Minimum Spanning Trees

June 10, 2020

0.1 Connecting Computers by Wires

Total cost for optimal path above is $ 2 + 1 + 3 + 4 + 2 = 12 $

0.2 Building Roads

0.3 Formal Definition of MST

MST: - Input: - A connected, undirected graph $ G = (V, E) $ with postive edge weight - Output: - A subset of edges $ E' $ in $ E $ of minimum total weight such that the graph $ (V, E') $ is conencted

Its called minimum because were looking for the minimum total weight. Its called spanning because were looking for all the connected edges. Its called tree because $ E' $ is going to be for a tree

0.4 Properties of a Tree

- A tree is an undirected graph that is connected and acyclic
- a tree on $ n $ vertices has $ n - 1 $ edges
- Any connected undirected graph $ G(V, E) $ with $ |E| = |V| -1 $ is a tree
- An undirected graph is a tree if there is a unique path between any pair of its vertices

Any optimal solution must be acyclic simply because of the fact that if a cycle exists, then there are two path you can take. So if you remove a single path, then what you are left with is a tree.

1 Greedy Algorithms

Kruskal's Algorithm: > repeatedly add the next lightest edge if this doesn't produce a cycle - Intially our graph is empty so we just add the lowest value

- there is another edge with a cost of $ 1 $ so we add it
- Next One has a value of $ 2 $ so we add it
- But the one after that is $ 3 $ and we dont add it because of cycles
- The next lightest edge has a weight of $ 4 $
- we skip $ 5 $ because of cycle and we add $ 6 $

Prim's Algorithm: > Repeatedly attach a new vertex to the current tree by a lightest edge

- Select a root for the tree
- in each iteration, we attach a new node
- we do this by using the lightest possible edge

- Our tree has two nodes and we attach the next smallest node
- the next one has weight $ 2 $ and the one after that $ 6 $
- finally the last one is attached by an edge with weight $ 1 $

2 Cut Property

To prove both of Kruskal's and Prim's Algorithm work is to prove that they are safe. Meaning if at the current step, we have a subset of edges which we call $ e' $, whic is a subset of some optimal solution, then by adding this edge, according to one of these two strageties, to this set, gives us also a set of edges with is also a part od some optimal solution

Cut Properity

We know that $ X $ is a subset of some minimum spanning tree

Then consider some parition of the set of the vertices into two parts

What is important is that any two vertices that are joining lie in the same partition

Next we consider some lightest edge $ e $. What lemma states is tat if we add this edge to our $ X $ then the resulting edge will also be some MST

2.1 Cut Property Proof

we assume $ X $ is a part of some minimum spanning tree

We we then take $ e $ which is the lightet edge between $ S $ and $ V - S $

we know that $ X $ is a part of some $ MST $ $ T $ and need to show that $ X + {e} $ is also a part of a (possible different) $ MST $

To prove it, consider two cases

- case 1: if $ e $ is a subset of $ T $, then there is nothing to prove, so asume that $ e $ is not a subset of $ T $
- assume the new tree $ T $. adding $ e $ to $ T $ creates a cycle; let $ e' $ be an edge of this cycle that crosses $ S $ and $ V - S $
- since the edge joins the two parts and since it is a cycle it but go to both the left part and right part
- $ e' $ is the edge that joins the two portions
- then $ T' = T + {e'] - {e} $ is an $ MST $ containing $ X + {e}$: it is a tree, and $ w(T') <= w(T)$ since $ w(e) <= w(e') $

3 Kurskal's Algorithm

- Algorithm: repeatedly add to $ X $ the next lightest edge $ e $ that doesn't produce a cycle
- At any point of time, the set $ X $ is a forest, that is, a collection of trees
- The next edge $ e $ connects two different trees -say, $ T_1 $ and $ T_2 $
- The edge $ e $ is the lightest between $ T_1 $ and $ V - T_1 $ hence adding $ e $ is safe

3.1 Implementation Details

- use disjoint sets data structure
- initially, each vertex lies in a seprate set
- each set is the set of vertices of a conencted component
- to check wheter the current edge $ (u, v) $ produces a cycel, we check whether $ u $ and $ v $ belong to the same set

3.2 Example

- Initally, each vertex lies in a seprate set.
- Ne then start processing edges in order of non-decreasing weight
- First edge is $ A $ to $ E $
- note that we saw they were not in the same edge, so we connect edges and we update our disjointed set
-
- we go to the next shortest edge $ C $ to $ F $
- we update out data structure
-
- We repeat process for $ A $ to $ D $
- you might be thinkink of connecting $ D $ to $ E $ because its the smallest edge, but you would be wrong, becasue since $ E $ and $ D $ are already in the same set, you wold be creating a cycle
- so we just do $ A $ to $ B $
- finall we connect $ B $ to $ F $ and we finish
- Notice that our optimal path is $ AEDBCF $

3.3 Kruskal's Pesudo-Code

- `for all u in V`: loop through all vertices
- `MakeSet(v)`: makes a set of all vertices
- `X <- empty set`: init $ X $ to an empty set
- `for all {u, v} in edges in non-decreasing order`: we process all the edges in non-decreasing weight
- `if Find(u) != Find(v)`: we check if there is a cycle or not
- `add {u, v} to X`: if they are different, you can add the edges to path $ X $
- `Union(u, v)`: we update our data structure

3.4 Running Time

Sorting Edges: - $ O(|E| log|E|) = O(|E| log|V|^2) = O(2|E|log|V|) = O(|E|log|V|) $

Processing Edges: - $ 2|E| * T(Find) + |V| * T(Union) = O ((|E| + |V| log|V| = O(|E|log|V|) $

Total Running Time: $ O(|E|log|V|) $

4 Prim's Algorithm

- $ X $ is alaways a subtree, grows by one edge at each iteration
- we add a lightest edge between a vertex of the tree and a vertex not in the tree
- very simmilar to Dijkstra's Algorithm
- we will be using a pirority queue and thus will be setting all nodes to $ inf $
- we will then pick an arbirtary node and change its value to $ 0 $
- we process the adjacent nodes and change its value to 1 and 8 respectively
-
- we select the smallest weight and attach it to our current tree
-
- continue the cycle
-

4.1 Prim's Pesudo-Code

- `cost[u] <- inf, parent[u] <- nil`: we dont know the cost of anything so we initilize it to $ inf $. the array `parent[u]` is our tree
- `cost[u] <- 0`: we select a random vertex and make its cost $ 0 $
- `PrioQ <- MakeQueue(V)`: put all of the costs in the Queue. All of their costs except the initial one is $ inf $
- `v <- ExtractMin(PrioQ)`: you extract the min from the queue and attach it to the current tree
- `for all {v, z} in E`: what we are doing is iterating through all its neighbors $ z $
- `if z in PriorityQ and cost[z] > w(v, z)`: if $ z $ is in our queue and not our tree, and if the current cost of z is greater than the weight of the edges from $ v $ to $ z $, then this means that we've just found a cheaper way to attach the vertex $ z $ to the current tree
- `cost[z] <- w(v, z), parent[z] <- v`: we update the cost of the edge wright to $ z $ and we attach $ z $ to $ v $. meaning $ z $ is going to be the parent of $ v $
- `ChangePriority(Prioq, z, cost[z])`: we modify our Queue to reflect the changes

4.2 Running Time

The running time is: - $ |V| * T(ExtractMin) + |E| * T(ChangePriority) $ - for array-based implementation, the running time is $ O (|V|^2) $ - for binary heap-based implementation, the running time is $ O((|V| + |E|log|V|) = O(|E|log|V|) $

4.3 Summary

Kruskal: repeatedly add the next lightest edge if this doesn't produce a cycle, use disjoint sets to check whether the current edge joins two vertices from different components

Prim: repeatedyl attach a new vertex to the current tree by a lightest edge; use priority queue to quickly find the next lightest edge