Course: Introduction to Graph Algorithms

Professor: C Pandu Rangan

Department: Computer Science and Engineering

Institute: IISc

Week: 04

Lecture 15 Bellman Ford P2

Namaskara we will continue our discussions on Bellman and Ford algorithm. We have defined the class of walks Wi, okay the set of all walks from s to v with at most i edges.

$$W_i(v) = \{\text{Walks from } s \text{ to } v \text{ with } \leq i \text{ ~~e~~ edges}\}.$$

alpha i is the weight of the minimum walk, alpha i v is the weight of minimum walk in Wi it is v it is depending on v.

$$\alpha_i(v) \text{ is the weight of Min Walk in } W_i(v).$$

So these alpha i's are satisfying certain interesting properties if G has no negative cycle if G has no negative cycle then alpha i v equals minimum over u not equal to v of alpha i minus 1 u plus weight of uv.

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

Not only this we have alpha n minus 1u equal to delta n minus 1u which is equal to delta u for all vertices this is true.

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

These two properties. allow us to write the following program that would start from alpha 0 then in one iteration we will build alpha 1 for all vertices. And finally we stop at alpha n minus 1

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

So here is the code for this Bellman Ford, this takes the directed graph G, ve weight function W source s and G is G has no negative or 0 cycle and we are going to use an array this is like a pseudo code.

$$\text{Bellman-Ford}\left(G = (V, E), \omega, s\right)$$

So we are going to use an array this array is d array div is supposed to hold alpha iv

$$d_i[v] = \alpha_i[v]$$

The purpose of this array d is to hold this value so we have to compute in our code the values div since our goal is to have alpha iv the alpha iv based recurrence is used for the computation of the d values okay. So I initialize d0v to alpha0v for all v.

$$d_0[v] \text{ to } \alpha_0[v] \quad \forall v$$

So this is the first step, in the first step d0s is 0 and d0v is infinity for all v belonging to V minus s v minus small s okay.

$$d_0[s] = 0; \quad d_0[v] = \infty \quad \forall v \in V - s$$

So you have initialized now second step will be the for loop you have to compute. For i equals 1 to n minus 1 so I am going to compute. when i equal to 1 d1 array for i equals 2 the d2 array and so on okay. So for i equals 1 to n, n minus 1 di v equals di minus 1 v, div is di minus 1 v and then we are going to check for each uv in E if this is the condition we are going to check. If div is greater than di minus 1u plus weight of uv then we are going to update this. div is equal to di minus 1u plus weight of uv and parent of v is u okay.

```
For $i = 1$ to $n - 1$
For each $v \in V,\ d_i(v) = d_{i-1}(v)$
For each $(u, v) \in E$
If $d_i(v) > d_{i-1}(u) + w(u, v)$
$d_i(v) = d_{i-1}(u) + w(u, v)$
$p(v) = u$
```

So when we do this n minus 1 times we get third step. return dn minus 1v for all v belonging to V, comment dn minus 1v is delta v the weight of the shortest path is obtained.

$$\text{Return } d_{n-1}(v) \;\; \forall v \in V \;\; \backslash\backslash \, d_{n-1}(v) = \delta(v)$$

Okay straight away all I have done is that implemented the cyclic dependency is broken by the alpha values cyclic dependency was there for the delta values. So I created a new class where we can go smoothly in a single direction and obtain the values that we want, so this is called the iterative method, the values that you want to obtain is to be obtained by solving an equation since we cannot solve that equation we use the relation to iteratively and converge to the answer we are getting, we are reaching our answer in n minus 1 stages, okay. So this is valid if G has no negative cycle what happens if G has a negative cycle.

If G has a negative cycle we can still do this computations the d arrays will have some values but those values may not represent the weight of the shortest path, they are just some numbers that is it okay. Therefore you should not output. You should not output the d array value, dn minus 1 array value, if G has a negative cycle but how do we know whether the G has a negative cycle or not the input can be an arbitrary graph. If the input has no negative cycles if we know that we can straight away run this code because we know, G has no negative cycle we know this kind of computation will lead to the answer so there is no issue if G has no negative cycle.

But if you have no idea whether G has a negative cycle or not you can still do this computation, And then you can detect using the dn minus 1 values whether G has a negative cycle or not that is the beauty of this method okay. The dn values can be used to check whether G has a negative cycle or not. Here is the property of the values in dn minus 1 array, okay.

So theorem if G has no negative cycles or 0 cycles we know that the delta v values satisfy Bellman equation, delta v is less than or equal to delta u plus weight of uv

$$\delta(v) \le \delta(u) + w(u,v)$$

This we know for all u we know that this is Bellman equation. If G has no negative cycle we know that delta v is nothing but delta n minus 1v and delta u is delta n minus 1u plus weight of uv.

$$\delta_{n-1}(v) \le \delta_{n-1}(u) + w(u,v)$$

And if G has no negative cycle we know that delta n minus 1 is nothing but alpha n minus 1v is less than or equal to alpha n minus 1u plus wuv.

$$\propto_{n-1}(v) \leq \propto_{n-1}(u) + w(u,v)$$

But alpha n minus 1 values are in the d array in our program that we have in the pseudo code that means dn minus 1v is less than or equal to dn minus 1u plus weight of uv.

$$d_{n-1}(v) \leq d_{n-1}(u) + w(u,v)$$

So look at the condition that we have derived if G has no negative cycles. or 0 cycle if G has no negative cycle or 0 cycle dn value satisfy this condition. So the negation of that the contrapositive, what is contrapositive if P implies Q not Q implies not P. okay not P implies not Q implies not P, I will write it as not here also.

$$P \Rightarrow Q$$
$$\text{NOT } Q \Rightarrow \text{NOT } P$$

So the contrapositive of what we have proved is the following if there is an edge u, v such that d of n minus 1v is greater than d of n minus 1u plus weight of uv then G has a negative cycle, okay if there is a uv such that dn minus 1v is greater than dn minus 1u so look at this implication.

$$(u,v)\colon d_{n-1}(v) > d_{n-1}(u) + w(u,v)$$

This implication shows that I can check whether G has a negative cycle or not, therefore compute up to dn minus 1 then, you just check whether there is a negative cycle or not if there is a negative cycle simply say that G has a negative cycle do not return the d values because d values do not hold the shortest path weights, d has some numbers that is all okay, but if G does not have a negative cycle we know that d has the correct value you can return it. Therefore, this is the way to deal with arbitrary graph, okay if G is an arbitrary graph. That is it may have positive cycle negative cycle 0 it can have anything it really does it just a weighted graph is given okay.

When a weighted graph is given I can always compute I can initiate d0 array and then I can compute dn minus 1 array that I can do for all graphs. They just need the input weights of the graph other than that you do not need anything. So, if G is an arbitrary graph first compute dn minus 1 array of values. That is you are going to compute dn minus 1v for all v belonging to V for all vertices you are going to compute this okay.

Compute $d_{n-1}[\,]$, $d_{n-1}(v), \forall v \in V$

Now this is what you are going to do next for each u, v belong to E if dn minus 1 of v is greater than dn minus 1u plus weight of uv,

For each $(u,v) \in \mathrm{E}$ such that $\left(d_{n-1}(v) > d_{n-1}(u) + w(u,v)\right)$

If this happens for a particular edge return, that is terminate the program return the message G has a negative cycle, because G has a negative cycle just terminate with the report that G has a negative cycle, that is it if this happens even for one edge when you encounter just one edge you do this.

So at the encountering of the first edge satisfying this condition you terminate the program and report that G has a negative edge. Suppose this has not happened for any edge that means you will execute the step 2 but you will not return anything because this condition will fail. dn minus 1v will be less than or equal to dn minus 1u plus w for all edges. Now you know that G has no negative cycle since G has no negative cycle you know that dn minus 1 is carrying the correct values. So step 3 return d n minus 1 v and previous of v for all v belonging to V.

$$d_{n-1}[v], \; p(v), \; \forall v \in V$$

So you will execute step 3 only step 2 does not return anything, if step 2 returns the control is returned the code is terminated you will never go to step 3, so if step 2 returns you are terminating with a message that G has a negative cycle. Notice that I am not returning the d values, p values, nothing. I am only reporting that G has a negative cycle. That is because if G has a negative cycle, all the computation I have done, I have done of course, they are useless. the d values are not shortest path weights at all.

So I am not going to return any of them okay. So this is the complete Bellman Ford algorithm. The Bellman Ford algorithm if G has no negative cycle returns the shortest path weights. If G has a negative cycle it reports that it has indeed a negative cycle okay. Very clever and nice algorithm the complexity is very easy to determine okay

Complexity of Bellman-Ford algorithm. It is very simple there are n minus 1 iteration in each iteration you are working on all edges on behalf of each edge you do one comparison and one assignment therefore the complexity is clearly n minus 1 times m. There are n minus 1 iterations if the G is arbitrary after finding dn minus 1 in n minus 1 iterations you make one more pass on all the edges that will be another m therefore the complexity is order nm.

$$(n-1)m + m \qquad = O(nm)$$

This is to compute dn minus 1 array, this is to check if G has a negative cycle, therefore you are able to solve the problem in order nm time single source shortest path problem on an arbitrary graph is solved in the following manner okay. If the graph has no negative cycles it will give the answer. This will not give any answer if the graph has got negative cycle because if the graph has negative cycle computing the shortest path distance and other things, we cannot use this approach. Why we cannot use this approach? All this approach is based on Bellman equation. Bellman equations are valid only when the graph has no negative cycles.

If the graph has no negative cycle Bellman equation itself does not hold good and none of these steps would make any sense and hence the value you have makes no sense. That is the reason why for a graph that has a negative cycle I simply report that G has a negative cycle and do not return any values but if I am convinced if I am confirmed that G has no negative cycle. then the dn values will be returned okay. This completes the discussions related to Bellman Ford algorithm thank you.