Parallel Algorithms

Prof. Sajith Gopalan
Department of Computer Science & Engineering
Indian Institute of Technology, Guwahati

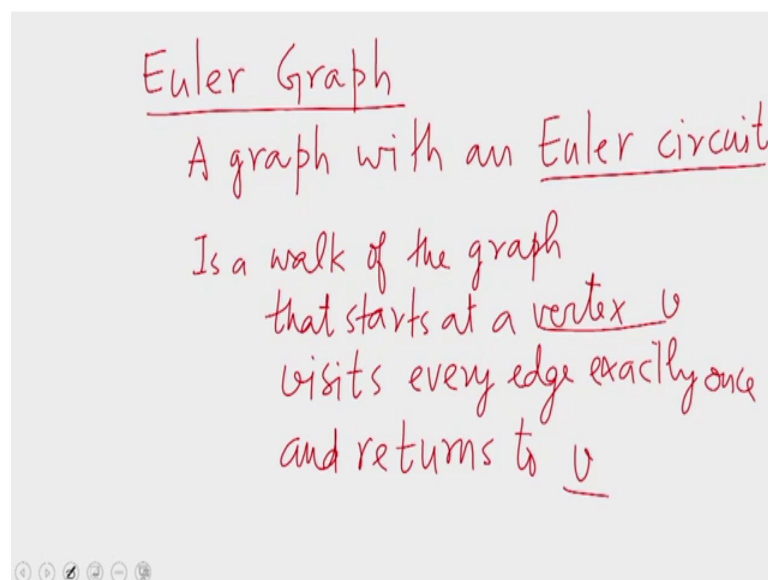
Lecture – 15
Applications

Welcome to the 15th lecture of the MOOC on Parallel Algorithms. In the previous two lectures we saw an optimal algorithm for ranking a link list which runs on an EREW PRAM, that algorithm ran in order of log n time using n by log n processors. Today we shall see some Applications of this Optimal this Ranking Algorithm.

Today's applications have to do with trees. So, we shall find some algorithms for routing a tree, finding the number of descendants of a node in a tree, finding the pre order traversal post order traversal of the tree, etcetera. But then all these algorithms use what is called an Euler technique.

So, let us see what the Euler technique is.

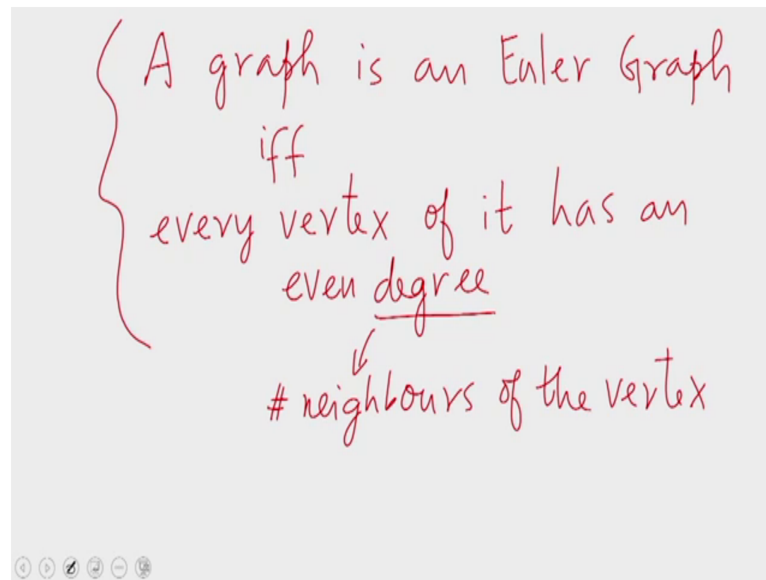
(Refer Slide Time: 01:17)



First let me define what is called an Euler graph. An Euler graph is a graph with an Euler circuit. An Euler circuit in turn is a walk in the graph which is a traversal of the graph that starts at a vertex v, visits every edge in the graph exactly once and returns to v.

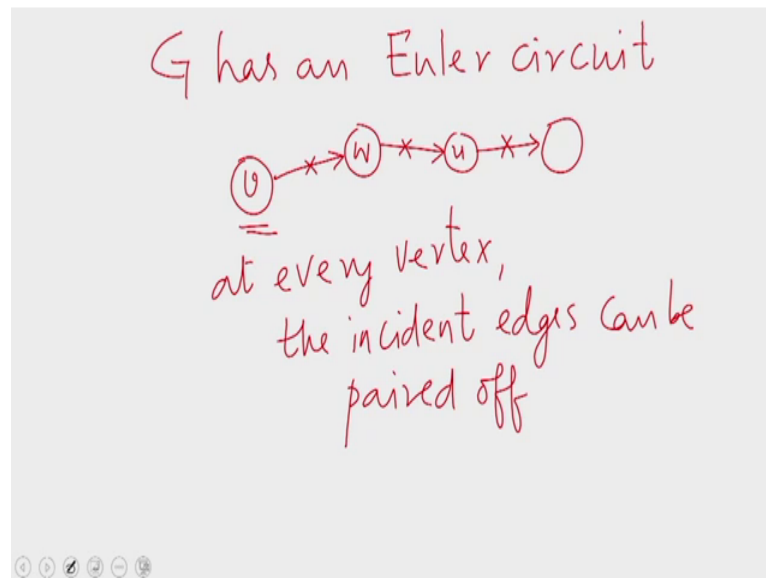
So, as you can see the vertex v has no real significance, you could start the walk at any node. So, if you simplify the definition it would mean that an Euler circuit is just any circuit on the graph that visits every vertex exactly once not every graph has an Euler circuit.

(Refer Slide Time: 03:01)



A graph is an Euler graph or it has an Euler circuit precisely when every vertex of it has an even degree. Where the degree of a vertex is the number of neighbors the vertex has; the degree of a vertex is the number of neighbors the vertex has which is the same as the number of edges incident with the vertex. So, this is a well known result in graph theory which is easy to show.

(Refer Slide Time: 04:01)



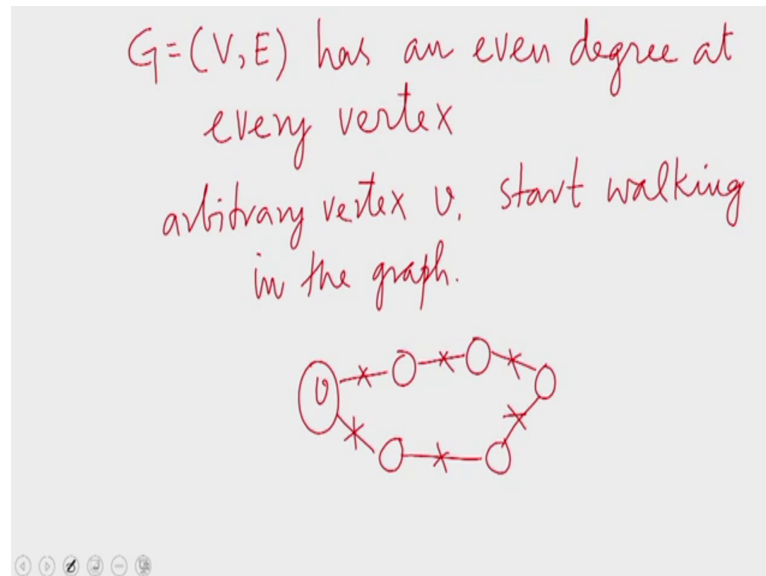
We will quickly see a proof suppose G has an Euler circuit, then let us say we start the circuit at some vertex v we start the walk at some vertex v comes to some vertex and then move on to another vertex and so on. So, this vertex w we enter through an edge and then leave through a different edge.

So, let us say we cross out every edge that we travels. So, when we pass through w, we cross we cross over two edges of w. So, w is now two edges less. So, let us say we continue doing this for every vertex that we encounter. So, every vertex that we pass through will lose two edges. So, as we pass through the Euler circuit we find that the start vertex v and the present vertex where we have reached are the only two vertices with odd degrees. Every vertex that we have passed through has lost an even number of vertices; an even number of edges. So, every vertex other than the start vertex v and the present vertex has an even degree.

Now, this is violated only when we come back to the start vertex. When we come back to the start vertex then every vertex in the graph has an even degree; every vertex that we have encountered has lost exactly two edges. So, let us say we complete the circuit in which case we have passed through every single edge and we have come back to the start vertex. So, every vertex that we encountered we entered through one edge and left through another edge. So, this is a unique pair. So, what we find is that; at every vertex in the graph the incident edges can be paired off. At the start vertex v we start through an

edge this edge will be paired off with the last edge that through which we come back to v. So, at we also we managed to do this, every edge the edges are all paired off the incident edges are all paired off which means every vertex in the graph has an even degree which proves the statement in one direction; that shows that if the graph is an Euler graph then every vertex of it has an even degree.

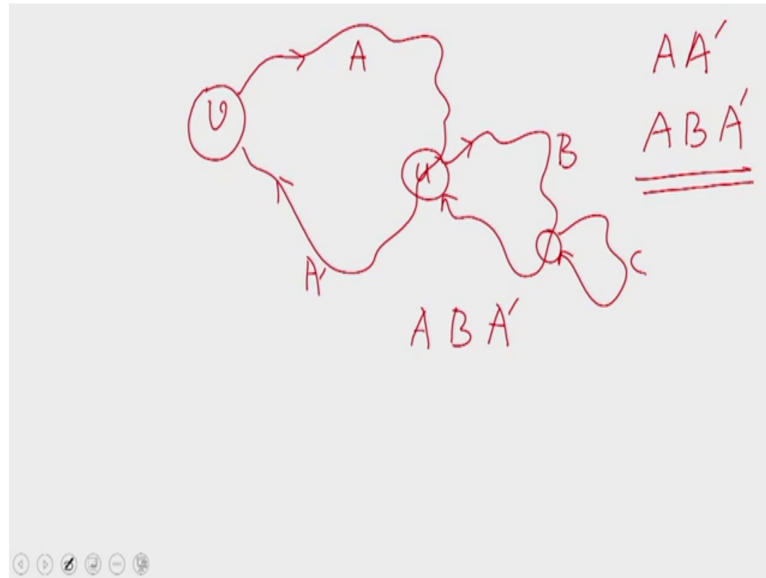
(Refer Slide Time: 06:59)



Now, proving the other way round to show that a graph that has an even degree at every vertex, such a graph is oil notice what we want to show. So, we assume that G has an even degree at every vertex. So, let us say we start with an arbitrary vertex, and start walking in the graph. Always make sure that an edge that you travels is crossed out, so that you will never use that edge again

So, you start at vertex v keep traversing while crossing the edges let us say after some time. We come back to v we have to come back to v because every vertex has an even degree. So, the travels will never get stuck whenever you enter a vertex there will be an edge that is not crossed out for you to flow out of that vertex, so eventually you will come back to v. But then when you come back to v it is not guaranteed that every single edge has been traversed, you might come back to v before traversing every single edge.

(Refer Slide Time: 08:45)



So, the situation could be like this. You start at vertex v and come back to v without traversing every single edge. All these edges have been crossed out that is they are effectively deleted from the graph, but the remaining graph is an even degree graph. Every vertex in the remaining graph still has an even degree; that is because every vertex that we pass through lost an even number of edges. If there are edges remaining in the graph then there is at least one vertex on this path which has remaining edges assuming that the graph is connected.

So, let us say u is one such vertex u is a vertex that has got remaining edges. So, let us say now we start another walk from you and then continue until we come back to u. Again it is not guaranteed that we have exhausted all the edges in the graph, all that we know is that at every vertex that is remaining in the graph still there is an even degree. Therefore, by inductive hypothesis the remaining graph ought to be Euler. Or we can argue with this way you have started from v and have come back to v if all the edges are not exhausted there at least one vertex u in the path that we in the walk that we traced which has remnant edges. So, start at that vertex u and again continue the same process until you come back to u. Then you can stitch the two paths together.

So, let us say in the original walk the walk from v to u is labeled A, and the walk from u to v is labeled A prime. And let us say this new walk from u to u is labeled B, then we can consider the concatenation of A with B with A prime which is a way of starting from

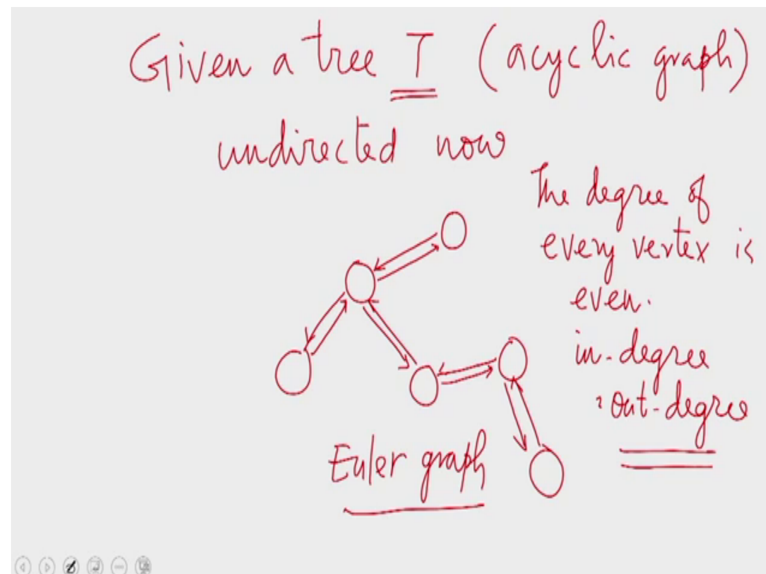
v and then coming back to v. You start from B the track A get to u and then track B which will take you back to u and then you track A prime to come back to B.

Now, this is a walk which will take you from v to v and as an enhancement of the original path A A prime. So, we are enhancing A A prime to A B A prime. So, we have found a larger walk from v to v. If there are still remaining edges in the graph there must be some vertex in this walk from which we can again repeat the process. Let us we could start with another vertex and repeat this process. When all these paths are stitched together at the point when there are no edges remaining in the graph we will have an Euler tour of the graph.

So, which shows that if the vertex degree of every single vertex is even in the graph the graph has an Euler circuit; which means a graph is Euler if and only if every vertex in the graph has an even degree.

So, we are going to make use of this principle two devise some tree algorithms.

(Refer Slide Time: 11:57)

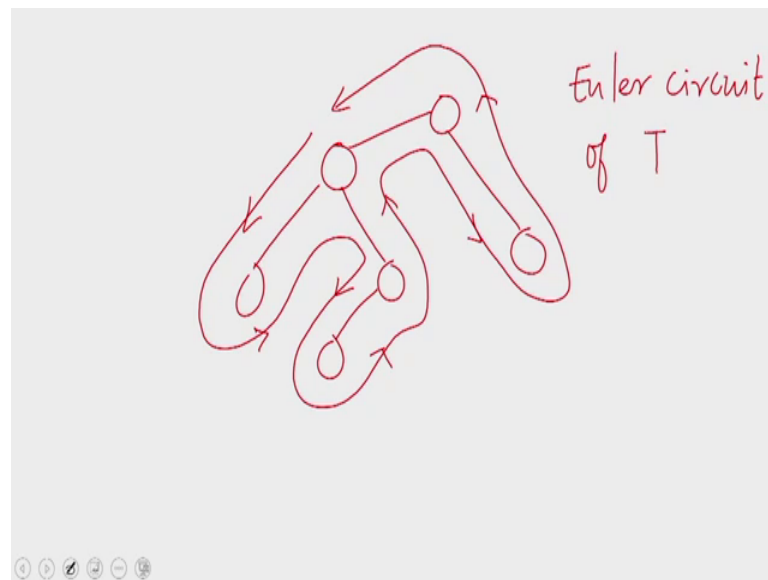


So, let us say we are given tree T. So, let us say the tree is undirected at the moment, a tree is nothing but an acyclic graph. So, let us say we are given the stream. Suppose every edge of the tree is replaced with two directed edges; that is this edge we are replacing with two directed edges in this fashion.

When we do this irrespective of the original structure of the tree; now we know that the degree of every vertex is even. That is because the in degree of every vertex is equal to the out degree. Therefore, the resultant graph is now Euler, and undirected tree converted into a directed graph in this fashion will give us an Euler graph. Therefore, we can find an Euler circuit of this.

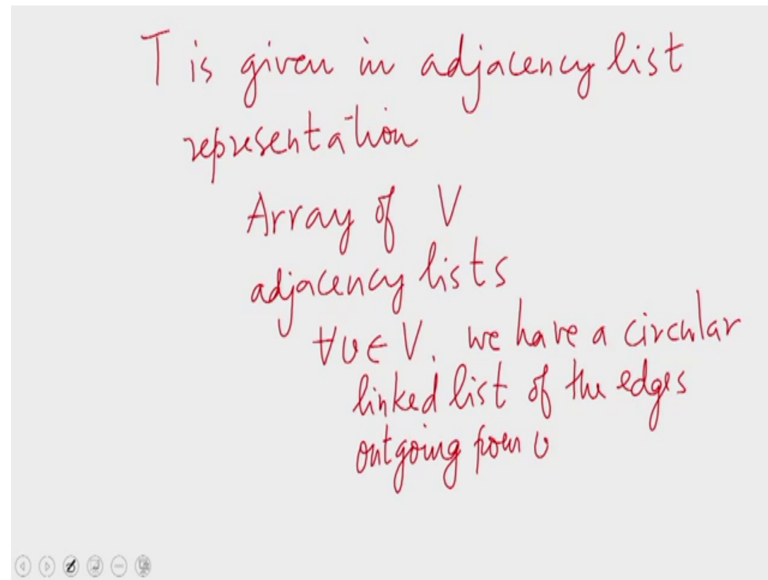
What I mean by an Euler circuit of such a tree is this.

(Refer Slide Time: 14:05)



We can traverse the tree in this fashion. Because every edges now available in both the directions we can traverse the graph in this way. This is what I call an Euler circuit of the tree. So, our first goal is to find the Euler circuit of a tree that is given in adjacency list representation.

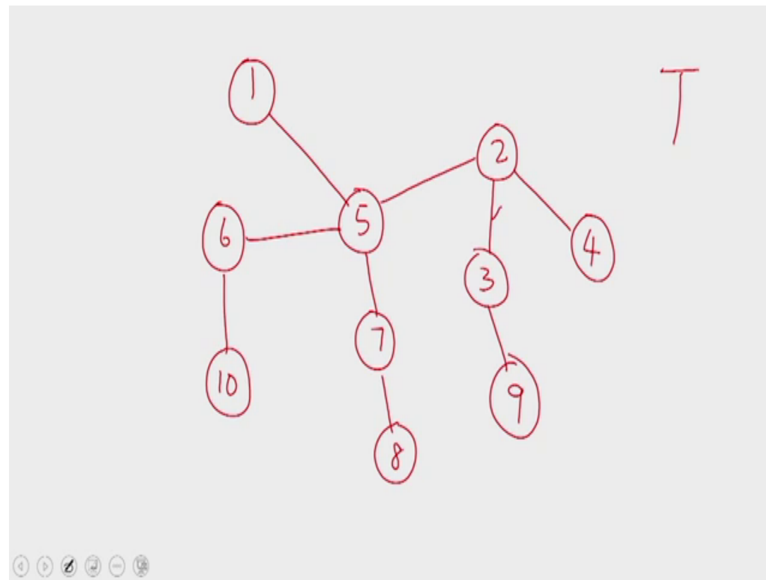
(Refer Slide Time: 14:59)



So, we assume that the tree T is given in adjacency list representation. So, in the adjacency list representation we have an array of the vertices of the graph. We have an array that represents the set V the set of vertices. And then we have what are called adjacency list entries for each V that is a vertex, we have a circular linked list of the edges outgoing from v.

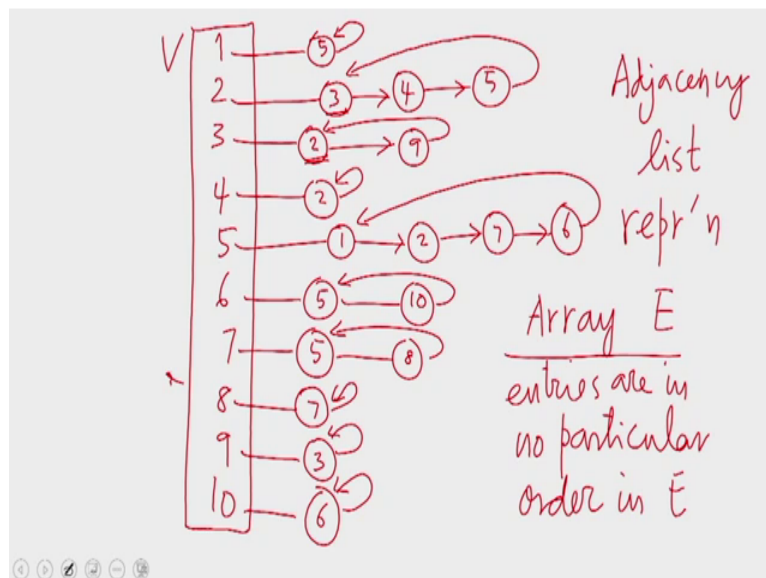
When we have such a list for every single vortex we say that the graph is given in adjacency list is representation. So, let us look at the adjacency list representation of an example tree.

(Refer Slide Time: 16:39)



So, let us say this is the graph that we have which is a cyclic and therefore it is a tree. So, given, this tree in its adjacency list representation which looks like this.

(Refer Slide Time: 17:17)

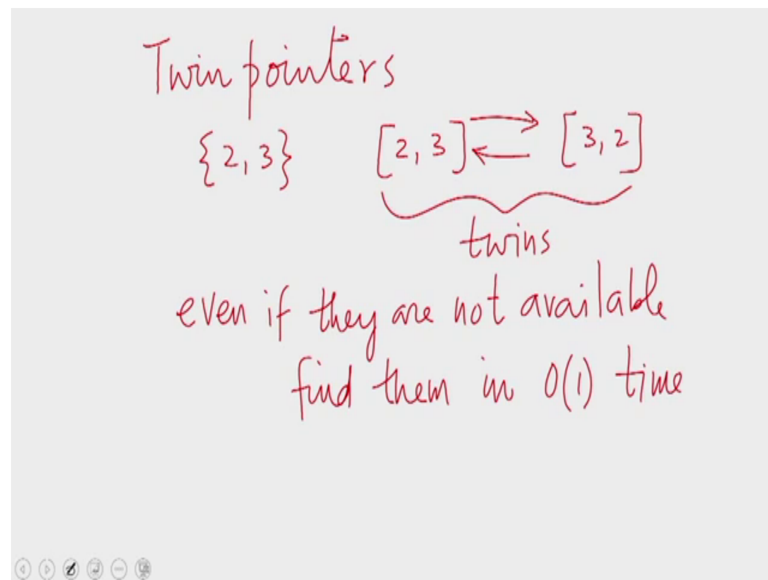


There are 10 vertices in the tree. So, let us say these vertices are available to us in an array, each array element holds a pointer one has only one incident edge which is the edge from 1 to 5. So, it holds only element 5 which is in a circular linked list containing itself. 2: has three neighbors 3 4 and 5, we assume that all these pointers are flowing to the right. 3: has 2 and 9. 4: has 2 acids only neighbor. 5: has 1, 2, 7, and 6, so its

neighbors. 6: has 5 and 10. 7: has for 7 has 5 and 8. And 8 has 7 alone, 9 has 3 alone, and 10 has 6 alone.

So, this is the adjacency list representation of the graph we saw in the previous screen. So, this is the tree T of which we have the adjacency list representation. So, we can actually see that this adjacency list representation, in fact is off the tree in which every edge has been replaced with two directed edges. For example, consider the edge between 2 and 3 in the in this picture that is this edge the edge from 2 to 3, when we replace this with two directed edges we have one edge from 2 to 3 and one edge from 3 to 2. So, when you look at the adjacency list of 2 you find that the first entry is 3, which is which corresponds to a directed edge from 2 to 3. And when we look at the adjacency list of 2, we find that the first entry is 2 which corresponds to the directed edge from 3 to 2. So, the undirected edge from 2 to 3 is replaced by 2 directed edges from 2 to 3 and 3 to 2 respectively.

(Refer Slide Time: 20:29)

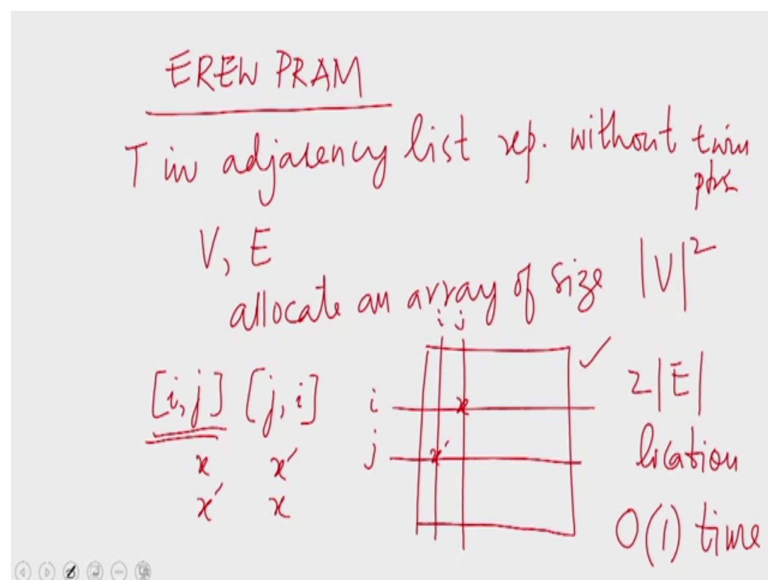


We additionally assume that the adjacency list representation has twin pointers. Twin pointers are available between the twins. For example, we saw that the undirected edge 2 3 is replaced with two entries; 2 3 and 3 2 in the adjacency list representation which are this element and this element respectively. We call them the twins of each other. A twin pointer disappointed between the twins

So, what we assume is that all the twin pointers are available which means 2 3 the entry 2 3 points to the entry 3 2 and the entry 3 2 points to the entry 2 3. That is these two underlined entries point to each other. So, here in the adjacency list representation we assume that the set of vertices v is available in an array. The adjacency list entries are all available in an array E, but we assume that the entries are in no particular order. That is a physical order in which they are given in the array E has no particular relation to the logical order. The logical order is what we have drawn here. So, they are all jumbled together in array E. And these entries have twin pointers, which means the entry to three will point to the entry 3 2 and the entry 3 2 will point to the entry 2 3.

So, if we do not draw the twin pointers here because that will clutter the picture. So, what we assume is that the twin pointers represent here. Even if the twin pointers are not available, we can find them in order 1 time. What we do is this.

(Refer Slide Time: 23:11)



On an EREW PRAM suppose we have T in the adjacency list representation without twin pointers. We have the adjacency list representation without the twin pointers, which means we have the two arrays V and E within E we do not have the twin pointers. So, to take the to find the twin pointers what we do is this we allocate an array of size V squad which we visualize as a two dimensional array. In this array first what we do is this suppose we have as many processors as there are edges in the graph. So, we station those

processors in the array E; the processes that we have are stationed on the adjacency list entries.

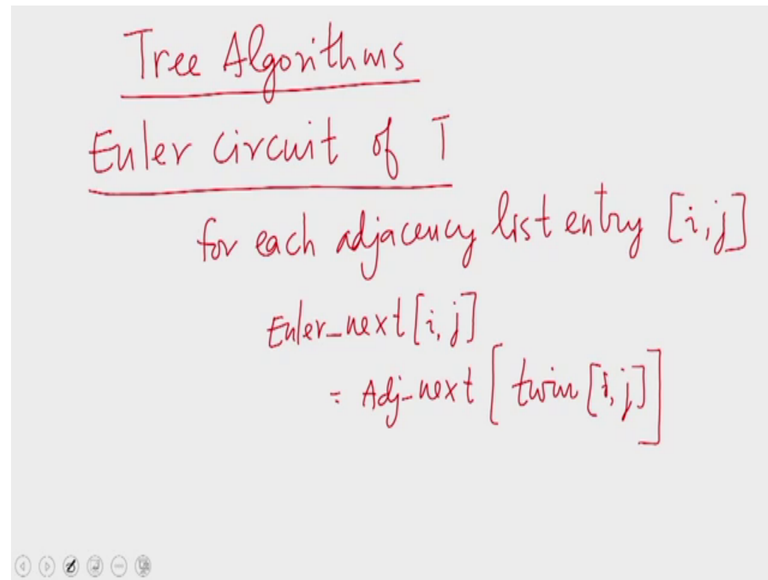
So, for the twins i j and j I; for every edge i j. For the twins i j and j i let us say their actual locations within E are x and x prime. That is the entry i j is in the x-th location of E and the entry j i is in the x-th prime location of E, then the processes sitting at i j and j i we will do this. The processor sitting on i j we will go to the i j-th location of this two dimensional array and write x-there. At the same time the process of sitting on the j i-th location; we will write x prime in the j i-th location of the two dimensional array. The process sitting on entry j i we will write x prime in the j i-th location of the two dimensional array.

Now these two writings happen simultaneously and since the rights are in different locations there is no write conflict. Now, the processors will interchange the processes sitting on i j will access the j i-th location of this two dimensional array from where it will read off x prime. So, i j will now get the value x prime. At the same time the process sitting on entry j i will be reading the location i j, from where it will get the value x. So now, the i j-th entry knows that its twinness at location x prime and the j i-th entry knows that it is twinness at location x; that is precisely what we want when we have twin pointers.

So, we have now managed to establish the twin pointers with the help of a two dimensional array. All we had to do was to allocate an array of size mode V squared, but since we did not access every single location of this array the actual use of the array is restricted to mod E locations or twice of that. For every edge we have two locations to be accessed, so we end up using exactly twice as many locations in this array, and the time required for establishing the twin point does this order of 1.

So, as I said before even if the twin pointers are available we can find them in order of 1 time on an EREW PRAM with exactly as many processes as there are edges. So, we assume without loss of generality that the tree is available to us in the form of an adjacency list with twin pointers.

(Refer Slide Time: 27:49)

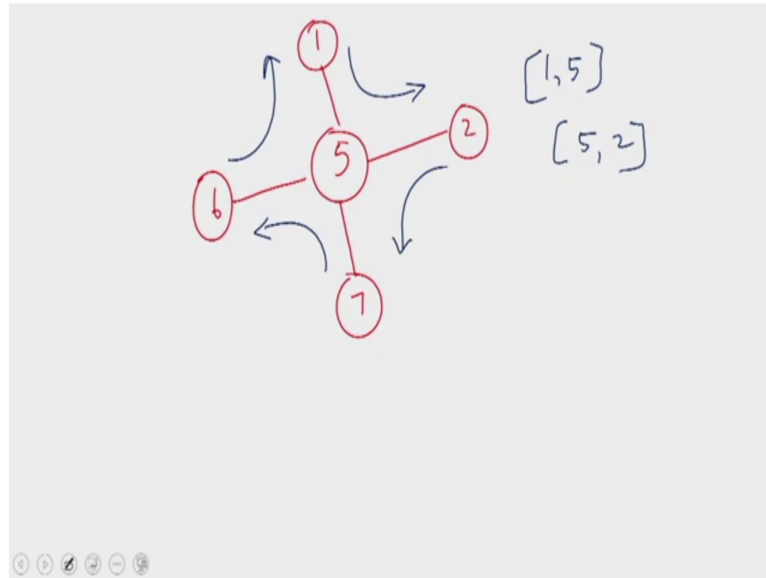


Now, let us come to some tree algorithms. The first algorithm is to find an Euler circuit of T. To find an Euler circuit of T what we do is this: for each adjacency list entry i j we define the Euler circuit next of i j as the adjacency list neighbor of the twin of i j.

Suppose the we do this for every single adjacency list entry simultaneously. So, assuming that we have one process station that the adjacency list entry i j what we do is this: we access the twin of i j since the entry i j has a twin pointer we can access the twin in order 1 time. Once we have the twin we take its adjacency list next finder the same address will be the address of the Euler circuit next of i j. That is after traversing i j in the Euler circuit you will have to traverse the twin of the adjacency list neighbor of j i.

So, let me make this clear with an example.

(Refer Slide Time: 29:51)

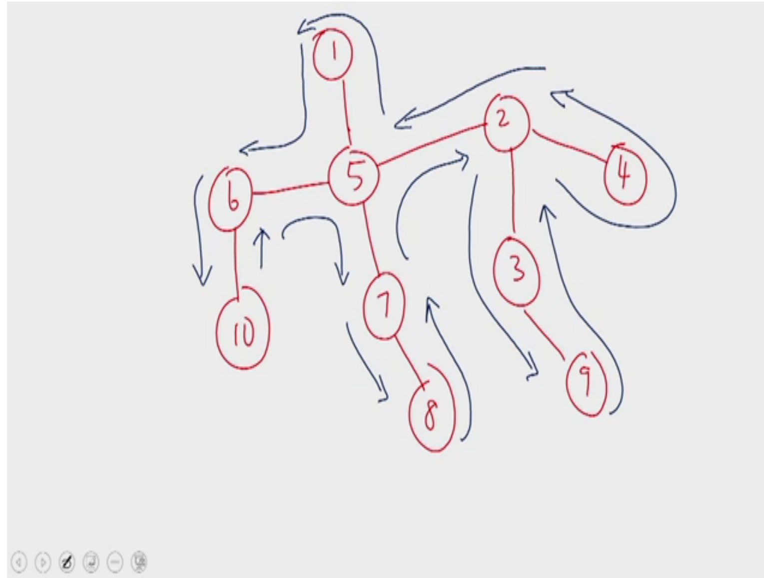


In our graph we have this vertex 5 which has four neighbors. So, the four neighbors of 5 are 1, 6, 7, and 2. So, here what we say is this: for the edge 1 5 the edge going from 1 5 the directed edge from 1 to 5 the Euler circuit successor has to be the adjacency list successor of the twin of 1 5 the twin of 1 5 is 5 1. So, if you look at the adjacency list of 5 you find that its successor is 2; the successor of 1 is 2, and the successor of 2 is 7, and the successor of 7 is 6, and the successor of 6 is 1. So, accordingly here if we set the pointers we find that we should set the pointers in this order.

So, what this means is this: for the directed edge from 1 to 5 the successor has to be the directed edge from 5 to 2. So, 1 to 5 has 5 to 2 as its successor which means if you come into the vertex through the edge 1 to 5 you have to go out through the edge 5, 2. So, we are now setting up a way for the Euler circuit to flow through 5 when it comes in through the edge 1 5 it has to go out through edge 5, 2. Then later on you might come into vertex 5 through the edge 2 5 and then you will have to flow out through the edge 5, 7. Later when you flow in through 7, 5 you have to flow through flow out through 5, 6. And then when you come in through 6, 5 you have to go out through 5 1.

So, this is how we are setting up the Euler circuit at vertex 5. So, at every vertex in this manner we are pairing off the incoming edges with the outgoing edges. So, let us see what this means for the entire tree.

(Refer Slide Time: 32:01)



So, this was our original graphs. So, when we set up the Euler tour in do spanner we find that: the flows are in this direction which will take us back to the original point. So, when you start from vertex 1 you pass through vertex 5, 6, 10; and then back to 6 then 5, 7, 8; and then back to 7, 5; and then 2 3, 9; and then back to 3 2, 4; and then back to 2; then back to 5; and back to 1; and that completes the Euler circuit.

So, when you stitch up the pointers in this fashion you obtain an Euler circuit of the graph.

(Refer Slide Time: 33:35)

$O(1)$ time using $|E|$ processors

T is a tree

$|E| = O(|V|) = O(n)$

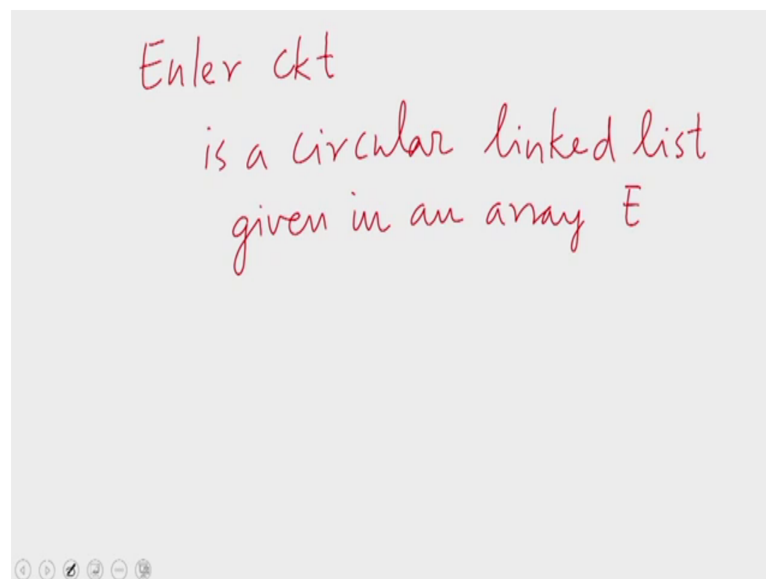
n processors $O(1)$ time

EREW PRAM

Which we have managed to do in order 1 time with one single processor for every adjacency list. But then since T is a tree its edge set this of the same order as its vertex set in size if we denote the size of the vertex set by n we see that mod E is order of n.

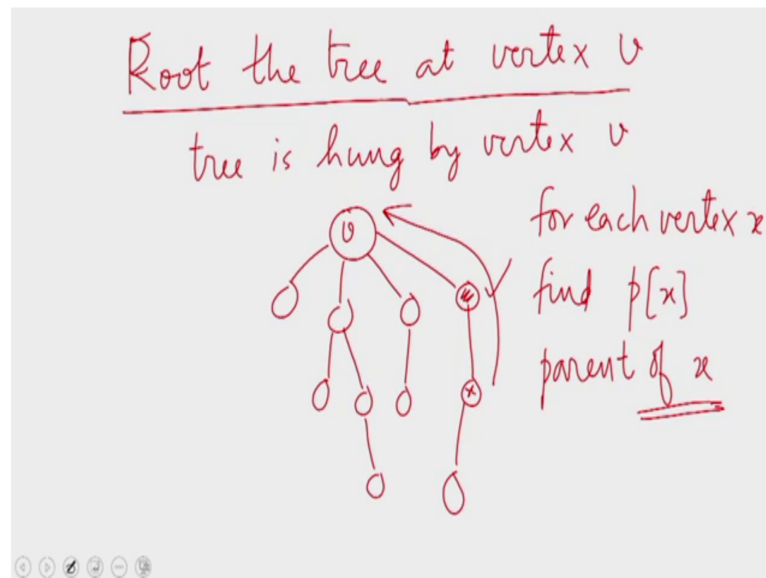
So, we require order of n processors and the algorithm runs in order 1 time. Therefore, with n processors we can say the algorithm runs in order of 1 time; on an EREW PRAM. This is because we have in used concurrent reads or concurrent writes anywhere. So, the Euler circuit of the tree can be found in order of 1 time on an EREW PRAM. So, the Euler circuit is essentially a circle circular linked list made up of the adjacency list entries.

(Refer Slide Time: 34:55)



So, this you have to understand: the Euler circuit that we have found is a circular linked list given in an array T, which contains the adjacency list entries. So, we have managed to stitch the original adjacency list entries through a new set of pointers and this stitched up circular linked list is what the Euler circuit is.

(Refer Slide Time: 35:45)



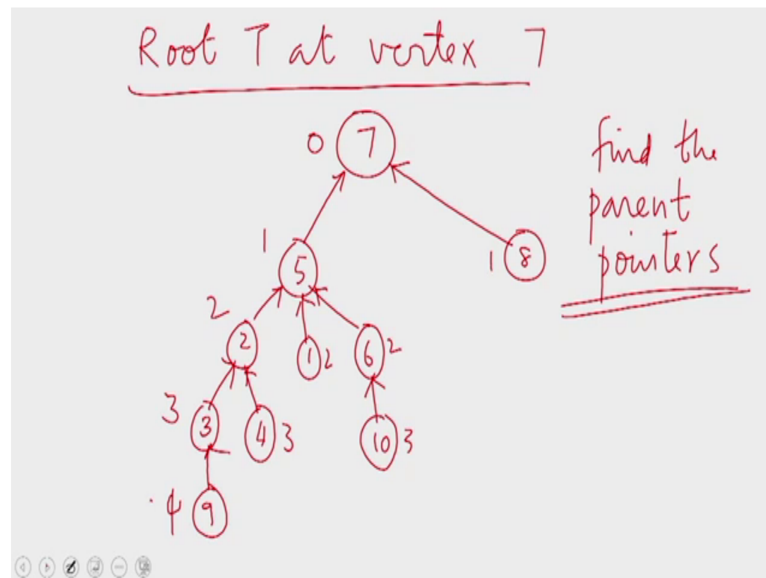
Now, let us see how we can make use of this Euler circuit to achieve various ends.

So, the first thing that we want is to root the tree at some vertex v. When we root the tree at vertex v what we assume is that the tree is hung by vertex v. Therefore, all the paths that lead you away from v are hanging down; let us say by gravity. So, vertex v is at the top, vertex v is children are all at a level below v, the grandchildren our father below and so on. So, we assume the tree is hung by vertex v on a nail.

So, when we root the tree what we require is this for each vertex x find p of x the parent of x. The parent of a node x is the first vertex on the path from x to vertex v. So, the parent of this node x is this one. That is on the path from x to v this happens to be the first vertex after x.

So, likewise we want to find the parent of every single vertex x, when we do this we say we have rooted the tree. So, let us continue with our original example and see how the tree will look like when we have the rooted it at vertex 7.

(Refer Slide Time: 37:45)



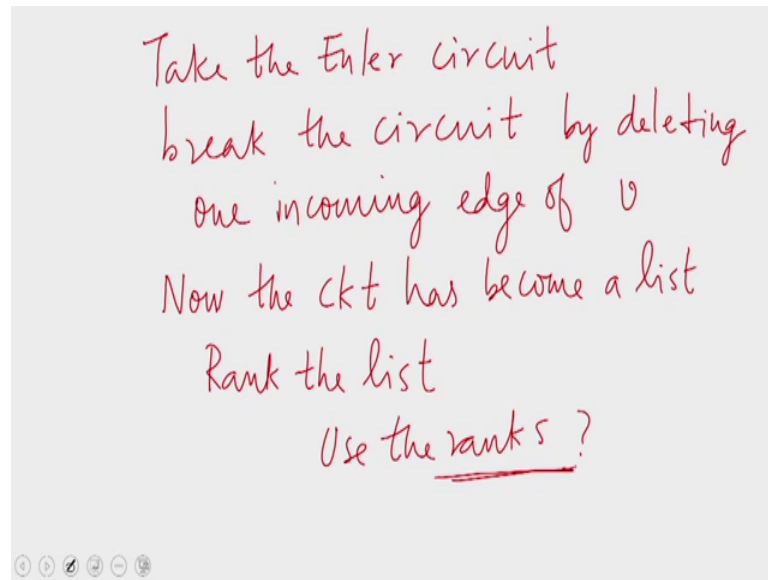
So, for the same graph we can now redraw the diagram with 7 at the top. And 7s neighbors 5 and 8 one level below, and then 5s neighbors array from 7 will now be the grandchildren of 7; they will be one level further below. The neighbors of two father from 5 are 3 and 4 they will now be children of 2. 6 has one child 10 which is father from 5 that will be the child of 6. 3 has one child 9.

So, when we arrange the vertices in this manner we have rooted the tree at 7. Now the parent pointers will can be defined in this manner. The parent of 10 is 6; the parent of 6 is 5 and the parent of 5 is 7 and so on. So, all the parent pointers are now flowing from bottom up. So, when we root the tree at a vertex what we require is to find the parent pointers at every single node. Find the parent pointers S shown in this figure. So, this is the problem we have at hand. How do we solve this using the Euler circuit.

So, what we have is this the tree T is given an adjacency list representation, we can assume that twin pointers are available, then with end processes we can find the Euler circuit of this in order of 1 time. Now, with the Euler circuit available with us how do we find the parent pointers when we root the tree at a particular vertex v.

What we do is this.

(Refer Slide Time: 39:57)

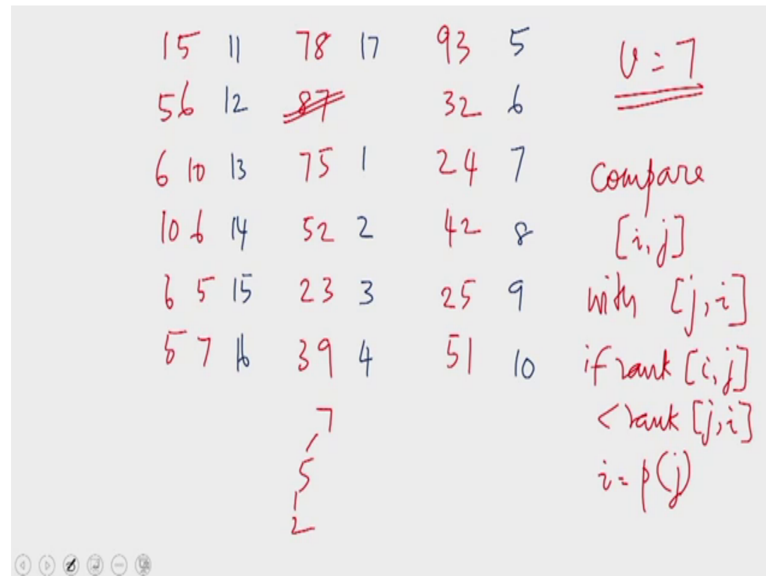


Take the Euler circuit break the circuit by deleting one incoming edge off vertex v. Now the circuit has become a list. The list has a length of order of n because the list is made up of the adjacency list entries. So, exactly one edges missing from the Euler circuit which is an edge that is incoming into vertex v.

Now, once we have this list we can rank the list. So, this is not a circular list anymore, this is a regular list let us say we rank the list. And then how do we use the ranks to root the tree?

So, I claim that the ranks can easily root the tree. So, let us see how this can be done.

(Refer Slide Time: 41:39)



In our example the Euler circuit we had was this. This was the Euler circuit that we have, which is a circular linked list. So, in this Euler circuit the successor of 1 5 is 5 6; the successor of 5 6 is 6 10 and so on. And then the successor of 5 7 is 7 8; the successor of 3 9 is 9 3. And finally, the successor of 5 1 is 1 5 closing the circuit.

So, let us say we break open in this circuit by deleting one incoming edges one incoming edge into vertex 7. So, vertex v here is vertex 7. So, let us say we break open the circular linked list by deleting one incoming edge of vertex 7. So, let us say we choose to delete this 8 7. And then we rank the remaining list.

So, the list is now starting with vertex 7 5 if we rang this the rank will be like this; we rank the edges in this manner. And then let us say we compare every vertex i j with its twin. Let us look at the ranks of i j and j i. In the Euler circuit we see that when we root the tree at 7 consider the edge 5 2 in this case. So, the edge 5 2 will be traced before we trace edge 2 5. So, when we compare the edge 5 2 with its twin to 5 we find that 5 2 has a lower rank than 2 5.

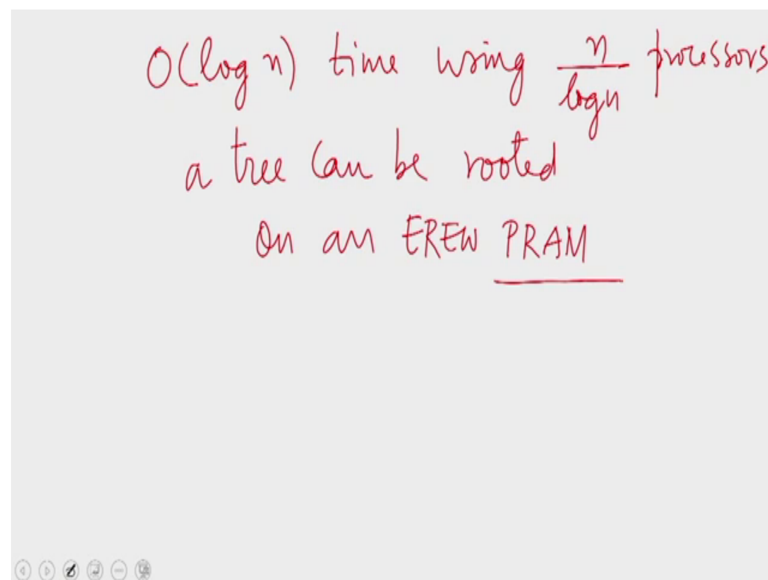
Therefore, 5 is closer to 7 then 2. On other words when we root the tree at 7 the edge from 5 to 2 must be a parent child pointer. Whereas, the edge from 2 to 5 is a child to parent point, which means 5 has to be the parent of 2. So, that gives us the idea that when an edge i j is compared with its twin and if we find that i j has a rank which is less than

the rank of j i then i is the parent of j. If the rank of I or rank of i j is less than the rank of j i, then i is the parent of j.

So, accordingly here when we look at vertex 7 5 we can consider a vertex 5 7; 5 7 has a higher rank which means 7 is the parent of 5; 7 is the parent of 5. When we consider what take the edge 5 2 which has a rank of 2 and its twin 2 5 which has a rank of 9 we find that 5 is a parent of 2. So, if you write the parent relation in this fashion you will find that what you get is exactly what we had before; namely this, which means the algorithm is now clear enough when we are given a tree T in adjacency list representation with twin pointers. All we need to do is to find the Euler circuit and then delete an incoming edge of the designated route, and then the Euler circuit becomes a regular linked list you rang the linked list.

And then for every edge i j compare the rank of i j with the rank of j I; the twin of i j if the rank of i j happens to be less than the rank of j i then I is the parent of j otherwise j is the parent of i. So when we do this, we would have root at the tree which means the cost of rooting the tree is order of n and the time required is order of log n.

(Refer Slide Time: 46:23)

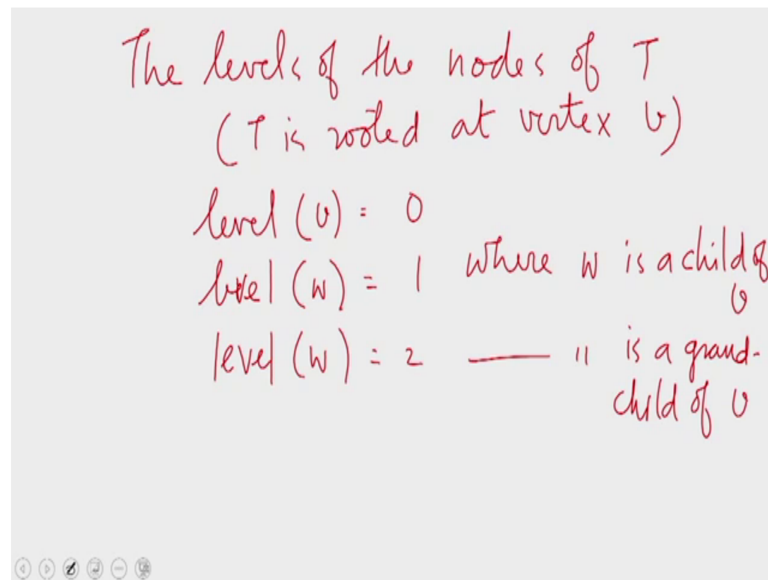


So, in order of log n time using n by log n processors a tree can be rooted on an EREW PRAM. Here the costliest step is that of ranking the linked list; after finding the list we have to rank the list for which we use the optimal algorithm that we studied in the previous class that runs in order of log n time using and by log n processes. Every other

step can be executed an order of 1 time using n processes, which means the same as executing in order of log n time using n by log n processer. So, if we have n by log n processes every single step can be executed not of log n time. Therefore, the total time taken by the algorithm is order of log n.

So, this was the first three algorithm that we saw today that of rooting the tree. Now, let us consider another problem.

(Refer Slide Time: 47:37)



Let us say we want to find the levels of the nodes of T. So, here we assume that T is rooted at vertex v. So, the level of vertex v is defined as 0 the level of w is 1; where w is a child of v. For every child of v the level is defined as 1, and for every grandchild of v the level is 2. This is how we want to define the levels.

So, how can the levels be defined now? The levels can be defined again with a single invocation to list ranking.

(Refer Slide Time: 48:55)

Let us consider the Euler 2 starting at 7 5. So, this was our Euler 2, after deleting the edge 8 7. And then let us say we assign values in this manner for parent child edges, we assign a value of one a rank of initial rank of 1. For child parent edges we assign a value of minus 1 an initial rank of minus 1. Then 7 5 is a parent child edge 5 2 is a parent child edge, 2 3 is a parent child edge, 3 9 is a parent child edge, but 9 3 is the other way; 3 2 is the other way; 2 4 is a downward edge; 4 2 is an upward edge; 2 5 is in upward edge 5 1 is a forward edge, and 1 5 is an upward edge, 5 6 is a forward edge, 6 10 is the forward edge, 10 6 is an upward edge, 6 5 is an upward edge, 5 7 this an upward edge, and 7 8 it is a forward edge.

So, let us say these are the initial ranks of the list and then using the list ranking algorithm we can find the prefix sum of these values. So, if you find the prefix sum you find that the sums are these. So, we find the prefix sums in this manner. So, let us see what exactly happens here, this will be clear when we look at the root at tree let us look at the diagram of the root at tree.

We start from vertex 7 when we go down to 5 we count up. So now, our count is one when we go from 5 to 2 we count up once again. So, the count is 2 here at vertex 2, when we come to vertex 3 are count is 3, when we come to vertex 9 or count is 4. And then when we go back to 3 we count down again or countless again three which matches which the with the original count we had obtained. We go back to our countess, now two

we go down our countess three again. So, four all vertical at vertex 4 also we have we get a count of 3. And then back to 2 the count is 2 back to 5 the count is 1, down to 1 the count is 2, back to 5 the count is 1. And then down to 6 the count is 2, again down to 3 the count is 3 again, and then going backwards when we reach 7 the count is 0 when we come to 8 the count is 1, and then the list ends.

So, when we assign counts in this manner what we find is that whenever we visit the vertex visit a vertex, the count at that point is equal to the level of the vertex. So, that is all we have to do now. After finding the prefix sums in this manner all we have to do is to use the prefix sum to find the rank of every single vertex. So, the process of which is sitting on the node; the parent child node coming into a vertex can assign its level. So, look at the edge 7 5 the edges from 7 to 5. So, this is a parent child edge coming in to 5 and it has a value of 1. So, this edge can now decide that vertex 5 has a level of 1. Similarly 5 2 is a parent child edge with our rank of 2. Therefore, this edge can now decide that vertex 2 has a rank of 2.

So, when you decide ranks in this manner there will be no conflict there is only one parent child edge coming into a vertex and the process of sitting on that edge can decide the rank of the level of the child. The rank that we obtained when we commuted the prefix sums is identical to the rank the level of the child. So, the algorithm establishes that vertex 5 is at level 1 vertex 2 is at level 2. Similarly consider another parent child edge which is 5 6, which decides that 6 has a level of 2 and so on. So, in this manner we can find the level of every single vertex.

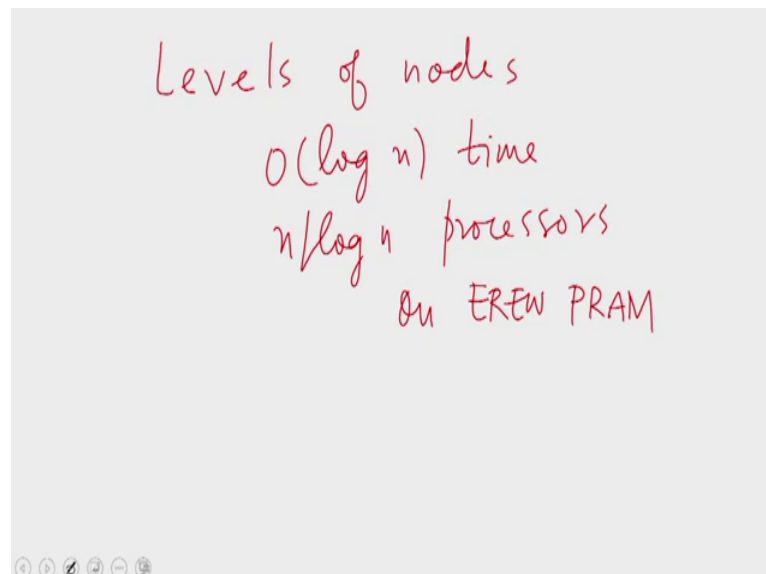
So, once again to summarize to find the levels first we have to root the tree. Once the trees rooted, we take the same Euler circuit with one edge deleted, but here now we initialize the weights differently for every parent child edge we assign a rank of 1, and for every child parent edge we assign a rank of minus 1. With these initial ranks assigned we perform a list ranking once again, and then we get a rank for every single vertex from the list ranking algorithm. And then for every single parent child edge the rank of that edge will be assigned as the level of the child node; that is the destination node of that edge.

So, in this example we had edge 5 2 with a rank of 2. Therefore, vertex 2 is assigned a level of 2. So, in this manner we can find the level of every single node. The level that

we get at every single vertex is precisely the distance from the root to that vertex. That is because, for every forward edge we are counting 1 and for every backward edge we are counting a minus 1. Therefore, when we come to one single vertex all the paths that; all the deviations that we took from the path from the root to that node have been nullified because of the reversals we had. That is any forward movement along any such deviation would be neutralized by the backward moment when we came back to this path.

Therefore, the rank that every single node will be exactly the distance from the root vertex v to that vertex.

(Refer Slide Time: 56:03)



So, this establishes that the level of every single node in a tree can be found in order of log n time using n by log n processors on an EREW PRAM. This algorithm we obtained as a further extension of the rooting algorithm we saw before.

So, in the next class we will see some more tree algorithms. This is it from this lecture, hope to see you in the next lecture.

Thank you.