Homework 9 - shortest path in graph

Problem 1:

What is the big-Oh space complexity of Dijkstra's? Justify your answer.

ANS

Based on the implementation in the homework, the representation of the graph is an adjacency list, and using the a circular queue to implement Dijkstra's algorithm.

Assume that the number of vertices of a graph is ***n**,* and there are ***e*** edges in it. The big-Oh space complexity is $O(n + e)$.

We define one struct graph_node_t for each node, another struct graph_edge_t for each edge in the graph. All ***n*** nodes of the graph are stored in a Double Linked List, for each node there are two Double Linked Lists storing the out neighbors $\boldsymbol{e_{to}}$ **(<*e*)** edges and in neighbors $\boldsymbol{e_{from}}$**(<*e*)** edges.

For the implementation of the algorithm, we use one Double Linked List to store visited nodes, meanwhile using one Circular Queue to store un-visited nodes. The maximum size of DLL or Queue is ***n.***

In the worst case, the number nodes and edges stored in DLLs or Queue would be equal to ***n*** and ***e.*** Thus, the largest number of nodes is ***an,** which is* ⇒ $\boldsymbol{n}$**,** the largest number of edges is ***be,** which is* ⇒ ***e.*** The big-Oh space complexity is $O(n + e)$.

Problem 2:

What is the big-Oh time complexity for Dijkstra's? Justify your answer.

ANS

Based on the implementation in the homework, the representation of the graph is an adjacency list, and using the a circular queue to implement Dijkstra's algorithm.

Assume that the number of vertices of a graph is ***n,*** and there are ***e*** edges in it. The big-Oh time complexity is $O(n^2)$.

For each u in Graph: u.cost =INFINITY run **n** times.
Pushing all nodes to Q, all cost is infinity runs **n** times.

For the implementation of the Dijkstra's algorithm, we create queue for un-visited nodes storage, the size is the number of all nodes in the graph ***n.***
The first loop that while *queue is not empty* need to operate ***n*** times, at each time we will pop one node with minimum cost and push it into visited double linked list.
The second loop while $position < queue -> size, position ++ \ size --$, the operation time is like $n + n(n-1) + (n-2) \ldots. + 1 = \frac{n}{2}$ times. The third loop while traverse all nodes in visited dll, the operation time is like $1 + 2 + 3 \ldots + (n-1) + n = \frac{n}{2}$ times. For both second and third, the largest operation times is also ***n.***

Thus, the time of the implementation for Dijkstra's algorithm is $\boldsymbol{n^2}$

The total time would be $O(kn^2 + mn + ce)$. The big-Oh time complexity is $O(n^2)$.

Problem 3:

Write up the *proof by induction* of the correctness of Dijkstra's algorithm. .

ANS

Assume that the number of vertices of a graph is ***n***, and there are ***e*** edges in it. It is a directed graph with non-negative edge weights,
One double liked list is created to store nodes which might be reachable with the minimum cost for the start node. One circular queue, with size ***n***, is the storage for un-visited nodes of graph.

Initialization

For Graph[G] vertex **n**,edge **e**

all graph $\mathrm{nodes} -> \mathrm{cost} = \infty$, put all nodes into queue Q={start, a, b,...x}

$\mathrm{start} -> \mathrm{cost} = 0$, dll visited L={start},

There will be three types of nodes in list L

* From start node to itself, the cost is zero
* the node has a shortest path with start node, the cost is min[start.cost+edge.cost]
* the node is not reachable to start node, the cost is infinity.

Proof by induction

#begin proof

Proposition: we prove this fact by induction that the Mth node that's indexed into visited list L, all *List_i i*n L must have found the shortest path *Short[List_i]* and it is equals to *Cost[List_i]*

While M = 1, List={star} =>Cost[start] ==short[start]==0；

Let U be the chosen node indexed into L next.

While M=2, there must be a edge contained directly between start and U pushing into the list L. The shortest path is the edge from the start node to first node, Cost[u]=start.cost+ cost(start,u), which is definitely smaller than infinity.

Then Assume that while M = k, all nodes in list L have found the shortest path, the fact is true.

Then let V be the chosen node indexed into L next.

While M = k+1, find the smallest cost in Q, return the corresponding nodes V. V which is the node going through other nodes List_i has the short distance reached by start. This path A could be like start -> U ->V, we need to prove Cost[v]==short[v].

#contradiction begin

Assume that the fact is false, which means there exists another path with shorter distance from start to V, this path B could be like $\mathrm{start} -> X -> Y ->$V, this path is the short[v].

This path go through the Y which is un-visited node in Queue, the Y could be Y or some others un-visited nodes. For now, distance of path $\mathrm{B} == \mathrm{Cost}[y] + \mathrm{Cost}(y, v) == \mathrm{short}[v]$.

Cause Y is un-visited, $\mathrm{Cost}[v] <= \mathrm{Cost}[y]$

$$=> \mathrm{Cost}[v] < \mathrm{B} == \mathrm{short}[v]$$

Cost[V] is the shortest path comparing with B instead of Cost[y]+Cost(y,v).

Thus, the assumption is invalid.There is not another path with shorted distance from start to V.

#contradiction end

Thus, while M=k+1 V which is the node going through other nodes List_i has the short distance reached by start. This path A could be like $\mathrm{start} -> U ->$V, Cost[v]==short[v].

The proposition is true.

#proof end.