors, output them in ascending order of the node ID, line by line.
- You are strongly recommended to use the `set_intersection()` algorithm to implement this member function. http://www.cplusplus.com/reference/algorithm/set_intersection/. This will be a good practice to use libraries provided by C++.

4. `ShortestPath x y` – Returns the shortest path from node `x` to node `y`.

```
ShortestPath 0 4
0 Library_Building
3 Kadoorie_Biological_Sciences_Building
4 Haking_Wong_Building
ShortestPath 1 5
1 Hui_Oi_Chow_Science_Building
2 University_Street
4 Haking_Wong_Building
5 Chow_Yei_Ching_Building
```

- If there are more than one shortest paths, you can output any one of those.
- If node x cannot reach node y, output "`No path found.`".

How to return the shortest path? We may use the **breath first search** technique as follow:

Step 1. Initialization.

- We define 3 STL containers that help the searching process
  i) `queue<int> q;` - Please visit http://www.cplusplus.com/reference/queue/queue/ to learn more about this STL container.
  ii) `map<int,int> previous;` - For remembering the previous node in a path.
  iii) `map<int,bool> visited;` - For remembering if a node is visited or not.
- For each node `x` in the graph, initialize `visited[x]` to `false` and `previous[x]` to `-1`.
- Put `source` in the queue `q` by `q.push(source);`
- Set the source node as "visited" by setting `visited[source]=true`.

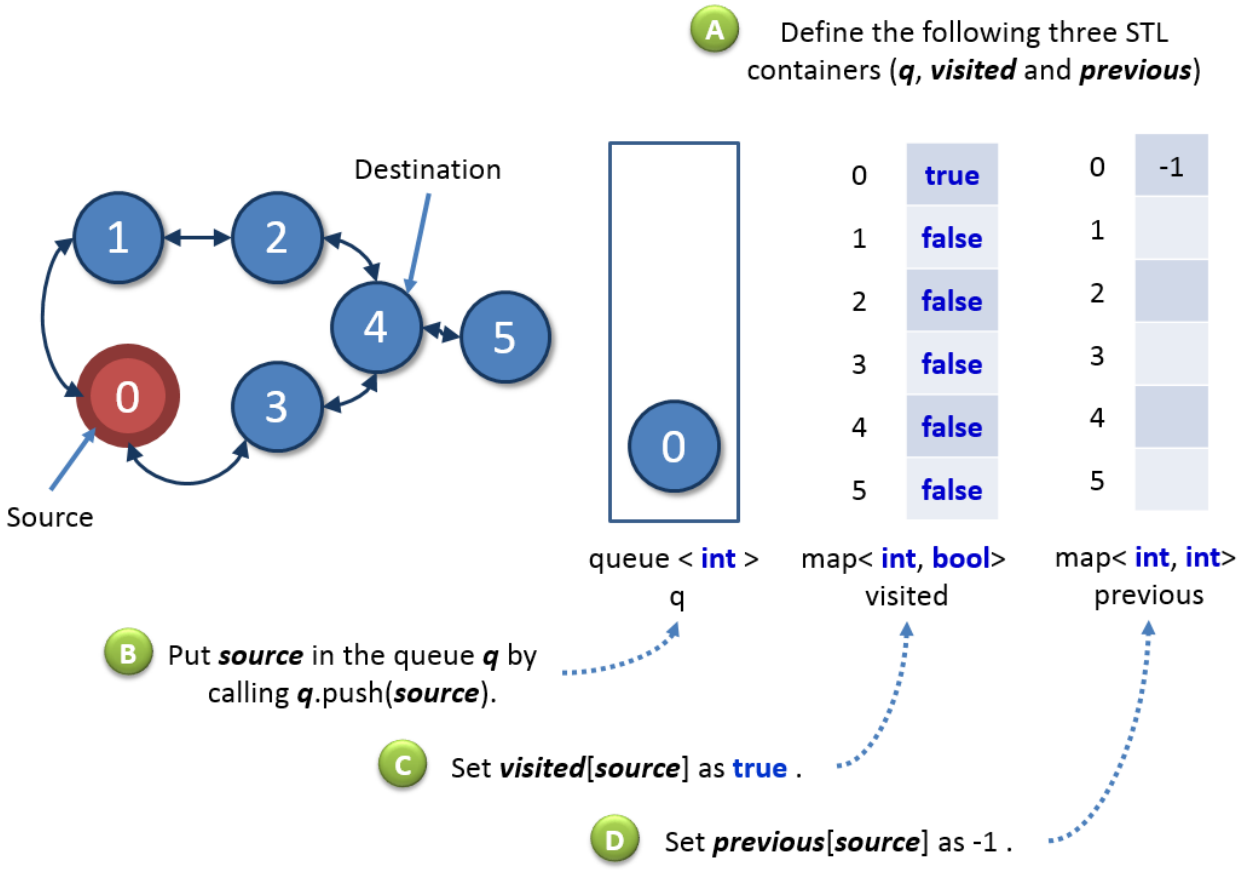

Figure 2a. Initialization of the searching process.

Step 2. Start searching.

- While the destination node is not reached and the queue `q` is not empty, we keep on searching.

```
while (/*the queue q is empty*/ == false && foundDestination == false){



}
```

i) Dequeue one node from queue, and store the node in `current`.

```
int current = q.front();
q.pop();
```

ii) For each neighbor `n` of the current node that `visited[n]` is `false`, push them to the queue `q`. You may want to search in the C++documentation about the member function of the queue container for this purpose.

iii) Set node `n` as visited by setting `visited[n]=true`.

iv) Set `previous` of `n` as `current` by setting `previous[n]=current`.

v) Repeat i) if the destination node is not reached and the queue `q` is not empty.

- Figure 2b-2d give a step-by-step illustration to help you better understand the searching process.

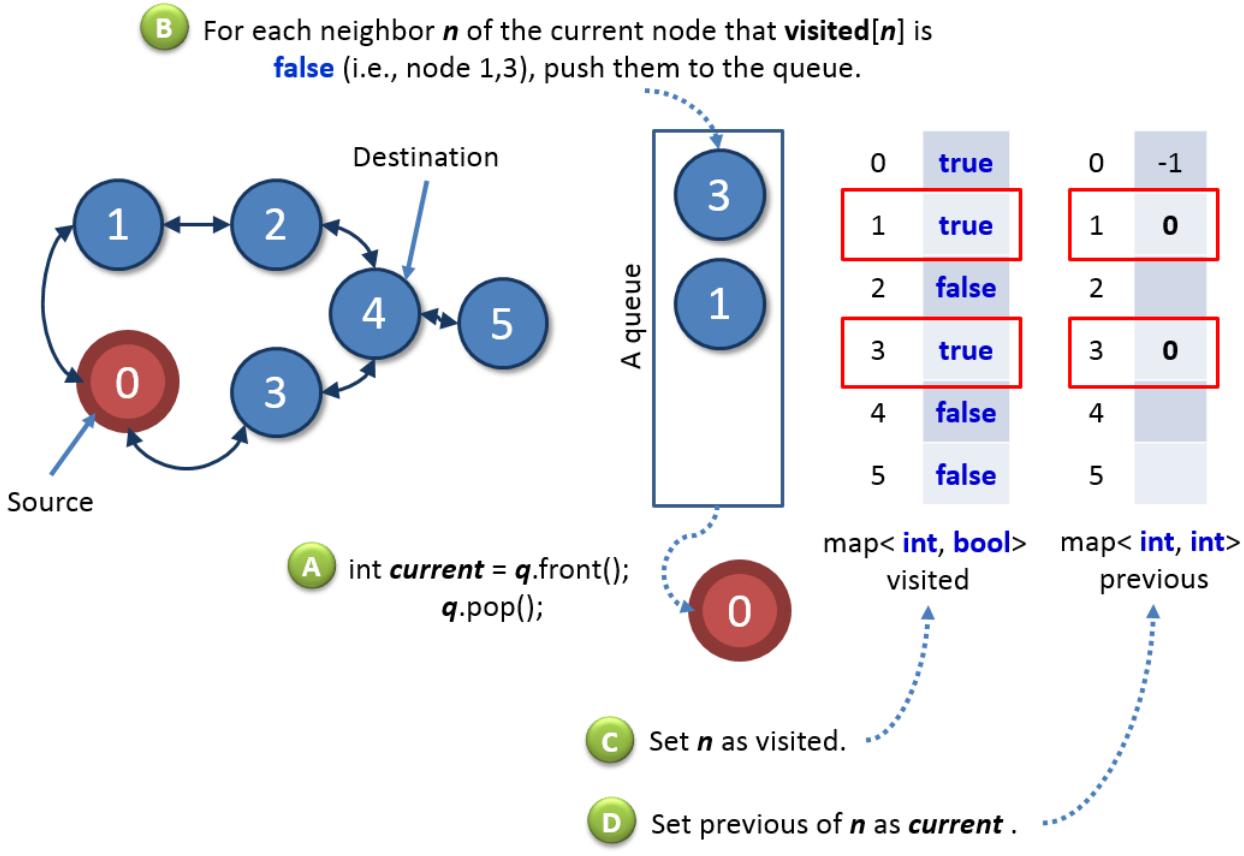

Figure 2b. The instance after exploring the neighbor of node 0 (the source node).

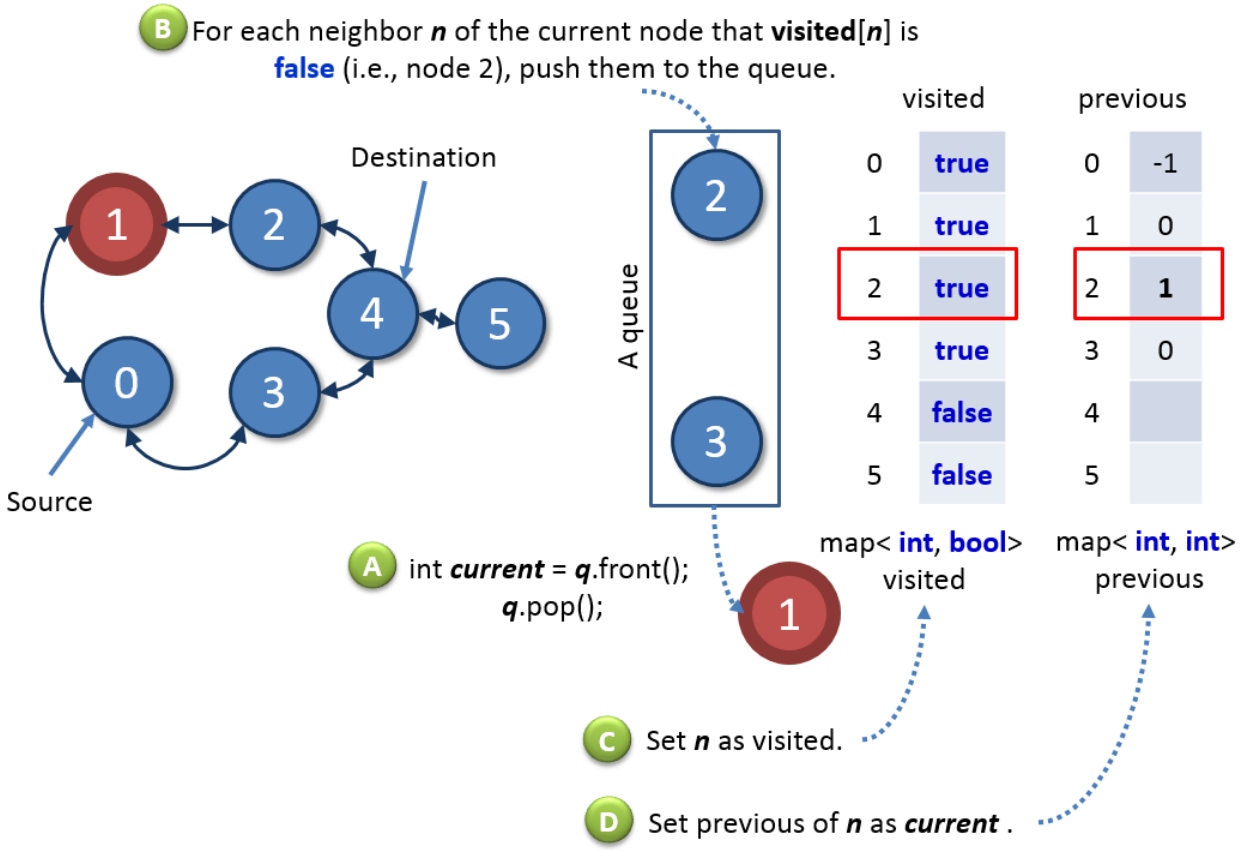

Figure 2c. The instance after exploring the neighbor of node 1.

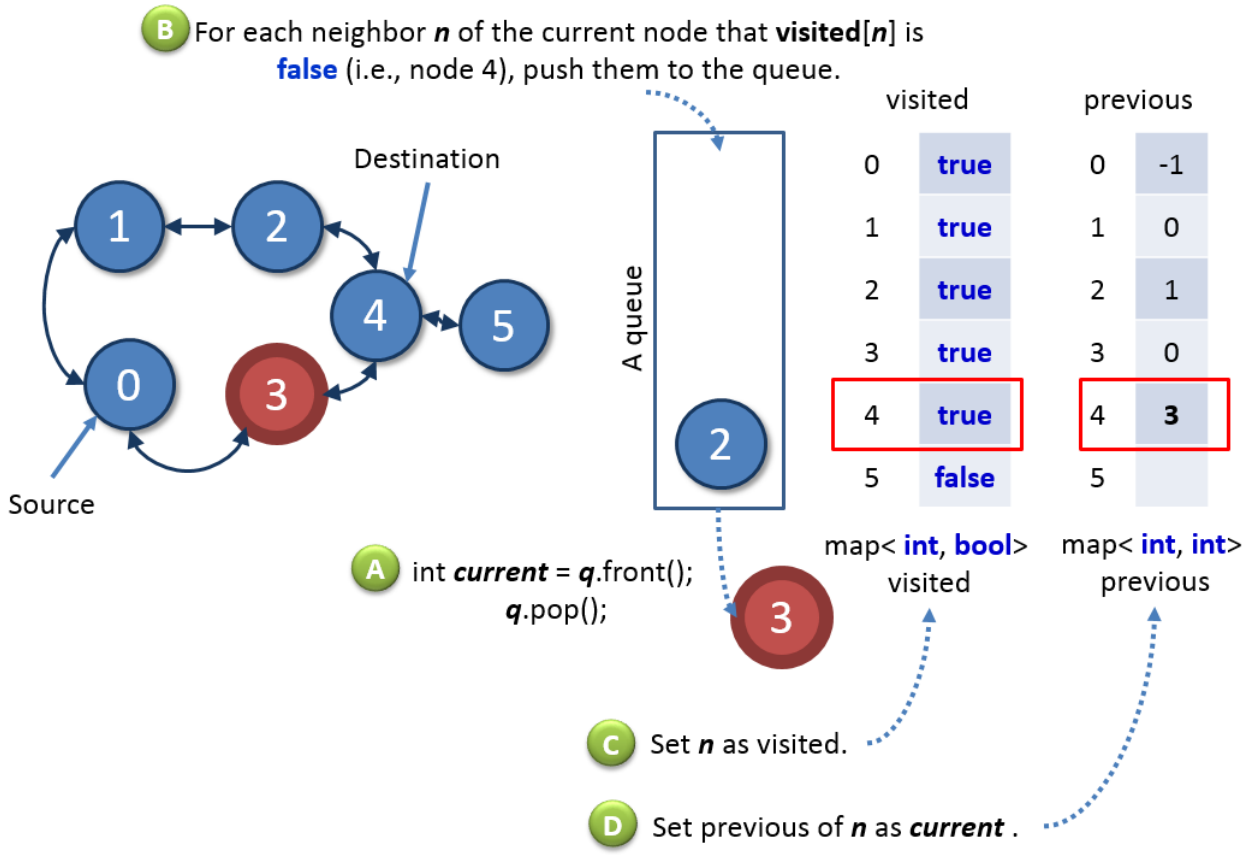

Figure 2d. The instance after exploring the neighbor of node 3, since the destination node is reached, we can break the while loop. The shortest path information is now stored in `previous`.

Dear Students.

We wish you enjoy this assignment. Please feel free to let me know if you face any difficulties when working on this assignment. I am happy to help you ☺. We wish you enjoy learning programming technologies and tools in this course!

Best regards,

Kit

-- END --