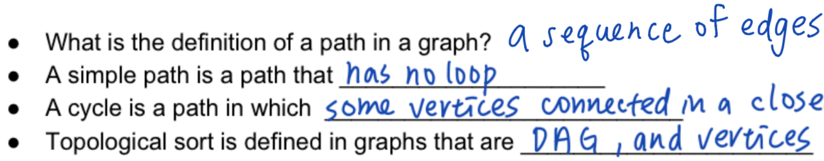
EE599 HW#7

Student Name: Shi Lin Chen

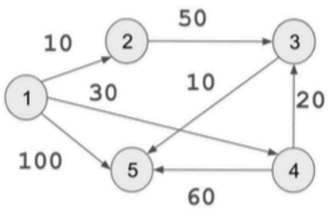
Student ID: 2991911997

1.



- What is the definition of a path in a graph? a sequence of edges, join a sequence of vertice
- A simple path is a path that has no loop
- A cycle is a path in which some vertices connected in a closed chain $(v_1 = v_n)$
- Topological sort is defined in graphs that are DAG, and vertices have order!

2.



	1	2	3	4	5
1	0	10	∞	30	100
2	10	0	50	∞	∞
3	∞	50	0	20	10
4	30	∞	20	0	60
5	100	∞	10	60	0

K = 0

	1	2	3	4	5
1	0	10	∞	30	100
2	10	0	50	∞	∞
3	∞	50	0	20	10
4	30	∞	20	0	60
5	100	∞	10	60	0

K = 1

	1	2	3	4	5
1	0	10	60	30	100
2	10	0	50	∞	∞
3	60	50	0	20	10
4	30	∞	20	0	60
5	100	∞	10	60	0

K = 2

	1	2	3	4	5
1	0	10	60	30	70
2	10	0	50	∞	60
3	60	50	0	20	10
4	30	∞	20	0	30
5	70	60	10	30	0

K = 3

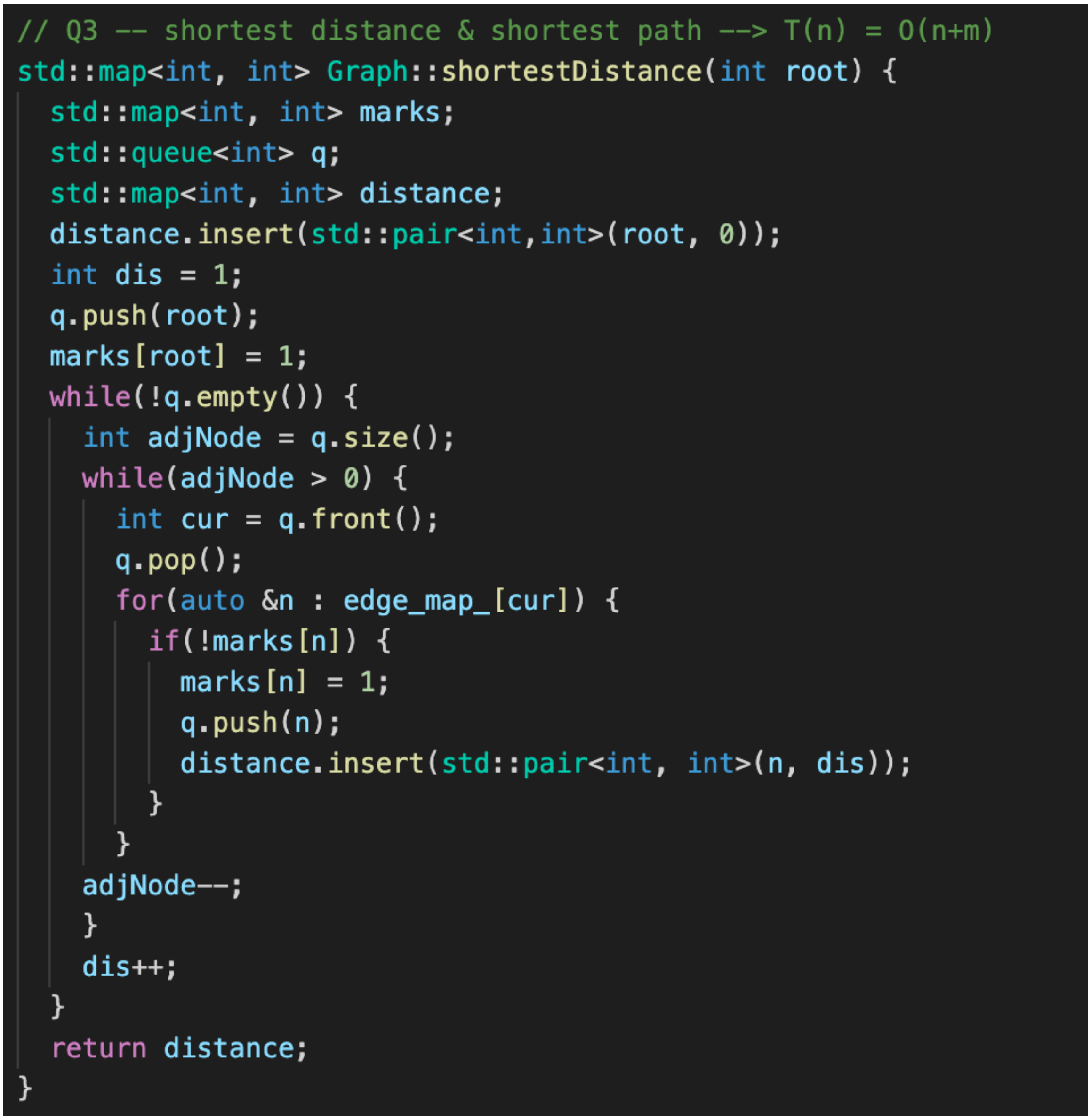
	1	2	3	4	5
1	0	10	50	30	60
2	10	0	50	∞	60
3	50	50	0	20	10
4	30	∞	20	0	30
5	60	60	10	30	0

K = 4

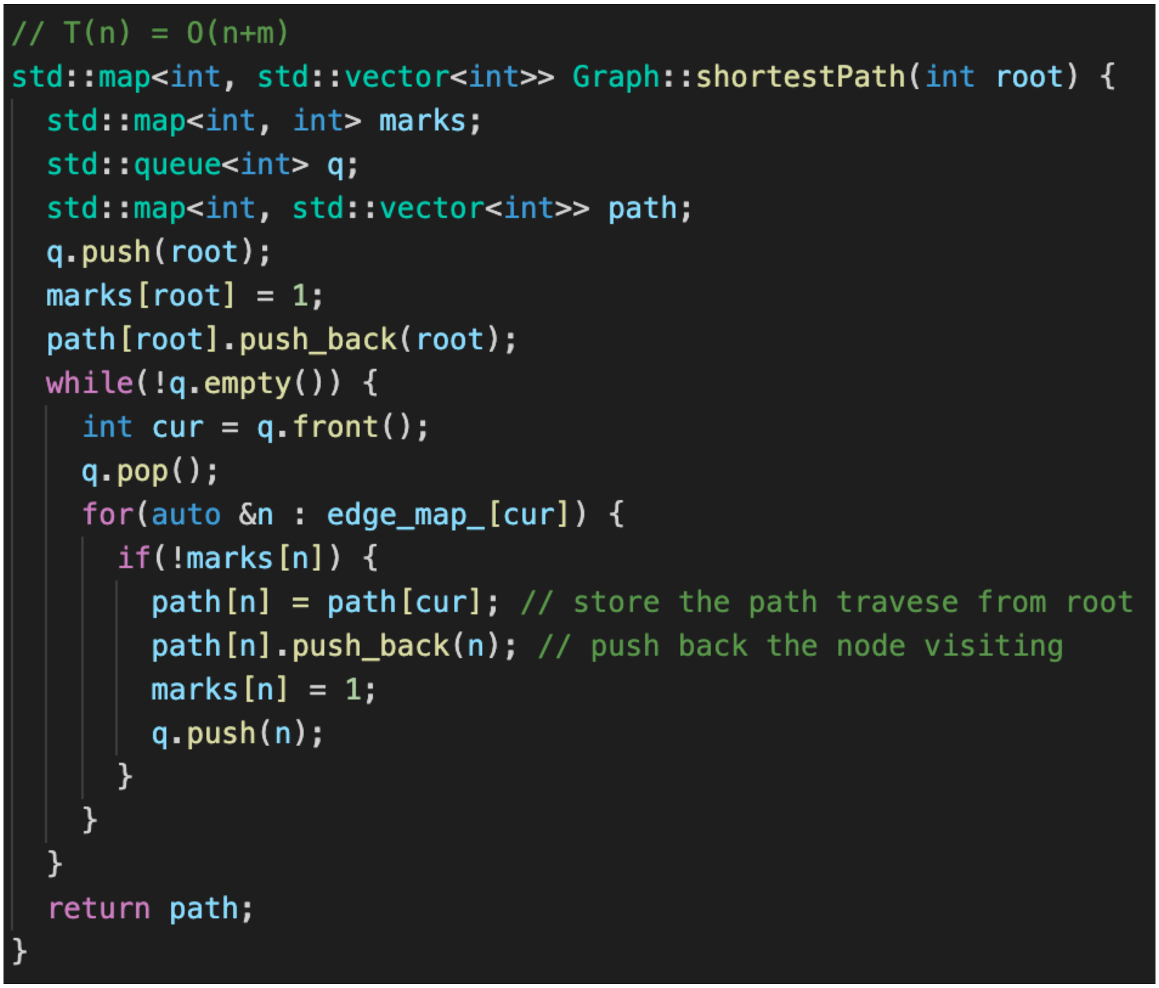
	1	2	3	4	5
1	0	10	50	30	60
2	10	0	50	∞	60
3	50	50	0	20	10
4	30	∞	20	0	30
5	60	60	10	30	0

K = 5

3.

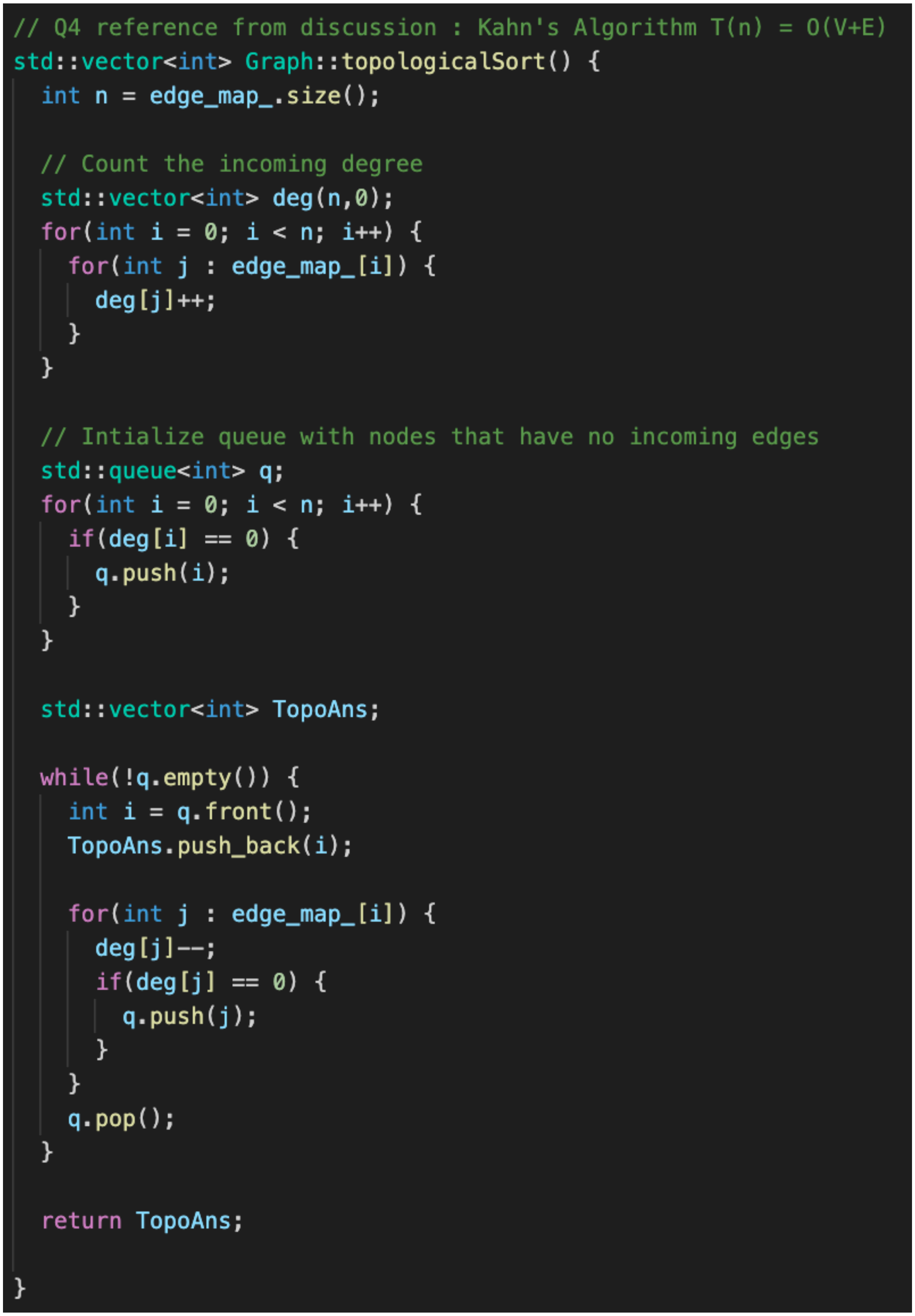


```
// Q3 -- shortest distance & shortest path --> T(n) = O(n+m)
std::map<int, int> Graph::shortestDistance(int root) {
  std::map<int, int> marks;
  std::queue<int> q;
  std::map<int, int> distance;
  distance.insert(std::pair<int,int>(root, 0));
  int dis = 1;
  q.push(root);
  marks[root] = 1;
  while(!q.empty()) {
    int adjNode = q.size();
    while(adjNode > 0) {
      int cur = q.front();
      q.pop();
      for(auto &n : edge_map_[cur]) {
        if(!marks[n]) {
          marks[n] = 1;
          q.push(n);
          distance.insert(std::pair<int, int>(n, dis));
        }
      }
    adjNode--;
    }
    dis++;
  }
  return distance;
}
```



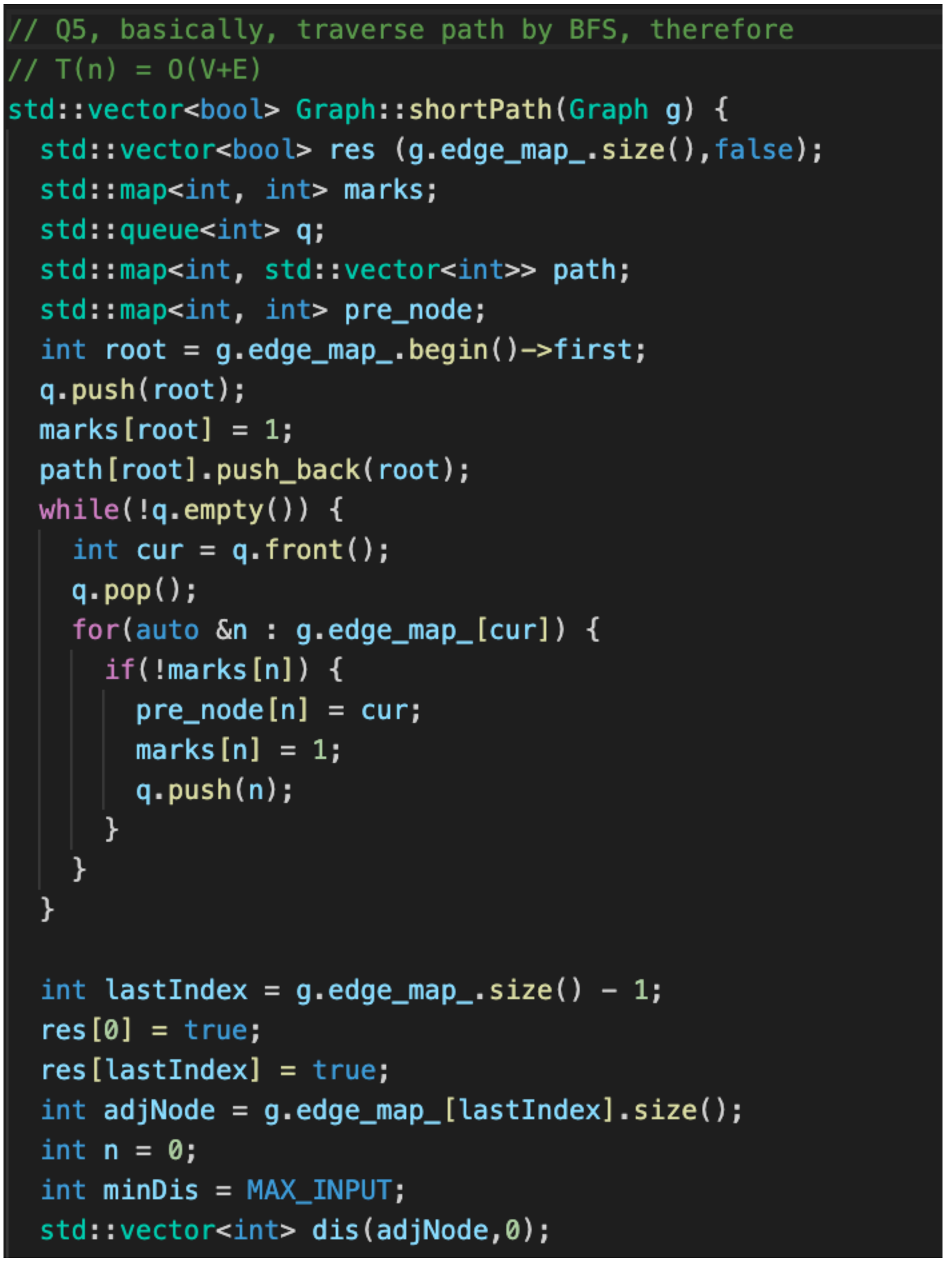
```
// T(n) = O(n+m)
std::map<int, std::vector<int>> Graph::shortestPath(int root) {
  std::map<int, int> marks;
  std::queue<int> q;
  std::map<int, std::vector<int>> path;
  q.push(root);
  marks[root] = 1;
  path[root].push_back(root);
  while(!q.empty()) {
    int cur = q.front();
    q.pop();
    for(auto &n : edge_map_[cur]) {
      if(!marks[n]) {
        path[n] = path[cur]; // store the path travese from root
        path[n].push_back(n); // push back the node visiting
        marks[n] = 1;
        q.push(n);
      }
    }
  }
  return path;
}
```

4.

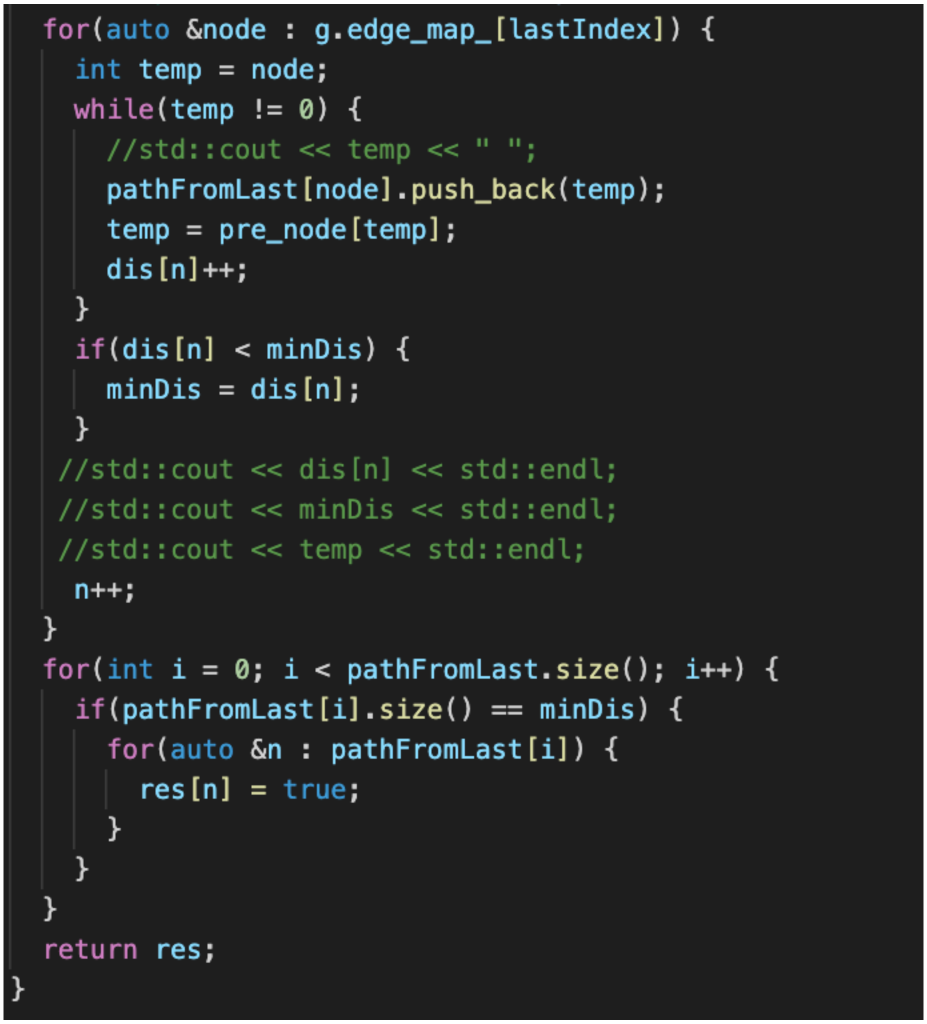


```cpp
// Q4 reference from discussion : Kahn's Algorithm T(n) = O(V+E)
std::vector<int> Graph::topologicalSort() {
  int n = edge_map_.size();

  // Count the incoming degree
  std::vector<int> deg(n,0);
  for(int i = 0; i < n; i++) {
    for(int j : edge_map_[i]) {
      deg[j]++;
    }
  }

  // Intialize queue with nodes that have no incoming edges
  std::queue<int> q;
  for(int i = 0; i < n; i++) {
    if(deg[i] == 0) {
      q.push(i);
    }
  }

  std::vector<int> TopoAns;

  while(!q.empty()) {
    int i = q.front();
    TopoAns.push_back(i);

    for(int j : edge_map_[i]) {
      deg[j]--;
      if(deg[j] == 0) {
        q.push(j);
      }
    }
    q.pop();
  }

  return TopoAns;

}
```

5.



```cpp
// Q5, basically, traverse path by BFS, therefore
// T(n) = O(V+E)
std::vector<bool> Graph::shortPath(Graph g) {
  std::vector<bool> res (g.edge_map_.size(),false);
  std::map<int, int> marks;
  std::queue<int> q;
  std::map<int, std::vector<int>> path;
  std::map<int, int> pre_node;
  int root = g.edge_map_.begin()->first;
  q.push(root);
  marks[root] = 1;
  path[root].push_back(root);
  while(!q.empty()) {
    int cur = q.front();
    q.pop();
    for(auto &n : g.edge_map_[cur]) {
      if(!marks[n]) {
        pre_node[n] = cur;
        marks[n] = 1;
        q.push(n);
      }
    }
  }

  int lastIndex = g.edge_map_.size() - 1;
  res[0] = true;
  res[lastIndex] = true;
  int adjNode = g.edge_map_[lastIndex].size();
  int n = 0;
  int minDis = MAX_INPUT;
  std::vector<int> dis(adjNode,0);
```



```
  for(auto &node : g.edge_map_[lastIndex]) {
    int temp = node;
    while(temp != 0) {
      //std::cout << temp << " ";
      pathFromLast[node].push_back(temp);
      temp = pre_node[temp];
      dis[n]++;
    }
    if(dis[n] < minDis) {
      minDis = dis[n];
    }
   //std::cout << dis[n] << std::endl;
   //std::cout << minDis << std::endl;
   //std::cout << temp << std::endl;
    n++;
  }
  for(int i = 0; i < pathFromLast.size(); i++) {
    if(pathFromLast[i].size() == minDis) {
      for(auto &n : pathFromLast[i]) {
        res[n] = true;
      }
    }
  }
  return res;
}
```