DSA PROJECT

Team Name : Eye of Sauron
Members Name : Aadhaar Goel (0001) , Aditya Gupta (0010) , Ayush kumar Giri (0030)

What are we doing in our Project ?

- We are doing comparative analysis between two flavors of the Dijkstra algorithm by implementing them.
  Two flavors are :
  1. Lazy Implementation (Using min heap)
  2. Eager Implementation (Using Indexed priority queue)

- Along with that we have implimentated the class of Indexed Priority Queue (IPQ).

Why this project ?

- As we all know, Dijkstra is an important algorithm for finding the shortest paths between nodes from a single source in a graph for non-negative weight edges.
- It is widely used in Google Maps,GPS-based navigation tools , Airline route planning,Railway networks , IP Routing ,Friend recommendations.
- And there are two flavors of it one is with min heap (Lazy Implementation) and one is with Indexed priority queue (Eager Implementation) which is an upgraded version of Lazy Implementation.
- And we got to learn about the upgraded version of priority queue.

WHAT WE HAVE AS A INPUT :

n >> no of nodes ,

m >> no of edges ,

m triplet of u v w (from node u to node v with weightage)

We will store these triplets in the adjacency list.

IMPLEMENTATION OF DIJKSTRA USING MIN HEAP

First we will declare

Array of n size which will store the shortest distance from source.

Mean Heap which will store pair value(distance , node) ,**distance** from the source to **node**.

This algorithm uses mean heap and first value in the pair is distance because there is a rational behind that , suppose we have two pairs for distance for the same node

As we want to encounter the **shortest** distance first, we use min heap and the priority is on the basis of **distance, that's** why the algorithm will take it as the first value.

Initially it will be initialized to infinite for all nodes and zero for the source node (obviously the shortest path to itself will be zero).

Dijkstra function will have information of source nodes

And Min heap(pq) have (0 , source) node pair

Now we will iterate over the min heap until it gets empty.

Get the very first node obviously that will be having shortest distance from source

Compare if it is larger than previously store distance from source if yes then continue;

Otherwise iterate over its neighbor and now **push them in the min heap if their distance (parent + their weight) is less than previously store distance** otherwise neglect it. And loop will do it iterative and keep update the shortest path in distance Array…(bfs)

And this is where LAZY IMPLEMENTATION becomes a bit slow.

Now what is the reason behind this ???

This algorithm is greedy, BFS-like algorithm it will traverse level wise and push it's children from next layer in PQ, as it is a graph same node can have two or more parent from previous layer , so may happen that while traversing in one layer two or more node pushes pair of distance and child in PQ for same node.

So for same node there may have multiple pairs in our PQ as we have to push it there is no way to overcome this because if we want to update the one of the previously stored node's distance we need to first search it is O(n) in min heap then update in O(1) which can be done in O(log n)

by IPQ and we don't need to push many pairs of distance and node value for the same node this make Eager's Implementation better and efficient.

Why Both implementations exist if Eager's Implementation is better as seen above ?
Lazy perform better when density of graph is very much low but less efficient for dense.

Showing it by generating input of different type and comparing how many IPQ implementation perform better.
Time Complexity
Lazy O((V+E) log E) V - vertices , E- no of edge
Eager O((V+E) log **V**)