Course: Introduction to Graph Algorithms

Professor: C Pandu Rangan

Department: Computer Science and Engineering

Institute: IISc

Week: 03

Lecture 11 Belman Equation Part 2

Namaskara we have been looking at the properties of the minimum weights of the shortest paths. We are looking at the single source shortest path problem from the source to other vertices. We are interested in finding the shortest path. Towards that we are looking into the mathematical properties of the minimum weights or the weights of the shortest paths. If G is an instance of the single source shortest path problem

$$G = (V, E, w, s)$$

and if delta v is the weight of shortest path from s to v. Then we have derived the following equations when G has no negative cycles. When G has no negative cycles, we have proved that delta s equal to 0

$$\delta(s) = 0$$

and delta v equal to minimum over the edges, all the incoming edges, (u, v) minimum over (u, v) of delta u plus weight of (u, v).

$$\delta(v) = \min_{(u,v)} \{\delta(u) + w(u,v)\}$$

We can extend it and make it simpler looking by adding other conventions. But in a sense this is the equation called Bellman equation and we have seen an example where if G has zero cycle, this equation allows us to have infinitely many solutions and if G has a negative cycle, of course these equations do not make any sense and as expected it does not have any solution we have also shown the non-existence. Okay, to summarize here is the theorem, let G equal to (V, E, w, s) be an instance of single source shortest path (SSSP) problem. If G has no negative cycle or zero cycles, then the equations xs equal to 0

$$x_s = 0$$

xv equal to minimum over all the incoming edges, (u, v) over xu plus weight of (u, v), small v, has a unique solution and the unique solution.

$$x_v = \min_{(u,v)} \{ x_u + w(u,v) \}$$

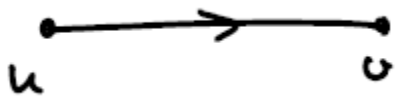
And the unique solution is xv equal to delta v for all v belonging to V, this is that unique solution. Okay, and this gives us, I mean assuming that we are able to solve, we are getting the weights of the shortest paths. But what we want is not only the weights of the shortest path, we want those paths themselves. How do we explicitly obtain the paths with this length? We have to look into that and fortunately the solution contains enough information about the shortest path itself okay.

$$x_v = \delta(v) \quad \forall \ v \in V$$

So in the unique solution for a vertex v we will have xv equal to xu plus weight of (u, v) for some edge (u, v).

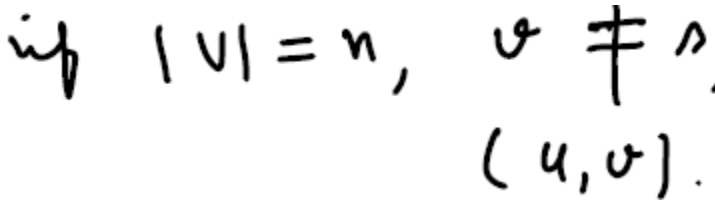
$$x_v = x_u + w(u,v)$$

This is an edge incoming to v or it is an edge ending at v, okay. It is like this, u and is an edge ending at v.



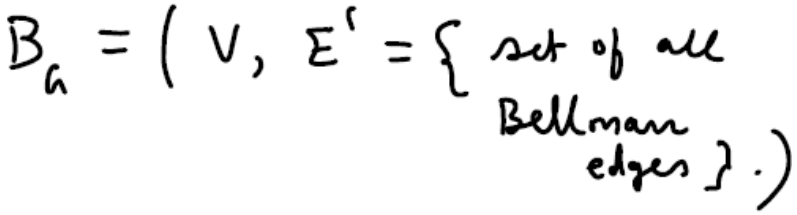
So this how the solution is if you look into the final solution it will have this form and we call this edge (u,v) which is ending at v as the Bellman edge corresponding to the vertex v. This (u,v) is called Bellman edge responding to v okay. In the solution, there is an edge that is ending at v and that edge is called Bellman edge. So for each vertex there is only one edge because in that solution we have an edge ending at that vertex. So one Bellman edge for each vertex, but if you look at s there is no edge ending at s, in fact delta s is straight away set as 0 there is no Bellman edge associated with it. So for all vertices other

than s we have a Bellman edge associated with this, therefore if cardinality of V equal to n, other than s there are n minus 1 vertices, for each vertex v not equal to s we have an associated Bellman edge (u,v),



Therefore there are n minus1 Bellman edges, these edges are edges of the graph, okay, and there are n minus1 Bellman edges. We construct a graph, it is called the Bellman graph associated with G because from G you form the Bellman equation and then when you solve you get these edges identified as a part of the solution.

So what is a Bellman graph? Bellman graph B associated with G you can say, the same vertex set, BG is same vertex set and the edge set is the set of all Bellman edges okay, the edge set E dash is set of all Bellman edges, so this is a graph with same vertex set but only n minus 1 edges, okay.



We are going to show that this graph is in fact a tree, okay. There are no cycles first we will show that there are no cycles. In fact it is a directed graph therefore it is a tree like structure and this is called out tree. Out tree means from the root all edges will be directed towards the leaf. This is a root, all edges will be like this. This is the root, s is going to act as a root and to all other vertices we are going to have paths, okay, we will show all of them.

So this Bellman graph is going to be, this BG we will show as in fact an out tree. Out tree means, from the root all edges are directed towards the leaf. In tree means all the edges are directed towards the root. They will all go towards the root. Here all of them are going away from the root, okay. This is called out tree. We will show that this is basically an out tree. First we will show that there are no cycles okay. BG has no cycles. Suppose it has a cycle right assume that BG has a cycle, let C equal to v0, v1, v2, vn where vk equal to v0 be a cycle in BG.

$$C = \langle v_0, v_1, v_2, \ldots v_k \rangle$$

$$v_k = v_0 \text{ be a cycle in } B_G$$

Assume that there is a cycle, we are going to arrive at a contradiction, okay. We are going to arrive at a contradiction so the C is like this, starting from v0 to to v1, v1 to v2, v2 to v3 and so on, vk-1 to vk and vk is nothing but v0, vk is v0, it comes back.

These are all the edges in the Bellman graph. Therefore you have v1, delta v1 equal to delta v0 plus weight of (v0 v1), it is equal because it is a Bellman edge, right. This is a solution, therefore we have this equality.

$$\delta(v_1) = \delta(v_0) + w(v_0, v_1)$$

And delta v2 is delta v1 plus weight of (v1 v2). So, in this way we are going to have for each edge an equation, right, and finally delta vk equal to delta vk-1 plus weight of (vk-1 vk), right, for each edge we have an equality, you add all of them. All right, here you are going to get weight of (v0, v1), weight of (v1, v2), weight of (vk-1, vk). So you're going to get the weight of the cycle, right? You're going to get the weight of the cycle when you add all of them.

$$\delta(v_1) = \delta(v_0) + w(v_0, v_1)$$

$$\delta(v_2) = \delta(v_1) + w(v_1, v_2)$$

$$\vdots$$

$$\delta(v_k) = \delta(v_{k-1}) + w(v_{k-1}, v_k)$$

So here, adding, you get sigma i equal to 1 to k of delta vi is equal to sigma i equal to 0 to k-1, here v0 to vk of delta vi plus weight of the cycle C. All these things they add up to weight of the cycle C, all the edges of the cycle are listed here, their weights are added so you get the weight of the cycle there. But these two terms are equal, therefore you can cancel them.

$$\sum_{i=1}^{k} \delta(v_i) = \sum_{i=0}^{k-1} \delta(v_i) + w(C)$$

Why they are equal? if you look at v1 to vk, right, delta v1 plus delta v2 up to delta vk,

$$\delta(v_1) + \delta(v_2) + \cdots + \delta(v_k)$$

is same as delta v1 plus delta v2 plus delta vk-1 plus, vk is v0, delta v0,

$$= \delta(v_1) + \delta(v_2) + \cdots + \delta(v_{k-1}) + \delta(v_0)$$

bring this v0 here, you see that this is delta v0 plus delta v1 up to delta vk-1.

$$= \delta(v_0) + \delta(v_1) + \cdots + \delta(v_{k-1})$$

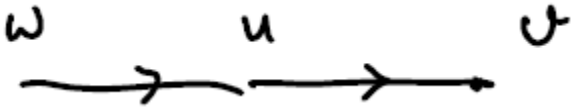
So that is what is there on the right hand side. So this implies sigma i equal to 1 to k, this is i equal to 1 to k of delta vi is same as sigma i equal to 0 to k-1 of delta vi, both terms they are same value so you can cancel them out. okay,

$$\Rightarrow \sum_{i=1}^{k} \delta(v_i) = \sum_{i=0}^{k-1} \delta(v_i)$$

So when you cancel them out, you get 0 equal to W(c) implying that C is a zero cycle.

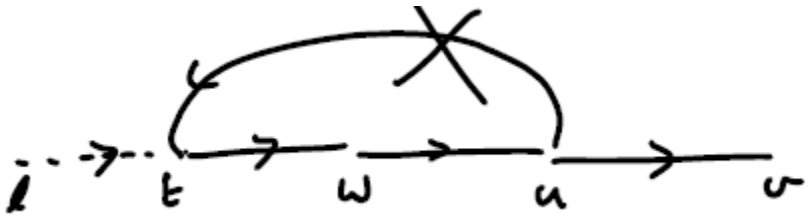
$$0 = W(c)$$

But we have assumed that we do not have a zero cycle or a negative cycle in the graph, okay. our graph has no negative cycle no zero cycle. Only on such graph we can formulate this equation and then attempt to solve. So we are assuming that G has the property. C as a 0 cycle contradicting the assumption that G has no 0 cycle. We have assumed that G has no 0 cycle but here we are showing a 0 cycle, if we assume that Bellman graph has a cycle, therefore Bellman graph has no cycles okay. Since Bellman graph has no cycle we get the following consequence, right.
Let us take any vertex v, take any vertex v, other than s, okay, s has no Bellman edge. So there is a Bellman edge ending at v, let us say (u, v) is the Bellman edge ending at v. For u, there is a Bellman edge ending at u let that be w.

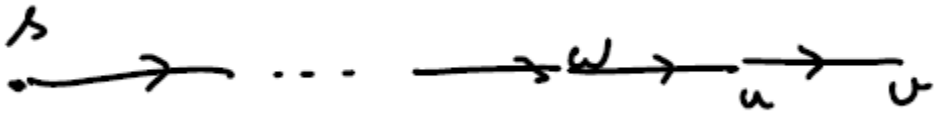


So in this way this chain keeps going backwards, okay, this chain keeps going backwards. Every time you are going to see a new vertex that is not considered already, when you go back you will keep finding newer and newer vertex. Why? you will never

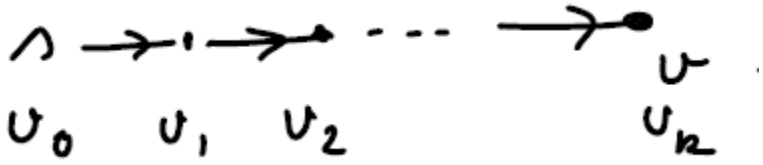
encounter a vertex that you have already visited, if it so it forms a cycle. Let us see, from v you have come to, sorry, from u you have gone to v, from w you have gone, from, say from t, but to t suppose you have come from one of the already visited vertex then it will be like this, right, and this is going to form a cycle. Such an edge will not be there, therefore such an edge will not be there to t probably you will come from some other vertex, okay, some l.



It will all be newer vertices every time you are going to come from a distinct vertex that is not encountered in our backward journey. Therefore when you keep on getting newer vertices, at some point you will get s, there is no cycle and then every time you are means some, so it is going to be like that. From s you cannot go any further backwards because s has no Bellman edge, ending at s. Therefore the process would stop at s and things will look like this.



When this backward extension stops at s, we see that there is a path from s to v in the graph. So for every vertex there is a path from s to that vertex, therefore it is going to be a out tree, no cycles. This path is called Pv. In fact this path is a shortest path from s to v, therefore this Bellman edges sequenced like this, they lead to a path, that path is in fact a shortest path, okay. Why this path is a shortest path? you have a selection of edges and they go to s. But how come they constitute a shortest path? The proof is simple, very similar to the argument that we have given for cycle. We will, let us say we have a path from s to some vertex and from there to some vertex and so on and finally from there to v, we have seen that such a, just for the sake of ease of reference we call this s as v0 the next one as v1 and the next one as v2 and v itself will be vk for some k, okay.



Because it is a path we name it like this so that we refer the edges as (v0, v1), (v1, v2) and so on so that it becomes easy to do index based computation, okay. Now in this (vi minus 1, vi), (vi minus 1, vi) is a Bellman edge, okay.

$$(v_{i-1}, v_i)$$

Therefore we have delta vi is equal to delta vi-1 plus weight of (vi minus 1, vi), weight of (vi minus 1 vi),

$$\delta(v_i) = \delta(v_{i-1}) + w(v_{i-1}, v_i)$$

This implies weight of (vi-1 vi) is equal to, move the delta v, so delta vi minus delta vi minus1, okay.

$$\Rightarrow w(v_{i-1}, v_i) = \delta(v_i) - \delta(v_{i-1})$$

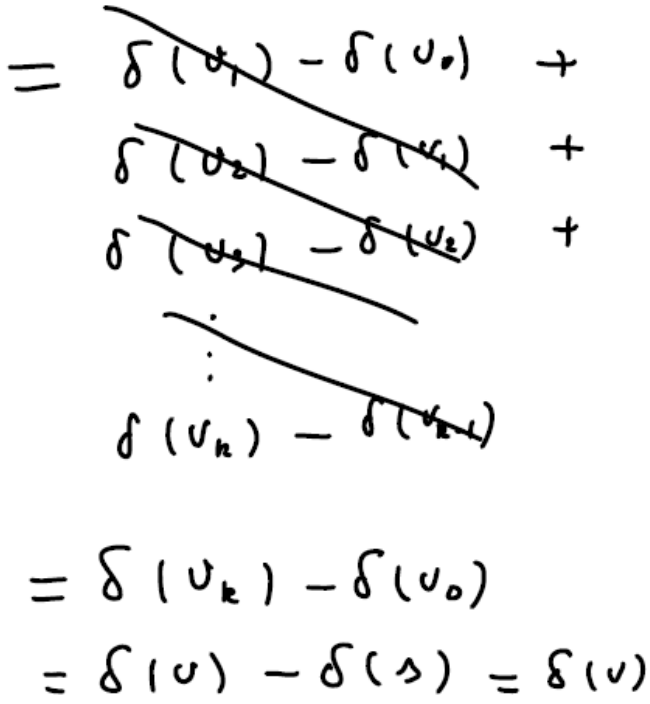
I have moved delta vi minus 1 to the other side. So, for each i I have this equality. What is the weight of the path? Weight of the path is, let us say this is path Pv, weight of Pv is weight of (v0 v1) plus (v1 v2) plus v1, is equal to sigma i equal to 1 to k of w(vi minus 1, vi) that is equal to sigma i equal to 1 to k of d vi minus d vi minus 1.

$$w(P_v) = \sum_{i=1}^{k} w(v_{i-1}, v_i) = \sum_{i=1}^{k} \delta(v_i) - \delta(v_{i-1})$$

Because w(vi minus 1 vi) is dvi delta vi-delta vi-1, not d, it is delta I am sorry, it is delta. The sum on the right hand side is called the telescopic sum, okay. sigma i equal to 1 to k of delta vi minus delta vi minus 1.

$$\sum_{i=1}^{k} \delta(v_i) - \delta(v_{i-1})$$

How does this look, for i equal to 1, it is delta v1 minus delta v0 plus; for i equal to 2, delta v2 minus delta v1; plus delta v3 minus delta v2 and so on finally for k it is a delta vk minus delta vk – 1. Here, delta v1 and delta v1 cancels, delta v2 and delta v2 cancels and in this way all of them will cancel except vk and v0. Therefore, this sum is actually delta vk minus delta v0. vk is v, right, therefore this is equal to delta v, v0 is s, delta s but delta s is 0, therefore this is equal to delta v.



$$= \delta(v_k) - \delta(v_0)$$

$$= \delta(v) - \delta(s) = \delta(v)$$

In other words weight of Pv is delta v.

$$w(P_v) = \delta(v)$$

That shows Pv is a shortest from s to v. So all you have to do is that while solving the equation you not only obtain the delta v values, there are edges whose weights define those equation just save those edges and those edges are useful in constructing the shortest paths, okay. Just those n minus1 edges you hold, that is enough. They are all useful in constructing the shortest path. But if you hold them in a messy way constructing the shortest path would become hard, so, we are going to maintain it in a very simple and an elegant way, okay. It is based on the following observation. If you look at the way in which we have derived the equation for delta v, delta v equal to delta u plus weight of (u,v), this is a shortest path, this is a shortest path and this is the last edge in the shortest path.

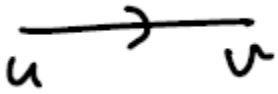
$$\delta(v) = \delta(u) + w(u,v)$$

So from s, u, v, this is the last edge. Okay, therefore u is a vertex previous to the vertex v in that path.

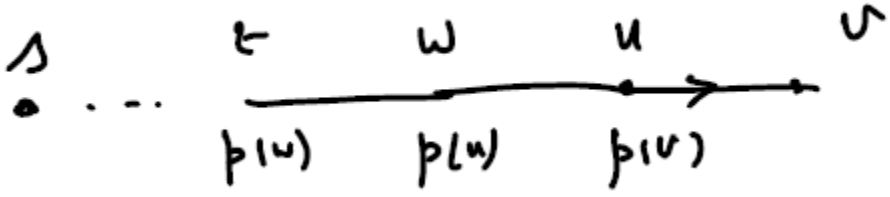
All we are going to do is that we save the edge in the following way, for each v we maintain the previous vertex, so u equal to p(v), so for each vertex there is only one previous vertex. Therefore, I can use an array indexed by vertices.

$$u = p(v)$$

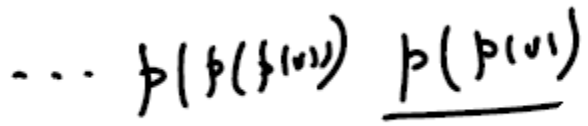
p of s is undefined. There is no Bellman edge. There is only one Bellman edge for each vertex. That edge is ending at v. So if (u,v) is a Bellman edge, then previous of v is defined as u.



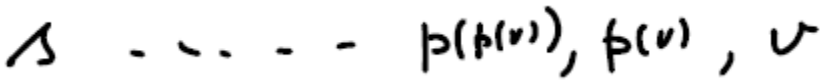
Now for u there is a previous of u, that is how that chain was constructed. There is a Bellman edge ending at u, that we have considered that will be previous of u and for the other one there will be, so for v, this is u, this is previous of u, for u there is a w that will be previous of u. Sorry this is previous of v. For v this is previous, for w is previous of u, for w there might be some t which is previous of w, in this way it goes all the way up to s. s has nothing previous to it, this is the path pv.



So this p(u) now can be written as p(p(v)). This p(w) can be written as p(p(p(v))), because this is one vertex, its previous is this and so on.

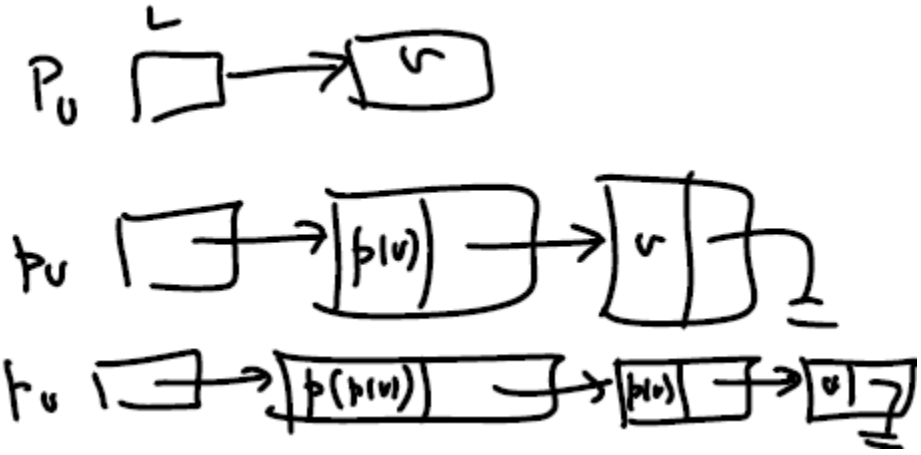


So if you have a previous array, you keep looking into the previous array and keep obtaining the sequence of vertices. First start from v go to p(v), from there go to its p, which is denoted as previous of previous of v, p(p(v)). From there again go to p(p(p(v))). Keep going until you get s, right, then you are going to get the vertices of the path. What is the actual vertex sequence of the path? It is v, p(v), p(p(v)) and so on up to s, okay.

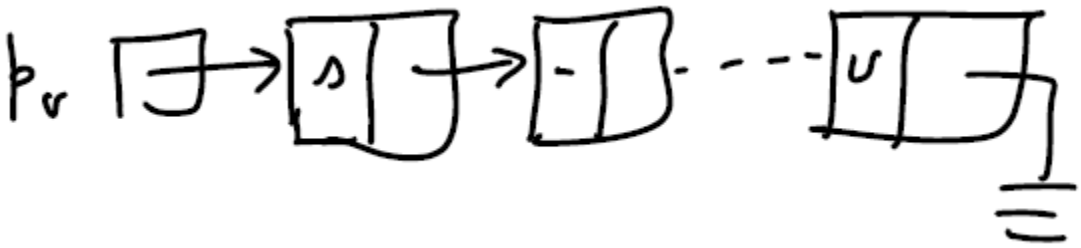


So this is how you would get. Starting from v, keep applying the p function or p array entry consultation, until you hit s. s has nothing previous to it so you are going to stop there. But this is outputting the vertices of the path in the reverse order. What you want actually is the path from s to v, the shortest path from s to v. But those vertices which are in the shortest path are obtained in the reverse order. So in order to output properly p of v, what is that you should do? Print this in the reverse order, generate it, but print it in the reverse order. For example, if you want to have a linked list of vertices of P of v. How can you create that linked list? It is very simple.

First L this is a linked list for P of v. First it will point to v. Insert in the beginning of the list. then L this p of v will become p(v) and that will point to v. Insert, find p(p(v)) but insert at the beginning of the list, always insert at the beginning of the list. Then p of v will be a list pointing to p(p(v)), this you have found out, this is inserted in the beginning and so on.

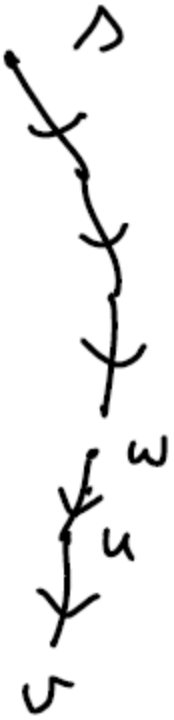


So finally p of v will be a list pointing to s, when you find s you insert it, finally it will have v.



You can see that this linked list is in fact containing the vertices of the path p of v in that order. That is because I have inserted in the beginning, it is a kind of reverse growth. Therefore as you see the linked list the vertices are appearing in the natural order from s to v, therefore we are going to only construct p of v. p of v is used as and when required, as and when you require a shortest path, that P array will be used to actually build the path.

I am not going to output the paths, I am going to output only p(v), if you require physically the path you generate it. Starting from the vertex apply p(v), p(p(v)) and so on go to s and output those sequence of vertices, you have the physical path. Therefore paths are not explicitly output. Paths are implicitly output through p array, okay. The P array is called parent array or previous array, parent array or previous array, this is the name or interpretation of this p array. p array is called implicit representation of shortest path, okay. Why it is called the parent? If you look at the Bellman tree, okay, so this is how the path v, parent of v this is u, (u,v) is an edge and for u let us say (w,u) is the Bellman edge.



This Bellman edge tracked backwards go up to s, therefore it is an out tree. In that out tree u is a parent of v. For v, I maintain only one piece of information, namely its parent, that is all.

So every vertex has got only the pre or parent information. With this information, you can explicitly construct the path by chasing the parent array pointers or parent links up to s, up to the root, okay. That is the reason why this called implicit, means hidden. You are not giving for every vertex the long list of vertices of shortest path. For every vertex you give only p(v,) that is all you are given, okay. Therefore our goal is given G equal to V, E, w, s. G has no negative or 0 cycles, is given, given that G has no negative cycles.

$$G = (V, E, w, s)$$

Output delta v, p(v) for all v belonging to V.

$$\delta(v),\ p(v) \quad \forall\ v \in V$$

We know that delta s equal to 0, p(s) is undefined.

$(\delta(s) = 0, \quad p(s) = \perp)$

There is no previous to s. This we know. All others we compute and solve, right. We solve Bellman equation and then we can compute it and in this way we would proceed. okay, so this completes our discussions on Bellman tree, in fact Bellman tree is also known as shortest path tree. Bellman tree is also known as shortest path tree. Because, the path in the tree from s to a vertex is indeed a shortest path. But I am not explicitly representing the tree, I am representing the tree using a single array, for each vertex I maintain only the previous vertex in the path. Okay, this compact representation allows us to build efficient algorithms. And as and when you need the path explicitly use the P array and construct the path as outlined earlier, okay. So this completes our discussions on Bellman equations, Bellman edges, Bellman trees and shortest path trees and this allows us to represent our goal in a more concise manner. Here is the clean statement of the single source shortest path problem. Thank you.