CS-201

Lab – 6

Johnson's Algorithm

Implementation details and Time Complexity Analysis using array, Binary heap, Binomial heap and Fibonacci heap

By

Ashish Sharma

2019MCB1213

Algorithm Details: Johnson's Algorithm for All Pair Shortest Paths finds the shortest paths distance between all possible pairings of nodes in a graph.

The core idea is of reweighing the edges and making them non negative (so that Dijkstra's algorithm can be applied) preserving the relative ordering of the paths. (so that shortest paths don't change)

For reweighing Bellman Ford Algorithm is carried out once on the graph with an extra source added and distance to other nodes from this source as 0.If a negative cycle exists it is detected by the Bellman Ford Algorithm and the algorithms returns -1.After this Dijkstra can be applied once from each vertex to get the shortest paths which are stored in a matrix.

Implementation Details:

The input is given in form a V x V matrix to the Johnson's Algorithm function. Another source vertex is added with no incoming edges and outgoing edges to all vertices with 0 distance which gives a new (V+1)x(V+1) matrix which is passed to the Bellman Ford function.

Now the edges are reweighed using the values obtained from bellman ford function. A new VxV matrix is formed with difference of shortest distance obtained from new source of finishing point and starting point of edge in directed graphs added to the old weights.

For convenience always a (V+1)x(V+1) matrix is given as an argument to all functions and only the relevant vertices are used.

Time complexity of Dijkstra is theoretically known to be number of edges times decrease key plus the number of vertices times extract min.

- Using Array it comes out to be O(E*1+V*V) hence O(V^2)
- Using Binary Heap it is O(E*logV+V*logV) which reduces to O(E*logV) for connected graphs.(as E>=V-1)
- Using Binomial Heaps it is O(E*logV+V*logV) in worst case
- Using Fibonacci Heaps it is O(E*1+V*logV) amortized .

Johnson Algorithm's time complexity = Time(BellmanFord)+ V*Time(Dijkstra)

- O(V*E+V^3) in arrays
- O(V*E+V*(E+V)*logV) = O(V*(E+V)*logV) in case of binary heaps and reduces to O(V*E*logV) for connected graphs (as E>=V-1)

- O(V*E+V*(E+V)*logV) in case of binomial heaps
- O(V*E+V^2*logV) in case of fibonacci heaps

Better the time complexity the more complicated the codes get. Also we observe time complexity is bettered by changing the data structure we store the information in as is the case here. This is because we intend to use a specific functionality of a data structure for a specific purpose which is fast for one and slow for others.

In Array based implementation only two arrays one storing the minimum distances(all cases require it)and other whether the nodes are visited or not are required along with few variables.

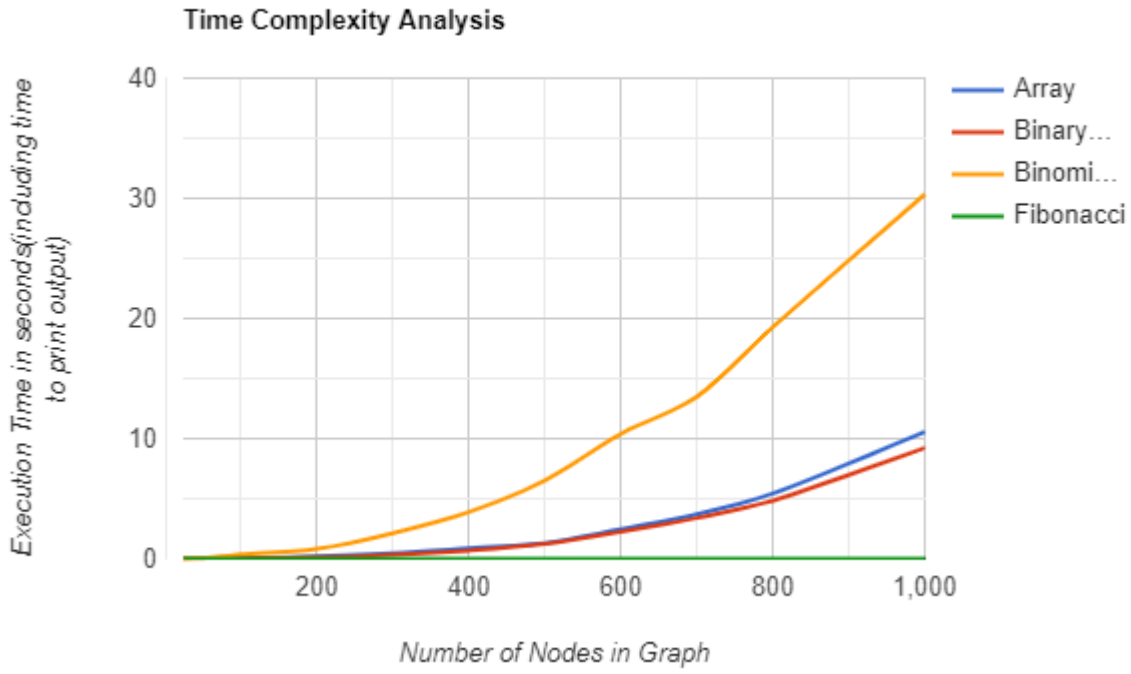
In binary heap based implementation we store the values in a min heap which follows structure property as well, represented by a 2D array(1D for vertex and 1D for minimum distance).

In binomial heap based implementation again we store information in binomial trees which are chained together from their roots.

In Fibonacci heaps we use doubly linked list at each child level and the parent level , which causes too much book keeping problems as we need to modify too many pointers.

Plot and Table of time taken for Johnson's Algorithm function(in s) along with printing the output(which is just a constant term and doesn't effect the relative ordering of the times) v s number of nodes

Implementation ->	Array	Binary Heap	Binomial Heap
Number of Nodes	Time Taken	Times Taken	Time Taken
25	0.0012	0.0029	0.0182
50	0.0063	0.0107	0.056
100	0.024	0.0247	0.355
200	0.198	0.119	0.8197
300	0.4689	0.3176	2.13
400	0.8783	0.6827	3.9
500	1.313	1.252	6.52
600	2.4662	2.2413	10.416
700	3.6996	3.395	13.4854
800	5.429	4.849	19.304
1000	10.56	9.23	30.35



The order of time complexities we observe from the graph is : binomial>array>binary(binomial the most inefficient and binary the most efficient)

Which is different from expected as binomial should be the least among the three but greater than Fibonacci(not plotted as code gave segmentation fault).

When number of nodes are less all the three procedures take around the same time with array based implementation taking least time for execution for many cases.

As we increase number of nodes time taken by array based implementation increases, matching with the expected theoretical order. (binary heaps starts to take lesser time for computation)

This may be occurring because of some specific test cases . Here in my graph, up to 100 nodes array was better than binary heap implementation but from 200 nodes onwards binary heap implementation took lesser time than array .

Similar might be the case with binomial heaps but my pc takes too long for large number of nodes and hangs. I mean for even larger number of nodes

binomial heap implementation may take lesser time than binary heap asymptotically.

Binomial heaps might be taking more time because of too much amount of pointers being changed and

So , I think theoretical trend is not followed in less number of nodes because we require too much book keeping i.e. updating pointers and too many function calls.

Asymptotically my code implementation seems to match with theoretical calculation in binary heaps and arrays but not in binomial.

Fibonacci if successfully implemented would have come out to be the least among all.

Conclusion:

For Johnson's algorithm we apply concept of reweighing without effecting the relative ordering of paths which is done with help of Bellman Ford Algorithms with an extra new source node.

Johnson's Algorithm involves 1 execution Bellman Ford and V executions of Dijkstra.

Observed order of time complexity from best (least time complexity) to worst.

(most efficient)Binary Heaps < Arrays <Binomial Heaps(least efficient)

Expected Order:

Fibonacci Heaps < Binomial Heaps < Binary Heaps < Arrays

It is expected that binomial heap implementation should be more efficient than binary heap and array implementation which is not observed up to 1000 nodes but may occur if number of nodes are further increased like it happened in the case of arrays vs binary heaps.

Also Fibonacci Heap implementation of Dijkstra is expected to be the most efficient for Johnson's Algorithm.