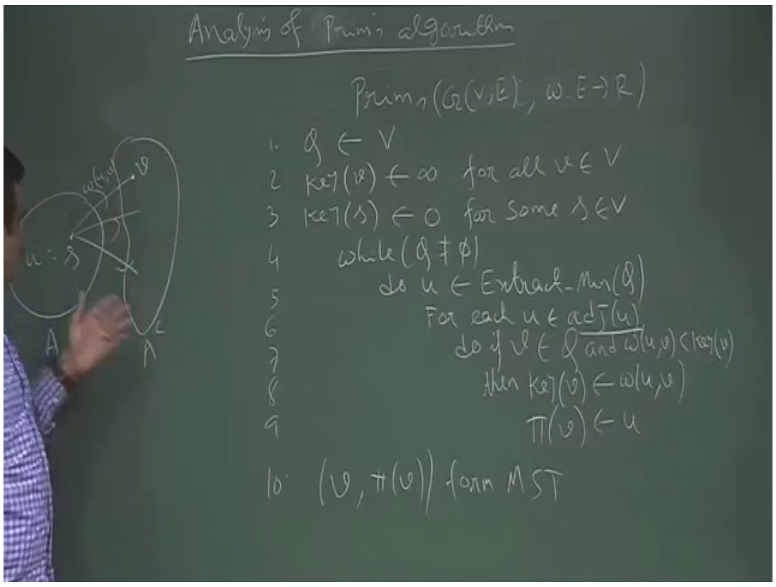
An Introduction to Algorithms

Prof. Sourav Mukhopadhyay
Department of Mathematics
Indian Institute of Technology, Kharagpur

Lecture – 40
Graph search

(Refer Slide Time: 00:39)



Analysis of Prims algorithm

```
Prims(G(V,E), w: E → R)
1.  Q ← V
2   key(v) ← ∞ for all v ∈ V
3   key(s) ← 0 for some s ∈ V
4       while (Q ≠ φ)
5           do u ← Extract-Min(Q)
6               For each v ∈ adj(u)
7                   do if v ∈ Q and w(u,v) < key(v)
8                       then key(v) ← w(u,v)
9                           π(v) ← u
10  (v, π(v)) form MST
```

So, we talk about graph search, how you can explore a graph. So, before that let us just finish the analysis of prims algorithm from the last class. So, last class, we have described the prims algorithm and we have taken an example, how it is working. So, now we will talk about the time complexity of this algorithm so that is called analysis of prims algorithm. So, as we know prims algorithm is to find the minimum spanning tree. So, input is a graph, we have a undirected graph v comma E and we have a edge weight which is basically E each edge is associated with a weight.

So, the algorithm is goes like this. So, we assigned each vertex to a priority queue. So, we put everything into the priority queue. We maintain this Q based on the degree of each vertex, and we assign the degree of each vertex is infinity because this is the initialization because initially nothing has been explore, so that is why you put degree to is vertices infinity. So, nothing is exploring yet, but we have to have the starting vertex.

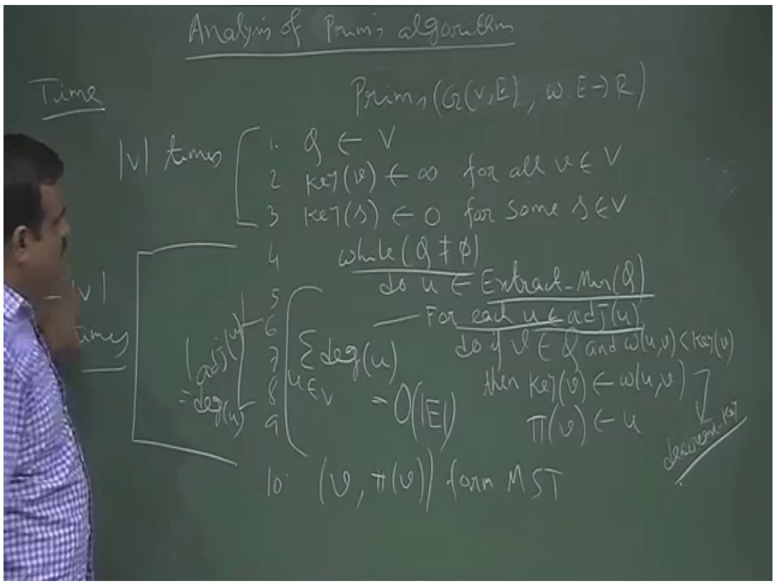
So, if we recall the prims idea is a greedy choice we start with a vertex s and we capture all the vertices which are connected with edge is to other vertices whose edge weight is minimum, so that is the greedy option, so that is why prims algorithm is greedy choice. So, which we have to start with a vertex and this vertex is we can choose arbitrary. So, for some arbitrary vertex s, so we have to have a starting vertex, so that we can decide. So, any vertex could be the starting vertex in prims algorithm, so that we have to choose arbitrary. And now this loop is for while Q is not empty Q is starting with all v.

Now, we extract the minimum from this Q minimum is based on the key value. So, initially minimum will be the s. So, this will be the minimum initially s will be extracted and this will be u. And when we extract something from Q, it will be deleted from Q and it will be added in say what is a set comma called A set say. So, it will be deleted from Q and then which we all the adjacency vertex of u which will the whole adjacency vertex of u and suppose this vertex are called v vertex, these vertex are called v vertex. So, this is A compliment c.

Now, we check this weight by v vertex if this is in Q. So, if this is, so initially everybody has to be in Q because initially u is basically s other than you everybody is in Q. So, their degree was infinity, now we have to change the degree by this weight w of u comma v because this was infinity and in the later stage this is u. So, we check the degree of this and we change the degree. So, this is the relaxation. So, this step is called. So, we check the degree, if the degree is this, this step is called decrease key, decrease key operation. And this step is called relaxation step, now not relaxation this, this is called decrease key operation. So, basically this is not relaxation, this is the decrease key operation.

So, basically we are decreasing the key like this. So, because these are having a degree which is more than the degree what we have the edge weight what we have right now. So, we have to update that. So, we update that I mean put a. So, who is the responsible for this update this guy. So, we put a link to that to have the minimum spanning tree at the end. And now again we extract the minimum, so that minimum will be the minimum among the bridge edge. So, this is our A set, A set means everything in A is basically A compliment is Q, A is basically Q compliment. Now, there is a theorem we have prove in the last class that all the bridge edges if we consider A to A compliment whichever is the minimum has to be in those minimum spanning tree, and that will reflect in their degree.

(Refer Slide Time: 05:23)



Analysis of Prim's algorithm

Time

```
Prims(G(V,E), w: E→R)
1. Q ← V
2  key(v) ← ∞  for all v ∈ V
3  key(s) ← 0  for some s ∈ V
4     while (Q ≠ ϕ)
5        do u ← Extract-Min(Q)
6           For each v ∈ adj(u)
7              do if v ∈ Q and w(u,v) < key(v)
8                 then key(v) ← w(u,v)
9                      π(v) ← u
10  (v, π(v)) form MST
```

|V| times

|V| times

$|adj(u)| = deg(u)$

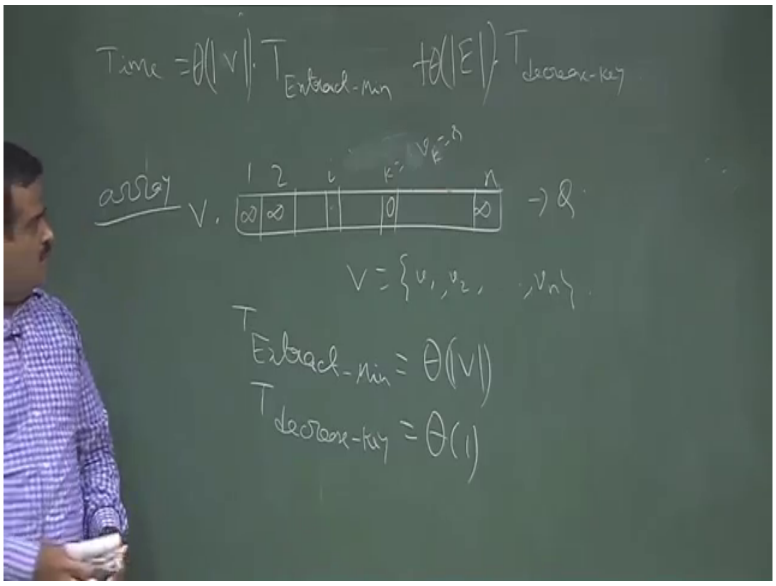
$\sum_{u \in V} deg(u) = O(|E|)$

decrease key

Now, talk about time complexity of this algorithm, so the time complexity of this algorithm. So, this initialization step is for order of so this is basically order of v times v is the degree of the vertices. So, we can just write order of v, actually this should be order of cardinality of v, but anyway we are just denoting this v means anyway to be very specific we have to have this. And this outer loop is also order of this inner loop is v times by this initialize some step and this outer loop while Q is empty because Q is initialize by v. So, this is also v times. So, v times we are doing the extract minimum from the Q. So, it depends how much time our Q is taking to extract minimum. So, depends which data structure we are using for the Q, we will come to that analysis.

Now this step the inner loop inner loop is for this for each of these we have to going for the adjacency list of u. So, this step will take. So, this step will take adjacency list of u. So, this we are doing for all v. So, this is basically summation of this. So, is called degree of the cardinality of this is basically degree of u. So, this step will be take summation of the degree of u, u belongs to v. So, this is basically we know this is basically order of E by handshaking lemma summation of degrees order of v. So, this time we are doing the decrease key. Again decrease key we will depend on the data structure we are using to have this. So, what is the total time complexity, time complexity is order of v plus order of v times extract mean plus order of E times decrease key. So, the time complexity for prims algorithm is basically.

(Refer Slide Time: 07:50)

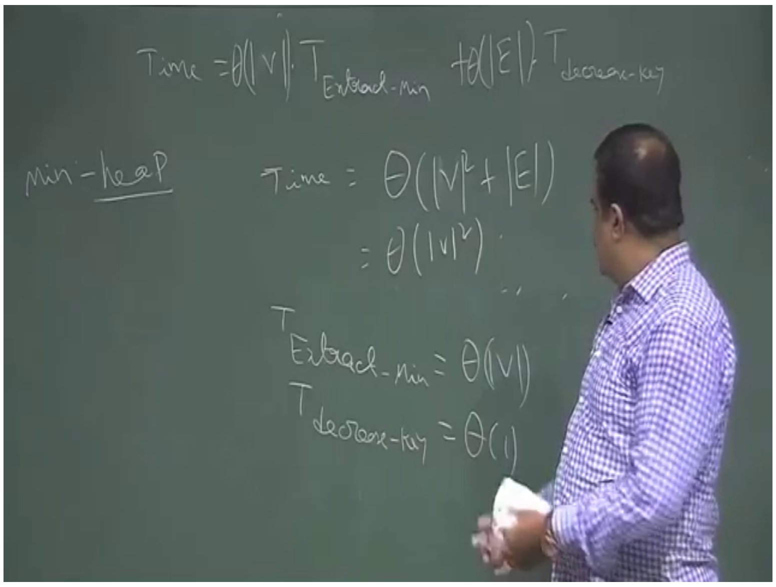


So the time for prims algorithm is order of, so it is basically order of v times time to extract the mean. So, it depends how we extract the mean time for extract the mean plus E times time for the decrease key operation because we are decreasing the key value of a node. So, we have to go to that particular node and we have to decrease this. Now depending on so this is the order we must write this as order from. So, this is the time complexity for prims algorithm in general.

Now, suppose we depending on which data structure we are using to have this Q, suppose we are using array for our Q. Suppose, we are using array data structure simple array, annotate array so that means, we are putting everything. Suppose, our V is say or vertices v 1, v 2, v n suppose there are n vertices. So, we have a array, so V array is 1, 2 up to n, so this basically v n, v 1 like this. So, simple annotate array. So, this is our Q say so we are having a array for our priority queue. So, each of this element are Q this is the key value. So, this is the key value each is insulates by except some value s, s is say v k. So, this is say k, so this is say v k. So, v k is basically s a v k is basically s starting vertex. So, except this everything is infinity. Now, this is changing because this is the every time we are checking, so we are extracting name. So, how much time, we will take them find the minimum from this array. So, minimum extract mean will take, so extract mean will take how much time because this is on order array. So, we have to scan all the element. So, this is basically order of v times.

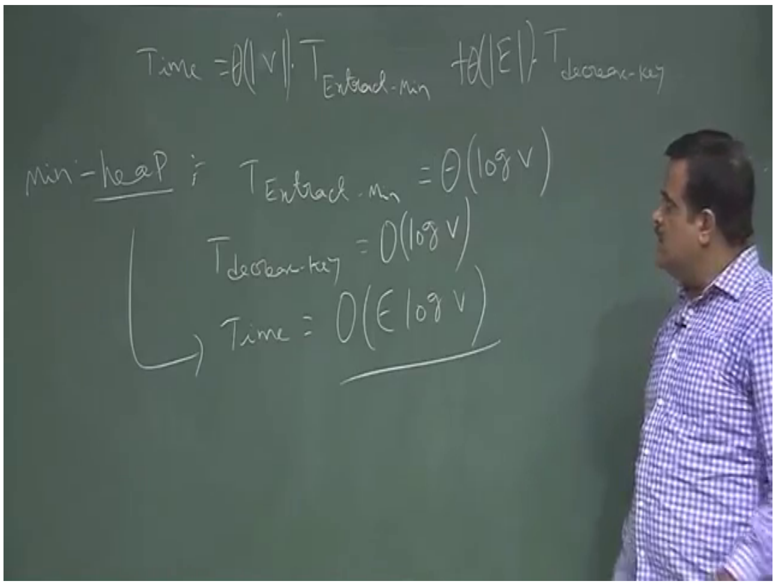
And now decrease key. So, decrease key means suppose we have to decrease these values. So, this is the i th. So, we have to go to this position we have to change. So, decrease key will take constant time. So, decrease key will take constant time, because we are going to that particular position particular position and we have then we are changing the value. So, in that case what is the complexities, this is basically order of v and this is order of one. So, this for array, this time is basically order of v square.

(Refer Slide Time: 10:56)


$Time = \theta(|V|) \cdot T_{Extract-min} + \theta(|E|) \cdot T_{decrease-key}$
min-heap
$Time = \Theta(|V|^2 + |E|)$
$= \theta(|V|^2)$
$T_{Extract-min} = \theta(|V|)$
$T_{decrease-key} = \theta(1)$

For array implementation time is order of V square plus E I mean this is the cardinality basically. Now we know this order of E bounded by order of V square, so this is basically order of V square. So, order of n square if there are n vertex in this array in this graph. Now, suppose we are using a priority we are using a Q where heap min heap for this priority queue min heap. We know the max heap, similarly min heap is basically the heap we know heap is a data structure which is basically an array which is viewed as a binary tree. So, it is a array i-th element note if i th element than the child set 2 i to i plus 1, so that way we viewed the heap. So, for heap if we use heap for this priority queue min heap, min for min heap minimum is there in the root. So, for min heap property is every node is less than from his child, so that is the property of min heap. So, if we use heap data structure for this Q priority queue min heap particularly.

(Refer Slide Time: 12:20)



$\text{Time} = \theta(|V|) \cdot T_{\text{Extract-min}} + \theta(|E|) \cdot T_{\text{decrease-key}}$

$\underline{\text{min-heap}}$: $T_{\text{Extract-min}} = \theta(\log V)$

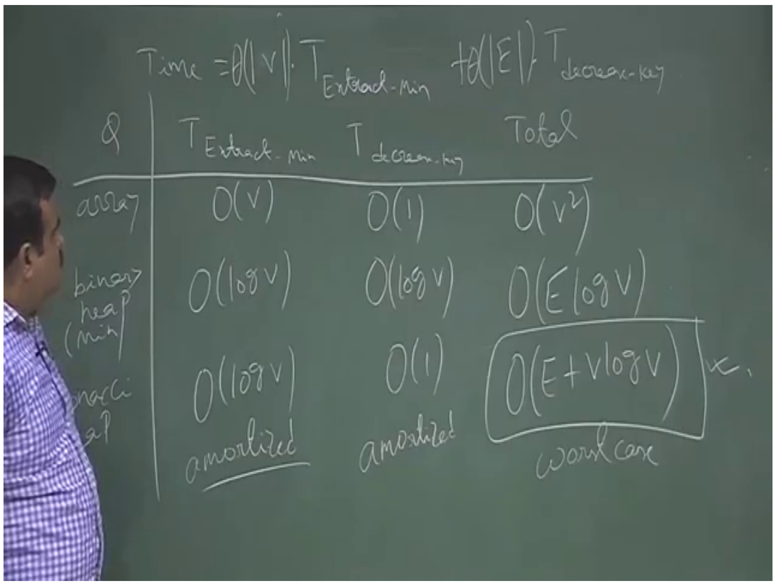
$T_{\text{decrease-key}} = O(\log V)$

$\text{Time} = \underline{O(E \log V)}$

Then what is the time for extract min, so extract min means we know the for minute min is in the root. So, for min heap we know for min heap we know, this is the minimum, if it is min heap. Now, we need to extract this, to extract this we can change with the last element here and that will destroy the heapify property, here min heap by property then we have to make it a heapify again, so that will take login time. So, it is basically order of log of V, so that is the extract min.

Now, what about the decrease key decrease key mean suppose we are going to decrease this this key value. So, this have being a value. Now, suppose it got decrease due to the algorithm the whatever value it was have being is more than the direct link from that u to that vertex. So, this has to be change. So, if you change that then it may violate the many, many property. So, we may up to call min heapify here, so that will again take order of login because we do not know the position. So, order of log in. So, both the operation decrease key and extract min will take order of log v so that means, if we add these two time complexity here. So, for min heap it will take time is basically, so order of log v and order of E log v. So, basically it is order of E log v. This is if we use the heap implementation heap as a priority queue.

(Refer Slide Time: 14:27)



$\text{Time} = \theta(|V|) \cdot T_{\text{Extract-Min}} + \theta(|E|) \cdot T_{\text{decrease-key}}$

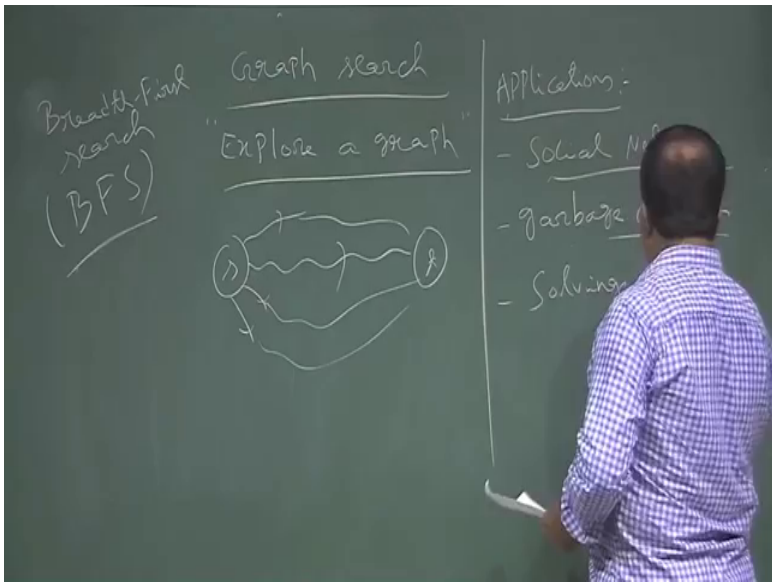
Q	$T_{\text{Extract-Min}}$	$T_{\text{decrease-key}}$	Total
array	$O(V)$	$O(1)$	$O(V^2)$
binary heap (min)	$O(\log V)$	$O(\log V)$	$O(E \log V)$
nacci ap	$O(\log V)$ amortized	$O(1)$ amortized	$\boxed{O(E + V\log V)}$ worst case

So, now, there is another data structure which is called Fibonacci heap. So, let us write this in terms of so priority queue T extract we will write this in a table extract minimum time and then T decrease key and this is the total time. Suppose we are writing this in a table. Now, suppose this we are using an array for this priority queue, the data structure we are using here. So, this will take order of V, this will take order of one this will take order of V square, so cardinality of V square basically. And if we take the binary heap; that means, in particularly min heap then we have seen this will take order of log v, this will also take order of log v and this will take them order of E log v.

Now, if you take another data structure, which is called Fibonacci heap. So, this is there in the text book Fibonacci heap then we can so this will take order of log v amortized. So, this analysis is amortized analysis and this will take order of one, so this is also amortized. So, the total will be order of v. So, order of v into log v, and order of one means decrease key. So, this is E. So, there basically it will take E plus v log v. So, total time complexity is E plus v log v. So, this is the best among these. So, this is also worst case amortized analysis. So, this is the best data structure for this analysis. So, this is the analysis of prims algorithm. So, basically depending on the data structure. So, if we use the Fibonacci heap, the amortized cost in the worst case is order E plus v log v. So, this is the based we know for the prims algorithm.

So, now, we will start the next topic. So, this is the prims algorithm time complexity. So, this time complexity will be same as when we will talk about (Refer Time: 17:19) algorithm for finding the shortest path. So, just to there will not discuss this whole details. So, we have to recap we have to come back here to have the analysis for analysis this table.

(Refer Slide Time: 17:50)



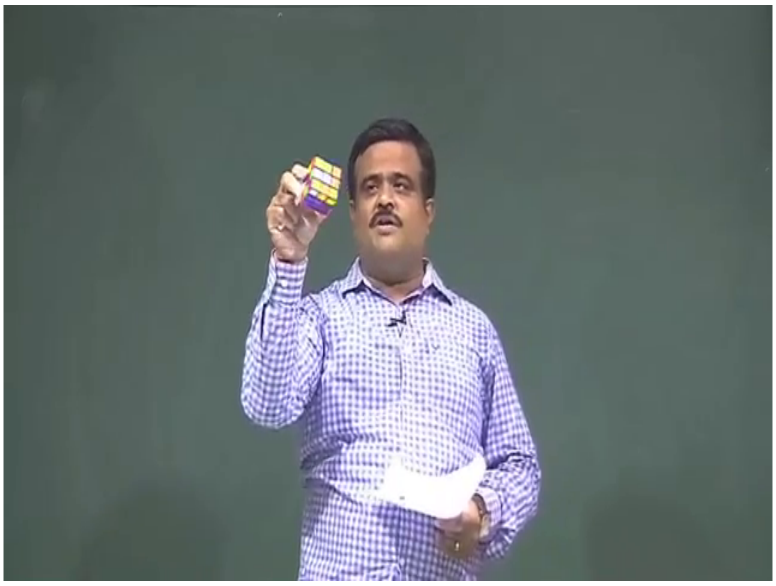
So, let us start with the new topic, which is called graph search. So, how we can explore a graph. So, this is called basically graph search. So, this is to basically explore a graph. So, what do we mean by exploring a graph, so that means, suppose we have a vertex says source vertex and we have a another vertex t, we want to find a path from s to t. So, that is one way of exploring a graph. May be we want to find all the path from s to t, we want to find all the path from s to t, so this is one way to explore a graph. Or maybe we want to find the ah shortest path from s to t we want to find the shortest path we will come with the weight, we may not have weight there in a graph. So, this is a given graph we have a graph, we want to explore it. So, this is one way we can find out the path from a source vertex to another vertex, we may need to find out all the paths from source vertex to other vertex.

So, we will talk about to breadth first search, this is one way tom exploring a graph, breadth for search this is in short come BFS. So, this is one way to exploring a graph. So, there are many application of exploring a graph. So, let us just jot down some of the applications of. So, one application may be social network like Facebook. So, Facebook we can consider each user has ignored, and suppose I want to find my all of friends. So, we have to explore the graph. So, or suppose there is a fried, there is a node user I want to see whether I can reach to that user through my friend. So, this is a one applications.

Another application is maybe garbage collection. So, any so most of the opening system is having this garbage collector like if we have if we use some memory and if you no longer using that suppose we use (Refer Time: 20:30) and then if you free that without freeying that suppose we are no longer using. So, it need to be free, so that other next time we can use this, so that step we can do. So, so basically we have to we have to see whether we can reach from this two that space. So, if we cannot reach that means, that is not using, so that way we can that is one applications then web crawling.

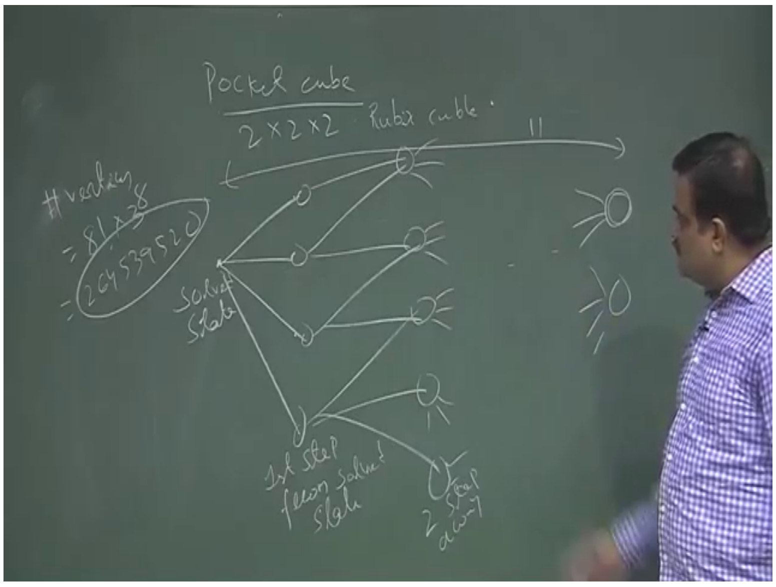
So, if we have a data, if we have a new web page I want to have a access from the other. So, Google has to access all the new web page, so that is the also exploring a graph. And solving puzzle like rubix cube how we can solve the rubix cube. So, I think I have a rubix cube let us bring the rubix cube. So, this is the rubix cube or this is called pocket q, so this one. So, solving this kind of rubix cube, so solving puzzle. So, that is they are these are the applications for this.

(Refer Slide Time: 21:54)



So, let us start with the rubix cube or the pocket Q, how this can be applied there. So, how we can associate this graph search with the pocket Q. Suppose we have a this is a 3 cross 3 cross 3 rubix cube, but suppose we have we take the lower version suppose we are have a 2 cross 2 cross 2 pocket cube.

(Refer Slide Time: 22:01)



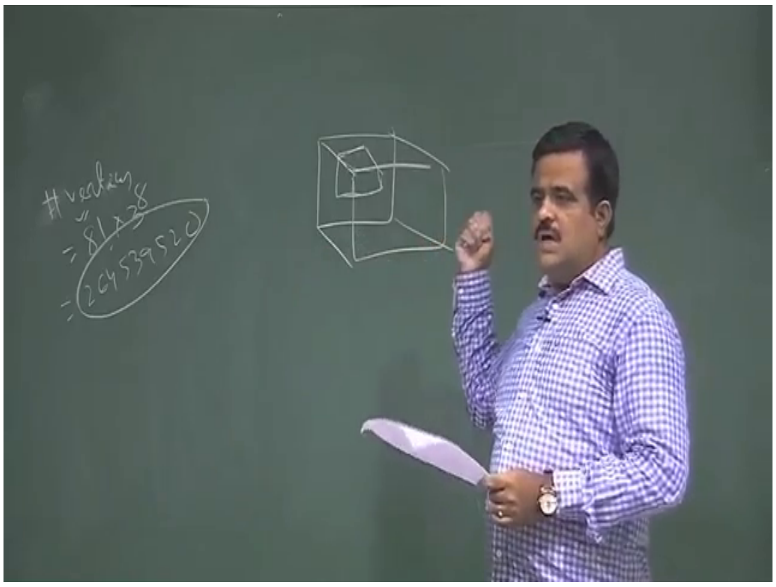
Suppose, we have a 2 cross 2 cross 2 pocket cube or this is rubix cube. So, how we can think, this rubix cube as a graph. We take each position has a graph and node of a graph vertex, this is a position we take node this is the vertex. If we change one little then it is another vertex like this. So, if you change this, this is another vertex. So, this is another vertex like this. So, each position each state this is one state, each state is a vertex of a graph and suppose we are in solve state.

Now, if we take just one move then there will be another state, so so that is the basically suppose we at are solve state solve state means all the colors are matching in a same this is not solve one, but solve state means all the color that is also is state that is also vertex of a graph. And at the solve state if we just have one change 90 degree change then that will be another state suppose these are the vertices which are basically these are the vertices which are basically one step. So, these are the basically one step away from the solve state that means, these from this we can have a one move to go to the solve state. So, some solve state we did one change to get this state. So, this is basically one step from solve state.

Similarly, if from there we can have two move another move. So, this is basically two step array from the solve state like this. So, this is may be here. So, like this. So, this is the two step away from solve state. So, these way we will continue we will reach here A. So, this is the last state like this. So, this is called this is a example of a breadth first search, breadth first tree basically. So, this tree is called breadth first tree ah b of phase and we you will be surface to know this dimension this is called dimension or this is also called got number. This is basically 11 for 2 by 2 by 2 rubix cube.

So, now, how many state are there in this a rubix cube 2 by 2 by 2, the number of vertices basically number of vertices, vertices are basically factorial 8 into 3 to the power 8. So, this many vertices we have, so this many possible. So, this is huge number this is basically 264539520 this is the number of state of this graph, but dimension is 11. You can there is a proof that we one can show the dimension is 11 this is called got number or the dimension of this, this is for 2 by 2 by 2 rubix cube.

(Refer Slide Time: 26:11)



Now, how to show this graph has this number of vertices is this. So, basically how to show this. So, just try to have a 2 by 2 by 2 rubix cube. Now, if you take this one state like this one, one part of like this. So, basically what are the so there are basically eight ways we can choose this, so eight weight means, so eight are the possible that cube we have the small cube so that means, factorial eight this is the possible ways and each way there are 8 dimensions and the three ways we can change that. So, this is basically 3 to the power 8, so that is why it is this. So, this is the breadth first search is basically this the rubix cube we just go from the opposite direction. From solve state we, so how to reach to the another state the one step away, two step away like this. So, this is the if we do the other way round and that is called breadth first search. So, we will formally have the algorithm for breadth first search, so that will do in the next class.

Thank you.