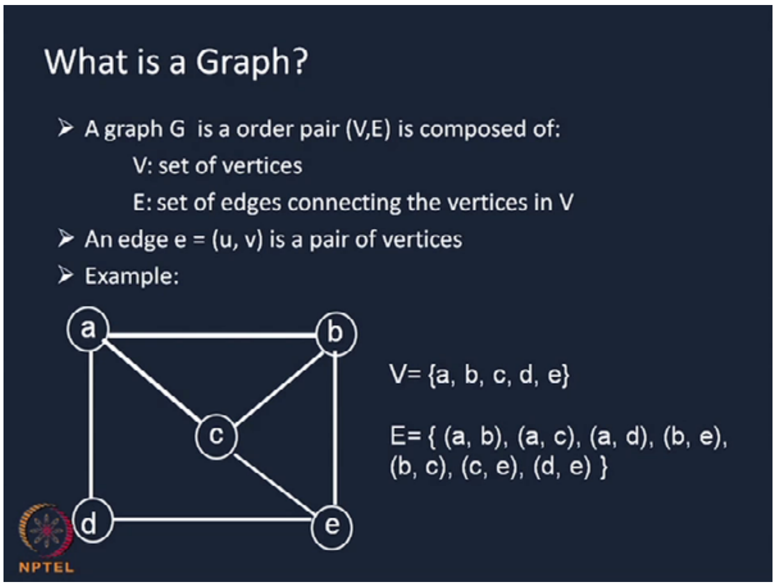
Optimization
Prof. A. Goswami
Department of Mathematics
Indian Institute of Technology Kharagpur

Lecture -19
Shortest Path Algorithm

In this lecture, we want to discuss the shortest path algorithm; before discussing the shortest path algorithm just will discuss little bit about what is a graph for this directed graph, what is undirected graph, what is weighted graph? And, then we will go to the shortest path algorithm. Basically, if you see the we have derived shortest path using the C P M and path method. Now, we want to derive shortest path algorithm using the earlier other method that is graphically; let us see first the definition.

(Refer Slide Time: 00:50)



What is a Graph?

- A graph G is a order pair (V,E) is composed of:
  V: set of vertices
  E: set of edges connecting the vertices in V
- An edge e = (u, v) is a pair of vertices
- Example:

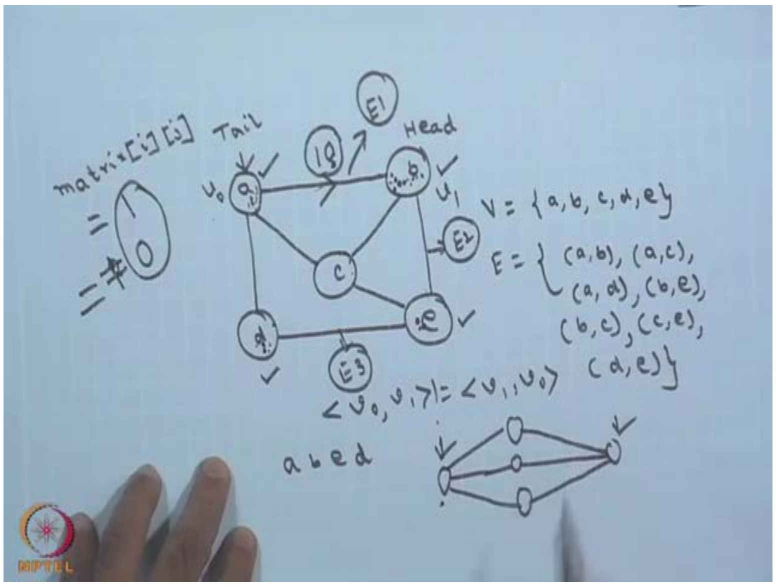
V= {a, b, c, d, e}

E= { (a, b), (a, c), (a, d), (b, e), (b, c), (c, e), (d, e) }

a
b
c
d
e
NPTEL

What is a graph? A graph we say that is a order pair V, E which is a composed of what is V is the set of vertices; E is the set of edges connecting the vertices. So, basically a graph will have a set of pairs of the form V, E; V is the set of vertices, E is the set of edges which connects the vertices and edge; what you say that is e equals (u, v) which is a pair.

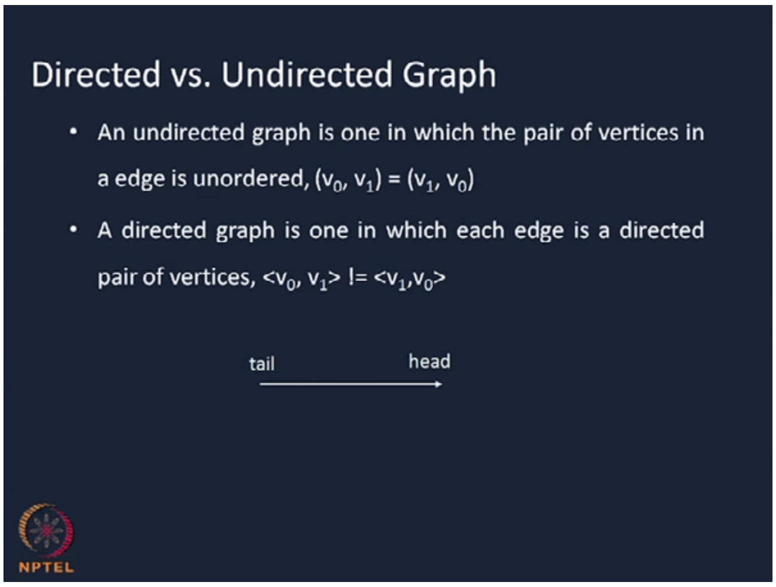
(Refer Slide Time: 01:23)



So, if you take the example over here what I have drawn over there. But it will be better if it tell from here; if you are having say your vertices a b c and d, e these are the vertices are there. If I have drawn a lines from here a to b, a to c, a to d and there are d to e, c to b, c to e and b to e. So, what are the vertices here v set of vertices vertices will be a, b, c, d and e; what we have written over here; v is a, b, c, d, e and . And, the edges; the edges will be the pairs what are the pairs from the figure if you see the pairs one pair is a b; another pair is a c, other pair is a d.

So, (a, b), (a, c), (a, b), (a, c) and (a, d) this is one side after that from b you can write down (b, c) and (b, e). So, (b, e) and (b, c) this is another one; from c it is (c, e) and then (c, e) the other one is (d, e); if you see on the figure over there. So, the we are drawn it the ((Refer Time: 02:41)) equals V is the vertices. And, therefore a b c d e these are the vertices. And, for that the edges are one edges is a to b, another edges a to c, other one is a to d I have written that; the next one is b to e that is b to c b to e; the other thing is d e and c to e. So, therefore b c I have to not written twice (b, c) is coming only once. So, if you draw any graph for the graph there will be a set of vertices and the set of edges like this.

(Refer Slide Time: 03:20)



Directed vs. Undirected Graph

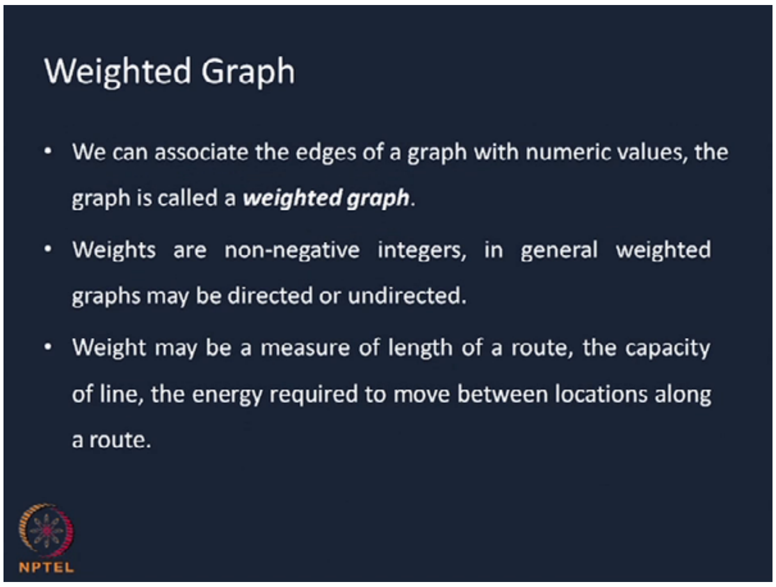
- An undirected graph is one in which the pair of vertices in a edge is unordered, $(v_0, v_1) = (v_1, v_0)$
- A directed graph is one in which each edge is a directed pair of vertices, $<v_0, v_1> \; != \; <v_1,v_0>$

The next one is what is directed graph and what is undirected graph? A directed graph is one in which the pair of vertices in a edges is un it is unordered; it is unordered means undirected graph sorry undirected graph is one in which the pair of vertices in a pair of vertices in edge is uncovered. Uncovered means what I have written that is (V 0, V 1) should be equals to (V 1, V 0) that is if you form any activity from V 0 to V 1 whatever activity time it takes from V 1 to V 0 if you take it will take the same time.

So, undirected means it may be from V 0 to V 1 or from V 1 to V 0 whereas just the opposite one is the directed graph. If you see the directed graph is the edge from where in this case your (V 0, V 1) this should be equals to V 1 not equals to (V 1, V 0). So, if you have V 0 to V 1 is not equals to V 1 to V 0; that is there will be only one side edge; one edge either it will be form V 0 to V 1 or from V 1 to V 0. So, just like whatever I have drawn that is undirected graph there is no direction. So, it may be from a to b from b to a whereas if you make anything like this one arrow like this; then there will be a direction the direction is a to b. Next if you see for this one; we have written this is the tail and this is the head.

So, basically this I have told earlier also; this is the tail and this will be the head. So, this is your directed and undirected graph.

(Refer Slide Time: 05:04)

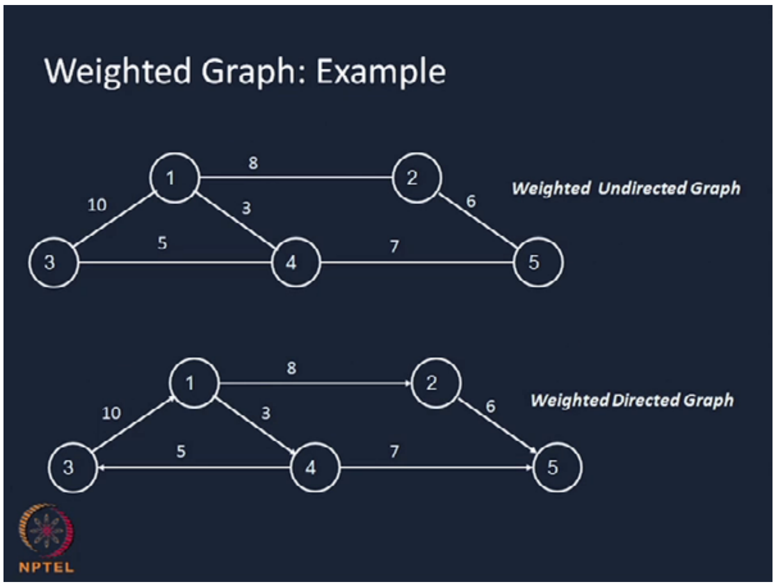

Weighted Graph

- We can associate the edges of a graph with numeric values, the graph is called a ***weighted graph***.
- Weights are non-negative integers, in general weighted graphs may be directed or undirected.
- Weight may be a measure of length of a route, the capacity of line, the energy required to move between locations along a route.

Next is what is weighted graph? We can associate the edges of a graph; edge we have told edges means the directions the pair (a, b) edges of a graph with numeric values; then the graph we call it as the weighted graph. This numeric values or weights we say it may be nonnegative integers in general in it is in. But weight may be a measure of length of route, the capacity of line, the energy required to move between locations among a load, among a route like this or in other sense I want to say that what is the weighted graph? In the weighted graph if you have the you see this figure to move from a to b; you need you have to spend either sometimes or some cost or some labor whatever; it may be to go from vertex x to vertices b and that amount may be say 10 units.

So, this 10 units wherever you are put in then we call it as the weighted graph like this way I can put weight for each and every graph.

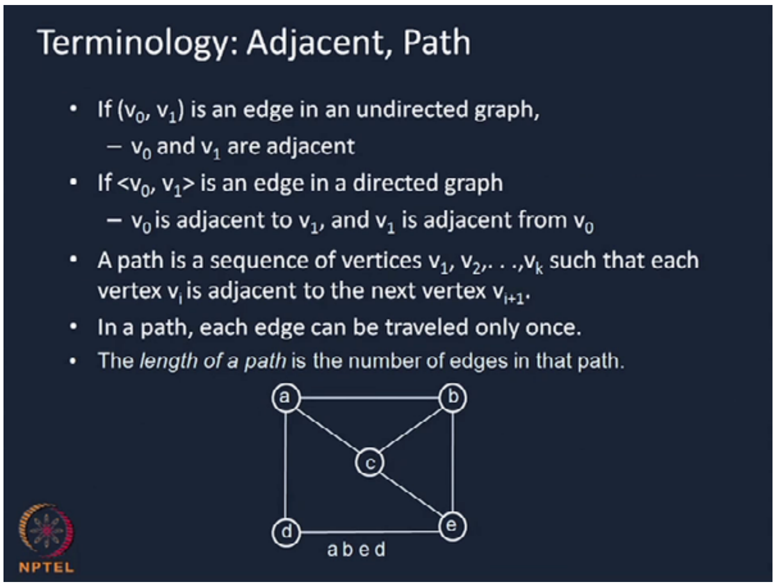
(Refer Slide Time: 06:10)



Weighted Graph: Example

So, you see the example; the first one is the weighted undirected graph you are put the weights if you see 1 to 2 some weights are given 8; 2 to 5 weight is 6 like this way the weights are given. So, these are un weighted undirected graph whereas if you see the next figure; then there are directions from 1 to 2; 1 to 4 say similarly 4 to 3 like this; and weights are given. So, the next example is for the weighted graphs.

(Refer Slide Time: 06:4)



Terminology: Adjacent, Path

- If (v_0, v_1) is an edge in an undirected graph,
  - v_0 and v_1 are adjacent
- If $<v_0, v_1>$ is an edge in a directed graph
  - v_0 is adjacent to v_1, and v_1 is adjacent from v_0
- A path is a sequence of vertices $v_1, v_2, \ldots, v_k$ such that each vertex v_i is adjacent to the next vertex v_{i+1}.
- In a path, each edge can be traveled only once.
- The *length of a path* is the number of edges in that path.

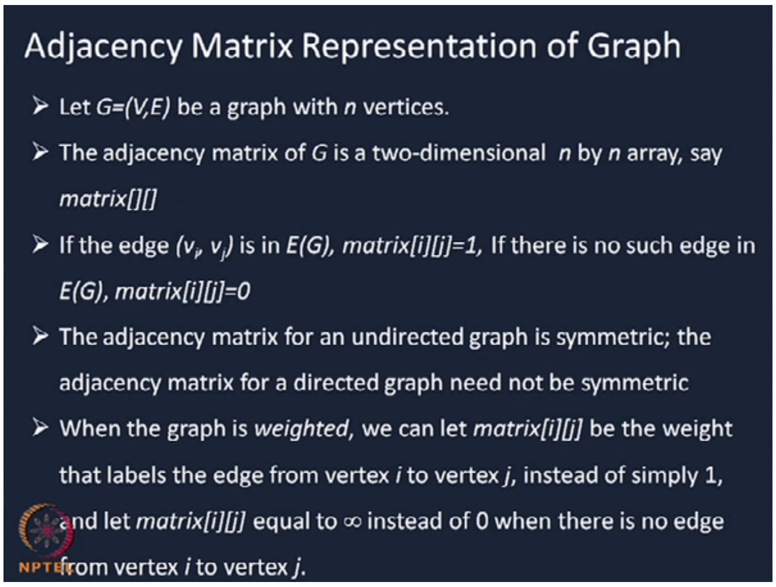
Now, we can associate termination some notions we will use if (V 0, V 1) is an edge of an undirected graph. Then, you say that V 0 and V 1 are adjacent or in other sense in this figure if you are drawing a, b you can say a b is adjacent or this d e; this d e is adjacent. And, if this a, b is an edge of an directed graph basically this is I have given a direction.

So, a, b we are calling is an edge of the directed graph whereas d e is edge of one undirected graph; of course both cannot occurs simultaneously you have to understand it that either there will be directed graph or there will be undirected graph. So, if you are having the directed graph for that if 2 edges are there a to b then these V 0 and V 1; this we call as the adjacent.

Next is a path is a sequence of vertices V 1, V 2. V k a path is a sequence of vertices V 1, V 2, V k such that each vertices V i is adjacent to the next vertices V i plus 1. So, you are a having path in a sequence just like if you see a b, a b, e d if you want to go like this a b, e d something like that; for this path whenever you are moving it is a sequence of path moving here to here, from here, from this position starting position this then next position after that this position and after that this position. If you see if this is the starting position next one will be this is if this is V 0.

Then, this one will be V 1 that is if it is V i it will be V i plus 1 like that way it will be going; you have to note on thing that in a path each edge can be travelled only once; each edge sorry each edge can be travelled only once each edge that is this edges can be travelled only once. And, the last one is the length of a path will be the number of edges in that path that is if I am moving in the direction of a, b, e, d. Then, number of edges this is your edge one I am say E 1, E 2 and E 3 this sum of E 1, E 2and E 3; E 1 E 2and E 3 the edges which has travelled sum of this 3will give you the length of that particular path.

(Refer Slide Time: 09:20)


Adjacency Matrix Representation of Graph

- Let *G=(V,E)* be a graph with *n* vertices.
- The adjacency matrix of *G* is a two-dimensional *n* by *n* array, say *matrix[][]*
- If the edge *(v_i, v_j)* is in *E(G), matrix[i][j]=1,* If there is no such edge in *E(G), matrix[i][j]=0*
- The adjacency matrix for an undirected graph is symmetric; the adjacency matrix for a directed graph need not be symmetric
- When the graph is *weighted*, we can let *matrix[i][j]* be the weight that labels the edge from vertex *i* to vertex *j*, instead of simply 1, and let *matrix[i][j]* equal to ∞ instead of 0 when there is no edge from vertex *i* to vertex *j*.

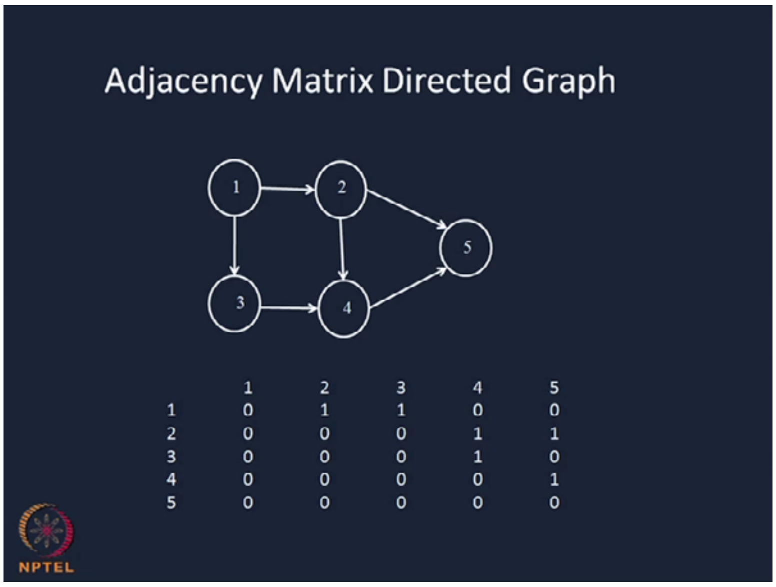
Then, the adjacent matrix representation that is this graph can be represented in terms of some matrices.

So, let G equals (V, E) be a graph with vertices; usually the adjacent matrix we are denoting if you see here we have given it the notation matrix which is a 2 dimensional array; we have given the notation matrix here. So, if the edge (V i, V j) is in e j then matrix i j equals 1; if there is no such edge then matrix i j equals 0 or in other sense whenever you are writing the matrix say here matrix i j; whenever you are writing matrix i j; if there is an there is one edge between V i and V j. Then, we say this value is 1; if there is no edge then we say this is equals to 0.

So, the value will be either 1 or 0depending up on whether for one path i, j for the vertices (V i, V j) if it is there then we say matrix i j equals 1; otherwise we say that the matrix i j is 0. The next one is the adjacency matrix for an undirected graph is symmetric; whereas the adjacent matrix adjacency matrix for the directed graph may be symmetry may not be symmetry. For the adjacency matrix for the undirected graph must be symmetric because there is no direction. And, when the graph is weighted we can laid the matrix in the i j you weight that label the edge from vertex i to vertex j instead of simply 1 or in other sense we want to say just like if you see this figure. In this figure wherever you are moving from a to b we have associated one weight that is 10.

So, in that case the value will not be 1; the value will be 10. So, in this case whatever we are doing the if there is an edge some value is associated, some weight is associated in the matrix we put the weights. Let us see the example.

(Refer Slide Time: 11:33)

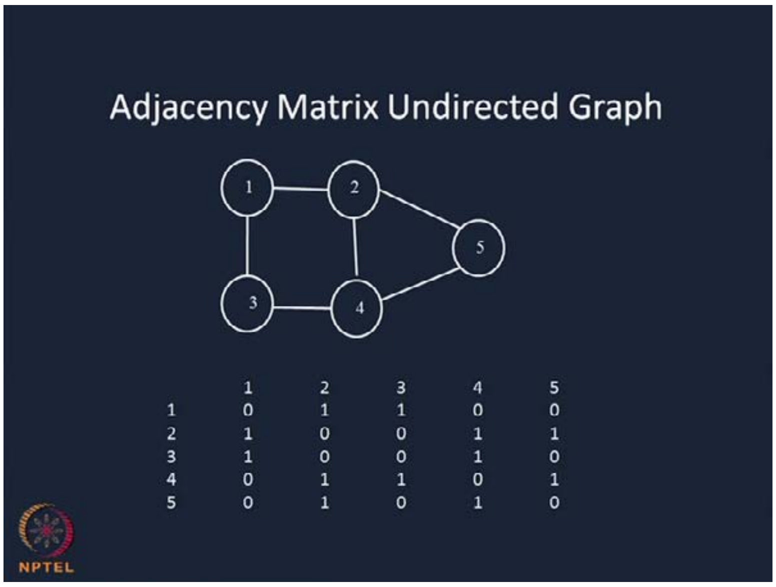

Adjacency Matrix Directed Graph

	1	2	3	4	5
1	0	1	1	0	0
2	0	0	0	1	1
3	0	0	0	1	0
4	0	0	0	0	1
5	0	0	0	0	0

In the next example it will be very clear for the directed graph if you see there is a directed graph 1, 2, 3, 5. So, you have made a matrix; in the matrix there are 5 rows and 5 rows; 5 rows and 5 columns because you are having the vertices are 1, 2, 3, 4, 5. So, one to one if you see there is no path obviously so it is 0; 1 to 2 there is a path so you have put 1. Similarly, 1 to 3 there is a path we have put 1; 1 to 4 there is no path so it is 0 if you see the element of 1, 4.

Similarly, the element of 1,5 this is equals 0 and the other rows are coming like this. For the second row 2 only 2 2 only 2, 4 there is an path from 2 to 4 and 2 to 5. So, this two are one and one only for the second row like this way you are putting the next elements of this. So, therefore if there is one edge from or path from vertex to another vertex corresponding element i j will become 1; if there is no path the corresponding element will become (0, 0).

(Refer Slide Time: 12:44)



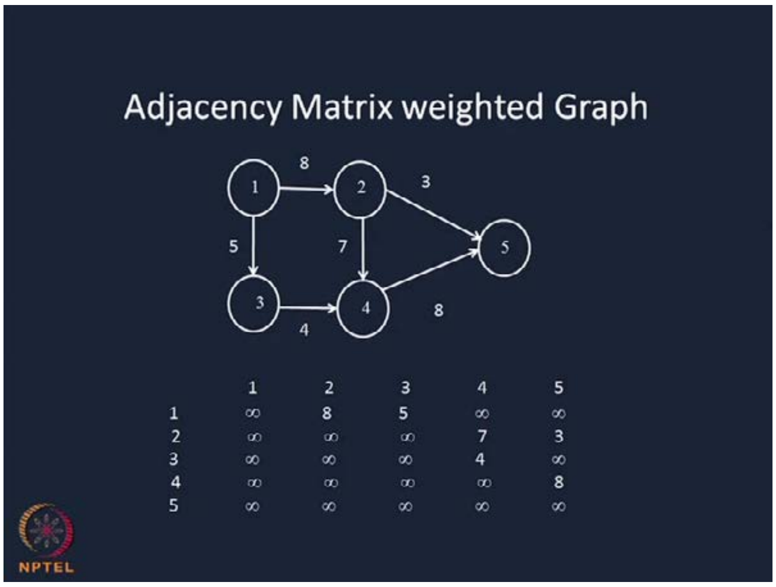
Adjacency Matrix Undirected Graph

	1	2	3	4	5
1	0	1	1	0	0
2	1	0	0	1	1
3	1	0	0	1	0
4	0	1	1	0	1
5	0	1	0	1	0

Next one you see the undirected graph; for the undirected graph also same thing is happening 1 0; 1, 1 is 0; 1, 2, 3, 4 like this way you are doing. The only thing 2, 1 if you see here from 2, 2, 1 also we have made it one because from one to there is no directions.

So, it may be 1, 2; it may be 2, 1 both are same; for this is in from 2 to 1 it is one and like that way other this is the only difference between directed and undirected graph. Because in directed graph if there is path from 1 to 2 only 1 to 2 will be 1 whereas for if for the undirected graph if there is a path from 1 to 2. Then, it will be 1 to 2 will be 1 whereas 2 to 1 also will be 1; the other things will remain the same. So, this is the example of undirected graph.

(Refer Slide Time: 13:39)



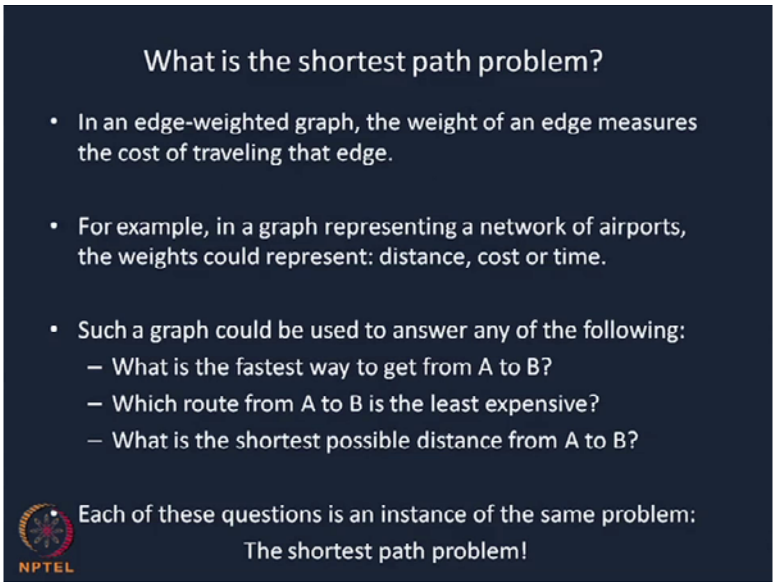
Adjacency Matrix weighted Graph

	1	2	3	4	5
1	∞	8	5	∞	∞
2	∞	∞	∞	7	3
3	∞	∞	∞	4	∞
4	∞	∞	∞	∞	8
5	∞	∞	∞	∞	∞

Now, the next one is the adjacency matrix weighted graph; if you see the weights are given for this one whenever you are putting the weights what we have done if you see 1 to 2; 1 to 1 there is no path. And, we have put and cost of infinity that is and hypothetical value has been assigned infinity instead of 0. Because you have some numeric values over here the weights are non negative in general non negative.

So, therefore we are putting very high value infinity over here when there is no path. So, far whenever for the weighted graph the matrix can be done like this.

(Refer Slide Time: 14:21)



What is the shortest path problem?

- In an edge-weighted graph, the weight of an edge measures the cost of traveling that edge.
- For example, in a graph representing a network of airports, the weights could represent: distance, cost or time.
- Such a graph could be used to answer any of the following:
  - What is the fastest way to get from A to B?
  - Which route from A to B is the least expensive?
  - What is the shortest possible distance from A to B?

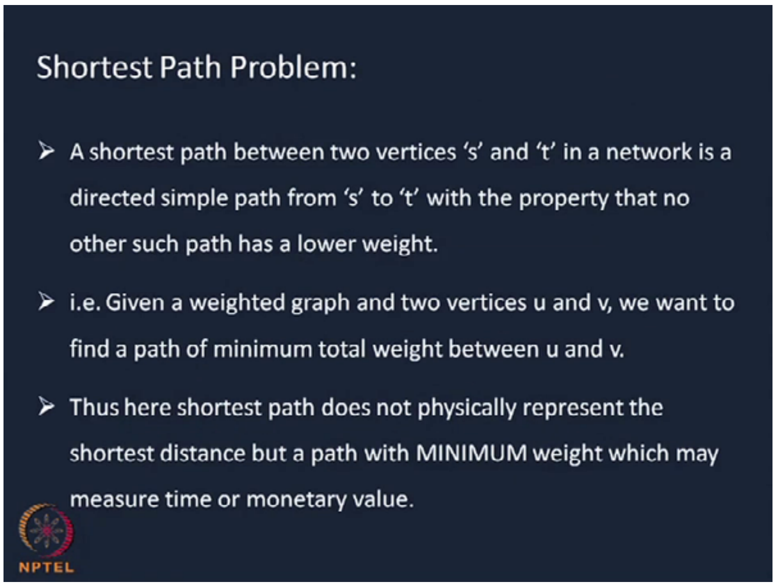
Each of these questions is an instance of the same problem: The shortest path problem!

Next one is what is the; so this are the basic preliminaries you can say that what we require; then what is the shortest path problem? In you already if you see you have done the shortest path problem; that is from starting from one note if you see this note; starting from this note if I want to come back to this note; how to do it? This basically we are telling that you are having a problem something like this or it is something like this also it may be there say it is like this; this you have done in the path C P N. So, this is you are starting position, this is you are ending position; from this position to this position in which path should we move. So, that the cost will be less; basically this is your shortest path.

So, in an edge weighted graph; the weight of an edge measures the cost of travelling that edge; that is whenever you are moving from one vertex to another vertex whatever cost is associated the cost may be in terms of the time; is may be in terms of label, in terms of money whatever it may be that we call it as the weights. For example, in a graph representing the network; a network of airports the weight got should represent it may be distance, it may be cost or it may be time; such a graph can be used to answer any one of the following questions basically. The question same question may be asked in the different way if you see. Then, what is the first question, what is the fastest way to move from a to b?

So, if you see the next question is which route from a to b is the least expensive; and the last one is what is the shortest possible distance from a to b. So, basically so in which way I should move from one path, one route to a to b route; what should be the route to move from a to b such that the cost will be minimum and what should be that path? So, basic problem is this one; for each of this questions we have the solution for the shortest path algorithm.

(Refer Slide Time: 16:37)

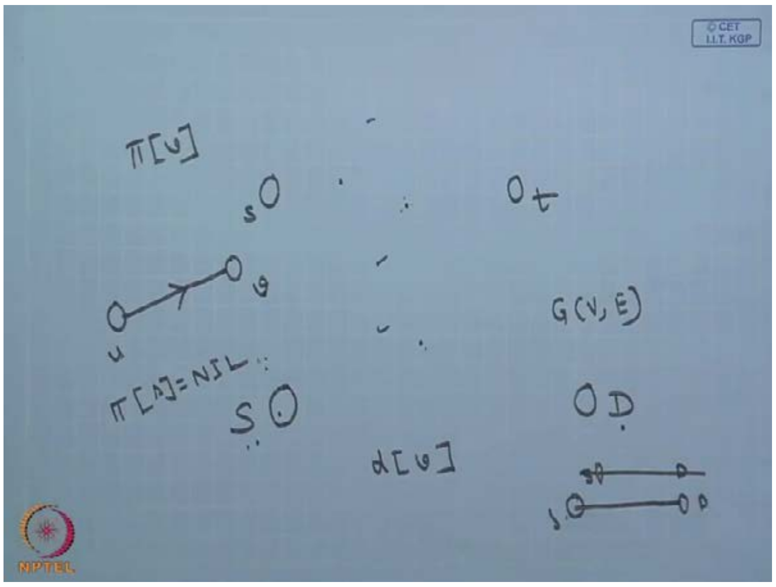


Shortest Path Problem:

- A shortest path between two vertices 's' and 't' in a network is a directed simple path from 's' to 't' with the property that no other such path has a lower weight.
- i.e. Given a weighted graph and two vertices u and v, we want to find a path of minimum total weight between u and v.
- Thus here shortest path does not physically represent the shortest distance but a path with MINIMUM weight which may measure time or monetary value.

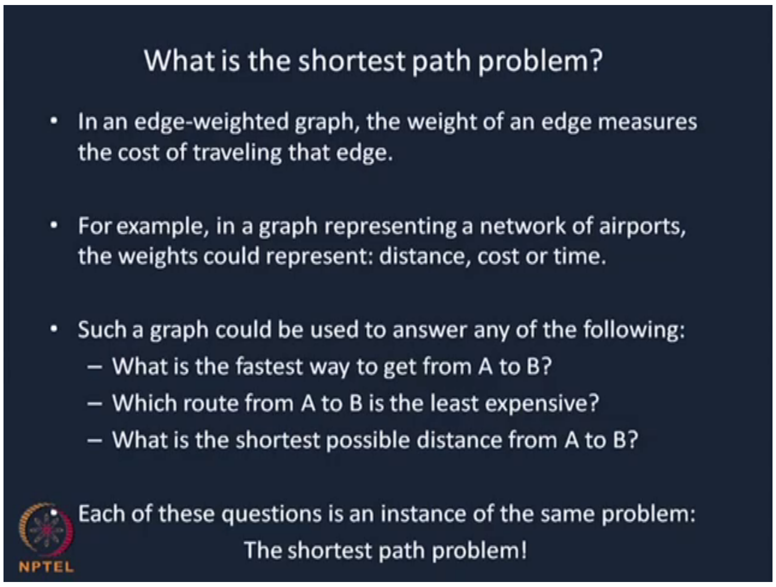
A shortest path between 2 vertices s and t in a network is a directed simple path from s to t with the property; that no other such path has a lower weight please note this one; if you see this one.

(Refer Slide Time: 16:53)



I want to move from this to this in between so many things are there. So, whenever I have want to move from s to t; then what should be the shortest path? So, that I can move from s to t such that there is no other lower weights are available to move from s to t; that is the lowest one that is if you see in a weighted graph and 2 vertices u and v. We want to find a path of minimal time weight between u and v. So, minimal time take between to move from s to t that we are calling as the shortest path. And, you have to note one thing that shortest path does not physically represents the shortest distance. But basically the path consist which gives you the minimum weight please note this one; it is not that the physically shortest distance. But it will be the minimum weights which you are moving along the path; so this is the shortest path.

(Refer Slide Time: 17:53)



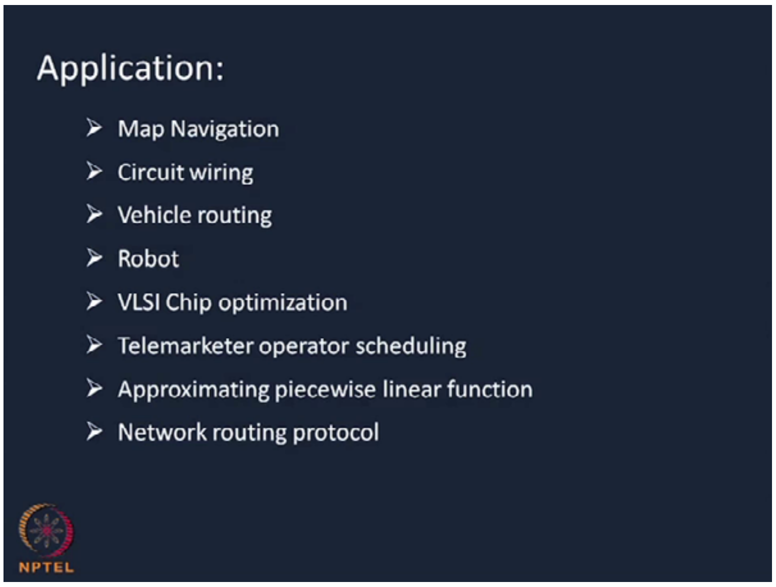
What is the shortest path problem?

- In an edge-weighted graph, the weight of an edge measures the cost of traveling that edge.
- For example, in a graph representing a network of airports, the weights could represent: distance, cost or time.
- Such a graph could be used to answer any of the following:
  - What is the fastest way to get from A to B?
  - Which route from A to B is the least expensive?
  - What is the shortest possible distance from A to B?

Each of these questions is an instance of the same problem: The shortest path problem!

Now, what is an shortest path problem? So, you are shortest path problem is this one.

(Refer Slide Time: 18:06)

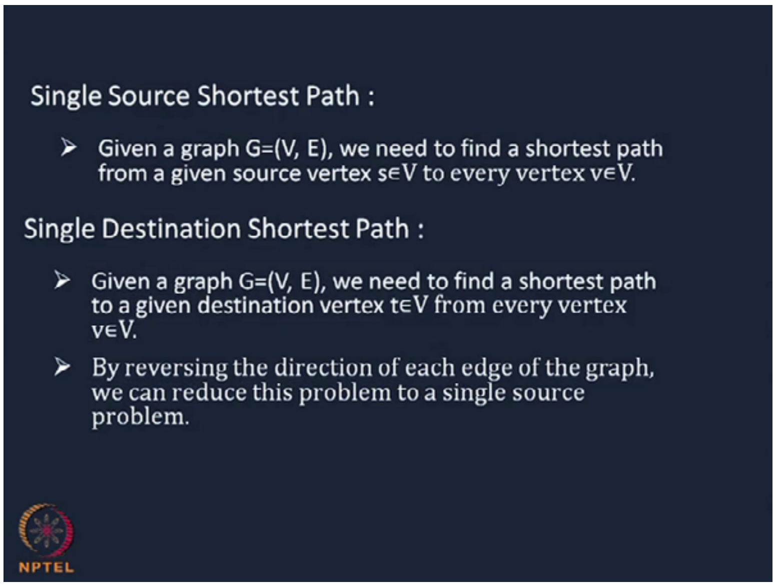


Application:

- Map Navigation
- Circuit wiring
- Vehicle routing
- Robot
- VLSI Chip optimization
- Telemarketer operator scheduling
- Approximating piecewise linear function
- Network routing protocol

What are the applications? The applications if you see you are writing in map navigation; that means whenever you want to move from one place to another place; we can use this shortest path problem. In circuit wiring whenever you are drawing the circuits to move from one circuit to another circuit the vehicle routing; that is you have to drop some materials at different locations of a city; what path it should follow for that one in robots also use in VSLI chip optimization this is use, in telemarketing, telemarketer operator scheduling it is use, approximating piecewise liner function we can use it; network routing protocol also if you want you can use it.

(Refer Slide Time: 18:50)



Single Source Shortest Path :

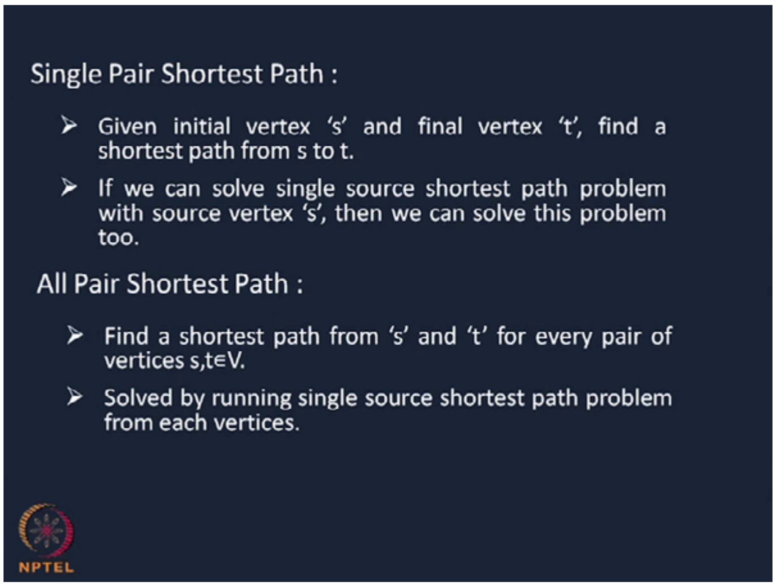
- Given a graph G=(V, E), we need to find a shortest path from a given source vertex s∈V to every vertex v∈V.

Single Destination Shortest Path :

- Given a graph G=(V, E), we need to find a shortest path to a given destination vertex t∈V from every vertex v∈V.
- By reversing the direction of each edge of the graph, we can reduce this problem to a single source problem.

And, now single source shortest path basically whatever we will discuss that is the single source and single destination shortest path. So, single source shortest path means I want to say the path starts from a source only; from only one sou. If you see for this problem if you see the path may start from here, from here, from here, from here, but and it is destination also may be any one of this. But the problem which we want to discuss; basically the problem which we want to discuss here there will be only one source that is this one and there will be one destination that is this. So, please note that for a given graph G (V, E); we want to find out only the shortest path for which there is only one given source S and there is only one given destination say D. So, of course by reversing the direction of this D and S of each graph; we can reduce this problem always to a single source problem.

(Refer Slide Time: 20:09)


Single Pair Shortest Path :

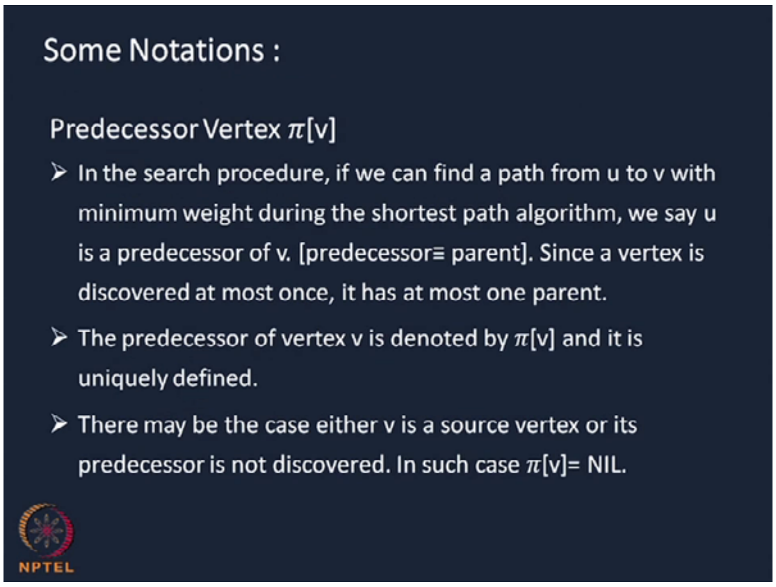
- Given initial vertex 's' and final vertex 't', find a shortest path from s to t.
- If we can solve single source shortest path problem with source vertex 's', then we can solve this problem too.

All Pair Shortest Path :

- Find a shortest path from 's' and 't' for every pair of vertices s,t∈V.
- Solved by running single source shortest path problem from each vertices.

So, we are we want to work with single source, single pair shortest path problem. And, now the single pair shortest path you are having the given initial vertex s and final vertex t; our aim is to find out the shortest path from s to t. If we can solve the single source shortest path problem with source vertex s then we can solve this problem too obviously. Because we if we are starting from an initial vertex and we are moving to the destination then we can solve the shortest path problem. And, it has another property that is all pair shortest path problem. In the shortest path problem not only we want to find out the shortest path between source and destination. Basically we are trying to solve the shortest path between any 2 vertex; whenever you are completing the shortest path algorithm we can develop the shortest path among the source and the destination as well as the shortest path between any 2 vertices. Please note this one shortest path between any 2 vertices which actually we are not done using the C P M method or the path method.

(Refer Slide Time: 21:20)



Some Notations :

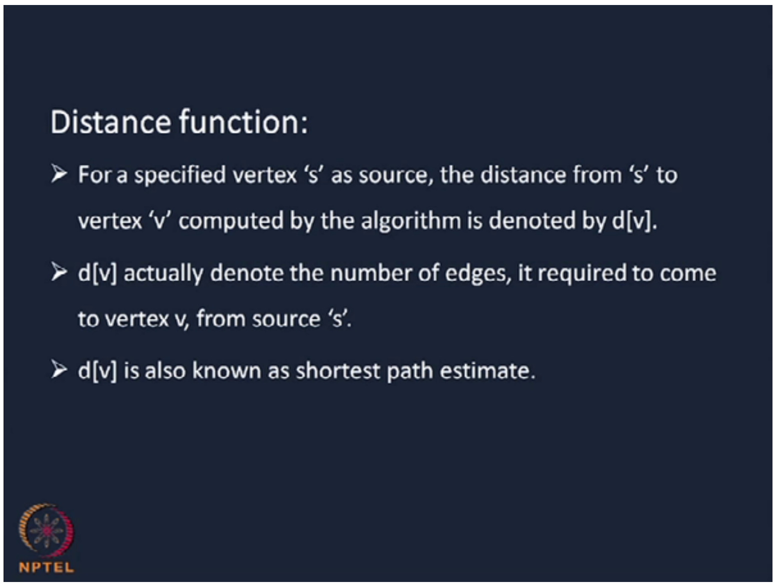
Predecessor Vertex $\pi[v]$

- In the search procedure, if we can find a path from u to v with minimum weight during the shortest path algorithm, we say u is a predecessor of v. [predecessor≡ parent]. Since a vertex is discovered at most once, it has at most one parent.
- The predecessor of vertex v is denoted by $\pi[v]$ and it is uniquely defined.
- There may be the case either v is a source vertex or its predecessor is not discovered. In such case $\pi[v]$= NIL.

So, some notations which I will use actually in the development of the algorithm. The first one is the predecessor vertex pi (v). Predecessor vertex pi (v) is in the search procedure, if we can find a path from u to v with minimum weight; that is you are moving from vertex u to v with minimum weight during the shortest path algorithm. Then, you say that u is the predecessor of v or in other sense if you see if this is your u if this is your v; there is an shortest path between u to v. Then, we say that that the there is one predecessor; u is the predecessor of v or in other sense you can say predecessor is nothing but the parent that is u is the parent of v. So, basically this pi array tells you what are the predecessors of different vertex? For a vertex what is the corresponding predecessor that is being written on this pi array. And, this predecessor is always has to be uniquely defined and there must be there may be a case where v is a source vertex.

So, if a source vertex is there the predecessor of the source vertex is always nil; that is if this is the source vertex say s then pi of s this is equals to nil. So, for the source vertex it is always nil otherwise you are getting if from u to v there is a shortest path. Then, the predecessor of v u is v; we say that the u sorry predecessor u is the predecessor of v; this is one notation.

(Refer Slide Time: 23:17)



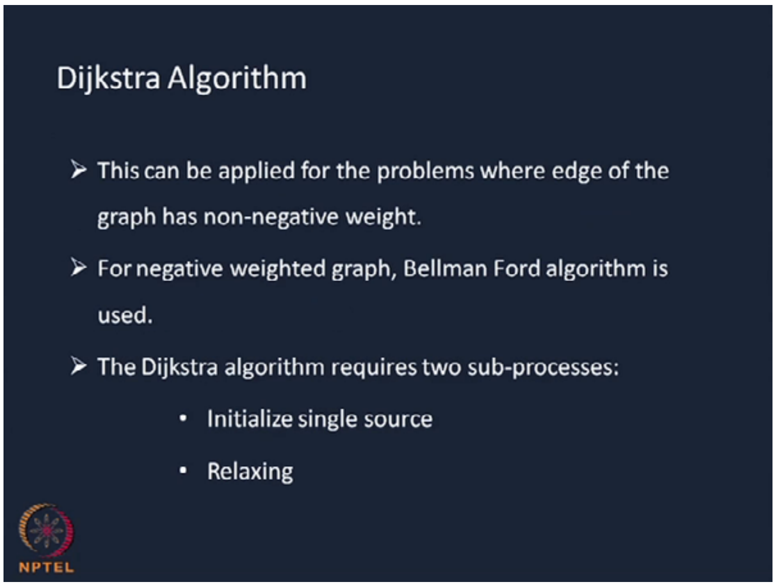
Distance function:

- For a specified vertex 's' as source, the distance from 's' to vertex 'v' computed by the algorithm is denoted by d[v].
- d[v] actually denote the number of edges, it required to come to vertex v, from source 's'.
- d[v] is also known as shortest path estimate.

The other notation is the distance function for a specific vertex s as source; the distance from s to vertex v is computed algorithmically and we denote it by d [v]. So, the distance we denote it by d [v]. So, the distance is between if you have the source s, if you have the destination d this two are there; you have a source, you have a destination. So, what is the distance between s and d that is basically told in this d [v]?

So, basically d [v] denotes your number of edges its requires to come to the vertex v from source s. And, also it is known as the shortest path estimate that is from move from s to v what is the shortest path that also we write over here.

(Refer Slide Time: 24:09)



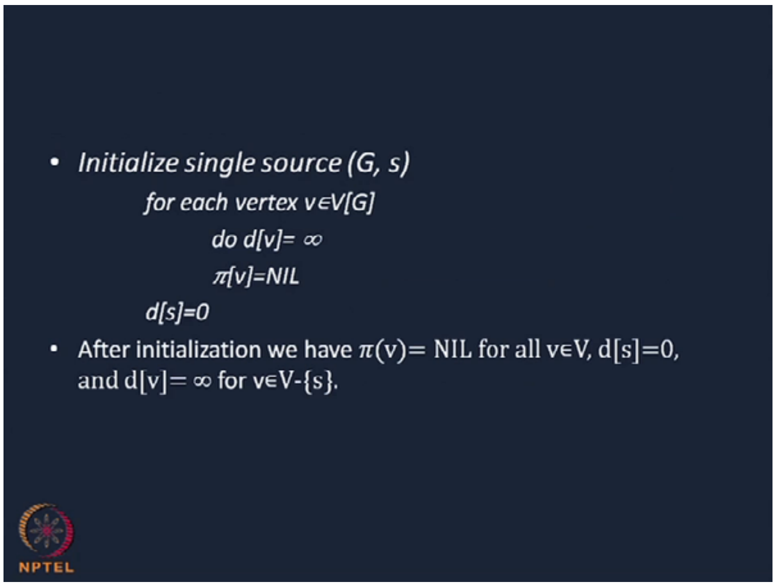
Dijkstra Algorithm

- This can be applied for the problems where edge of the graph has non-negative weight.
- For negative weighted graph, Bellman Ford algorithm is used.
- The Dijkstra algorithm requires two sub-processes:
  - Initialize single source
  - Relaxing

Now, there is a famous algorithm which we call it as Dijkstra algorithm; this Dijkstra algorithm is used will basically discuss this Dijkstra algorithm to find out the solution of the find out the shortest path. This algorithm can be applied for the problems where edge of the graph has the non negative weights number one. For non negative weights the bellman ford algorithm is also available; although we will work with the non negative weights. And, we will discuss only the Dijkstra algorithm because due to shortage of time; we cannot discuss you both the cases that is weighted case and non negative weighted cases we cannot discuss. If you see the Dijkstra algorithm is best the only two things; one is your you initializing the singles source and another one is your relaxing.

So, we have to know what do you mean by we are initializing the source and what do you mean by we are relaxing?

(Refer Slide Time: 25:15)



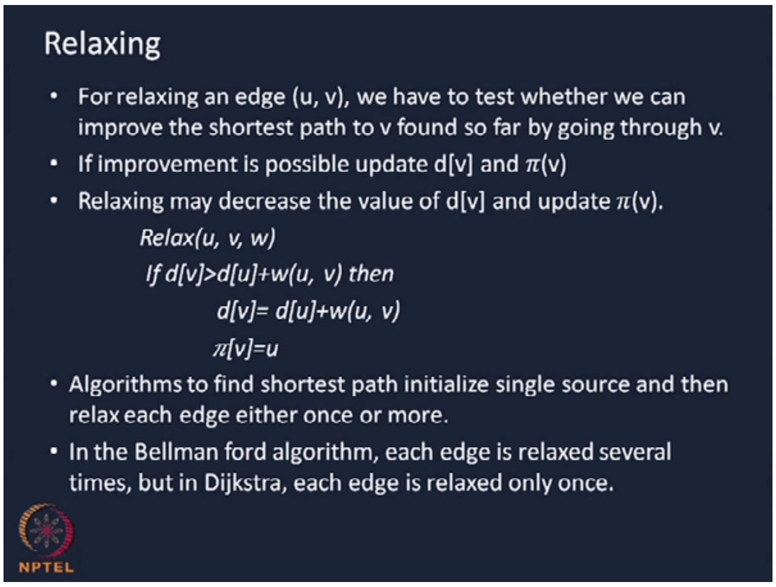
- *Initialize single source (G, s)*

```
    for each vertex $v \in V[G]$
        do $d[v] = \infty$
        $\pi[v] = NIL$
    $d[s] = 0$
```

- After initialization we have $\pi(v)$= NIL for all $v \in V$, $d[s]=0$, and $d[v]=\infty$ for $v \in V$-{s}.

So, what is initializing source? Initializing single source (G, s)if you see what we have written for each vertex v belongs to V [G] you take anything; you this is the initializations single source a graph; for that reason if you see we written (G, s) that is it has a single source for each vertex v what you will do? You will make the distance vertex v equals infinity and your pi v what is your p i (v) if you remember we have told in the just this one initialization. For initialization we are using pi whereas for the other one we are using the distance we are using v.

(Refer Slide Time: 26:06)



Relaxing

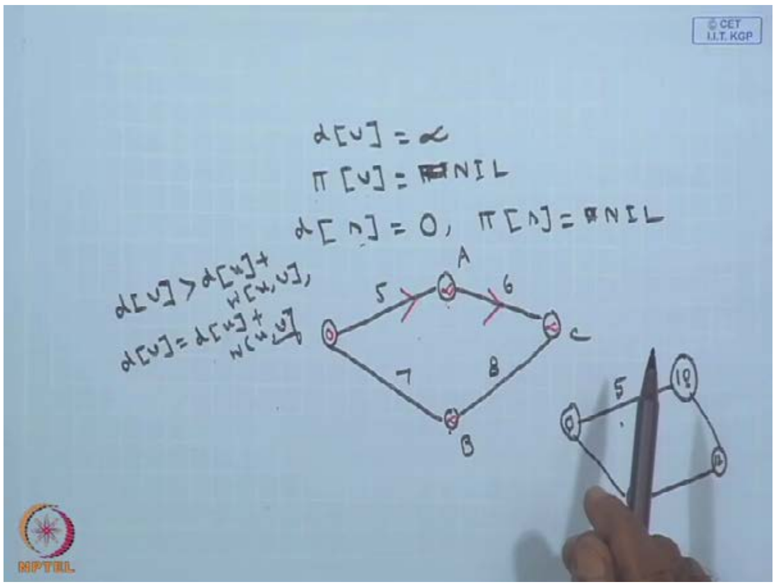
- For relaxing an edge (u, v), we have to test whether we can improve the shortest path to v found so far by going through v.
- If improvement is possible update d[v] and $\pi(v)$
- Relaxing may decrease the value of d[v] and update $\pi(v)$.

```
Relax(u, v, w)
 If d[v]>d[u]+w(u, v) then
     d[v]= d[u]+w(u, v)
     π[v]=u
```

- Algorithms to find shortest path initialize single source and then relax each edge either once or more.
- In the Bellman ford algorithm, each edge is relaxed several times, but in Dijkstra, each edge is relaxed only once.

So, pi v your d v is just your predecessor vector you are storing in pi and whereas the distance vector you are storing in v. So, here what you are doing for the each vertex v belongs to V [G] what you have to do? You have to make the d that is d [v] equals infinity.

(Refer Slide Time: 26:29)



You are making d v that is equals to infinity and initially you are making this pi (v) this is equals to nil. And, pi (v) equals to nil whereas if s is the source then your d s value that will be equals to 0; to make it simple for you suppose you have a graph like this I am just drawing the graph very easy. We have a graph like this the nodes we are defining as say o, a, b, c and the distance suppose it is 5, 6, 7 and 8.

So, basically what you are doing initially the value is if you see initially we are writing this value as vertex value as 0 over here. And, all other vertex values will be infinity; this we call as the initialization this you are making as 0. And, all this elements all these vertex values you are making as infinity; that we have written in the algorithm if you see the algorithm over here d [v] equals to infinity. And, your pi (v) is nil whereas for the source d s equals to 0. And, similarly after initialization your pi (s) that is the predecessor of source there will be nothing. So, predecessor of source always you will write down as nil.

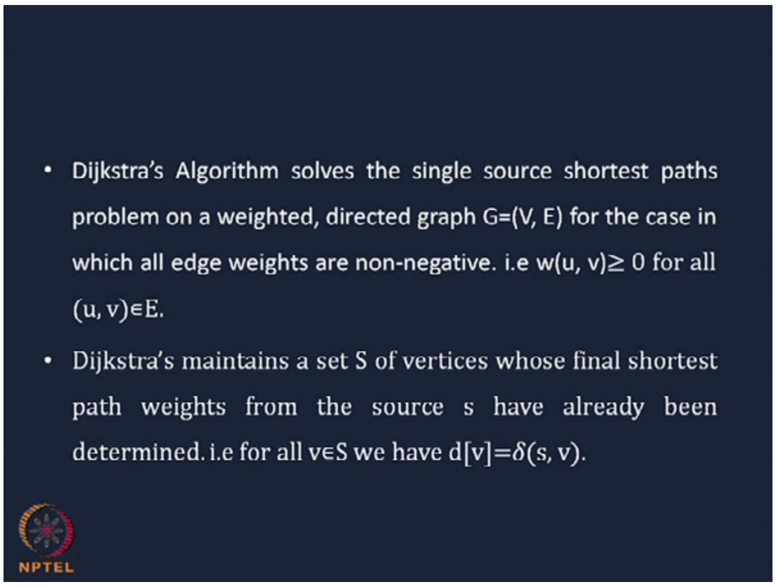
So, we have to note this one that the distance of source is always 0 initially. And, the predecessor of source is nil whereas the distance of any other vertex we will initialize with infinity; and your the predecessor of each of them will be equals to nil. Because we have not evaluated that. The next one is the procedure that is the relaxing procedure; for relaxing and edge u to v we have to test whether we can improve the shortest path whatever we are found that one or not; whatever shortest path we found that is correct or not. So, if there is a already you have obtain a path from u to v that is if you see in this figure.

Suppose I have derived a path like this let me denote it by this red color pen this is a path. So, I have to basically there is a path and if you see the distance will be 5 plus 6; 11 can I find out any other alternative path by which the distance will be less. And, that actually we find out by this relaxing method; the basic idea is this one. So, in the relaxing method what we are doing here if the we are trying to we can improve the shortest path from u to v. So, that by going through v by improving if improvement is possible; then you has to update the value of d [v] and pi (v).

And, relaxing may be it may decrease the value of d [v] and update pi v this is basically done by this small algorithm; if you see the relax (u, v, w) this is the algorithm if d [v] is greater than d [u] plus w [u v]. Then, d u will be equals to the smaller value that is d [u] plus w u v and your pi u v equals to so let me just write down what I am saying that is if d [v] it is greater than d [u] plus w [u v] if this is true in that case what you will do you will replace d [v] by the smaller value d [u] plus w [u v]. So, what does it actually means the meaning is again I am just drawing this 2 parts; 3, 4 parts. If this is the initial value suppose this value is 10, this value is 5, this value is 12 from here this value is 7 say some links are there; I am not going to this links what I want to say your d (v) is this value 10 by some path can I get some value such that if you add with this with this whether the value is less than this or not. If it is less than this one in that case your d [v] will be replace by d [u] plus w [u v].

So, this gain I will explain with an example. So, this is the first one relaxing one and algorithms for finding out the shortest path will initialize the source. And, then relax edges is either ones or more that means by relaxing you can initializing in single source. And, you can relax each edge with ones or more that is you can reduce the value of each edge more than ones also. And, in bellman's algorithm sorry in Dijkstra algorithm your relaxing only once whereas in a bellman's algorithm relax its several times. This is the basic difference between the Dijkstra algorithm and the bellman ford algorithm. Next if you see the next is the Dijkstra algorithm solves.

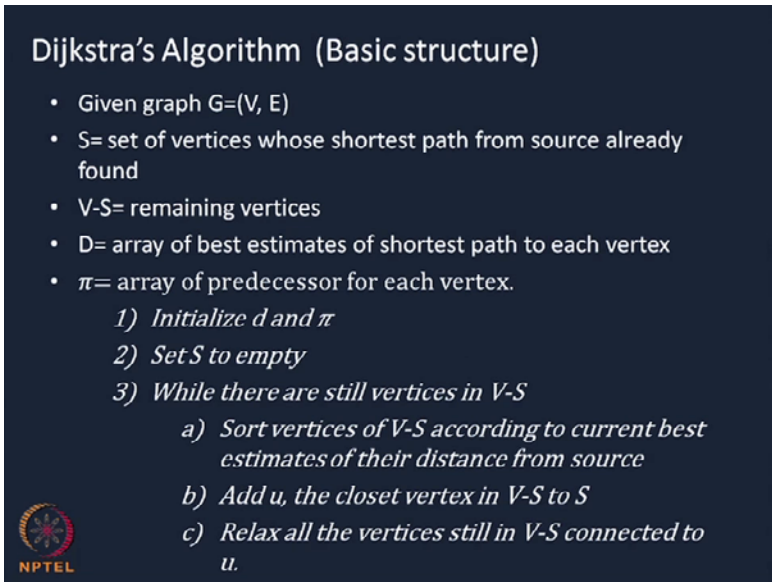
(Refer Slide Time: 32:01)



- Dijkstra's Algorithm solves the single source shortest paths problem on a weighted, directed graph G=(V, E) for the case in which all edge weights are non-negative. i.e w(u, v)≥ 0 for all (u, v)∈E.
- Dijkstra's maintains a set S of vertices whose final shortest path weights from the source s have already been determined. i.e for all v∈S we have d[v]=δ(s, v).

This single source shortest path problem as on a weighted G equals (V, E) for the cases in which all edges are weighted and they are non negative that is w (u, v) greater than equals 0. And, Dijkstra algorithm maintains a set of set S of vertices whose final shortest path weights from a source to a they source S have already been determined; that is we are having d [v] equals gamma (s, v) this already we are doing.

(Refer Slide Time: 32:33)



Dijkstra's Algorithm (Basic structure)

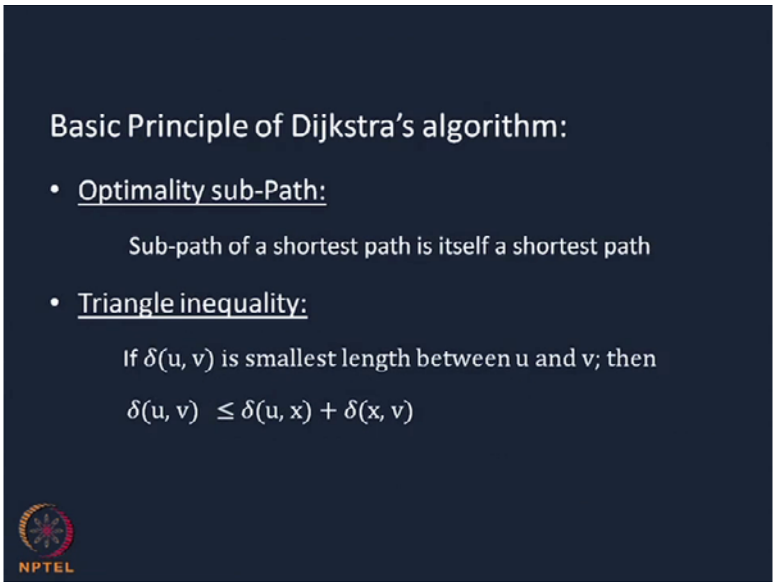
- Given graph G=(V, E)
- S= set of vertices whose shortest path from source already found
- V-S= remaining vertices
- D= array of best estimates of shortest path to each vertex
- π= array of predecessor for each vertex.

```
1) Initialize d and π
2) Set S to empty
3) While there are still vertices in V-S
    a) Sort vertices of V-S according to current best
       estimates of their distance from source
    b) Add u, the closet vertex in V-S to S
    c) Relax all the vertices still in V-S connected to
       u.
```

What is the basic structure of the Dijkstra algorithm? You are given a graph G equals (V, E); S is the set of vertices whose shortest path you have to find out. And, V to S are the remaining V minus S; V is the all the vertices S is the set of vertices whose shortest path will be finding V minus S obviously will be the remaining number of vertices your D as we have told basically it is the small d array of best estimate of shortest path of each vertex this is small d not D. And, pi this is the array of predecessor of each vertex which we have define earlier.

So, initially what you are doing I have given 1, 2, 3 like this you have to initialize your distance and the predecessor; that is you are finding what is initialize d and predecessor? Then, the set S which contains all the elements that is initially empty; while there is some still vertices in V minus S you have to short vertices V minus S according to current best estimate of their distance from the source; that is whatever remaining vertices are there what is the minimum distance that you have find out. Then, you have to add you the closet vertex in V minus S to S and we will relax the all the vertices still V minus S is connected to u.

(Refer Slide Time: 34:11)



Basic Principle of Dijkstra's algorithm:

- Optimality sub-Path:

  Sub-path of a shortest path is itself a shortest path

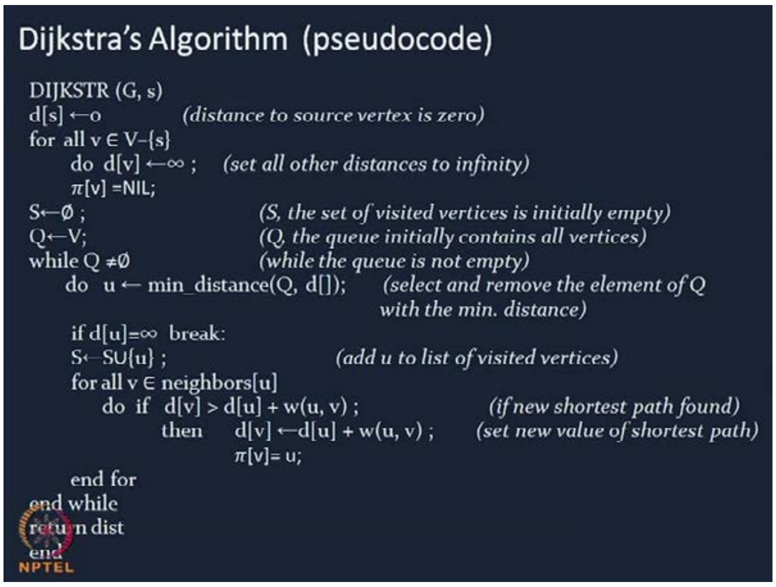
- Triangle inequality:

  If $\delta(u, v)$ is smallest length between u and v; then

  $\delta(u, v) \leq \delta(u, x) + \delta(x, v)$

I the basic property what we are using basically one is we are trying to find out two things; one is the optimal sub path. And, another ones is the triangular in equality this are the basic principle is used; your sub path of a shortest path is itself a shortest path. This is the basic principle any sub path you take of a shortest path always that must be the shortest path. And, the if you are having a small length between u and v; then always you take any other 2 points that will always remain the shortest. And, this we call as the triangular inequality.

(Refer Slide Time: 34:50)

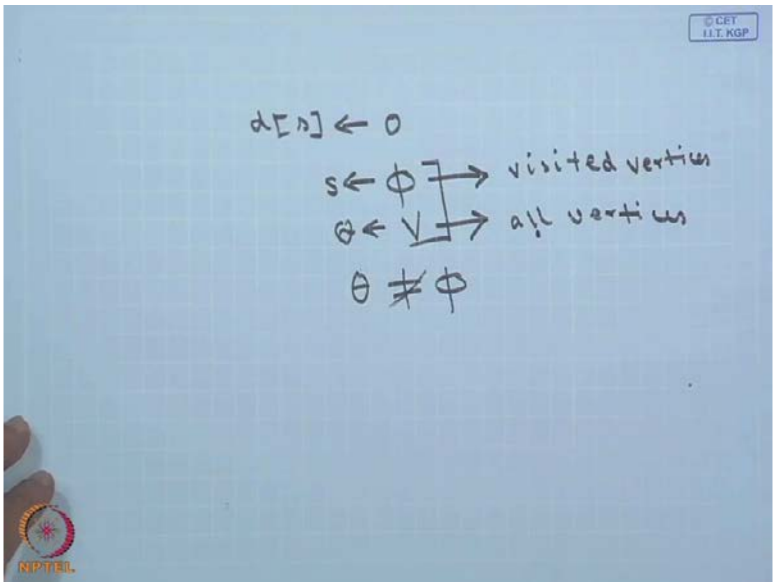


Dijkstra's Algorithm (pseudocode)

```
DIJKSTR (G, s)
d[s] ←o              (distance to source vertex is zero)
for all v ∈ V–{s}
      do d[v] ←∞ ;    (set all other distances to infinity)
      π[v] =NIL;
S←∅ ;                          (S, the set of visited vertices is initially empty)
Q←V;                           (Q, the queue initially contains all vertices)
while Q ≠∅                     (while the queue is not empty)
      do  u ← min_distance(Q, d[]);     (select and remove the element of Q
                                          with the min. distance)
      if d[u]=∞  break:
      S←SU{u} ;                          (add u to list of visited vertices)
      for all v ∈ neighbors[u]
            do  if  d[v] > d[u] + w(u, v) ;           (if new shortest path found)
                  then     d[v] ←d[u] + w(u, v) ;     (set new value of shortest path)
                           π[v]= u;
      end for
end while
return dist
end
```

Now, the Dijkstra algorithm I have just rewritten in terms of the algorithm here; which we have written Dijkstra (G, s) that is a single source. Initial line if you see you are making d [s] that is equals 0 actually that is not properly visible.

(Refer Slide Time: 35:07)



$d[s] \leftarrow 0$

$S \leftarrow \phi \rightarrow$ visited vertices

$Q \leftarrow V \rightarrow$ all vertices

$Q \neq \phi$

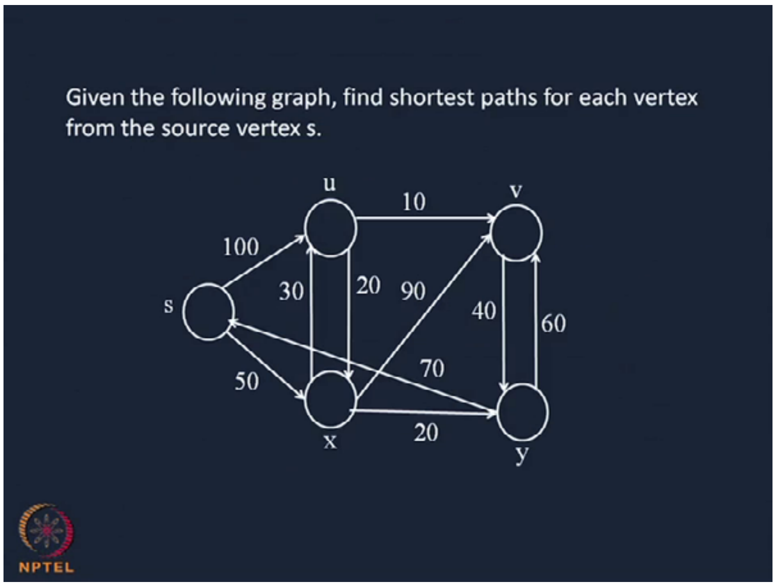
If you see d [s] this you are making initially 0; next is for a all v belongs to V minus s you are making d [v] is you are making as infinity pi (v) equals nil. And, your s is approaching 0; S is null this 3 things that is if you see the slide for all do d [v] equals infinity pi (v) equals nil. And, your S is 0; S is null and your Q is V this 2 are important I will discuss; please to go the slide here.

So, this 3 parts do d [v] equals infinity pi (v) equals nil this part basically you are doing as the initialization as with discussed it has 2 basic parts; one is the initialization and another one is the your one is initialization and another one is the relaxing. So, the first part we have done that is the this one what is S? S is the set of vertices set of visited vertices; please note this one set of visited vertices; how many vertices already we have visited. And, obviously this will be initially empty because we have not visited anything. So, S is the number of visited vertices and you Q initially it contains all the vertices initially Q is all vertices.

And, so what I will do basically initially S is empty your Q is all the vertices once I am visiting vertex; then that will be moved to S and that will be deleted from this. And, ultimately our aim is we will repeat our process; relaxing process until your Q is not equals phi; for this condition we will move from there. So, let us see. So, the next part is while Q not equals phi in the slide if you see; the next one is while Q not equals phi do u equals minimum distance of (Q, d) that is you have to calculate; what is the minimum distance? Then, if d u equals infinity will break otherwise S equals S union u. And, then for all v vertices belongs to neighbors of u do if d [v] greater than d [u] plus w [u, v]. Then, d [v] will be d [u] plus w [u v] and pi (v) equals u again this part if you see the last part is basically the relaxing part.

So, the it is consisting basically of 2 parts; one is the initialization part, another one is the relaxing part. In the initialization part your initializing 2 things; one is the distance array and another ones the predecessor array. And, then in the relaxing part your revisiting all the vertices and if possible you are trying to reduce the cost. And, so this 2 together will give you the shortest path. Now, let us take one example and how it works.

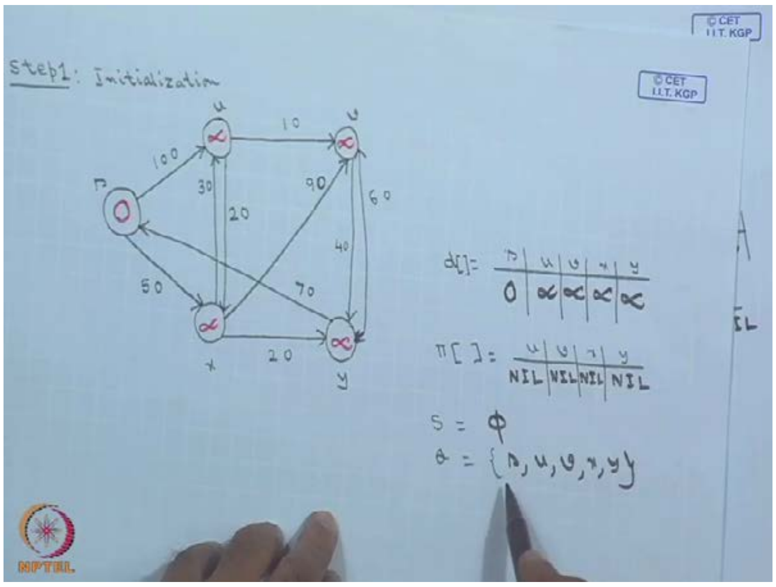
(Refer Slide Time: 38:27)



Given the following graph, find shortest paths for each vertex from the source vertex s.

So, let us check this example in the flowing graph if you see we are having s. And, from there you want to visit the single source problem; this is not the single destination problem of course. Because you may visit to any vertices but the source is same. So, it is if you see from the graph it is directed graph number 1; number 2 is the it is one after directed graph it is also weighted graph weights are given. So, let us see what happens in the next steps. Let us see here.

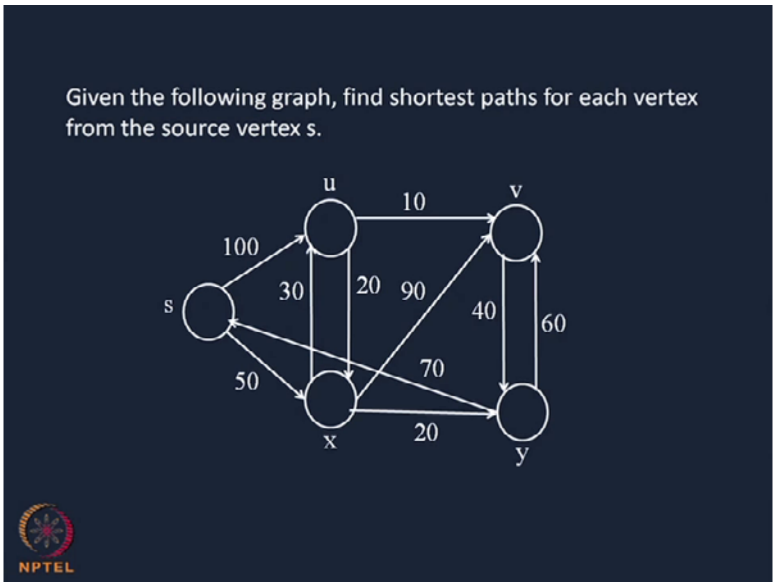
(Refer Slide Time: 39:07)



So, step one is here initialization problem; step one is the initialization. In the initially what is happening you are having this thing; the graph if you see the graph the graph I have drawn over here whatever graph problem was there that is s to u, u to v like this way all the. And, all the weights are also given over here like 100, 10, 30, 20 like this. Now, let us see the problem. In the problem itself you are having the initialization problem that is s to u, u to v what I have told you we have to find out what is your d; first what is the distance and what is your pi? So, your distance obviously it will be s is 0 as I told you since it is source. And, for u all are infinity this is the basic thin. In the initialization process the distance of all vertex which will be infinite and distance of s is 0 whereas whenever your calculating your pi we have not written source over here. Because for source nothing is coming source is already here. And, for all others the predecessors obviously of u, v and all these are nil the predecessors of this one are nil.

So, once I have written this I have to calculate this matrix; once I have calculated this matrix. Then, this distance values you put it here distance of source s is 0 you write down here 0 distance. And, all others basically they are infinity, infinity x is this and this is infinity. So, you are I have written it rates so that you can understand it properly. So, you are calculating first what is the distance array; from the distance array on each vertex you are writing now the values. For the initialization you know source of a distance is 0 whereas source of other vertices infinity. And, you have written all this whereas for the predecessors for all of them predecessors initially is nil. So, initially your S is phi and your Q is all the vertices that is s, u, v, x and y; I think the first step is clear that your S is phi and your q is s, u, v, x and y.

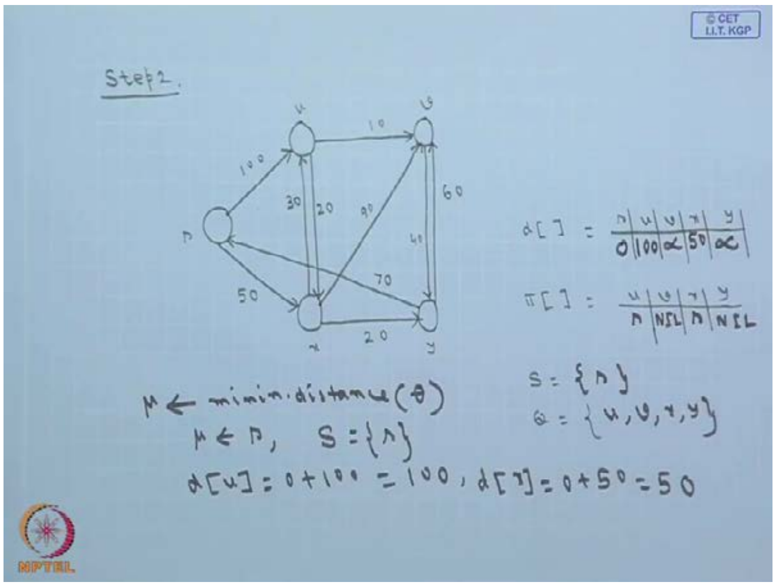
(Refer Slide Time: 41:49)



Given the following graph, find shortest paths for each vertex from the source vertex s.

Now, let us come to the step 2.

(Refer Slide Time: 41:57)



Step2.

d[ ] =

s	u	v	x	y
0	100	∝	50	∝

π[ ] :

u	v	x	y
s	NIL	s	NIL

$S = \{s\}$

$Q = \{u, v, x, y\}$

$\mu \leftarrow$ minim. distance(Q)

$\mu \leftarrow s, \quad S = \{s\}$

$d[u] = 0 + 100 = 100, \; d[x] = 0 + 50 = 50$

For the step 2 basically again you are having this thing; you are having this one you have to first calculate what is the distance, what is the pi? And, then from there you will calculate the this one. So, for calculation of this what is your mu say I am just writing mu. What is your mu? Mu is the minimum distance of Q that is what do you mean by minimum distance of Q? Minimum distance of Q means I want to say which vertex is having lowest value over here; you see the earlier one graph in this these vertex you have supplied the values which vertex is having the lowest value obviously the source will have the lowest value; the source value is 0. So, therefore your mu is here. Since, the source value if you see it is 0 the this sis the minimum value therefore your mu is actually it is s. And, your S basically now it will become s I will write down that your S is basically s.

So, once I have obtain what is the distance? Now, from this neighbor what is mu; you are calculating which are having the lowest value of the vertex that vertex you choose. Now, from that vertex you find out what is the neighbor of this vertexes? The neighbors of this vertex one neighbor is u and another neighbor is s. Neighbor means there is directed graph from s to u or s to v not the opposite way; if there is direction from s to u your telling ((Refer Time: 43:42)) u is neighbor. So, neighbor of s is u and the neighbor of s is x. So, now what I have to do? I have to find out the distance of you and the distance of x; what is the distance of u? Distance of u is the value of the vertex s plus this weight whatever you have given; value of this plus the weight from moving from s to u. Therefore, your s u if you see d u that is distance of u basically it will be the value of the source s value of the vertex s that is 0 plus weight from moving from s to u that is 100.

So, you can write down from here that your d [u] is equals to 0 plus 100. So, this is equals to 100. And, on the same way I have to calculate what is d x; what will be the d x? d x is what was your predecessor; predecessor is s what is the value of the vertex says vertex says is 0 plus the weight of this path x to s that is 50. Therefore, d x will be equals to 0 plus 50 and that is equals 50 So, you can write down d (x) that is equals 0 plus 50. So, this is equals 50. So, now you got a change over here the value of d u the distance of u, the distance of x are not now infinite. So, it is now fixed. So, now what are the values let me write down over here.

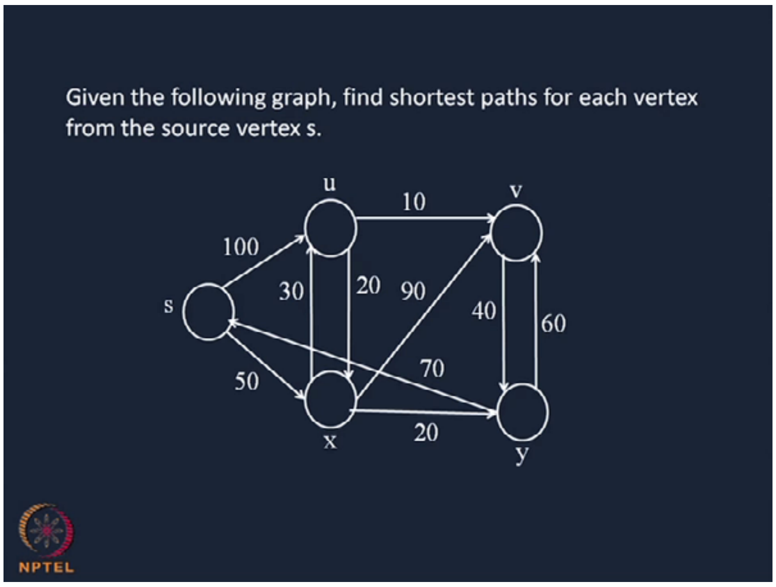
So, for this let me just calculate first the distance is what from s to x it is 0 obviously; for distance of u is now 100 just now we calculated distance of x; that is equals to distance of x is 50, v is infinite and y is infinite what about the predecessor? Predecessor which the predecessor of u now? Predecessor of u is x, predecessor of x is s, predecessor of u is a x, predecessor of x is s. So, change it predecessor of u is s predecessor of x is s for v and y it is nil.

So, once you have drawn this your s is this thing. So, from Q what will happen earlier your Q was {s, u, v, x, y}. Now, already you have visited s and that has come to S. Therefore, you should delete this s from Q; so that now you will get Q equals u, v, x and y. So, once I have written this now write down the distances for this it is 0 still; for the next one it is 100; for this one it is 50 and for this is it is infinite; and for this it is infinite still. Now, whenever so like this way first you are calculating the distance no first you are finding out whatever vertex are there which are as the minimum value; the minimum value obviously 0 among all this vertex values.

So, you are finding what is the neighbor of this vertex; after you finding the neighbor you are finding the distance of those neighbor using this formula; that is value of that vertex plus the weight for going to from vertexes to u like that. So, once I have done this you are filling up the array d distance array, you are filling up the predecessor array for u; from that figure it is clear predecessor of u is s, predecessor of x is s we have written it. Accordingly the set S and Q your operating over there and then you are writing the distances over there. If you see once I have obtain this from s to u this is the shortest path; this sub path is the shortest path for moving from s to u you cannot minimize this distance value 100 in any way. If I want to move from s to u in by any means I cannot minimize or I cannot make it lesser than 100. So, therefore what I am doing I am putting an array over here.

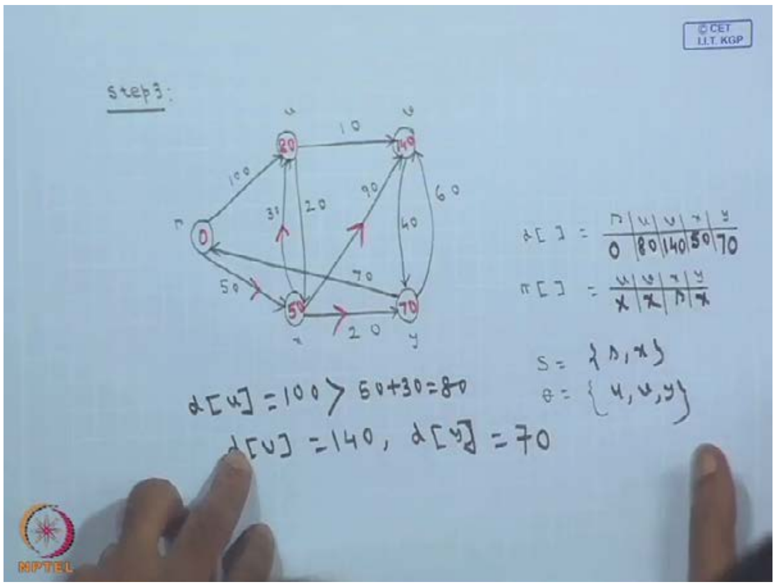
Similarly, to move from s to x this sub path also gives you the minimum distance; you cannot minimize for that. So, the sub paths which I will not be able to minimize; that initially I am putting by this red arrow. So, this is your step 2. Now, let us go the step 3.

(Refer Slide Time: 48:35)


Given the following graph, find shortest paths for each vertex from the source vertex s.

Now, in step 3.

(Refer Slide Time: 48:39)



d[ ] =

s	u	v	x	y
0	80	140	50	70

π[ ] =

u	v	x	y
x	x	s	x

S = {s, x}

Q = {u, v, y}

d[u] = 100 > 50+30 = 80

d[v] = 140, d[y] = 70

Again you are having now in step 3 what I have to do first; this was the figure for the step 2. So, from this figure first I have to find out which vertex is having minimum; now from which vertex you will check? You will check from only the vertex u, v, x and y; you do not have the vertex S now. So, basically I will go through the vertex whatever is there in the set Q; because after each step for the vertex it will be one vertex will be removed. So, among the vertex u, v, x and y if you see the vertex x has the minimum value that is 50; the vertex says at the minimum value what are the neighbors? The neighbor is one is u this arrow is there; one arrow is to their v, another arrow is there y.

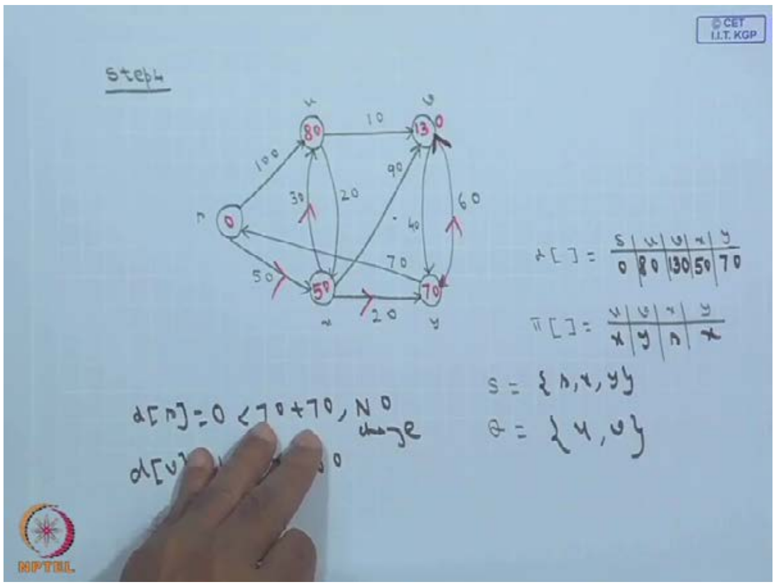
So, from this remaining vertex u, v, x, y we are finding minimum distance is there for the vertex x that is 50 and the adjacent vertex of x are u, v, x and y; what is the distance now we have to calculate? The distance say d [u] if you see what will be d [u] now; d [u] will be the distance of the vertex x plus this one; that is d [u] will be equals to your d [u] this is equals I am just showing one you can calculate afterwards; d [u] will be vertex this plus d [u] initially this is 100; initial value is 100 if you see. And, what is the distance now 50 plus 30.

So, 50 plus 30this is equals 80; so 100 is greater than this. So, therefore I have to now update this d [u] from 100 to 50 plus 30; 80 what about d [v]? Your d [v] will be d [v] if you calculate 50 plus 90 plus 140 you d [v] will be 50 plus 90 that is equals to 140. Similarly, your d [y] this will be d [y] will be 50 plus 20; 70 this is also will be replaced because in both cases alpha are greater than. So, what are the distances you are getting from s to s it is 0; from u now earlier if you see it was 100 now it is replaced by 80; from v it is 140; we have calculated from x it remains fixed that is 50 and for y it is 70; for y if you see it is it is 70.

Similarly, for now you see the what are the predecessor for u? The predecessor is now it is not the 80 is there 80 coming from through this path. So, the predecessor of u has changed now x. Similarly, for v also the predecessor is x; for x the predecessor is obviously s and for y also the predecessor is x. So, you if you know write down the values now it is 0; it is eighty this one is 50, this one is 140 and the last one is 70. So, now the if you is the arrows will be changing; this is the minimum value to move from here to I will not go this direction. If you go this direction the value is 100 whereas if you move from this to this the distance is 80. So, one path will be this to go to this path 50 plus 90 140 and then the 50 plus 20 70. So, this is the s to u.

So, basically whenever you are moving the steps the shortest path is being change changing; which we are showing by the arrows over here. Now, come to the on the same way if you calculate for the step 4.

(Refer Slide Time: 52:33)



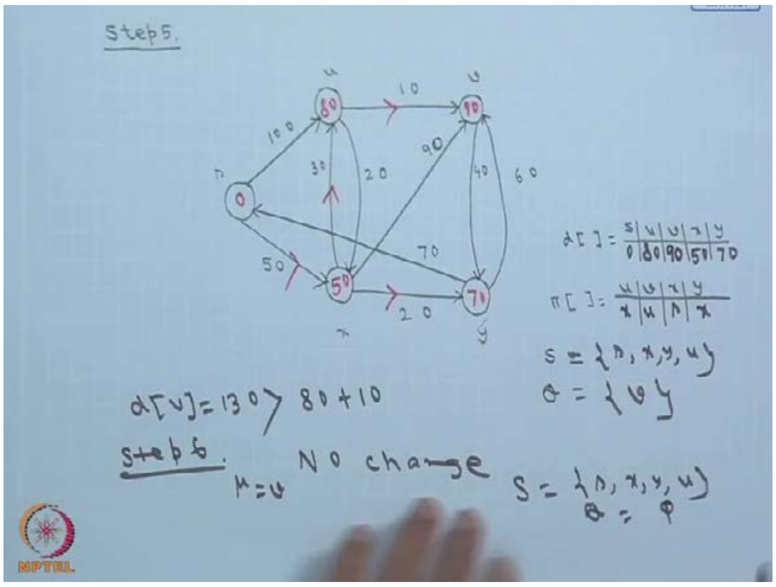
Step 4
d[ ] = s u v x y / 0 80 130 50 70
π[ ] = u v x y / x y s x
S = {s, x, y}
Q = {u, v}
d[s] = 0 < 70 + 70, No change
d[v]

Now, for the step 4 if you see over here sorry I have not written your minimum of s your S is here s and x whereas Q is u, v and y. So, now basically you will check between u, v and y what is the minimum one? The minimum value is here your y and for y if you see the vertexes s and v these are the neighbors. So, you will calculate d [s] and d [v]; your d [s] is equals to I am just writing from here d [s] initially is 0 and it is less than 70 plus 70.

So, basically there will be no change whereas your d [v] this was 140; d [v] is 140 and what d [y] d y is your 70 plus 60; so 70 plus 60. So, 140 is greater than this. So, your d [v] will be changed. So, v distance now will become 130; all other distances will remain as it is that is 0, 80, x is 50, y is 70 for u it is again if you see it is x; for v now it has changed to y, for x again it is s and for y it is for y the earlier one that is x. So, your S now will be s, x and y; y will be removed and because the your taking y was the initial value minimum value. So, q is {u, v}. So, the values will become now this is 80 remains fixed; this is 50, this is 70 and this one is 13 and 0; so this value as reduced. So, this path will remain as it is this whereas the path; this path was there if you see if you move from this to this the distance was 140; now distance is 130.

So, now this sub path is not the shortest one sorry I have to write down the new shortest path; in new shortest path will become 20 and this is 70 and 60 that is 130. So, arrow actually will be on this direction this is on this direction. So, this is this basically I wrongly wrote. So, this is 70 plus 60 that is equals to 130. So, this is the shortest path over here.

(Refer Slide Time: 55:11)



Now, let us go to the step 5; in step 5 what happens if you see in step 5 what happens you are having here only u and v. So, between u and v which one is the minimum; obviously again u is the minimum 80 and 130 and who is the adjacent; adjacent of u and v; one is u is x, another one is v. And, if you see adjacent of u is v that value will be more.

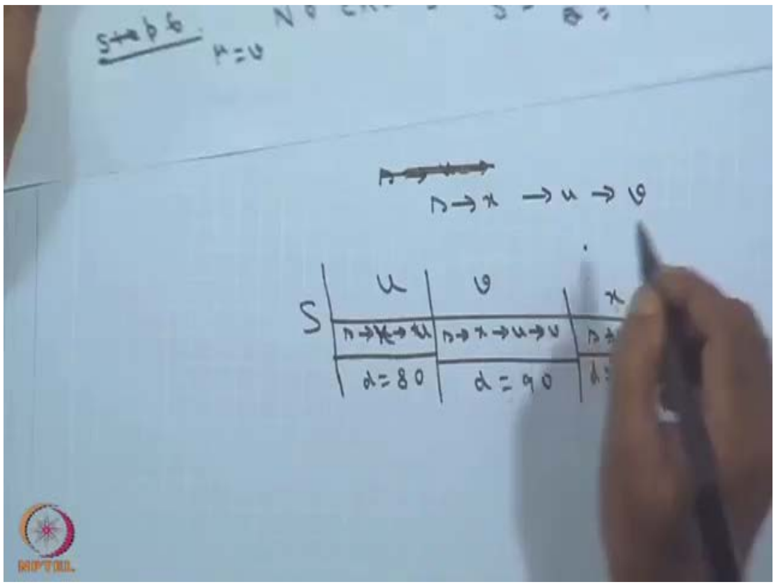
So, it will not be it will x will remain unchanged over here. Because you are not able to reduce it this is more 80 plus 30 whereas for this case if you see this 130 your d [v] what is your d [v] now; d [v] initially if you see your d [v] is 130. But if you calculate this will be distance is 80 plus 10 that is 90. So, 80 plus 10 which is greater than this. So, this distance 80 plus 10 this is 90. So, if I travel from here to here then my distance will be less; so this 130 has to be changed. So, since in that case your value of distance of v will be changing to 90 and all other values will remain as it is that is 0, 80, 50 and 70 whereas for u it is x, for v it is now u, for x it is x, for y it is x.

So, you see your S will now become a s, x, y and u your Q is v. What is the value over here then it is 0, it is 80, this is 50, this is 70, this is 90. So, now the arrow will be one is 50 plus 30; 80 plus 10; 90 to move on this direction and you can move on this direction also. So, 20 and now sorry if you move from here to here it is 50 plus 20this sub path shortest. But now this sub path is not shortest because this sub path was having 130 distance 70 plus 60. So, this is not a shortest distance.

So, earlier someone may be shortest path now it may not be the shortest path. So, now this is the figure over here; this in the step 7 what is happening you are in step 7. Now, if you see you are having only one case that is only v is remaining. So, what is the neighbor of v? Neighbor of v is only y. And, in that case the value will be if you see over here itself it is in that case 90 plus 40 130 which is greater than 70. So, this 70 will remain unchanged or in other sense in step 6; in step 6 there is no change we can say that in step 6 there is no change. So, since there is no change over here in step 6 because the in step 6 there was adjacent was only mu equals v; only one vertex was there, adjacent was y and the was cost more 90 plus 40. So, there will be no change.

So, this is the final graph because now your Q in this case your S will be now in step 6 your s will be x y and u and your Q will be equals to null. So, we are stopping the algorithm now; once we are stopping the algorithm so now you can say from here what will be the path?

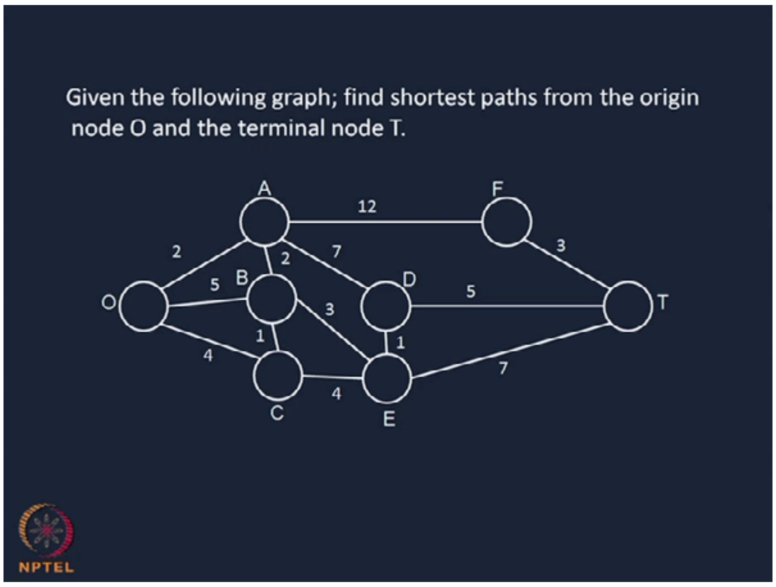
(Refer Slide Time: 58:53)



I will say it I will write it S that is the source from u if you want to move. If you want to move from s to u then what will be the path s to x, x to u I am writing this s to u, u to x what is the distance or cost? Your distance is 80 it is written over here. So, your distance is 80. If you consider v I want to go to move to v; if I want to move to v then s to x s to x x to u u to v and the distance is 90. So, if you want to move from here then s to x, x to u and u to v; and this value will be d equals 90 I have written this one wrongly this should be s to x and x to u. Because x to u and x to u, for u s to x, x to u, for v s to x, x to u, u to v minimum distance is 90 for x if I want to move there is only one case that s to x, s to x and the distance is 50. So, distance is 50 and for the y if you see if I want to move to y; s to x, x to y and the minimum value is 70. So, I am moving from s to x, x to y and your d is equals to 70. So, like this way if I want what is the minimum path then shortest path; the shortest path is s to u sorry shortest path is s to x, x to u and u to v this is your shortest path s to x, x to u, u to v this will be the shortest.

So, the beauty of this algorithm is that not only your finding out the shortest path from one from the end vertex. But you can find out any shortest sub path; if you want to move from one vertex to another vertex. Suppose I want to move from x to v then in that case the path may be s to u, u to v like this way. And, there is another problem which you solve of your own also on.

(Refer Slide Time: 1:01:13)



Given the following graph; find shortest paths from the origin node O and the terminal node T.

And, the same way the only difference is the it is undirected graph; if you see the graph is undirected graph. That means there may be I may go from o to a or I may go from a to o also; you can solve it of your own.

Thank you.