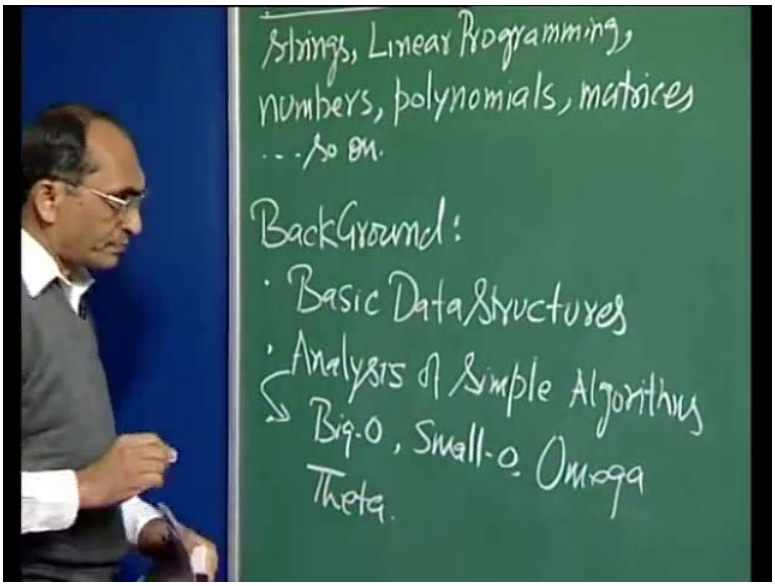
Computer Algorithms - 2
Prof. Dr. Shashank K. Mehta
Department of Computer Science and Engineering
Indian Institute of Technology, Kanpur

Lecture - 1
Graph Basics

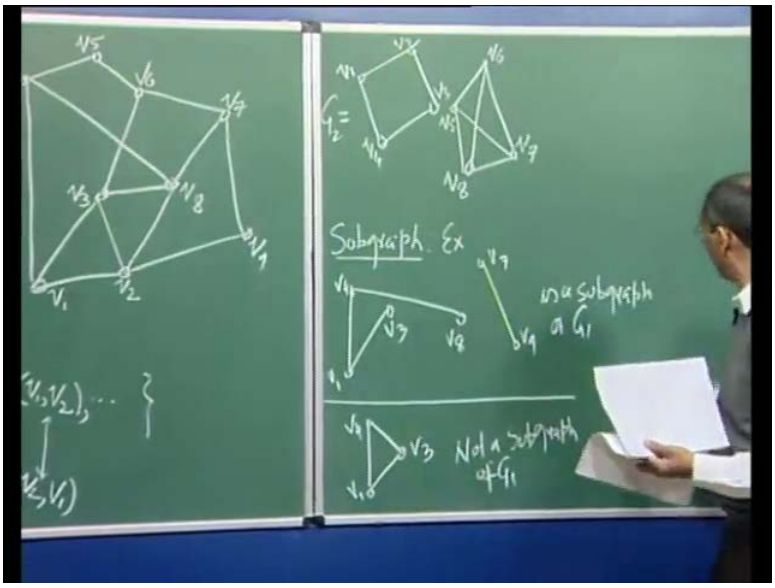
(Refer Slide Time: 00:28)



Hello. This is the course called Computer Algorithms 2; and in this course, we will introduce algorithms in many domains, including the algorithms in graphs, in geometry, in strings, on symbols, in linear programming, in numbers, polynomials, matrices and so on. In this course, we assume certain diagram that includes some knowledge of elementary data structures such as queues, stacks, binary search trees, heap etcetera. In addition, we expect that we were familiar with analysis of very simple algorithms such as sorting algorithm.

Yes, you are familiar with Big O notation, a small o, omega and theta. So, today, we will start with a very elementary algorithm in graph theory which probably you are familiar with. The algorithm is to determine the shortest distance between vertices in a graph. So, we will start with certain definitions. Initially, we will describe the concepts with examples. Then, will describe the problem and its solution. So, a graph looks like a structure such as this.

(Refer Slide Time: 03:45)



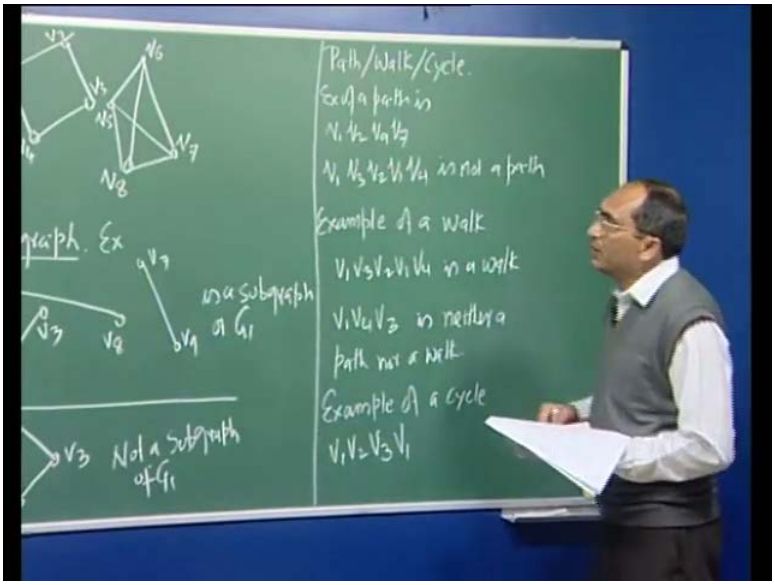
This is a representation of a graph. It has two parts. So, formally we say a graph is a couple of two entities, V stands for vertices and E stands for the edges. In this representation, the circles are vertices, which I might label as V 1, V 2, V 3, V 4, V 5, V 6, V 7, V 8 and V 9. So, the elements of V are V 1 to V 9. The second set is called the set of edges, which is a set of two sets containing two elements in it. So, a typical element in this graph is a subset of two elements V 1 and V 2. So, our E in this case, represents these subset of two elements. It is denoted by a line segment running between V 1 and V 2; so there are edges like V 1 V 3, V 1 V 4, V 4 V 5, V 5 V 6 and so on.

These are called edges. We denote typically the set of these two elements in this fashion. Hence, there is no difference between V 2 V 1 and V 1 V 2. We consider them as the same. Let us look at another graph V 1 V 2, V 3 V 4, V5 V 6, V 7 V 8. Let us call this as G 1. This is another graph G 2. V 1 is on nine vertices, G 1 is on 9 vertices and G 2 is on 8 vertices. The numbers of edges in this graph is 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, and 13. So, there are 13 edges. In this case there are 4, 5, 6, 7, 8, 9, and 10 edges.

Now, I am going to describe some of the concepts related to graphs. A sub graph of a given graph for example say I have a sub graph of G 1. This is another graph where the vertex set of sub graph, is a subset of the vertex set of the mother graph G 1. The edge set of the sub graph is also the subset of the edge set of mother graph. So, this is an example of a sub graph. This is a sub graph of G 1. The second notion I give you, is

actually another example of a graph, which is not a sub graph. This is a sub graph of G 1. On the other hand, this must be V 3. This is not a sub graph of G 1, the reason being edge V 3 V 4 does not belong to G 1. Hence, this is not a sub graph.

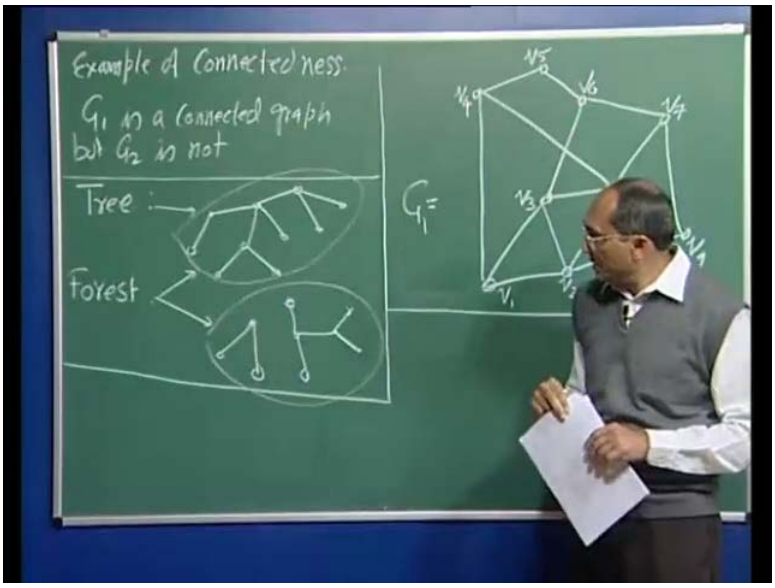
(Refer Slide Time: 09:34)



The next concept is that of a path, walk and a cycle. A path is a sequence of vertices of the graph, such that every successive vertex pair has an edge between them. So, in this case an example of a path is V 1 V 2 V 9 V 7. The condition is that no vertex should repeat in the path. This is a path. Now, V 1 V 3 V 2 V 1 V 4 is not a path, simply because V 1 is repeated. The concept of walk is when we allow repetition of a vertex, of an edge any number of times. Then, that becomes a walk.

So, in our example of G 1, V 1, V 3, V 2, V 1, V 4 is a walk. This is a walk, but V 1, V 4, V 3 is neither a path nor a walk. That is because, there is no edge between V 4 and V 3. Let us see the example of a cycle. A cycle is a sequence of vertices, where the first and the last vertex is the same and no other vertex is repeated. Again, every successive vertex pairs has an edge between them. So, here we V 1, V 2, V 3 is a cycle. Actually, I will write down this as V 1, V 2, V 3, V 1. This forms a cycle. So, let us go back and just verify that this V 1, V 2, V 3 is a cycle. The next concept we want to introduce is the notion of connectedness.

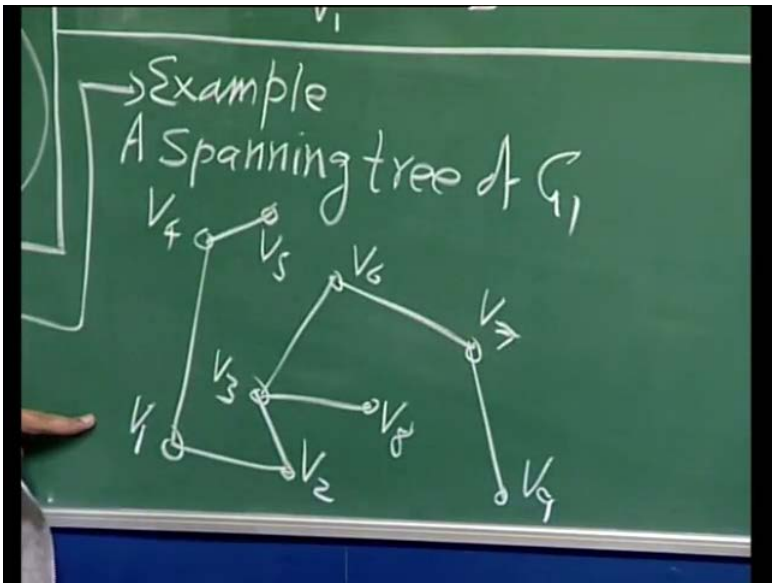
(Refer Slide Time: 13:52)



Next, we describe the notion of connectedness. A graph is connected, if there is a path between every pair of vertices. In our example, in graph G 1 there is a path between every pair of vertices if you take V 1 and V 7, we have V 1 V 3 V 8 V 7. We in fact, have more than one path V 1 V 2, V 8 V 7, V 1 V 2 ,V 9 V 7, etcetera. That way, every pair is connected. In the second example, in G 2 if you notice, there is no path between V 4 and V 8. In that case this graph will be considered to be not connected as against G 1 which is connected. So, we say that G 1 is a connected graph, but G 2 is not so. This is the notion of connectedness. Next, the tree is a special kind of graph.

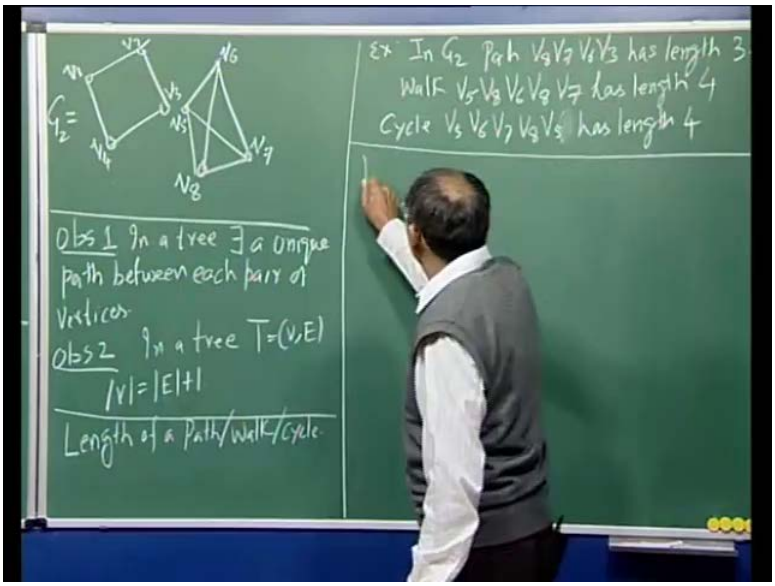
If a graph is connected, but has no cycle in it, such a graph is called a tree, neither of these two examples constituted for one thing, both of them have cycles; and in the second case, it is not even connected. So, an example of a tree would look like this. This is a tree. Of course, we related the notion to that of a forest. If we drop the condition of connectedness in the tree, then that becomes a forest. So, any graph without any cycle is called a forest. An example is this, is also an example of a forest. So, is this 1, both of these graphs is a single graph in this graph both of them becomes a forest. Next, I am going to describe a notion of a spanning tree. A spanning tree is a tree in the context of another graph.

(Refer Slide Time: 17:14)



So, we say of some graph G, the idea behind this is that, this is a sub graph of G, which is a tree which contains all the vertices of G. The word spanning refers to that. So, an example, here a spanning tree of G 1 could be V 4 V 1 V 3 V 2 V 6. This is one spanning tree of G 1. Notice that, it covers all the vertices of G 1 and it is a tree. It is a sub graph which means, it does not use any edges which is not present in G 1. Now, I am going to state a few observations regarding graph using only these notions.

(Refer Slide Time: 19:25)

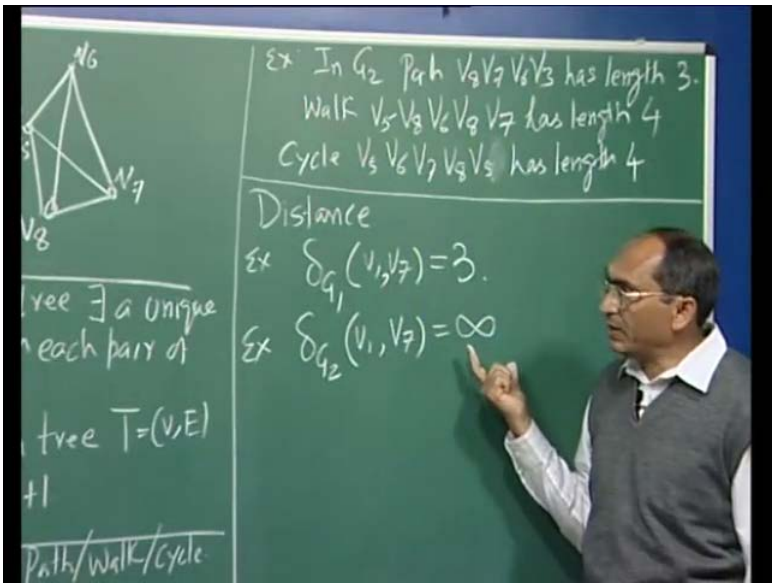


The first observation is that, in a tree there exists a unique path between each pair of vertices. You take a look at this tree. You see that there is exactly one path between every pair. Consider V 5 V 8. This is V 5. So, we have one V5 V 4 V 1 V 2 V 3 V 8. That is the only path between V 5 V 8. You pick any pair and you will notice that there is exactly one path. That is not true. In a general graph for example, between V 2 and V 3 there are several paths. One of them, being just V 2 V 3, the other is V 2 V 8 V 3, a third one is V 2 V 1 V 3 and a fourth being V 2 V 1 V 4 V 8 V 3, and so on.

Another useful observation is that, in a tree T equal to V comma E, the number of vertices is always exactly one more than the number of edges. Take a look at that example again. We have 9 vertices and 1, 2, 3, 4, 5, 6, 7, 8 edges. That is always the case. Now, again we define a few more notions. The one that we want to introduce next, is called length of a path or a walk or a cycle. In all the cases, the length refers to the numbers of edges in that entity.

So, let us take an example of a path V 8 V 7 V 6 V 5. This has got one, two, three edges. So, for example, in G 2 the path V 8 V 7 V 6 V 3 has length of 3. Walk V 5 V 8 V 6 V 8 V 7 it is certainly a walk and has length of 4 cycles, because there are four edges on it. Also, V5 V 6 V 7 V 8 and followed by V 5 has length of 4 cycles because there are four edges on it. Now, related to notion of length, we describe the notion of distance. Distance is associated with any pair of vertices in the graph. So, the idea is that given any two vertices, the length of the shortest path between the two vertices is called the distance of that pair.

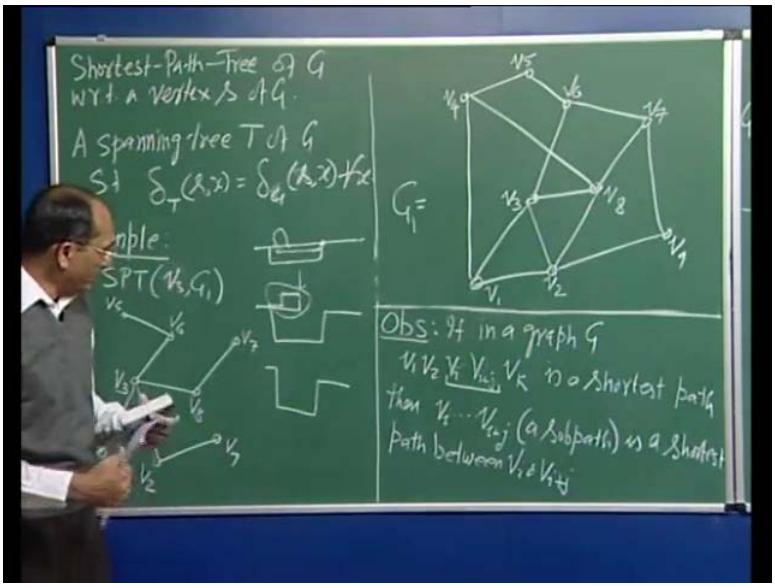
(Refer Slide Time: 24:22)



Here, example delta G 1 V 1 V 7 in G 1. Consider these two vertices V 1 and V 7. There are several paths and one can actually workout carefully and see that there is no path of length less than 3 between them. Of course, there are several paths of length 3. This one, this one, this one and the third is this, this and this. Other paths are of length greater than 3. So, the distance for this case is 3, another example, delta G 2 of V 1 and V 7.

This time I am looking at graph G 2 and the distance between V 1 and G 7. In this case, what we notice is that, there is no path between V 1 and V 7. Hence, we describe such distance to be infinite. There is actually no path we can reach from V 1 to V 7; hence, that is finite. After introducing the notion of distance, the next thing I am going to talk about the shortest path tree of a graph G with respect to a vertex S of G.

(Refer Slide Time: 26:37)



So, given a graph to give a specific vertex of G, we talk about a structure called shortest path tree of G. This is actually a spanning tree of G which satisfies the following property namely, that a spanning tree T of G such that delta T of S x is delta T delta G, rather of S x for all x. So, let us take an example. Consider the graph G 1 and vertex V 3. So, the shortest path tree is of V 3 and the graph is G 1. So, consider the following graph V 3 V 6 V 1 V 2 V 8 V 7 V 9. So, this is indeed a spanning tree. We cover all the vertices. We are looking at vertex V 3 and in context of V 3 if you notice, the distance is from V 3 to each vertex in G 1.

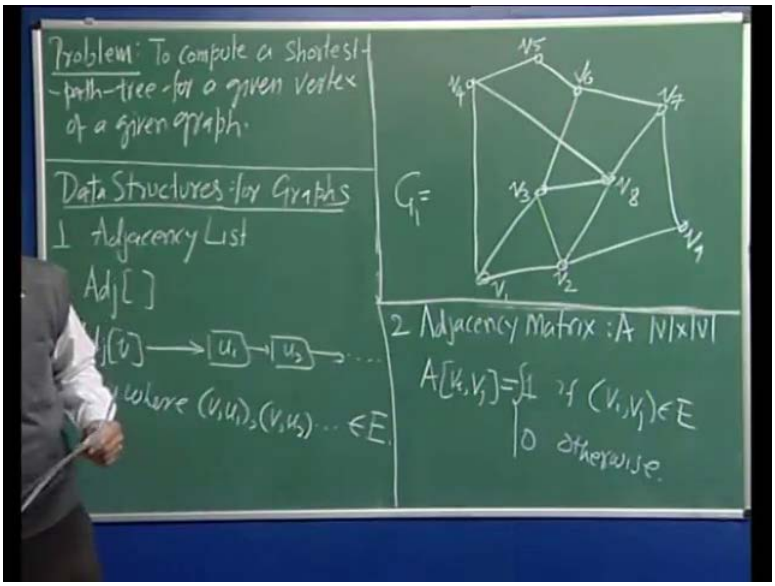
In this graph and in this tree T are same. For example, distance between V 3 and V 4, it is 2 here. The shortest path is from V 3 to V 4. G 1 is also of length 2 and same is true with respect to every vertex. So, this is an example of a shortest path tree of vertex V 3 in graph G 1. Now, I am going to introduce one more result and that is the following. If in a graph G if V 1 V 2 V k is a shortest path, then V i to V i plus j, a sub path is the shortest path between V i and V i plus j.

But, we are saying that this is a shortest path between V 1 and V k when you pick any segment of it, say from position i to i plus j. That segment also is a shortest path in the graph between those two respective vertices. Let us get a sense y, because suppose you had a shorter path between these two, so what you had was a V i here and V i plus j here.

You put this to cover a shorter path than this. Then, you can plug in, remove this and plug that path in now. What will happen is the length of this structure will be lesser, although this may not be a path. If this is not a path, then there will be cycles in it. You can delete them symbolically. Here, is original path and now I have found well from the looks that it does not looks like a shorter path. But suppose this path is shorter than this path, then you can remove this and you end up getting this.

I have deleted this segment and replaced that by this, which is supposed to be shorter. Then I get this path and this structure. Now, this need not be a path, say this is a cycle. In that case, we can cut off this. We notice, here is a cycle, this is a cycle and this is a closed walk, which starts here and ends here. So, I can delete this structure out of this thing and end up getting even shorter structure. If you remove all such closed cycles, you get a path between the same two end vertices, which will be strictly shorter than the original path. But, the original path was the shortest. Hence, this cannot happen and this better be the shortest path in the two vertices. So, far we had seen all the concepts and all we needed for our problem. Now, I am going to introduce the problem.

(Refer Slide Time: 34:03)

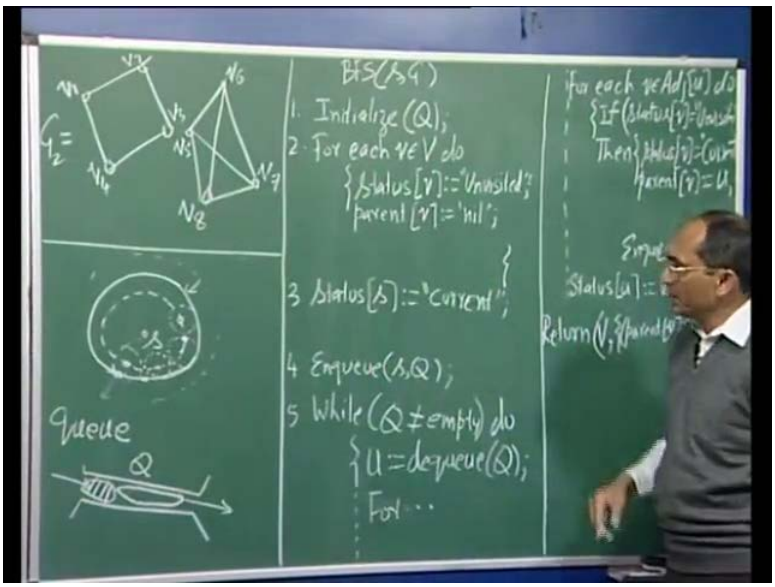


The problem is to compute a shortest path tree for a given vertex of a given graph. Now, we will introduce the algorithm for this and then see the correctness of that algorithm. But, before we go into the algorithm I will give the definition of two more entities in the two data structures. That one uses to describe graphs. So, these are very useful data

structures. The first one, the data structures for graphs is called adjacency list. This is an array, we may call A d j, where the argument is a vertex of a graph and A d j of a vertex V points to a list of all the vertices with which there is an edge with V.

So, this could be a list of U 1 U 2, where V U 1, V U 2 is epsilon, etcetera. All are in the edges of the graph. We also call such vertices as neighbours of V or we also call that U 1 is adjacent to V U 2 is adjacent to V and so on. The second data structure we may use is the called adjacency matrix. This is a matrix say A, of size number of vertices times number of vertices. This way it represents a graph, which is that A of V i comma V j is 1, if V i comma V j belongs to be edge set 0. These two data structures, we may find useful in describing an algorithm. Now, let me come to the main algorithm to solve the problem of computing the shortest path tree. Now, in order to get a sense of the algorithm, we will first see the basic techniques behind it to describe this algorithm. Let us recall, how light emanates from a point source let us say s?

(Refer Slide Time: 38:32)



A wave front of this light is the set of points where all the light reaches at the same time which starts from s at the same time. Further, if we want to compute the next wave front, then from Huygens principle you take each of these points. Then again from these local wave fronts, you take their cover. What you find is, this is the next wave front. The importance of the wave front is that, since it is the set of points where light reaches at the

same time therefore, in a homogeneous medium the distances are same from the source. So, these are all the points which are at the same distance from s.

Now, we are going to emulate this process in our algorithm. What we will do is that, we will start from vertex s, point s for our case. In the graph, the point will be a vertex. Then, identify all the vertices, which are adjacent to s. See, our graph is a discrete universe. So, we have immediate neighbours. I am going to constitute the first wave front after discovering all the vertices of s. Then, we will pick one of these vertices and find out all the new vertices, which are adjacent to that vertex which we have not yet discovered.

We will discover those and then we will take the next one and find all the new neighbours. We will discover from this and then we will go to the next one and discover the new vertices. Then, we will go to the next one and so on. After we have done that, we have found the second wave front and after completing a second wave front we will start with the vertices of this second wave front and again repeat this process. Of course, if there are back neighbours we are not interested in that. We only discover new neighbours and that is how we perform the third wave front and so on. This is the idea.

Now, in computing this, we are going to use a data structure Q where of course, we know that an element enters in to Q from one end and exits from the other. The advantage of using Q is following. Why we are trying to discover neighbours of a vertex of those which are pending? We have yet to discover those vertices. We will stay in this data structure. We will pick the four most near vertices. The vertex in the front, we will find out all its neighbours.

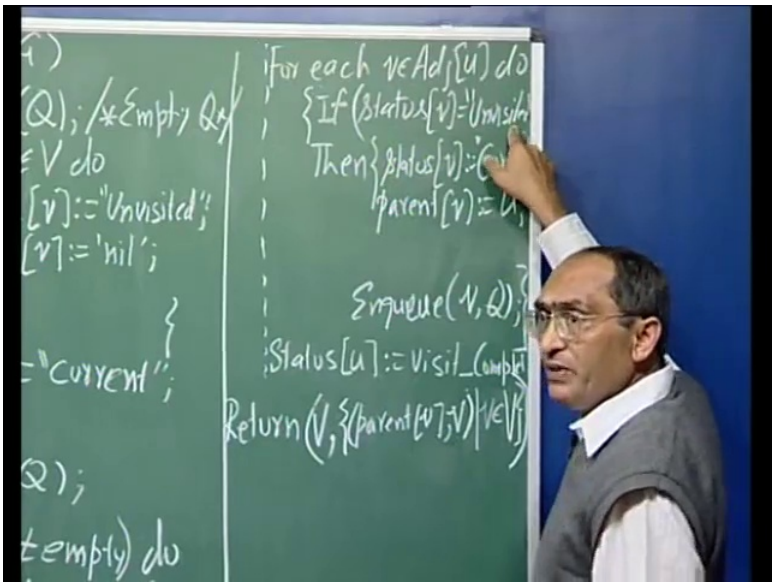
We will find the next wave front and all that. Once we have discovered we will enter them here. So, if there are vertices in this portion which are corresponding to the same wave front, the vertices of the next wave front will always be behind it. You will not start discovering their neighbours, until you are done with the neighbours of these vertices, because of the data structure justifies the use of Q. So, here we have our algorithm called breadth first search.

This algorithm only emulates this process in exploring the space starting from the vertices. Then we will see how we can use it to determine the shortest path tree. So, let us start with the algorithm. The first step is to initialise Q to an empty Q. Now, as you

notice that V, it is more neighbours of vertex once. Then this vertex is of no use for us. So, in case we revisit this. We want to make sure that we do not again do this, an exploration starting from that point or so called expansion from that point. For that purpose we are going to use a status.

Initially all the vertices will be set to status unvisited indicating that we have not visited that. Yet, there is one more information we are going to store which we will show later on. Initially, it is all nil. Now, s is our first vertex. We are going to set its status to be current and what this indicates is that currently I am going to search its neighbours. I am going to put this into Q. All those from which we have to search the neighbours are going to be set into this Q. Then, we begin this loop in which we pick the front element of the Q namely U. Then, for each neighbour of U we do something. We pick all the neighbours and check whether its status is unvisited.

(Refer Slide Time: 45:48)



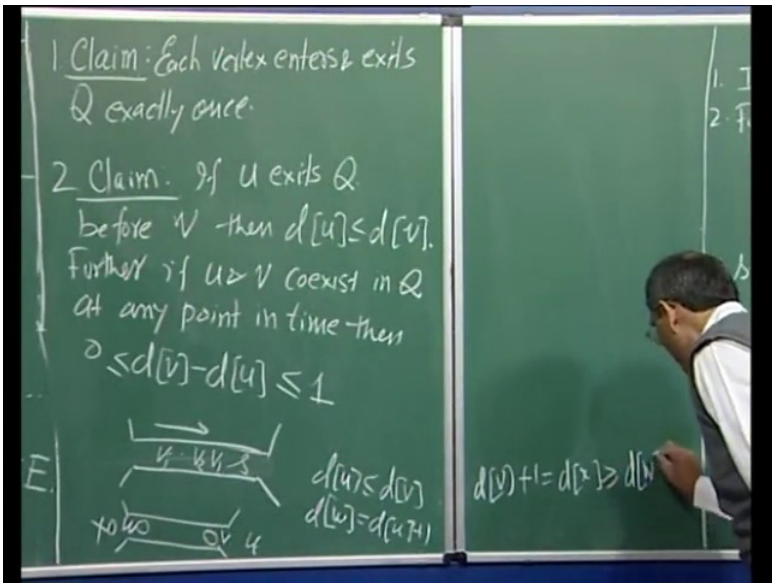
If it is a not unvisited, then we know that we have already reached it and are not interested in those vertices. But if it is being visited for the first time, then we are going to set its status to current because we are going to now search the neighbours of this vertex. Now, here I am going to tell you what this new parameter is, that is the parent of V. Here, we are going to record how we reached V for the first time. We reached V through U. So, we are going to set it to, parent of V is U. Now, this V which will give me the next set of vertices in the exploration has to go into Q.

So, n Q is entering the vertex into Q at the back. Once we have visited all the neighbours of U, we will set its status to visit complete. Now, this is of no interest to us. We will complete this. Then, later on I will show that we are going to get a spanning tree. From this, which is actually the shortest path tree of s? This spanning tree is following, it constitutes all the vertices of our original graph and edges in it are following parent of V, where every vertex and its parent, such pairs will constitute the edges of this graph. We will show you later on. Now, what we have seen here is that, we discover the vertices corresponding to one wave front means all the vertices are taken care of.

Then, we moved to the next wave front and so on. So, if we label the vertices with the wave front to which it belongs, then I will actually get the distance of that vertex from s. So, what we do that is initially the when we are initialising the vertices, we can set the distances of every vertex to be 0. We know the distance to V prime. It is distance of s. Well, actually that is taken care of. The distance of s is also 0. We do not have to do anything about it.

But, what we notice here is, as we discover a new vertex V from U, we are discovering a vertex of the mixed wave front. Hence, we will set a d of V as 1 plus d of U. So, what you learn here is that, the d of V is nothing but the wave front to which vertex belongs. Now, in order to prove the correctness of this algorithm I need to prove certain claims. So, I am going to first state the first claim. Let each vertex enter and exit from Q exactly once.

(Refer Slide Time: 49:36)



Now, to see this you must observe the following. Recall that our graph is connected. Now, in our graph if this denotes s, this is any vertex x. Then, let there be some path because of connectedness, denoted by this. Initially we know that s enters the Q right at the start and when we expand it, its neighbours enter the Q. Now, suppose up to certain point, let us say this is some y 1, y 2, y 3 and y 4. Say, suppose up to some point here vertices have entered the Q some point in time.

Then, when we expand this vertex, one of the neighbours will be y 3. So, either y three was discovered for the first time here or it was discovered earlier. Either way y 3 must also have entered Q. Hence, this also should have entered Q sometime and so on. Hence, eventually s must have entered Q as well. Now, what we see here is the use of the status ensures that every vertex enters exactly once. So we know it enters at least once. But, it does not enter again, simply because after exiting Q, we change its status. So, it exits from Q. We do certain processing and then set its status to visit complete.

Hence, it is not eligible to enter Q again, because for entering we check whether it is unvisited or not. So, it will never enter again. That guarantees that every vertex enters exactly once into Q and exits. Hence, exactly once the second claim states that the d values of vertices are non decreasing in the order of their exiting from Q. So, a vertex that exits earlier as a d value less than or equal to the vertex that exits later. So, we will say that, if U exits Q before V, then d of U is less than equal to d of V. Further, if U and

V coexist in Q at any point in time, then d of V minus d of U, which we know is greater than equal to 0.

This is not greater than 1. So, if they are present simultaneously in the Q, then the d value cannot differ by more than 1. How do we say this one? So, what we will see is that, if this is Q and this is the direction of Q, then let us say this is our U and this is V. See, the front and the end vertices are U and V. Then, when we are going to expand it, its children will enter here and there the d value will be the d value of u plus 1. So, the next vertices will have d values more than the d value of U. If we believe the first part, then the d V value of the U must be between the d value of 1 plus d value U.

So, that will guarantee the second part and the how do we see that the d values are monotonic? They can never decreases when we proceed and that is because such a statement is true. Initially, there is only s in it. Its d value is 0. Then, its children enter into it. So, if I want to show this initially, s and then its children comes say V 1 V 2 and on so on V j, their d values are 1. Then of course, when one of these is expanded the D values become 2.

So, in general if up to certain point the d values of the successive vertices that have entered the Q has only increased, then the last vertex here has highest value among all. Then, the next vertex that will enter here; now it is d is 1 plus d value of some earlier vertex U, this is the d of U plus 1. This vertex has d value greater than this. Hence, their children that will enter here will be of d value of 1 plus this. If this is say V and this is W, then we have inequality d of U is less than equal to d of V and d of W is d of U plus 1. Hence, it is d of V. The next vertex that comes here, say x plus 1 will be the D value of x which will be greater than equal to d value of W. So, this will remain monotonic. We will continue this discussion in the next lecture.