# 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 *.*

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 **(*<e*)** edges and in neighbors **(*<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* **,** the largest number of edges is ***be,*** *which is* ***e.*** The big-Oh space complexity is *.*

## 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 *.*

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 , the operation time is like times. The third loop while traverse all nodes in visited dll, the operation time is like times. For both second and third, the largest operation times is also ***n.***

Thus, the time of the implementation for Dijkstra’s algorithm is

The total time would be. The big-Oh time complexity is *.*

## 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, put all nodes into queue Q={start, a, b,...x}

, 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](E:/Dict/8.10.3.0/resultui/html/index.html" \l "/javascript:;) 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 V, we need to prove Cost[v]==short[v].

#[contradiction](E:/Dict/8.10.3.0/resultui/html/index.html" \l "/javascript:;) 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 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

Cause Y is un-visited,

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](E:/Dict/8.10.3.0/resultui/html/index.html" \l "/javascript:;) 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 V, Cost[v]==short[v].

The proposition is true.

#proof end.