CS201 REPORT
LAB 6
JOHNSON'S ALGORITHM

Submitted by: D .V .Sahan

Entry Number: 2019EEB1156

Submitted to: Dr. Puneet Goyal

Johnson's algorithm:

Implementation:

The Johnson's algorithm for All Pair Shortest Path includes the Bellman Ford Algorithm and the Dijkstra's Algorithm. In case of negative edges, the Johnson's algorithm changes all edges and makes them non negative and then Dijkstra's algorithm is applied for each vertex.

Since Dijkstra's algorithm can't be used for negative edges the Bellman Ford algorithm is used to modify the edges by adding a new source vertex which is connected to all the edges with edge weights equal to 0.

If a negative cycle exists then output will be returned as -1 without performing Dijkstra's algorithm.

The shortest pair distance from the source is stored in an array named dist[n]. Now the edge weights are modified using:

$G_{new}[u,v]=G_{old}[u,v]+dist[u]-dist[v]$

After the edge weights are modified the source vertex is removed and Dijkstra's algorithm is applied for all the vertices(it is called V times).

Once the shortest pair distance for each vertex is found then the edges are modified back by using the below formula:

$G_{new}[u,v]=G_{old}[u,v]+dist[v]-dist[u]$

Now the shortest path for each vertex is found.

The time complexity of Bellman Ford is O(VE) and the time complexity of Dijkstra's algorithm will depend on the data structure being used.

In the lab we have used 4 data structures and are analysing the time taken to implement Johnson's algorithm for large values of V.

The time complexity of Dijkstra's algorithm will depend on the Extract minimum and Decrease key time complexity of the data structure.

Here V stands for Number of Vertices

E stands for Number of Edges

Dijkstra's algorithm

Dijkstra's algorithm finds the shortest path from source to all vertices for a given graph but it fails for negative edges. In Johnson's algorithm,since the edge weights have become non-negative due to modification by the Bellman Ford algorithm Dijkstra algorithm will work for all cases.

Dijkstra's algorithm is a greedy algorithm which requires deleting the minimum distance(Extract Min operation) and the Decrease key operation.

The following data structures have been used for the Dijkstra algorithm's implementation:

1. Array
2. Binary Heap
3. Binomial Heap
4. Fibonacci Heap

We will analyse these data structures based on their time complexities of Extract min,Insert and Decrease key operations.

Theoretically these are the time complexities for the following data structures:

Data str./Operation	**INSERT**	**EXTRACT MIN**	**DECREASE KEY**
Array	O(1)	O(n)	O(1)
Binary Heap	O(logn)	O(logn)	O(logn)
Binomial Heap	O(logn)	O(logn)	O(logn)
Fibonacci Heap	O(1)	O(logn)	O(1)

Theoretically we can see that the Fibonacci heap has the best time complexity and the Array has the worst complexity among these data structures.

Array Method

If the Array data structure is used for Dijkstra's algorithm then Time Complexity for Dijkstra for a single node is $O(V^2)$. So total Time Complexity for Johnson's algorithm will be $O(EV+V^3)$ which is $O(V^3)$.

Binary Heap Method

If the Binary heap data structure is used for Dijkstra's algorithm then Time Complexity for Dijkstra for a single node is O((V+E)logV). So total Time Complexity for Johnson's algorithm will be O(EV+V(V+E)logV) which is O(EVlogV).

Binomial Heap Method

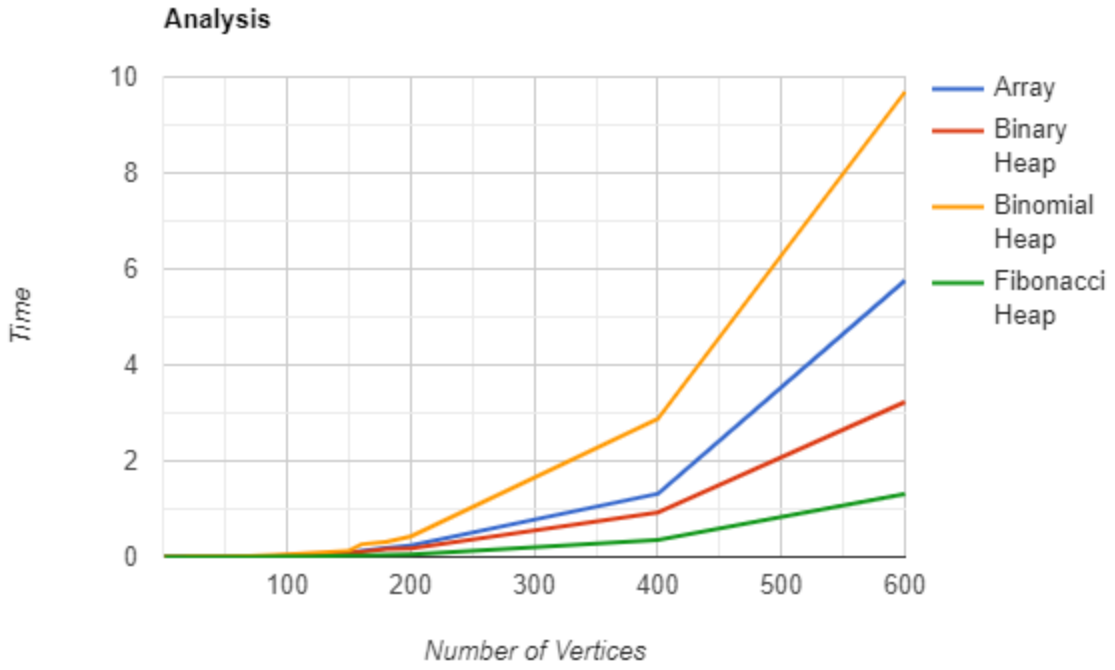
If the Binomial heap data structure is used for Dijkstra's algorithm then Time Complexity for Dijkstra for a single node is O((V+E)logV). So total Time Complexity for Johnson's algorithm will be O(EV+V(V+E)logV) which is O(EVlogV).

Fibonacci Heap Method

If the Fibonacci heap data structure is used for Dijkstra's algorithm then Time Complexity for Dijkstra for a single node is O(E+VlogV). So total Time Complexity for Johnson's algorithm will be O(EV+(E+VlogV)V) which is O((E+VlogV)V).

Analysis

Given below is the plot for Number of Vertices vs Time for these 4 data structures using my code for some randomised graphs with no negative edge cycles. The below plot has been plotted for about 20 vertices in the range from 1 to 600.



Note: Time is measured in seconds

Below are the values for which the graph has been plotted

Number of Vertices	Time taken(Array)	Time taken(Binary Heap)	Time taken(Binomial Heap)	Time taken(Fibonacci Heap)
1	0	0	0	0
3	0	0	0.001	0
5	0	0	0.002	0
10	0	0	0.003	0
15	0.001	0.001	0.004	0.001
20	0.001	0.001	0.006	0.001
30	0.002	0.003	0.009	0.002
50	0.004	0.004	0.011	0.002
70	0.007	0.005	0.019	0.004
100	0.022	0.031	0.052	0.007
120	0.041	0.044	0.086	0.014
150	0.08	0.097	0.13	0.027
160	0.124	0.105	0.266	0.03
180	0.184	0.17	0.314	0.041
200	0.236	0.177	0.428	0.057
400	1.317	0.93	2.875	0.356
600	5.76	3.23	9.687	1.31

From the plot we observe that the Binomial heap based method is taking the most time despite having better Time complexity than that of the Array based method.

The Array based method takes more time than the Binary heap and Fibonacci heap based method since it has more time complexity than these two data structures.
The Fibonacci heap based method takes the least time among all these 4 data structures which can also be seen theoretically(Fibonacci heaps has the best time complexity among these 4 data structures).

Conclusion

- For a smaller number of vertices the time taken for all the methods is almost the same. As the number of vertices increases the order of time taken for execution in lowest to highest order is as follows:
    1. Fibonacci heap
    2. Binary heap
    3. Array
    4. Binomial
- Theoretically the Array based method must take the most time but practically the Binomial heap based method takes most time despite having Time complexity of O(EVlogV). This may be happening because there are functions like merge heap and union which leads to increase in time taken for iexecutingmplementing the function.
- The Fibonacci heap based method has the best time complexity and takes the least time.