the longest shortest-path

S20201116 Lee Subin

1. Strategy used to solve this problem

* **main.cpp – for problem 1 and 2**

To find the longest shortest-path, I first started by calculating all-pair shortest paths using *Floyd Warshall Algorithm*. It is the dynamic programming method of finding all-pair shortest paths in a weighted and directed graph.

To implement *Floyd Warshall Algorithm,* you need to make a 2d array *distance* where each size of dimension is the number of vertices *V*. *distance[i][j]* will return the shortest path weight from vertex i to j on each iteration. Then, you put in the initially weights of given edges. When done with initializing, iterate *V* rounds and do the relaxation for each round. Relaxation done by comparing existing path (*distance[i][j]* ) with new path passing vertex *k* and updating if new path is shorter than the existing path.

* + pseudo code

init all values of distance [][] as INF

allocate initial weight to dist[][]

for k<-1 to n

for i <-1 to n

for j<-1 to n

if distance [i][j] > distance [i][k]+ distance [k][j]

then distance [i][j] <- dist [i][k]+ distance [k][j]

* + time complexity

For each round (number of each round is number of vertices in the graph), do relaxation. Relaxation is done for every source vertex and for each vertex, check with every destination, which is also same as number of vertices.

O(n\*n^2)= O(n^3)

Calculating in this way takes a long time since it’s complexity is big. To optimize this function, when distance from i to k or k to j is INF, then we skip the comparison code. Also, save distance from i to k and k to j as variable *cur* to avoid calculating it twice. Then, the optimized pseudo code will look like below.

* + optimized function pseudo code

init all values of distance [][] as INF

allocate initial weight to dist[][]

for k<-1 to n

for i <-1 to n

for j<-1 to n

if distance [i][k] != INF and distance [k][j] !=INF

cur = distance [i][k]+ distance [k][j]

if distance [i][j] > distance [i][k]+ distance [k][j]

then distance [i][j] <- cur

After calculating all-pair shortest paths, I declared priority queue to reorder paths in descending order. Each element of the queue should contain source vertex, destination vertex and path weight (which is the minimum sum of edge weight from source vertex to destination vertex). To implement this, I declared a class *Node* which contains source, destination, and path weight as elements. Then, declare a comparison function which compares path weight of each class *Node* and returns bigger element. Using this comparison function, declare a priority queue *p* with type *Node*.

Now, iterate all-pair shortest paths found by using Floyd Warshall Algorithm and push each pair of source, destination, and path weight in a class *Node* format into priority queue *p*. Then return the top Node of priority queue p and you can get the longest shortest path.

Also there is another way, which is not using the queue. To reduce space complexity, instead of declaring a queue, just used variable *source*, *destination* and *longest* to save the answer for the questions.

**-johnson algorithm – for problem 3**

For problem three, which has large number of vertices I needed to come up with smaller time complexity, which is Johnson’s algorithm. This algorithm first determines where there is a path between to vertices and then reweight according to it. And for those that have a path, run Dijksta’s algorithm.

* + time complexity

O(VE+(V\*2)lgV)

1. (src, dst, path weight) of the longest shortest-path found
2. 16000.graph

Text

Description automatically generated

1. 32000.graph

Text

Description automatically generated

1. 1000000.graph

lack of memory so could not run