

IIT Madras
ONLINE DEGREE

Mathematics for Data Science 1
Professor. Madhavan Mukund
Chennai Mathematical Institute
Lecture No. 12.6

Minimum Cost Spanning Tree: Prim's Algorithm

So, we are looking at minimum cost spanning tree and weighted graphs and we said there are two natural strategies which are simplified by two standard algorithms. So, the first one is Prim's Algorithm.


(Refer Slide Time: 0:25)

Minimum cost spanning tree (MCST)

- Weighted undirected graph,
 $G = (V, E), W : E \rightarrow \mathbb{R}$
 - G assumed to be connected
- Find a minimum cost spanning tree
 - Tree connecting all vertices in V
- Strategy
 - Incrementally grow the minimum cost spanning tree
 - Start with a smallest weight edge overall
 - Extend the current tree by adding the smallest edge from the tree to a vertex not yet in the tree

Example

- Start with smallest edge, $(1, 3)$
- Extend the tree with $(1, 0)$
- Can't add $(0, 3)$, forms a cycle
- Instead, extend the tree with $(1, 2)$
- Extend the tree with $(2, 4)$



Madhavan Mukund Minimum Cost Spanning Trees: Prim's Algorithm Mathematics for Data Science

So, we have a weighted graph, so a graph with a weight function which assigns a number to every edge. And let us assume that the graph is connected because otherwise, if it is not connected to begin with, we cannot superimpose a tree on it. What we want is a spanning tree, a tree which is the subset of the edges which connects the graph and if there are not enough edges to connect the graph to begin with, we do not even have a starting point.

So, assuming the graph is connected and it has weights. We want to find a minimum cost spanning tree which connects all the vertices in V . So, the strategy that we are going to adopt is to incrementally grow it. So, we start with the smallest edge and we will keep growing the tree by adding the smallest edge that we can to the current tree while retaining a tree. So, let us look at this example that we saw before.

So, supposing we start with the smallest edge. So, the smallest edge is the edge between 1 and 3. Now we want to grow this tree. So, the smallest edge with which so we need to grow it meaning we have to choose an edge which will extent this tree. So, we cannot choose for instance this edge right now because this edge is not connected to the tree. That is the smallest edge over all but it is not connected to the tree. I want to grow the tree. So, I can take any one of the edges coming out of here. But not that one that one is out I cannot do that one.

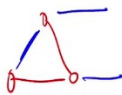
So, I can take any one of the edges which is leaving the tree and extend it. So, I take the smallest among those which is 0 to 1. And so now I have a tree which has 2 edges 1 to 3 and 0 to 1. Now I can take any edge which is leaving this tree. So, the smallest among them is this edge with weight 18. But unfortunately when I add this edge with 18 I create a cycle. And this is not a tree. So, I cannot do this, so I have to throw this away. And I have to go to the next one.

So, I go to the next one which is the 20. And now I have tree which connects four of the five vertices. And now finally because I have reach 2 and now allowed to add this edge. Because now this edge is connected to the tree that I have constructed so far. So, now add that edge and I will get this tree which if you remember we have said last time that this tree has weight 44. And you can check that we have actually found that one, so this is Prim's Algorithm.


(Refer Slide Time: 2:34)

Prim's algorithm

- $G = (V, E), W : E \rightarrow \mathbb{R}$
- Incrementally build an MCST
 - $TV \subseteq V$: tree vertices, already added to MCST
 - $TE \subseteq E$: tree edges, already added to MCST
- Initially, $TV = TE = \emptyset$
- Choose minimum weight edge $e = (i, j)$
 - Set $TV = \{i, j\}, TE = \{e\}$ MCST
- Repeat $n - 2$ times
 - Choose minimum weight edge $f = (u, v)$ such that $u \in TV, v \notin TV$
 - Add v to TV , f to TE



Madhavan Mukund Minimum Cost Spanning Trees/Prim's Algorithm Mathematics for Data Science I, W



So, formally what Prim's Algorithm does is it incrementally builds an minimum cost spanning tree. So, it keeps track over set of vertices which have been added and keeps track of set of edges which

have been added because just because of vertex has been added does not mean that all the edges between them have been added. So, if you look at previous one for instance when we have added 3 and 0, it does not mean that the edge 0, 3 is added. So, we have to separately note which edge is belong to the tree.

It is not just enough to say, it is not like a Dijkstra's algorithm where we said we have burned these vertices so we are done with them. We need to know which are the edges were used in order to construct the tree. So, we keep track of these two things separately TV, the tree vertices. And TE, the tree edges. So, initially everything is empty. The way we describe it we will start with the smallest edge. So, overall we look at all the edges in the graph and we pick the smallest edge that edge let it be called, let it be from i to j .

So, once I do that, then I am started with the minimum tree. Let a tree consisting of just one edge. So, the TV, the vertices in the tree are i and j and the edge is this edge e which I just added which is from i to j . And now what I will do at each step is I will take an edge which starts in T and goes out of T . So, that is the reason we could not introduce when we had this graph already drawn the reason we could not choose this edge is because it both end points are already in the tree that we have constructed. So, we need an edge like this or like this which starts in the tree and end outside the tree.

So, we choose a minimum weight edge such that u belongs to tree and the outside the other edge end point of the edge belongs to the other side. It is not in the tree. Among all these edges which has starting from the tree and going out you take the smallest one and add it. Once you add it you have added a new vertex to the tree. So, TV gets expanded to add v in it and TE gets expanded to add this edge. So, this is the algorithm for Prim.

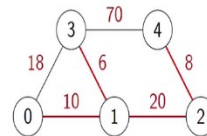
(Refer Slide Time: 4:28)

Prim's algorithm



- $G = (V, E), W : E \rightarrow \mathbb{R}$
- Incrementally build an MCST
 - $TV \subseteq V$: tree vertices, already added to MCST
 - $TE \subseteq E$: tree edges, already added to MCST
- Initially, $TV = TE = \emptyset$
- Choose minimum weight edge $e = (i, j)$
 - Set $TV = \{i, j\}$, $TE = \{e\}$ MCST
- Repeat $n - 2$ times
 - Choose minimum weight edge $f = (u, v)$ such that $u \in TV, v \notin TV$
 - Add v to TV , f to TE

Example



$$TV = \{1, 3, 0, 2, 4\}$$
$$TE = \{(1, 3), (1, 0), (1, 2), (2, 4)\}$$



So, again going back to that example so this is how we start. We start with both sets empty and now we take the smallest edge overall which is this one. This is how we start the algorithm. So, when we start the algorithm we say that 1 and 3 form a tree. So, the tree vertex at TV has 1 and 3 and the edge set consist of the edge 1, 3. Now I look at the smallest vertex which has one end point and TV and the other end point outside and it turns out to be the edge 0, 1. So,, I add 0 to TV and I add 0, 1 or 1, 0 because I am drawing it from 1 to 0 to this edge set.

Now I cannot do 18 because it is not inside. So, the next stage that I can do is 1, 2. Because I need to find an edge which is inside to outside. So, this edge is not allowed because I have this condition that you say for example 0 must be in the tree vertex set and 3 must be outside. But both 0 and 3 are inside. So, I have to drop that edge. So, I can only look at edges which leave the tree and go out. So, among those 1, 2 is the next one. So, I get 2 now in my tree set. And my tree edge set has 1, 2.

And finally, from here I can get 8. You are use that edge with weight 8 and get 4 into my edge set in a vertex set. And 2, 4 in the edge sets. So, now I have got all, so I did this $n-3$ times as the important thing to do. So, I basically started with 1, 3 and I did 1, 2, 3 times. So, did this $n-2$ times. So, I had five vertices I already started with two vertices. So, I have $n-2$ vertices to go. Every time I do add an edge I had one more vertex into my set.

So, after I do 10-2 times my vertex set has covered all the vertices remember to assume that the graph is connected otherwise it is not going to do work. They not be into able to connect it. So, this seems a little bit, I mean at one side seems reasonable another side it one might ask why is this going to work. I mean why does this particular strategy actually give me guaranteed the shortest smallest tree overall.

(Refer Slide Time: 6:27)

Correctness of Prim's algorithm

Minimum Separator Lemma

- Let V be partitioned into two non-empty sets U and $W = V \setminus U$
- Let $e = (u, w)$ be the minimum cost edge with $u \in U, w \in W$
- Every MCST must include e

Madhavan Mukund Minimum Cost Spanning Trees, Prim's Algorithm Mathematics for Data Science

So, to do this let us prove some small graph theory fact call the minimum separator lemma. So, what this says that supposing I take my graph. And I partition it partition it means I divide the vertices in two sets U and W . Now I look at all the edges which have end points on opposite side of this. So, there will be some e_1 there might be some other edge e_2 and so on. There might be multiple edges which go across this partition.


So, they have one end point in U the other end point in W . So, among them let us assume that we pick one which is smallest. So, let us just for moment assume all of them have different weight. So, we will see what to do with different with they have equal weight later on. But let us assume that they have smallest one supposing one of these has actually smallest. So, maybe this is the smallest one.

Then what is lemma says is that no matter how I construct an MCST on this graph that particular edge e_3 has to be in MCST. So, the intuition is that somewhere in my graph I will be separating U

from in my tree I would be separating U from W and the best way to connect U to W is via e_3 . So I must use e_3 in my MCST. So, this is the claim. So, let us see why this is true.

(Refer Slide Time: 7:49)

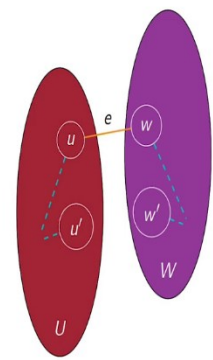
Correctness of Prim's algorithm



Minimum Separator Lemma

- Let V be partitioned into two non-empty sets U and $W = V \setminus U$
- Let $e = (u, w)$ be the minimum cost edge with $u \in U, w \in W$
- Every MCST must include e


- Assume for now, all edge weights distinct
- Let T be an MCST, $e \notin T$
- T contains a path p from u to w
 - p starts in U , ends in W
 - Let $f = (u', w')$ be the first edge on p crossing from U to W
 - Drop f , add e to get a cheaper spanning tree



Madhavan Mukund

Minimum Cost Spanning Trees: Prim's Algorithm

Mathematics for Data Science



So, as I said let us assume for now that all the edge weight in our graph are actually different from each other. So, it just simplifies the argument a little bit. So, this is the situation we are looking at. So, we have a tree and then this says for every partition. So, this is a universal property it says no matter how I partition may think as U and W . This property must hold so let me just assume this is some arbitrary partition on my graph. So, I have split my vertices into two sets which are disjoint which together cover all set.

So, partition means exactly that partition, if I partition my toys with my assistor then I have to take some and she has to take some and both of us get one or the other toy. And nobody leave gets left no toy gets left out. So, partition just means that the two sets are disjoint and the union is the full set. So, that is what partition is. So, U and W together cover all the vertices and there is no overlap.

So, now the question is that supposing I have a tree and I look at this graph this edge which I am promised must be in the tree. It says that the smallest edge which connects a vertex in U to a vertex in W . The smallest edge connecting this partitions must be in my tree. So, supposing it is not in my tree. So, I am assuming the for the sake of contradiction that I have built a tree which excludes this vertex which a lemma promises me should be in my tree.

Now other hand I have built a tree. So, this hypothetically this capital T is an MCST. It is a tree, it is a connected graph on the underline vertices. So, there is no problem going from u to w. the small u to small w, there must be path because it is connected. So, this path starts in the left partition. And it ends in the right partition. So, imagine that this is river separating two sides of the city you cannot go from this side to that side without crossing the river.

So, I am not using this edge e which I have want but I have to cross somewhere. So, there must be some other edge. Let me call it f where this path crosses. You cannot go from the left side from the red side to the purple side from capital U to capital W without crossing from this side to that side via some edge. And if I assume that I have not taken the edge e which I am interested in which is the smallest edge connecting U to W. I must have taken some other edge.

So, this is what the picture looks like. So, now we can see what happens. So, this is connected, so now supposing I had drop this edge and instead keep this edge. Then I claim that everything that was connected before is still connected anything that I can reach I can now go from there and reach. So, thing that I could reach by following this U to W edge I have this long path which goes via u, w and coming back to w prime and I can do it.

So, therefore, the connectivity does not change but I have replaced f by a smaller edge. So, therefore if I replace f by e I have got a smaller or a cheaper spanning tree in terms of cost. So, this contradicts the assumption that I have started out with a minimum cost spanning tree. I started of with an MCST and I have told you get a smaller one. So, therefore, this could not being the case.

So, either T was not in MCST or T was an MCST but these assumptions that e did not belong as false. So, therefore we have establish through this contradiction this lemma that says that if I take any partition of vertex set and I find the smallest edge going back in fourth across those that partition that must belong to every MCST that you build. So, if you choose that edge you are okay if you do not choose that edge you are in trouble. So, just lets dispose of this case of distinct edge weights before we look at Prim's Algorithm.

(Refer Slide Time: 11:25)

Correctness of Prim's algorithm

Minimum Separator Lemma

- Let V be partitioned into two non-empty sets U and $W = V \setminus U$
- Let $e = (u, w)$ be the minimum cost edge with $u \in U, w \in W$
- Every MCST must include (e)

- Assume for now, all edge weights distinct
- What if two edges have the same weight?
- Assign each edge a unique index from 0 to $m-1$
- Define $(e, i) < (f, j)$ if $W(e) < W(f)$ or $W(e) = W(f)$ and $i < j$

Madhavan Mukund Minimum Cost Spanning Trees: Prim's Algorithm Mathematics for Data Science

So, what if two edges have the same weight it is not a big problem. Because you just need to have a strategy to choose one or the other. So, you need to arbitrarily decide it one has smaller than the other in order to make this thing work. Which one will go into every tree? So, if you fix a strategy, so fix a strategy basically you assume that you have numbered the vertices in some fixed way from 0 to $m-1$.

And then you just decide that the ordering in such that either an edge. So, this is the numbering, so e with number i and f with number j now. So, supposing e and f are two edges I would have assigned each of them a number a different number between 0 and $m-1$. So, I look at now e comma i and f comma j and if the weight of e is smaller than the weight of j , I declare that e is this edge is smaller. But if they have equal weight then I will look at the index and I will say the index i is smaller than the index j then e comma i is smaller than f comma j . So, this gives me a weigh of kind of ordering all the equal vertices.

So, then my the lemma will say that it must include the smallest, smallest in terms of the ordering. So, that is what we say so it is not a big deal so it can be done. So, this lemma holds in general even if the graph has weights which repeat. So, now what is the impact of this lemma on Prim's Algorithm.

(Refer Slide Time: 12:36)

Correctness of Prim's algorithm



Minimum Separator Lemma

- Let V be partitioned into two non-empty sets U and $W = V \setminus U$
- Let $e = (u, w)$ be the minimum cost edge with $u \in U, w \in W$
- Every MCST must include e

- In fact, for any $v \in V, \{v\}$ and $V \setminus \{v\}$ form a partition



- In Prim's algorithm, TV and $W = V \setminus TV$ partition V
- Algorithm picks smallest edge connecting TV and W , which must belong to every MCST

Navigation icons: back, forward, search, etc.



Madhavan Mukund

Minimum Cost Spanning Trees: Prim's Algorithm

Mathematics for Data Science

Correctness of Prim's algorithm



Minimum Separator Lemma

- Let V be partitioned into two non-empty sets U and $W = V \setminus U$
- Let $e = (u, w)$ be the minimum cost edge with $u \in U, w \in W$
- Every MCST must include e

- In fact, for any $v \in V, \{v\}$ and $V \setminus \{v\}$ form a partition

- The smallest weight edge leaving any vertex must belong to every MCST

- In Prim's algorithm, TV and $W = V \setminus TV$ partition V
- Algorithm picks smallest edge connecting TV and W , which must belong to every MCST

- We started with overall minimum cost edge

- Instead, can start at any vertex v , with $TV = \{v\}$ and $TE = \emptyset$

- First iteration will pick minimum cost edge from v

Navigation icons: back, forward, search, etc.



Madhavan Mukund

Minimum Cost Spanning Trees: Prim's Algorithm

Mathematics for Data Science

So, in Prim's Algorithm, what we have at any given point is the set of vertices which are in the tree. So if I look at the set of vertices which are in the tree. And the set of vertices which are not in the tree. Let me call it W so this is the set-operation. So, $V - TV$, so I used it there also. So, this means all the elements which I get by removing the elements of TV from V . W and TV together have all the vertices and they are disjoint so it is a partition.

So, therefore, now if I look at what Prim's Algorithm does, it says I am currently looking at a tree which I have built and I am looking at all the edges which go out. Which connect my current tree to vertices are not in the tree. And I pick the smallest one. But that is nothing but this minimum

separator. For this particular partition I am picking the smallest one. So, therefore, the one that I am picking has to belong to us. So, I am not picking anything wrong.

So, every, every edge that Prim's Algorithm picks is guaranteed to belong to every MCST by this lemma. And since it does that and it picks exactly $n-1$ edges overall. All the edges are necessary and therefore they all are part of an MCST. So, that is correctness argument. So, in fact there is a slightly stronger statement we can make. If we look at a vertex V and we look at everything else the whole set- V .

Then if I look at the vertices the edges coming out of V they connect the vertex V to its neighbors and all those neighbors belong to the other partition. So, by this lemma among those edges which are coming out of a vertex all of them are disconnect, connecting the partition containing V alone to the remaining things. The smallest edge leaving a vertex is this minimum separator.

So minimum separator separating V alone from everything else and by this lemma that is smallest edge must belong to every MCST. So, basically if I started a vertex and I look at all the edges which are connected to it. And I pick the smallest one then that smallest one is guaranteed to be in every spanning tree, every minimum spanning tree. So, actually therefore it does not really matter that Prim's Algorithm started with this minimum cost edge that is the bonus.

The minimum cost edge is of course going to be the minimum separator between the partition of the two end points. So, that is fine but you can start anywhere. So, you can start with any vertex and we know that from that vertex the smallest edge leaving it is a minimum cost separator between it and the rest. So, I can start with the vertex V set to be just a single vertex and no adjust. And then I can apply Prim's Algorithm.

What Prim's Algorithm will first discover is the smallest edge which connects V to one of its neighbors. So, that will be my first edge and so on. So, Prim's Algorithm will work from any starting point, the first iteration will pick the minimum cost edge leaving v which by this lemma is correct.

(Refer Slide Time: 15:53)

Summary



- Prim's algorithm grows an MCST starting with any vertex
- At each step, connect one more vertex to the tree using minimum cost edge from inside the tree to outside the tree
- Correctness follows from Minimum Separator Lemma



So, Prim's Algorithm is a natural way to build a minimum cost spanning tree starting at any vertex as we saw. The way we first presented it was starting with the minimum cost edge but starting at any vertex you can build an MCST because of this minimum separator lemma. So, at each edge what you do is you take the tree you have already constructed. And we pick the minimum edge connecting that tree to the rest.

So, we extend a tree one edge at a time. And at each point because we have going from inside to outside the new edge is guaranteed to keep it a tree. And finally every edge that we get is guaranteed to be required by the minimum separator lemma. So, every edge that we add was required to be in the tree and therefore overall we have added no useless edges. So, we must have got a minimum cost spanning tree.