Course: Introduction to Graph Algorithms

Professor: C Pandu Rangan

Department: Computer Science and Engineering

Institute: IISc

Week: 05

Lecture 22 All Pair Shortest Path 5

Namaskara we will continue our discussions on the algorithms for all pair shortest path problems. We have seen that using the idea of k paths we could arrive at an algorithm with complexity order n cube, the algorithm was designed by Floyd and Warshall. Whether the graph is sparse or dense the complexity is always order n cube. If the graph has got less number of edges, is it possible to arrive at a more efficient solution than n cube, okay, that was answered positively by Johnson and we are going to take a closer look at Johnson's algorithm. It is a very clever algorithm, it is using the existing resources in a very clever way.

We know that the Dijkstra's algorithm can be used only for the graph that has got positive edge weights. End times Dijkstra's algorithms application will be more efficient however we need to have all the edge weights to be positive how do we achieve that Johnson's algorithm makes some clever moves towards it and arrives at an efficient solution for sparse graphs. The algorithm by Johnson runs in order n square log n nm time so look at this expression n square log n nm,

$$O(n^2 \log n + nm)$$

m is the number of edges, if m is order n then this becomes n square log n algorithm, if m is something like that like n or even if m is something like n log n, in both cases the complexity becomes order n square log n. So for graphs with less number of edges order n square log n is more efficient than order n cube algorithm.

$$m = n$$

$$= n \log n$$

$$= n^2 \log n$$

$$< O(n^3)$$

So Johnson's algorithm will be more efficient if the graph is sparse if the graph is a dense it has got a matching complexity if m equal to n square

$$m = O(n^2)$$

which is the property of a dense graph the complexity will be n cube anyway. So in the worst case it just matches with Floyd Warshall's algorithm but if it is sparse it turns out to be significantly more efficient okay.

First let us make one simple observation suppose all edge weights are positive if all edge weights are positive simply apply Dijkstra's algorithm n times starting from each vertex as a source vertex apply it as n times. We have an implementation of Dijkstra's algorithm with complexity n log n plus m.

$$n \log n + m$$

when you apply n times the complexity will be o n square log n nm this is the promised complexity of Johnson's algorithm okay.

$$n^2 \log n + nm$$

Therefore if all edge weights are positive we do not have to do anything simply apply Dijkstra's algorithm n times and you have n square log n plus nm algorithm

$$O(n[n \log n + m]) = O(n^2 \log n + nm)$$

but what if the graph is a mix of positive and negative weights of course we still want the graph to have no negative cycles. no zero cycles but it may have a mix of positive and negative edges if that is the case we simply cannot apply Dijkstra's algorithm. Dijkstra's algorithm does not work if edge weights have a mix of positive and negative weights therefore we have to use the only known algorithm that we have, we have seen so far Bellman and Ford algorithm but Bellman and Ford algorithm which is capable of handling mix of positive and negative weights runs in nm time. So n times nm if you apply Bellman and Ford n time then the complexity will be n square m and when m is n square n square times n square it can go as high as n power 4.

$$O(n.n.m) = O(n^2 m) \rightarrow O(n^4)$$

Therefore applying n times the Bellman and Ford is not a good idea okay but Bellman and Ford is capable of handling positive and negative weights, therefore how do we improve the complexity of n power 4. Tere is the technique this called the transformation technique it is very popular in mathematics. If a problem is hard to solve in one domain they transform it into another domain and there they solve it easily in the transformed domain. So often mathematicians will try to find an equivalent way of doing the same thing but in a more efficient manner, you call that as a shortcut you call that as an alternative way of doing and so on. But in computer science we value such ideas because these alternate methods sometimes could be computationally very efficient.

So transformation is a very powerful and an interesting idea. very popular in Mathematics we are using that in computer science here is a very good instance of that. We are going to transform the weights, let us assume that h is a function and we transform the weight in the following way, w dash uv is w uv plus h of u minus h of v.

$$w'(u,v) = w(u,v) + h(u) - h(v)$$

So where h is just an arbitrary function, h is a function from v to integers okay. Just let us assume that we have transformed the weights.

I am not changing the graph, I am only changing the weights. Now, if I take an arbitrary function h and define w dash transformed in this way the w dash has got the following interesting properties, P is a shortest path from i to j under w if it is a shortest path under w dash that means you can find a shortest path by working with w dash the same path can be used only the path the weights will vary we will see that in a moment. In other words I can solve the shortest path problem by taking the weight as w dash whatever the path I get I can take the same physical path as a shortest path under w as well. For any cycle c, wc is w dash c

$$w(c) = w'(c)$$

Therefore if the graph has got no negative cycle or no 0 cycle under w then even under w dash it will not have any negative cycles it will not have any 0 cycle because the costs of the cycle are preserved by this transformation.

Very easy to see simply add all the weights the h values will cancel out and you can see that the telescopic sum is identical wc is how are the weights of the paths related . The weights of the path are related like this wP is w dash P plus hj minus hi

$$w(P) = w'(P) + h(j) - h(i)$$

It is a depending on only i and j, h value of i and h value of j is known. So after finding the path we will have the weight of the path under w dash what we have to do is that. we take the weight under w dash and h of j subtract h of I you are getting the weight of the same path under w. Therefore after finding the shortest path under weights under w dash we do not change the path but we can change the weight by this simple operation very simple operation. We get the weights therefore I am going to solve the problem of all pair shortest path under w by solving the problem under w dash and then come back with respect to w.

This route I would take and this is much more efficient way to solve than directly working with w. We will see why going from w to w dash and working on w dash and coming back from w dash to w is more efficient than working directly with w. We have seen already if you work with w directly the only way is you have to apply n times the

Bellman and Ford and that is n power. Therefore working directly with w is not a good idea all right.

So what we do is we are going to find a h in such a way that w dash is positive ,okay w dash is positive for all edges, the transformation is done. So w is a mix of positive and negative values but w dash is always positive.

There is an advantage in working with the w dash that is positive you can apply Dijkstra's algorithm n times and get this complexity

$$o(n^2 \log n + nm)$$

Therefore all we have to do is that make sure that in the transformation we go from w to w dash and w dash has got all weights to be positive okay. So here is a trick, how to find that h the trick is use Bellman and Ford algorithm once to find a h, we have to find a h we are going to use Bellman and Ford algorithm once to find a h then use a h to transform w to w dash then work on w dash. only Dijkstra's algorithm I will apply on w dash okay. So this is the outline of the algorithm. So the algorithm at high level is this, use Bellman and Ford algorithm to determine a function such that w dash is positive these values are positive.

$$h: V \to I$$

Solve all pair shortest path problem by using Dijkstra's algorithm n times on G with weight function w dash because w dash is positive you can apply Dijkstra's algorithm and apply this. Let D dash be the shortest path weight matrix obtained. Then how do you construct the matrix D? Matrix D is the shortest path weights under w. This is the way to go from w dash to w. We have seen that already, right.

$$\delta(i,j) = \delta'(i,j) + h(j) - h(i)$$

And return D construct this. What is the total complexity? The total complexity is order mn. This is for Bellman. Bellman and Ford algorithm you are applying once, m and time. n square, n square is for constructing w dash from w and constructing delta from delta dash they take n square plus n square. I can add another n square actually it is order n square just 2n square

$$O(mn) + O(n^2 \log n + nm) + n^2 + n^2$$

and a single or application of Dijkstra's algorithm takes n log n plus m time this is single application

$$n(n \log n + m)$$

n times Dijkstra when you use it becomes n square log n plus nm

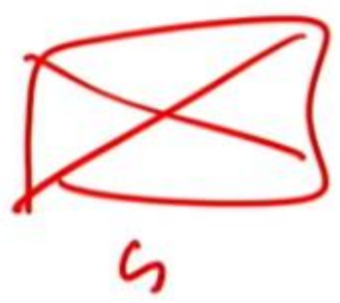
$$n^2 \log n + nm$$

and that is what is written this is the cost of n application of Dijkstra's algorithm so n square log n nm. So n square log n plus nm is the dominating term, therefore this sum has a total complexity given by O n square log n plus nm. okay this is Johnson's algorithm.

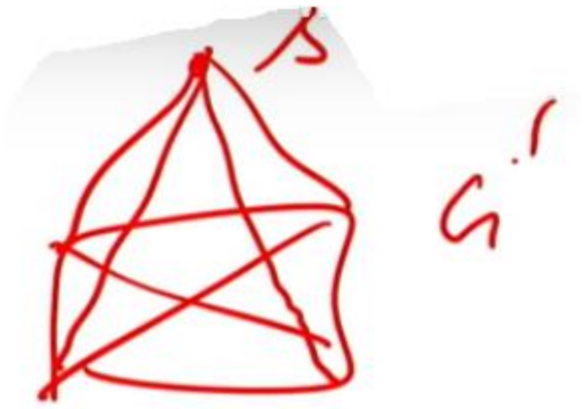
$$O(n^2 \log n + nm)$$

All we have to now do to complete our discussion is how we are building the h function. okay because once we know how to get the h function all things are done as discussed and we have an n square log n plus nm algorithm in our hand.

So it remains to discuss on the construction of h how do we construct the h function. Now if the graph whatever is the graph this is G

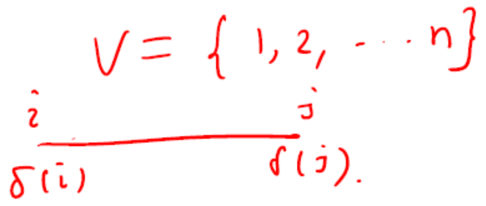


I am going to add a vertex s a new vertex s and connect that vertex s with every vertex of the original graph. So this graph is called G dash



So if G has n vertices G dash has got n plus 1 vertices. One more vertex is added okay and I am going to add n edges from s all vertex of G is connected and I give a weight 0 for the newly added edges the weight is going to be 0. okay now solve the single source shortest path problem on G dash.

G dash is still working with the weight function w, I have not done the transformation G dash has got the weight function w and new edges that are added they have weight 0. Now solve the single source shortest path problem using Bilman and Ford only Bilman and Ford can be used now and when you use that you are going to get the shortest path distance from s to all vertices okay. Let us say this is an edge uv this is u this is v this is an edge uv. So I have because I have used the Bellman and Ford, I have delta u computed I have delta v computed. If G has no negative cycle G dash has no negative cycles because in G dash I have added a vertex and then added only the 0 weight edges, they are all one way edges okay these are all one way edges, no new cycle or nothing is created by this okay they are one way edges and their weights are 0 therefore this is not introducing any new cycle the same cycle in G is what is there in G dash as well and they are all positive, nonzero. So I can apply the Bellman and Ford algorithm and get delta u and delta v for all vertices of course delta s is 0, delta s is 0 but that does not matter we are not interested in that value. Look at delta u and delta v what is the property of this delta u and delta So if you take any edge the delta u and delta v because we are considering the graph in which vertex at v is 1, 2, n so let me write vertex i, vertex j, delta i, delta j. The delta i and delta j they have a special property this we know this is the Bellman's inequality right.



So, by moving delta j to the other side you immediately see that wij plus delta i minus delta j is positive.

$$w(i,j) + \delta(i) - \delta(j) \geq 0$$

Therefore all I have to do is define a function h of i to be equal to delta i

$$h(i) = \delta(i)$$

This function if I use and if I design my delta if I define my w dash as wij plus hi minus hj it is guaranteed to be positive. it is guaranteed to be positive and that is it we have completed our discussions on the algorithm.

$$w'(i,j) = w(i,j) + h(i) - h(j) \geq 0$$

Therefore to summarize, all we have to do is that when a graph is given, add a vertex draw edges with weight 0, solve the Bellman and Ford algorithm and for each vertex i you would have got delta i set h of i equal to delta i. Define a new weight function w dash using this formula, define a weight function w dash ij using this formula w dash ij is guaranteed to be positive apply n times the Dijkstra's algorithm and solve the problem okay this completes the discussions and as a final remark if the original graph has got any negative cycle that can be detected by Bellman and Ford algorithm.

In G dash also the same negative cycle will continue to exist. Since we are running Bellman and Ford algorithm on G dash we would have detected that negative cycle and we can exit there itself without any further work. If the graph has got no negative cycle or 0 cycle, we proceed further with the transformation and solve the problem efficiently. This completes our discussions on Bellman and this completes our discussions on Johnson's algorithm. In our next session we will begin our discussions on minimum spanning tree. Thank you.