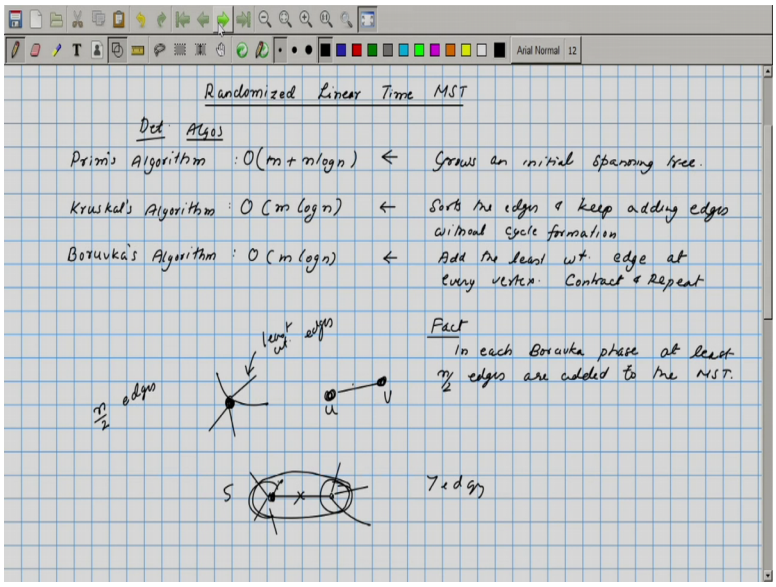
Randomized Algorithms
Prof. Benny George Kenkireth
Department of Computer Science & Engineering
Indian Institute of Technology, Guwahati

Lecture - 28
Randomized MST

In today's lecture, we will learn about Randomized linear time Minimal Spanning Tree algorithm.

(Refer Slide Time: 00:32)



So, the standard deterministic algorithm for solving the minimal spanning tree are say the Prim's algorithm and then there is Kruskal's algorithm and the third algorithm is the Boruvka's algorithm; these are the standard deterministic algorithm. So, Prim's algorithm if you remember it is about taking an initial spanning tree and then grow in it.

So, that if you do the correct implementation there algorithm can run in m plus n log n time. Whereas, Kruskal will run in order of m log n and Boruvka would also run in O of m log n. So, this algorithm basically grows an initial spanning tree. So, let us say you have the part of the spanning tree to start with and then that defines a certain cut, you add the least weighted cut edge keep on doing it repeatedly you will get Prim's algorithm. Whereas, kruskal algorithm you have an initial forest you sort all the edges and to keep on adding the edges one after another under the condition that the added edge do not introduce a cycle ok.

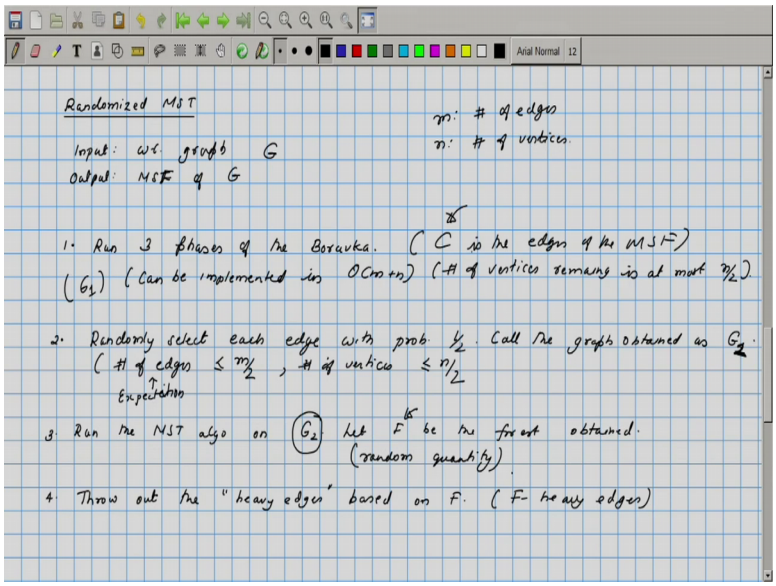
So, this basically sorts the edges and keep adding edges without cycle formation; Boruvka's algorithm on the other hand is you can think of it as a parallel algorithm at every vertex add the least weighted edge at every vertex. And then you let us say cluster it or you can contract the graph and repeat. So, two when you consider the least weight weighted edge at any particular vertex; say if you are looking at this particular vertex there might be lot of edges out of it. And the least weighted edge this can be done for every vertex because if you imagine this vertex as a single element and everything else in the remaining remainder of the graph.

So, if you think of the cut that is obtained by considering this vertex by considering there is a single vertex. This edge becomes the least weighted edge out of that vertex becomes the least weighted cut edge which by cut property will surely belong to every minimal spanning tree; assuming that the graph has distinct weights on all the edges. And this could be done for all the edges simultaneously it could happen that the least weighted edge is shared between two vertex; for example, this u v edge could be the least weighted edge for u, as well as for v in that case you will get only one edge instead of one edge for every vertex you will get in the worst case one edge for every two vertex.

So, you will get at least n by 2 edges when you do this in one; so, let us call that as the Boruvka phase. So, in each Boruvka phase at least n over 2 edges are added to the MST. So, you can keep on doing this till; so once you add these edges two vertices which share an edges being contracted you get a super vertex and then keep on repeating this for the new vertices formed. So, if you had let us say 5 vertices 5 edges out of this and you had let us say 4 vertices out of this and you contract them this edge is common that is gone.

And then there are 1, 2, 3, 4, 5, 6, 7 edges out of the block that is formed by or the super node that is formed by combining these two vertices. So, that is a Boruvka phase why be looking at it Boruvka is algorithm will become a crucial ingredient of the randomized linear time minimum spanning tree algorithm.

(Refer Slide Time: 06:09)



So, I will first describe the algorithm and then we will look at why this algorithm runs in expected linear time. So, let us call this as randomized MST ok. So, input will be a weighted graph and the output will be the MST; it could even be a weighted forest in which case we will be get a minimal spanning forest. So, output will be a we will think of it as a singly connected graph. So, the output is a minimal spanning tree of G. So, the first step is we will take this input graph and run the Boruvka algorithm on it 3 times. So, we will write as run 3 phases of the Boruvka's algorithm ok. So, let us add some commentary to the step; note that you can implement each Boruvka phase in linear time ok.

So, 3 phases would essentially mean again linear time; so can be implemented in O m plus n ok; m here is the number of edges and n is the number of vertices ok. After 3 Boruvka's phases are run each phase means you will look at all the vertices simultaneously and for each of them you add the nearest neighbor and then you contract them. So, that contraction is also included in a phase; so that entire thing can be executed in it is a linear time ok.

Now, after this has been done number of vertices remaining is at most n over 2 ok. So, why would be run this Boruvka's phase. well one thing is we could get a saving in the number of vertices, but the edges could still be large. So, what we do is we will throw away some of the edges ok. So, this is the only place where randomization really comes

in into this algorithm. So, we will randomly select the edges of this reduced graph. So, we can think of it has you toss a coin and the bias let us say it is p; we will choose P to be equal to half the randomly select each edge with probability half. So, you toss a coin, but bias half the coin results in a head then you include that edge otherwise you throw away.

So, we will get a graph and that graph we will all that as called the graph obtained we say G 1; our original graph was G, after we have shrunk we will get some particular graph let us call that as G prime or we will call this as G 1 and here whatever graph we get we will call it as G 2. Now note that G 2 can have at most n by 2 vertices or the number of edges is also reduced because the expected number of edges will now be half of the number of edges in G 1.

So, if G 1 contains at most m edges; then we can say a number of edges is less than I mean; so here when I say this is in the expected sense. So, expectation is less than n over m by 2 and number of vertices is also less than let us say n by 2. So, first step was to run 3 phases of Boruvka, there we would have got a forest we will call that as let us say C is the edges of the MSF; the edges that we will gather towards making our minimal spanning tree in the Boruvka phase those we will call as C.

And now we have reduced graph which we are calling it as G 2 ok. Now G 2 is a sub graph of the contracted graph we will try and construct the minimal spanning forest for G 2; G 2 may not be connected G 2 could be connected, but whatever it is we will just run the MST algo for; I would say on G 2 ok. So, now, the MST algorithm it is basically giving a minimal spanning forest. So, we should have been little more careful and said that this is a weighted yeah weighted graph; it need not be connected and the output is a minimal spanning forest. If G was a connected graph then the output would have been a minimal spanning tree ok.

So, we will run the MST algorithm on G 2 we will get a forest ok; so let F be the forest obtained. Now this F is a random quantity in the sense depending on the choices that we make in step 2; we could get different forests. In the second step we were selecting a sub graph of the original graph and for that sub graph there is a corresponding unique forest.

We can say unique because we will assume that the weighted graph that is initially given to us contains distinct weights on every edge. And therefore, any sub graph will also have that property and therefore, when we run the minimal spanning tree algorithm that

is this same algorithm when we run it recursively; the forest that we obtain will be unique forest. But this is unique only for the random choices that we have made; if you fix the random choices that we have made then for each such fixed choice we will get a unique forest; now that forest therefore, as a random quantity.

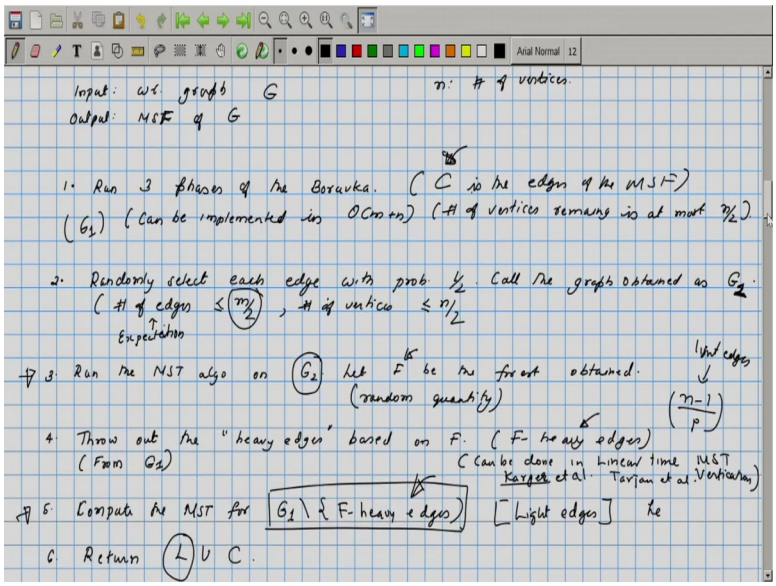
Now, once this forest has been fixed what we will do is; there are certain edges that we can throw away. Looking at the algorithm let us wonder why this algorithm might be a good algorithm? When does it not give linear time algorithm; that might be because there are a large number of edges, but here we are running the algorithm on reduced number of edges we run the minimal spanning tree on a smaller graph ok.

So, it might be possible to do it in less time, but after running why is this useful? After we have got this particular forest what we can do is; we can throw out certain edges ok. These are those edges whose addition will create a cycle in the forest and those edges in the cycle that we are considering those edges are going to be the heaviest edges. So, we will call these things as the heavy edges we will later on define more properly what are heavy edges.

So, right now I will just write down the algorithm. So, throw out the heavy edges; what are heavy edges? We will later on provide a definition based on F ok. So, we will call this as the F heavy edges; the key being us these F heavy edges will not be present in any minimal spanning tree.

So, it is safe to throw that out; so it does not depend on our choice of F edges that can be safely thrown out. The choice of which are the heavy edges might depend upon the particular forest that we had picked, but once we have identified that a particular edge is heavy for some particular forest that is a edge that surely is not going to be present in the minimal spanning tree; this we will see we will see a proof of it later.

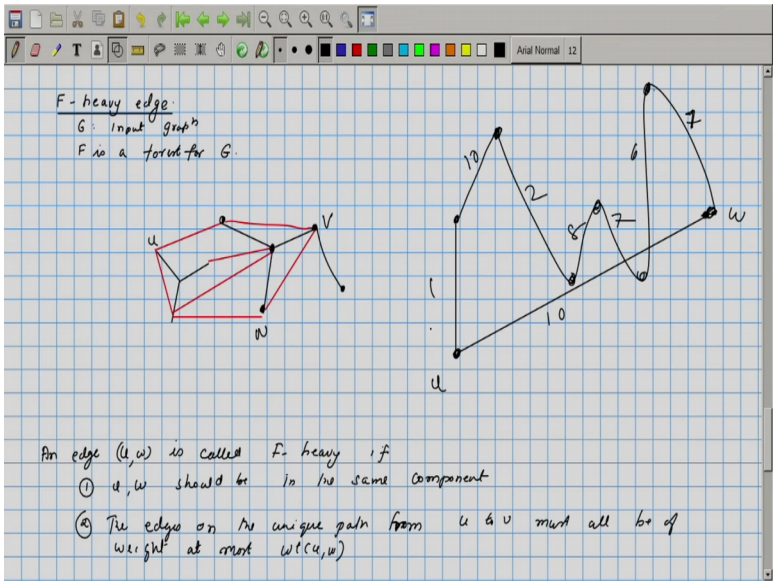
(Refer Slide Time: 15:18)



And once we have thrown out the F heavy edges the remaining edges are called as F light edges. So, these edges are being thrown out from G; from the original say graph G 1. So, now once the heavy edges F thrown out; we can compute the MST for ok; so these we will call as a light edges ok. So, you compute the minimal spanning tree for the light edges and then return. So, light edges if you call it as let us say L; we will return L union C that is the edges that we initially picked as the minimal spanning forest ok. This is always return correct answer; well these edges C had to be present in all minimal spanning trees.

Further this is the contracted graph and from the contracted graph the edges that we have thrown out are the heavy edges and we will see that heavy edges deserved to be thrown out. So, on the remaining graph if you compute the minimum spanning tree; though the new edges that you will obtain are L all of them need to be present. And therefore, if you return L union C that will be a minimal spanning tree of G. We need to figure out what exactly are these heavy edges once we figure out heavy edges deserved to be thrown out; we need to analyze this algorithm and see what is the overall running time.

(Refer Slide Time: 17:02)

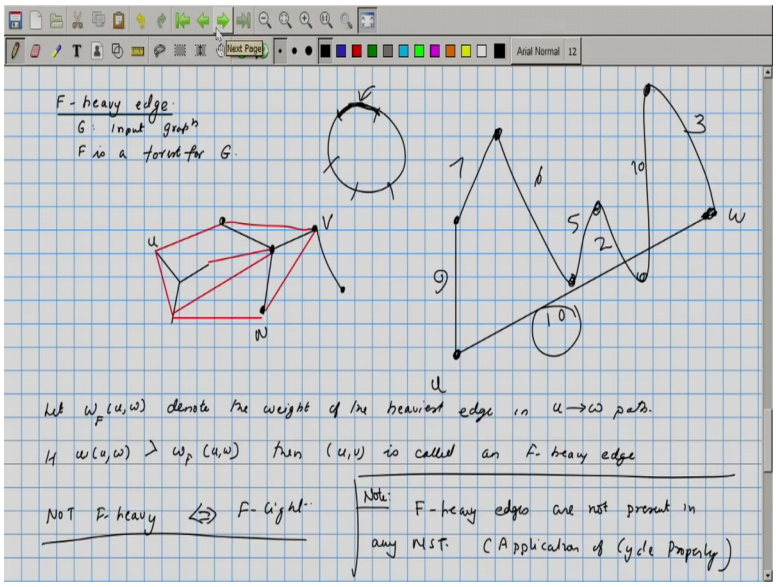


So, let me first provide you the definition of F heavy edges ok. So, F heavy edge means it is with respect to the particular forest F. So, F is a forest and let us say our initial graph was the was G and F is a forest for G ok; need not be minimal forest minimal spanning forest just some collection of edges. Now, if you have a forest ok; sorry let me say that the forest edges will be colored in black and the non forest edges let us say they are colored using a different color ok. So, now if you take any arbitrary pair of vertices u and v ok; between these two vertices here in this example, there is no path consisting of just the forest edges because they belong to different connected components, but suppose they are in a connected component.

So, let us say this is your vertices and if you look at v and w in the forest there is a unique path from v to w ok. Unique in the sense you are not you do not trace the same edge twice; if you allow only one transition through each edge, then there is a unique path from u to v. In fact, for any if you will take any tree and if you take a pair of vertices; there is a unique path from one vertex to another ok. Since we are looking at a weighted edge we can look at this unique path and look at the weights that are there on this path. There could be a direct path between v and w; so let us say this is some path from v to w. And probably all when these things have given weight and there could this is your u and this is your w; there could have been a direct path as well. Now let us compare the weight of the direct path to the weight of the individual edges in the path ok.

We will call an edge F heavy so an edge u w is called F heavy if 1; u come v should be in the same component connected component. 2; the edges on the unique path from u to v must all be of weight at most weight u v or u w ok. So, if you look at this path if its weight is 10 if everything here it is of weight less than or equal to 10; then this is called as a F heavy edge ok. So, this edges of 10 if these where the weights then this is not an F heavy edge because every weight is at most 10.

(Refer Slide Time: 21:27)



So, how do we determine F a particular edge is a F heavy edge? So, let us look at the unique path from u to w that is if there is such a path. And so let W F u v or u w denote the weight of the heaviest edge in u to w path. If the weight of u w is greater than W F u w then u v is called an F heavy edge ok. So, you look at the weight of u w every other edge on the path those of weight strictly less than the weight of u w.

So, if these were all weights say 9, 7, 6, 5, 2, 1, 3 then all these weights the u w paths are strictly less than the weight of u w ok. And therefore, u w will be called as an F heavy edge that is if all these edges when if this was the unique path; these are all edges in the path in that case u w would be called as an F heavy edge and this was 10 then this is not a F heavy edge ok.

So, the edges which are not F heavy; so not F heavy is by definition F light and the important fact for as us F heavy edges are not present in any MST; that should be clear

because that is just an application of cycle property. So, cycle property says that the maximum weighted edge in a cycle will be absent from all minimal spanning tree ok.

So, if you take the maximum weighted edge since we are looking at graphs where all the edge weights are unique; if you look at the maximum weight edge all the other edges are off strictly less weight. So, if you; so let us say there is a cycle and if you look at the maximum weighted edge in the cycle that edge can be thrown out from all minimal spanning trees that is what the cycle property says. So, here when you have a part between u and w and every edge on that path is of weight strictly less than the u w weight. If you add this u w into this particular path you will get a cycle and this becomes the heaviest edge of the cycle. So, that edge deserves to be thrown out from minimal spanning tree.

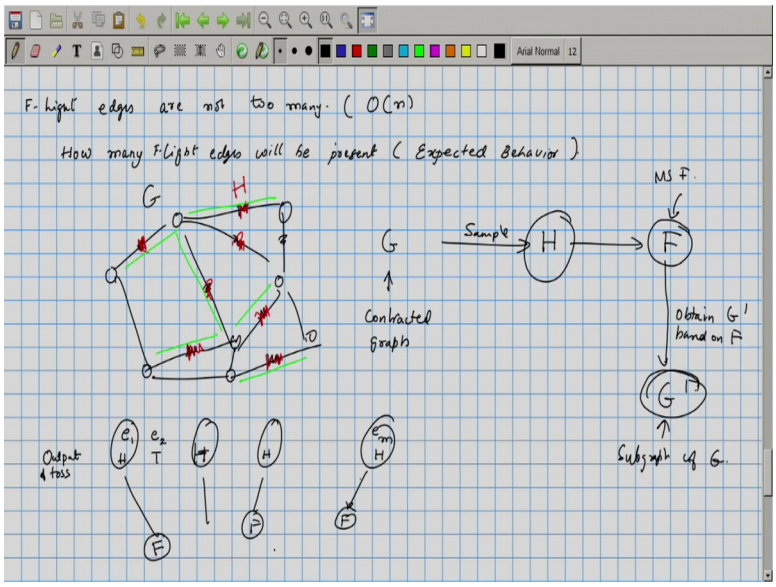
So, F heavy means it can surely be eliminated ok. Now let us just previous the algorithm the algorithm says you randomly choose a sub graph of G 1, where G 1 as a graph obtained after running 3 phases of Boruvka. From that randomly chosen sub graph you construct its forest which is going to be a random forest for G 1; based on that random forest throw out all the heavy edges.

Those heavy edges we know are not going to be present in any minimal spanning tree. We need to determine these heavy edges means we need to given a particular graph F; we need to classify the edges as heavy edges and light edges ok; this can be done in linear time. So, this is based on something known as MST verification algorithm, we will not go into details of those algorithm, but we will just assume that the F heavy edges and the F light edges can be segregated and this can be done in linear time the algorithm is by Karger et al ok.

So, Karger had given this randomized algorithm and the analysis was done by Tarjan et al. So, this is a algorithm given by Karger was shown to work in randomized linear time. So, that algorithm basically verifies that certain edges are heavy edges and the other edges are light edges for any arbitrary tree F. So, now we said that the heavy edges can be thrown out ok, but how do we analyze the total time; where did we get these savings ? So, how many heavy edges would be thrown out? The utility of this algorithm is significant only if large number of heavy edges are thrown out.

We have two recursive invocations of the MST algorithm here where the MST algorithm is a running on an input of size roughly m by 2 because the original graph; even after running multiple phases of Boruvka and after running 3 phases of Boruvka might have let us say m over two could have a significant fraction of the initial edges. The expected number of edges could be m over 2. And later on we have another invocation of MST; where all the heavy edges are removed.

(Refer Slide Time: 28:01)



What we will argue is the heavy edges are large in number or in other words we will show that the light edges for any F that is randomly chosen. So, that the light edges are not too many; too many for us here means let us say O of n ok; there are a linear number of edges in the size of the graph. So, let us look at this sampling a little more carefully to count the number of F light edges.

So, this is the question that we will answer how many light; so F light edges will be present. So, what we want is the expected behavior or the expectation of the number of light edges. So, what does this algorithm really do? So, let us say we had this contracted graph which we will call as let us say G for the time being.

So, this let us say this is the graph obtain after contraction; what we do is on each edge toss a coin ok. So, maybe this will be included, this will be included the marked edges maybe they will be included after a coin toss. So, based on the result of a coin toss from the graph G ok; we had obtained the graph let me just color it in red.

This is a sub graph of the original graph which we will call as H and for H we will obtain; so H is a smaller graph, for H we will have to obtain a spanning forest which would be let us say I have not put the weights, but suppose it is something like this ok. So, that is the way the algorithm progresses; from G this is the contracted graph we will sample we will go to H and from H we will go to F ok. So, F is the minimal spanning forest.

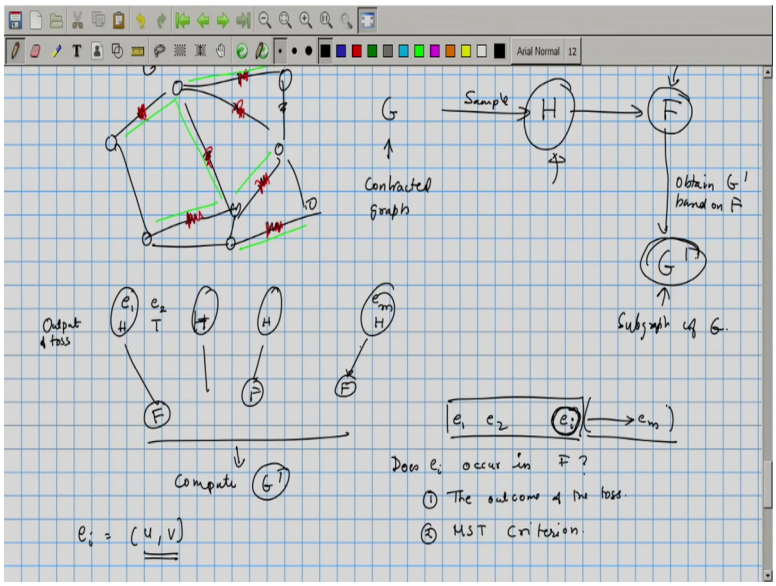
Now, based on F; obtain let us say G prime G prime is a sub graph of G and we are interested in how many edges are really present in G prime ok? What we will like to say is that in an expected sense large, number of edges will be thrown out of G based on this F. Let us look at the edges in G; we will write it in the increasing order of their weights the algorithm we do not sort the edges because that is too costly to do because if we sort all the edges your algorithms running time will be m log m which is more than linear time.

So, we will not sort, but let us imagine the vertices in the sorted order. So, we want to figure out which of the edges that would remain in G prime ok. So, after that contraction phase and after throwing out certain edges based for the intermediary minimal spanning forest; what are the edges that would remain ok? How many such edges would be there? Look at the edges in the original graph let us call it has e 1, e 2, e m.

So, this m is not the total number of edges in the input graph, but it is the number of edges in the contracted graph ok. So, let us say it was e 1, e 2, e m; what we had done is we toss a coin ok. So, let me just say that and I will denote the output of the coin toss. So, heads and some of them will be tail; tail head ok.

The heads are the ones that I would pick to obtain the graph H and then based on H in some of the edges in H. So, look at all these edges in H; few of them I will add to the forest. Once the graph H is fixed, it has a unique minimal spanning forest which are the edges of the forest we will need to worry about it. Once the edges of the forest gets fixed the edges which will be removed also gets fixed.

(Refer Slide Time: 33:21)



So, from F we will basically compute G prime ok; so the entire process we can view as; we have all these edges e 1, e 2, e m and we toss the coin and after the result of the coin tosses obtained we could take decisions. So, now, let me just look at the ith edge in the sorted order ok; will this edge be present in G prime? Before that will this edge be present in F ok? So, the question of whether this will be present in F. So, does e i occur in F; it depends on two things 1, the outcome of the toss. If the toss had resulted in a tail then certainly it is not going to be included in H which is not present in H is going to be included in the forest.

But even if this edge was present in H if the weight of that edge was too much or if it was not a suitable H for the minimal spanning forest it; could have been thrown out ok. So, I will write that as the MST criterion. So, only if this particular edge e i satisfies the outcome of the toss requirement and the MST criterion will it be present in the forest. Now the outcome of the toss is not dependent on anything else these are independent coin tosses; so it just depends on that particular toss. The MST criterion does not depend on any edges after e i. If you look at the sorted order the only edges which plays a role in the MST criterion are the edges which appear before e i.

There are many edges in the graph G which had gotten added to the sub graph H; amongst these look at those edges whose weights are less than e i, those alone would determine whether e i is present or not. So, look at this sub graph of H formed by

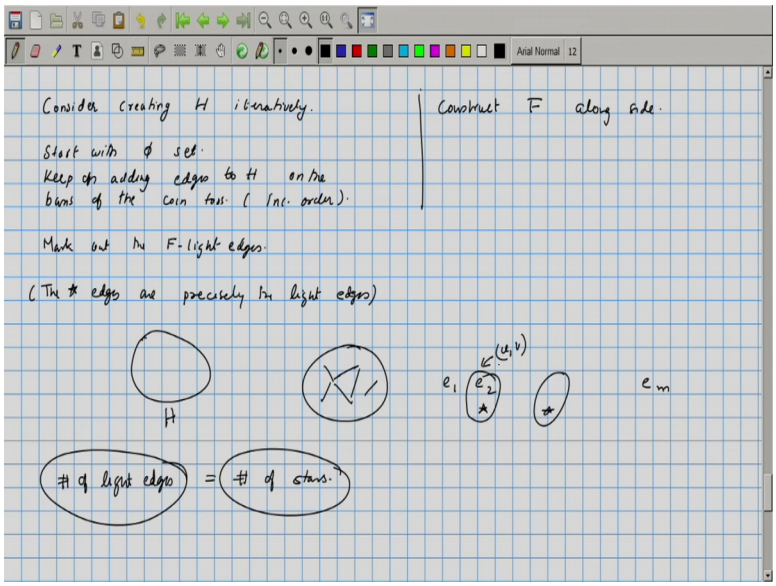
restricting the attention to edges which have weights which is strictly less than e i. Now, if you look at the edge e i. So, let us say e i is u comma v u and v are in the same connected component in this sub graph ok; which is a sub graph that we are looking at? Look at H; H is a particular graph which is a sub graph of G and inside H restrict your attention to only those edges whose weights are less than e i. They would have formed the sub graph amongst themselves; now if u and v are connected in that sub graph if you add the H connecting u and v, they will form a cycle whose weight whose largest edge is going to be u v and that is; that is why u is not going to be present ok.

So, this edge gets occurs in F in that sub graph if you add u and v they do not form a cycle in other words u and v belong to different components in that sub graph ok. So, if they belong to different components e i will be present; if they do not belong to different components e i will not be present. So, this decision entirely based on the outcome of the coin tosses e 1 to e i.

The result to the coin tosses starting from e 1 to e i determines the sub graph. And once the sub graph is determined the decision of whether e i is to be present in F or not is completely made. And therefore, the other edges do not play any significant role in whether e i occurs in F or not ok. Now what we are interested in is the light edges; how many light edges are there?

So, in order to analyze that we will do a small trick; what we will do is the following, let us see here we constructed the sub graph H ok. And then after H is ready we were looking at the forest and then we were looking at the sub graph of G by removing the F heavy edges. We will look at it in this increasing order of their weights itself; when an edges being considered that edge if it is included into H it might create a cycle in the already existing H ok.

(Refer Slide Time: 38:28)



So, consider creating H iteratively ok. So, which means start with empty; empty set keep on adding edges to H on the basis of the coin toss these edges considered in increasing order ok. So, we can imagine that the coin tosses have already been done for all the edges and we are just now taking those coin tosses and deciding which are the edges of H. While we do this we can also construct the spanning tree F side by side ok. Now when we add an H to H we can at that point wonder whether that edge will create a cycle in H; if it creates a cycle in H that is not going to be an edge for F.

In fact, that is going to be an F heavy edge ok. Once again while we are looking at construction of H in an iterative fashion; whenever we are adding a particular edge to edge that is if the coin toss resulted in a head, we could have added that edge, but at that stage we will check whether that edge creates a cycle in the already constructed spanning forest. So, we had let us say some particular set H and along with it we had a spanning forest a partial spanning tree. And while we are adding an H we look at whether adding that H creates a cycle in F.

If it creates a cycle in F that edges going to be a heavy edge ok. So, we will just mark out; so we will just ignore the heavy edges and we will just mark out the other edges the other edges are essentially the light edges ok. So, each edge I mean if you are looking at in this order e 1 e 2 e m certain edges I will just mark out this essentially means while considering e 2 if I look at the iterative spanning forest in that forest if I where to add e

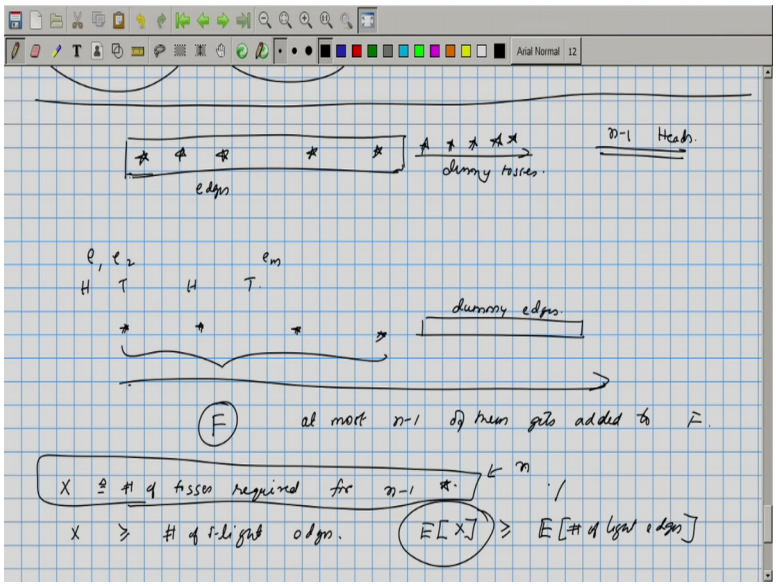
2; e 2 would have been a light edge; that means, adding e 2 will not create a cycle in the spanning forest.

Now, these light edges in whatever I marked as light edges need not be present in F; they would be present only if the coin toss was successful. While we are constructing H we said toss the coin keep on adding to H so, but we will just mark out an H. So, we could just reverse the order in which these things is being done; in the sense we will first check the edge if the edge can be added successfully without creating a cycle in F; we will mark it out. And after marking it out we toss a coin the toss results in a head; we will add that to the set H and that H surely would be there in the in F as well ok.

But note that the starred edges are precisely the light edges ok. Note that this entire thing is the analysis we are not trying to compute which are the light edges; you can think of it as an alternate definition of the light edges. That is if you consider the entire construction of H in an iterative sense that is in the increasing order of the weights; those edges which could have been added if the coin tosses successful without creating a cycle in F, those are precisely the light edges.

So, if we want to count the number of light edges; they are precisely then number of star edges. Amongst these star edges later on some of them would get added to F and the rest would stay there as light edges. So, summarize this by saying number of light edges is equal to number of stars ok. So, how do we estimate light edges? We will just need to count how many stars would be there ok. When does a particular edge get a star? Well first of all the preceding edges should have I mean if you look at the spanning forest by the previously selected edges; in that the vertices connecting e 2 or e i say u v they should be in distinct components.

(Refer Slide Time: 44:10)



We can think of this in a different way. So, the other edges which are not star does not play any role because we just need to constantly starred edges. So, we can think of there are all these starred edges coming up and for each of them you toss a coin. If the toss results in a head; you include that into the spanning forest they anyway get starred into H and by virtue of it being a light edge; it also gets added in the spanning forest ok. So, we have this sequence of stars, now once all the edges have been exhausted all the stars would also have been exhausted.

But let us say we keep on tossing further dummy coins which does not really depend on any edge ok. So, these are the edges and these are dummy tosses and we will keep on tossing till we get let us say n minus 1 heads ok. So, this is an mean in order to count the expected number of light edges we are counting in a slightly different way. So, there were stars and lot of edges these stars gets decided based on the ordering. If we had a particular ordering e 1 e 2 e m and if we had a particular outcome for the coin tosses heads, tails; once this is fixed the stars on the edges gets fixed. Certain edges would be starred edges and the others would be non starred edges; the non starred edges can all be thrown out because they are all heavy edges.

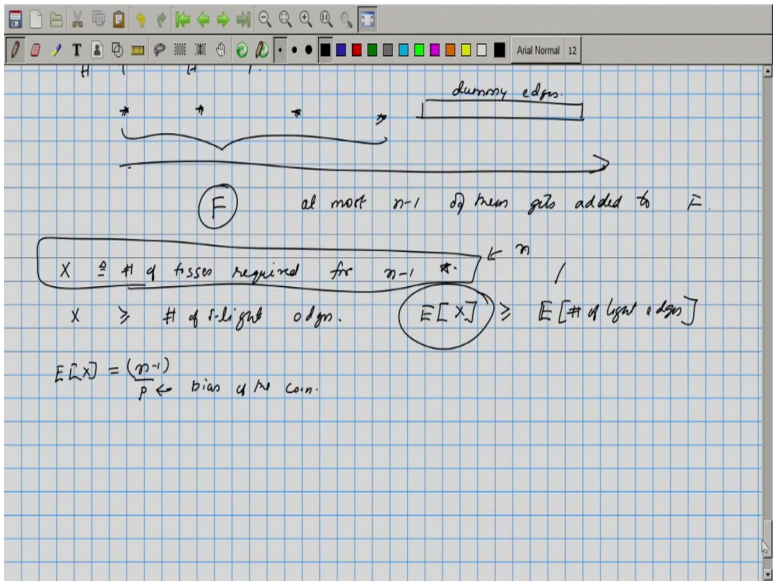
So, now let us just look at the starred edges there will be certain number of them and we append those with certain I mean let us an infinite number of dummy tosses or dummy edges. Some of these starred edges before the dummy edges would get added into F. At

most n minus 1 of them gets added to F because F is a spanning forest it can have no more the n minus 1 edges in it ok. So, if we look at the sequence of stars and say we keep on tossing till we get n minus 1 heads; that would be certainly more than the total number of stars. So, let us say X is a random variable denoting number of tosses required for n minus 1 stars ok. So, if we do it on this particular sequence by the time we get n minus 1 stars; we would have surely added all the edges to F.

I mean these coin tosses if it had resulted in a particular F all those edges would have been added or. So, in particular all the light edges would have been accounted. So, the random variable X; we can say it is greater than the number of F light edges ok. And since both these are positive we can say that the expected value of X is going to be greater than the expected number of light edges ok.

So, if we calculate this that is certainly an upper bound on the number of light edges, but X is just tossing coins till you get n minus 1 stars. But if each coin was of let us say bias P number of times you have to toss to get one heads is 1 by P; so, this will be at most; so this is expectation.

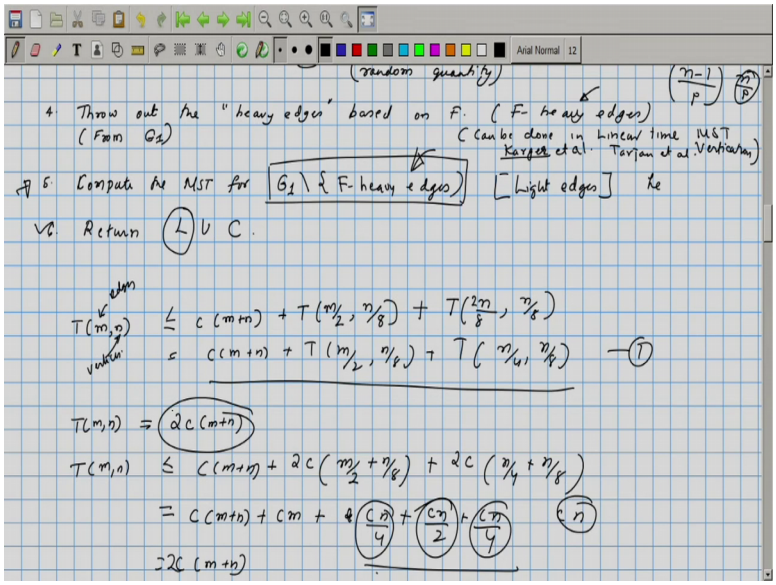
(Refer Slide Time: 48:19)



So, expectation of X is nothing, but n by P where P is the bias of the coin or n minus 1 by P ok. So, what this means is in our algorithm after throwing out these heavy edges; we are left with precisely n minus 1 by P light edges; where n is the number of edges in the

graph that you are looking at. So, now we can just put in all these inductive steps and compute what is a total running time P for as is 0.5.

(Refer Slide Time: 49:10)



So, let us look at the total running time; if you denote it by T m n; this is the total running time for a graph with if the input graph contains m vertices; sorry m edges and n vertices. Then this is T m n is going to be less than or equal to 3 phases that is let us say C times m plus n. So, we know the 3 phases and each of those phases takes linear amount of time. So, C times m plus n plus random selection that is again linear time; so that is absorbed into this. So, the first step is taken care of second step is taken care of last step just returns the spanning tree; so those three together takes C m plus n.

Now if you run the MST algorithm on G 2 G 2 will take T of m we expected number of edges in G 2 is going to be m by 2 and the expected number of vertices is n by ok. So, we had run 3 phases; so the number of vertices would be n by 8 each phase it gets divided by 2 plus if we had thrown out all the heavy edges; then G 1 will contain T of the number of edges that this remaining will be proportional to.

So, it is n minus 1 n minus 1 here being n by let us say we will bound this by n by P where n as will n by 8. So, n by 8 divided by half; so 2 n by 8 and number of vertices is n by 8 ok. So, this is equal to C m plus n plus T m by 2 n by 8 plus T n by 4 n by 8 ok; so that is the recurrence. So, T m n is going to be less than or equal to this quantity.

If you plug in T m n is equal to let us say when 2 C m plus n ok. Suppose this is the bound that we impose and then we will get when it satisfies this equation 1; T m n is going to be less than or equal to C m plus n T m by 2 n by 8 is at most 2 C m by 2 plus n by 8 plus 2 C n by 4 plus n by 8 ok. So, this is equal to C m plus n plus C m plus 2.

So, C n by 4 plus C n by 2 plus C n by 4 n by 4 n by 4 n by 2 and this is n by 2. So, this together is C n ok; so C m plus. So, this is C 2; C m plus n. So, we know that if we have T m n to be at most 2 C m plus n, then it satisfies the recurrence. So, this takes no more than linear amount of time in an expected sense; that is a randomized linear time algorithm for computing minimum spanning tree.

Thank you.