Course: Introduction to Graph Algorithms

Professor: C Pandu Rangan

Department: Computer Science and Engineering

Institute: IISc

Week: 04

Lecture 14 Bellman Ford P1

Namaskara, we will now begin our discussions on Bellman Ford algorithm. So we are going to work on a graph G which has vertex set V, edge set E, weight function w, a source function s, we are interested in solving single source shortest path (SSSP) problem. That means we have to find out delta v and parent of v for every vertex V. Towards this, we had a mathematical formulation and obtained Bellman equations for the delta values, but they have nonlinearity and circular dependency okay, nonlinearity and circular dependencies. These two things made solving the Bellman equation impossible, so we were looking for an iterative way to solve the Bellman equation and obtain the delta values. Towards that we have introduced the quantities like alpha iv and delta iv and we are going to iteratively build a solution.

We are going to still work towards obtaining the solution for Bellman equation, but then in the same equation we are relating alpha i values okay and we have derived Bellman equations for bounded walk and bounded paths that means there is a bound on the number of edges. This leads to a decreasing sequence leading to the answer we are looking for. So we have alpha 0 for all vertices, using that alpha 1 for all vertices and finally we will end up alpha n minus 1 for all vertices. This is an iterative process and that is what the Bellman and Ford algorithm is going to carry out.

$$\alpha_0 \longrightarrow \alpha_1 \longrightarrow \cdots \longrightarrow \alpha_{n-1}$$

Basically the Bellman and Ford algorithm will keep computing the alpha values. If G has no negative cycles, then whatever we are computing turns out to be delta 0 to delta 1 to delta 2 and so on to delta n minus 1.

$$\delta_0 \rightarrow \delta_1 \rightarrow \delta_2 \cdots \rightarrow \delta_{n-1}$$

This computation is valid for any graph, but if this computation is carried out on a graph that has no negative cycle, the computation that we are doing values we are generating are indeed this, that is because for such a G, alpha iv equal to delta iv for all v.

$$\alpha_i(v) = \delta_i(v) \quad \forall v$$

That is the reason why our computational process or whatever Bellman and Ford is doing is leading to an answer. Why does it lead to an answer? delta n minus 1 v is in fact equal to delta v.

$$\delta_{n-1}(v) = \delta(v)$$

So the answer we are looking for is obtained through a method of iterated improvement or successive approximation. So when we are unable to solve directly, we use the powerful mathematical idea of method of iterated improvement. In our context the method of iterated improvements are achieved through the alpha values. We will see the details, alpha v is less than or equal to alpha i-1 u plus weight of uv.

$$\alpha_i(v) \leq \alpha_{i-1}(u) + w(u,v)$$

For all edges uv, alpha iv is less than or equal to alpha i-, not only this you also have the following fact. alpha iv is equal to alpha i minus 1u plus weight of uv for some edge uv,

$$\alpha_i(v) = \alpha_{i-1}(u) + w(u,v)$$

for some edge uv this is also true, that means alpha iv is equal to minimum over u not equal to v of alpha i minus 1u plus weight of uv, okay?

$$\alpha_i(v) = \min_{u \neq v} \left\{ \alpha_{i-1}(u) + w(u,v) \right\}$$

So this is the Bellman equation for bounded walks, you can call this as, Bellman Equations for bounded walks, what we mean by bounded means the number of edges, it is limited. We are considering only the walks with less than or equal to i edges here, that is why these are called bounded walks, in general a walk can be unbounded any number of edges can be there. But we are focusing on the walks with the bound on the number of edges and alpha iv is less than or equal to alpha i minus1u plus vu and it is this.

$$\alpha_i(v) = \min_{u \neq v} \left\{ \alpha_{i-1}(u) + w(u,v) \right\}$$

So we are not able to solve the Bellman equation for deltas, that is because the Bellman equation for deltas have circular dependency but you can see that the Bellman equation for the alpha values, they do not have circular dependency, you can smoothly proceed.

Look at the dependency, alpha i is depending on only alpha i minus1. So if I have alpha i minus 1 s alpha i minus 1 another vertex, alpha i minus 1 yet another vertex and so on somewhere alpha i minus 1u right, alpha i minus 1u, I have alpha i values,

$$\alpha_{i-1}[s], \alpha_{i-1}[\ ], \alpha_{i-1}[\ ], \cdots \alpha_{i-1}[u] \cdots$$

using this, I can get alpha i values. All alpha i values are depending on alpha i minus 1 values,

$$\alpha_i[\ ], \alpha_i[\ ], \cdots \alpha_i[\ ]$$

that means from the values in the array alpha i minus 1 we can, by using let us use the word, by using the values in the array alpha i minus 1 we can compute the array of values alpha i, it is an array, okay.?

$$\alpha_{i-1}[\ ] \longrightarrow \alpha_i[\ ]$$

All values, so you have n values in alpha minus1 array, one value for each vertex and you compute. So how do we compute this? Here is a very simple pseudo code based on the equation, bellman equation that we have seen, okay, here is the Bellman equation I will show once again. Okay, here is the Bellman equation, so based on this Bellman equation we have to.

$$\alpha_i(v) = \min_{u \neq v} \left\{ \alpha_{i-1}(u) + w(u,v) \right\}$$

So, this is what we are going to do. Every edge has a potential to improve the alpha value of some v, especially uv may improve right, the value of alpha iv. Therefore here is the simple plan, pseudo code to improve okay, pseudo code to compute alpha i array based on alpha I minus1 array values. Based on these values, how do I do that? First of all, I will set alpha iv is equal to alpha i minus 1v,

$$\alpha_i[v] = \alpha_{i-1}[v]$$

because either it is this value or probably smaller value, remember it is a hierarchy, so the bigger set minimum either the minimum of the smaller set or it could become smaller. Let us see whether it is becoming smaller, alpha i equal to v, this is step 1 for all v.

First copy the array value alpha iv to alpha i minus 1v to alpha iv, then for each edge uv, do it in the next page so I will cut this. Computing alpha iv values by using alpha i-1 array values, how do we do that? alpha iv is equal to alpha i minus 1v for all v belonging to v this is step 1,

$$\textcircled{1} \quad \alpha_i[v] = \alpha_{i-1}[v] \quad \forall\, v \in V$$

and step 2 for each uv in E, for each edge uv in E. If alpha iv is greater than alpha i minus -1u plus weight of uv, that is you have found a smaller value, because you have found a smaller value you have to update it to the smaller value, that is the way to find the minimum. The way in which we find a minimum is we keep scanning the values, when we find something smaller you replace with a smaller, straightforward logic of finding the minimum. If alpha iv is greater than alpha iv plus v, then alpha iv equal to alpha i-1u plus wuv and we have to save the Bellman edges.

```
② for each (u, v) in E
      if $\alpha_i[v] > (\alpha_{i-1}(u) + \omega(u,v))$
          $\alpha_i[v] = \alpha_{i-1}(u) + \omega(u,v)$
```

So the current Bellman edge is the edge uv which is changing the value. We have already seen that we are going to use a convention of previous p array notation. Because Bellman edges are always last edges and in the last edge uv, u is going to be previous to v

$$\begin{tikzcd} u \arrow[r] & v \\ \uparrow & \end{tikzcd}$$

therefore in this edge uv, therefore previous of v equal to u, this is Bellman edge. This is actually current we will write it as, current Bellman edge, this Bellman edge will get updated, current Bellman edge.

$$p(v) = u$$

When the loop terminates, alpha iv will have the minimum and p of v will have the previous of the last edge, it will have the Bellman edge proper Bellman edge ending at p. So when the loop is terminated, when step 1 and step 2 are done, after this alpha iv will have minimum over because minimum over u not equal to v of delta u plus weight of uv, sorry not delta my apologies, it is alpha i minus 1, alpha i minus 1,

$$\alpha_i(v) = \min_{u \neq v} \{\alpha_{i-1}(u) + w(u,v)\}$$

every alpha array is fully consulted wherever there is a u and an edge uv this value is computed and minimum the smaller value is updated. Therefore, it will have the minimum, alpha i will have the minimum and pvv is the Bellman edge ending at v and pvv is the Bellman edge ending at v. Maybe I will write this cleanly next page, so that I will erase it this can be edited. So what is the consequence of executing? The consequence of executing is alpha i v is equal to minimum over u not equal to v of alpha i minus 1u plus weight of uv. This is a correct value that we want and pvv is the Bellman edge ending at v.

You have a current Bellman edge but that was updated. Finally when you finished with, alpha i will have the correct value and pvv is the Bellman edge ending at v, so this is called the improvement step, this is called improvement step Improvement step, it improves alpha i vector or array of values to alpha i array of values, alpha i minus 1 array of values.

$$\alpha_{i-1}[\ ] \longrightarrow \alpha_i[\ ]$$

Let us pass a moment to find out what is alpha 0 okay. Let us see alpha 0 array. What is alpha 0s? It is the length of the walk from s to s, from s to s, with maximum 0 edges, that is no edge at all, maximum number of edge at most 0 edge means no edge from s to s you have to go without using an edge 0.

$$\alpha_0[s] = 0$$

What about alpha 0 v for all other one? from s to v you have to go without using any edge maximum 0, edge you are allowed. There is no such walk, if there is no such walk it is infinity.

$$\alpha_0[v] = \infty$$

Therefore this is alpha 0, you know how to compute alpha i minus1 to alpha i, therefore use this improvement step n minus 1 times, from alpha 0 array build alpha 1 array, from alpha 1 array build alpha 2 array. In this way, you keep building it until you get alpha n minus 1 array, alpha n minus 1 array you build.

$$\alpha_0[\ ] \rightarrow \alpha_1[\ ] \rightarrow \alpha_2[\ ] \rightarrow \cdots$$
$$\alpha_{n-1}[\ ]$$

How do you go from one array to another array? The improvement step that we have shown okay. The improvement step that we have shown here, okay, tells you how to go from alpha I minus 1 to alpha i. Now, so from alpha 0 you go to alpha 1 and 2 and you stop here, alpha n minus 1, so you have alpha n minus 1 vector that means alpha n minus 1 s, alpha n minus 1 for another vertex, alpha n minus 1 for yet another vertex. So in this way you have alpha n minus 1 for all vertices you have.

$$\alpha_{n-1}[s], \quad \alpha_{n-1}[\ ], \quad \alpha_{n-1}[\ ], \quad \ldots\ldots$$

Since G has no negative cycles or 0 cycles this is nothing but delta n minus 1 s delta n minus 1 whatever is that vertex delta n minus 1.

$$\delta_{n-1}[s] \quad \delta_{n-1}[\ ] \quad \delta_{n-1}[\ ] \quad \ldots\ldots$$

This is nothing but because we know that we have already seen that alpha i equal to delta i, if G has no negative cycles but delta n minus 1 is nothing but delta, therefore delta s is computed, delta of this vertex is computed, delta of all vertices are computed.

$$\delta(s), \quad \delta(\ ) \quad \delta(\ ) \quad \ldots\ldots$$

In other words single source shortest path problem is solved. You can see that when we were trying to find the delta values directly when we were trying to find the delta values directly using Bellman equation we were caught in a mess there was cyclic dependency and we were not able to get it but now we are getting the delta values in a different way we are using Bellman like bounds for bounded walks those bounded walks have got nice relation which is very easy to compute in one go. You can start from alpha 0 and go to alpha 1, from alpha 1 you can go to alpha 2. So you are able to go and finally you get alpha n minus 1 for every vertex, alpha n minus 1 for every vertex is nothing but delta n minus 1 for that vertex. okay we have seen that alpha n minus 1v is nothing but delta n minus 1v delta n minus 1v is nothing but delta v because no path can have more than.

$$\alpha_{n-1}[v] = \delta_{n-1}[v]$$
$$= \delta(v)$$

So you have solved the delta v for all v by taking a slightly different route okay very impressive is not it you are stuck at a method for finding the delta directly but we used another method in that method we have proceeded iteratively and we have reached the delta values through an iterated improvement process okay. So we will this is a right point to break thank you.