CS201 REPORT-LAB 6

Lab : 6	**Name:** Shivam Pandey
Date: 22/12/20	**Entry-No:** 2019chb1055

Title: Johnson's Algorithm(All pair shortest path).

Objective : To implement Johnson's Algorithm for All Pair Shortest Path and analyse the algorithm's time complexity (considering the execution time on large graphs) considering different data structures based heaps implementation.

Introduction:

Johnson's algorithm is used to find the shortest paths between every pair of vertices in a given weighted directed graph, where the weights may be negative. Firstly, we run the Bellman Ford Algorithm in the modified graph which leads to non negative weights and then we run dijkstra, on the original graph using the modified weights.
Here in the Dijkstra implementation, we have used four different heap implementations. These are as follows:

1) Array- based implementation
2) Binary heap implementation
3) Binomial heap implementation
4) Fibonacci heap implementation

Using the Floyd Warshall algorithm, for the same purpose is quite inefficient wrt to time complexity analysis, since the earlier has $O(V^3)$,whereas Johnson's algorithm has $O(V^2 \log V + VE)$ time, where V is the no of vertices and E no of edges.

Time complexities of various functions involved:

OPERATION	BINARY HEAP	BINOMIAL HEAP	FIBONACCI HEAP
insert	O(log N)	O(log N)	O(1)
find-min	O(1)	O(log N)	O(1)
delete	O(log N)	O(log N)	O(log N)
decrease-key	O(log N)	O(log N)	O(1)
union	O(N)	O(log N)	O(1)

Reference:

https://www.geeksforgeeks.org/difference-between-binary-heap-binomial-heap-and-fibonacci-heap/#:~:text=In%20Fibonacci%20Heap%2C%20trees%20can,to%20be%20a%20Binomial%20Tree).&text=All%20tree%20roots%20are%20connected,a%20single%20'min'%20pointer

.

Algorithm:

- Firstly, we add a new vertex to the graph, then add edges from this vertex to each of the vertices of the graph, where the edge weight is 0.
- Then we run the ***Bellman ford*** algorithm on the modified graph. If there is a negative weight cycle, then it can be detected using this algorithm.Then we modify the weights of edges of the original graph using the relation: modified weight = original weight + distance calculated to reach u - distance to reach v(for an edge u-->v).
- Now we remove the added vertex and run the ***Dijkstra*** algorithm for each of the vertices present in the original graph.

- For the heap part, we use the above mentioned heap implementations and then compare the time complexities of the algorithms dealing with these implementations.

Time Complexity Analysis/ Results:

As, in this lab, we had run the Dijkstra Algorithm using different heap implementations, thus the execution time of each of these implementations were different.
For a graph consisting of **800** vertices and **24500** edges (using the random graph generator), the execution times of the codes dealing with these implementations were as follows:

Large Graph:

Heap (Data Structure)	Execution time(in sec)
Array Based implementation	14.564
Binary	13.977
Binomial	13.0822
Fibonacci	10.7701

For V = 1000 and E = 30000

Heap (Data Structure)	Execution time(in sec)
Array Based implementation	18.776001
Binary	19.22100
Binomial	16.77705
Fibonacci	13.447001

For a graph consisting of **45** vertices and **300** edges (using the random graph generator), the execution times of the codes dealing with these implementations were as follows:

Small Graph:

Heap (Data Structure)	Execution time(in sec)
Array Based implementation	0.001
Binary	0.08
Binomial	0.00077
Fibonacci	0.000042

Conclusions:

Fibonacci heaps are the ***most effective*** ones in terms of time complexity analysis, then followed by binomial, whereas in between array and binary, binary heaps are more effective if we look at a larger picture(i.e, considering many test cases).