CSCI 6470 Algorithms Scribe Notes

Rory Hibbler, Sowndarya Nookala, Bavesh Kamma, Bojja Meghana, Vinaya Birajdar

March 27 - April 3, 2023

Topics Discussed

- Greedy-Choice Property for MST Problem
- Prim's Algorithm
- Kruskal's Algorithm
- Theorem behind While Loop Iteration in Prim's algorithm

1 Greedy-Choice Property for MST Problem

MST: Minimum Spanning Tree: A tree that spans the entirety of a non-directed graph with the least amount of associated cost.
For any given graph G, there can be many different MST's with the same cost.
- For this reason, we can use a **Greedy Choice Algorithm** to find *a* MST for any graph G.
A **Cut** is a partition of a graph into two non-overlapping subgraphs containing all the nodes in the original graph.
A **light edge** is the lightest of the edges that cross a cut. Sometimes, there can be multiple light edges, all with the same weight.
Theorem:
Let G be a given graph. Then any light edge crossing any cut of the graph is in some minimum spanning tree of the graph.
Proof:
Let Graph 'G' = (V, E) and let C = (S, V - S) be cut for which (u,v) is a light edge and T be one of the minimum spanning trees generated for 'G'.
There are two scenarios:

1. If (u,v) $\in T$, then our claim is proved.
2. If (u,v) $\notin T$, then $\exists$ (x,y) that crosses the cut and (x,y) $\in$T where cost(x,y) $\geq$ cost(u,v), meaning either:
   a) T is not a MST, or
   b) Edge (x, y) in T can be **swapped out** with edge (u, v) with no penalty to the cost:

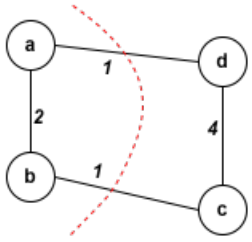
$$T' = T - (x, y) \cup (u, v)$$

Since T and T' have the same cost, and T is a MST, T' is also a MST for G.

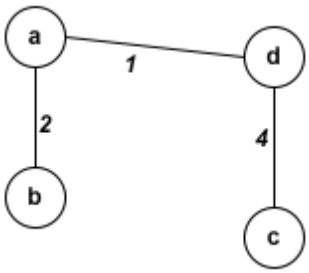
Thus, any light edge (u, v) crossing any cut of the graph is in *some* MST for graph G. ■

Note: The word "some" is emphasized because the light edge (u,v) can be in the first minimum spanning tree "T" or the other spanning tree M.

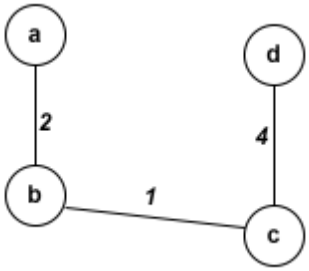
To understand this theorem let us consider the following example:



The graph 'G' has vertices V = {a,b,c,d}and edges, E= {ab=2,bc=1,ad=1,cd=4}
Let cut S = {a,b} then, V-S = {c,d} which has two crossing edges, (a,d)=1 and (b,c)=1
If (a,d) = 1 is chosen to be the light edge crossing the cut S. The minimum spanning tree "T1" that included (a,d) in it is,



Cost of the MST(T1) = 1+2+4 = 7
Another minimum spanning tree 'T' that can be generated with from G is,



Cost of the MST(T) = 1+2+4 = 7
In the MST 'T', adding the light edge (a,d) will create a cycle. This is because there is already a path from a to d via bc and adding another path without removing bc would lead to cycle. Also, we can observe that cost(bc) $\geq$ cost(ad).
Using the exchange method, we can add (ad) to T and delete bc. The updated spanning tree 'M' is as follows,

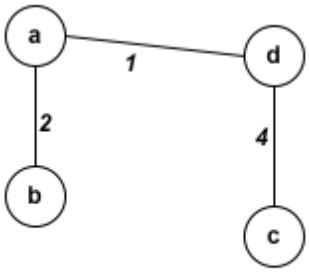


Fig. Updated MST 'M'

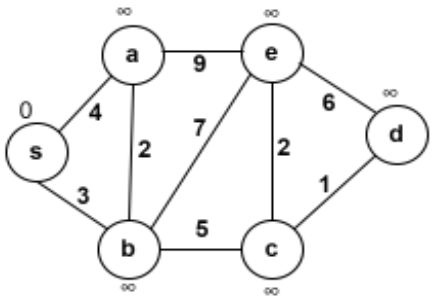
Here, (a,d) $\exists$ M and M is a minimum spanning tree with cost 7, which proves our theorem. This MST property leads to Prim's and Kruskal's algorithms.

2 Prim's Algorithm

Based on the Greedy Choice Property illustrated above, Prim's Algorithm grabs the shortest available path at every iteration:

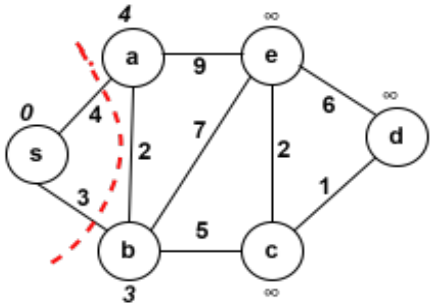
- start from any single vertex a, let $S = a, T = (\emptyset)$ where S is a collection of Vertices and T is a collection of Edges.
- Find the light edge (u, v) crossing the cut $(S, (V - S))$, then $T = T \cup \{(u, v)\}$, $S = S \cup \{v\}$
- While $(|T|) \leq (n - 1)$, repeat the above step;

How do we identify every light edge efficiently?
- A new cut evolves from an old cut; a light edge crossing the new cut may be identified with little effort;
How do we ensure that no cycles are created in the MST?
- Only light edges across the cut can be selected, meaning every iteration of Prim's will connect to a *new node* rather than an already connected node. No cycles will occur with $n - 1$ iterations.
An example graph 'G' has been taken to understand the working of Prim's using the algorithm given above.



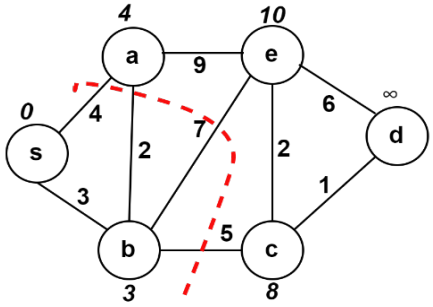
The vertex 's' is the source and has cost '0' while all the other vertices has cost set to ∞. The prev of all vertices initially is 'nil'. H = {s,a,b,c,d,e} and T = {}

1. When 's' is dequeued.
The cost of vertices 'a' and 'b' are updated to 4 and 3 respectively their prev is set to 's'.



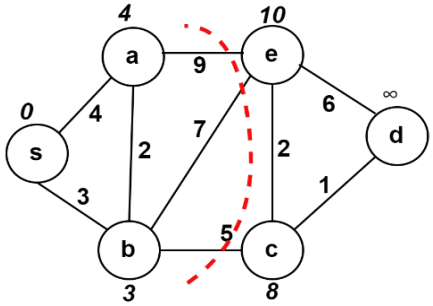
cost(s) = 0; cost(a) = 4; cost(b) = 3; cost(c) = ∞; cost(d) = ∞; cost(e) = ∞;
prev(s) = nil;prev(a) = s;prev(b) = s;prev(c) = nil;prev(d) = nil;prev(e) = nil;
The next vertex to be dequeued is 'b' since it has small cost. H = {a,b,c,d,e} and T = {s}

2. When 'b' is dequeued.
H = {a,c,d,e} and T = {s,b}



cost(s) = 0; cost(a) = 4; cost(b) = 3; cost(c) = 8; cost(d) = ∞; cost(e) = 10;
prev(s) = nil;prev(a) = s;prev(b) = s;prev(c) = b;prev(d) = nil;prev(e) = b;
Among the edges crossing the cut in the above picture, we can observe that ab=2 is the light edge. Hence, a is added to T next.

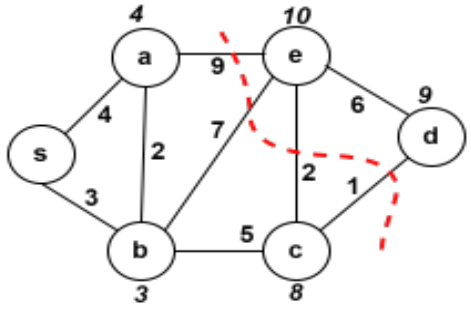
3. When 'a' is dequeued.
H = {c,d,e} and T = {s,b,a}



cost(s) = 0; cost(a) = 4; cost(b) = 3; cost(c) = 8; cost(d) = ∞; cost(e) = 10;
prev(s) = nil;prev(a) = s;prev(b) = s;prev(c) = b;prev(d) = nil;prev(e) = b;
Among the edges crossing the cut in the above picture, we can observe that bc=5 is the light edge. Hence, c is added

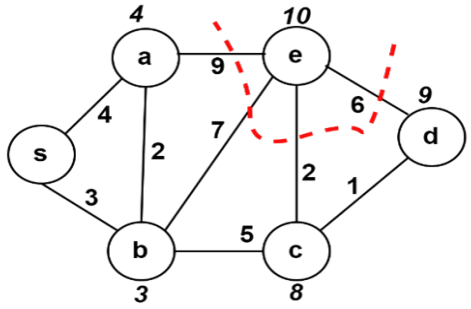
to T next.

4. When 'c' is dequeued.
H = {d,e} and T = {s,b,a,c}



cost(s) = 0; cost(a) = 4; cost(b) = 3; cost(c) = 8; cost(d) = 9; cost(e) = 10;
prev(s) = nil;prev(a) = s;prev(b) = s;prev(c) = b;prev(d) = c;prev(e) = b;
Among the edges crossing the cut in the above picture, we can observe that cd=1 is the light edge. Hence, d is added to T next.

5. When 'd' is dequeued.
H = {e} and T = {s,b,a,c,d}



cost(s) = 0; cost(a) = 4; cost(b) = 3; cost(c) = 8; cost(d) = 9; cost(e) = 10;
prev(s) = nil;prev(a) = s;prev(b) = s;prev(c) = b;prev(d) = c;prev(e) = c;
Among the edges crossing the cut in the above picture, we can observe that ce=2 is the light edge. Hence, e is added to T next.

6. When 'e' is dequeued.
H = {} and T = {s,b,a,c,d,e}
cost(s) = 0; cost(a) = 4; cost(b) = 3; cost(c)= 8; cost(d)= 9; cost(e)= 10;
prev(s) = nil;prev(a) = s;prev(b) = s;prev(c) = b;prev(d) = c;prev(e) = c;
The set H is empty and all the vertices are added to MST so we stop the algorithm here. Using the dequeued order and the previous pointers, the MST obtained is as follows,

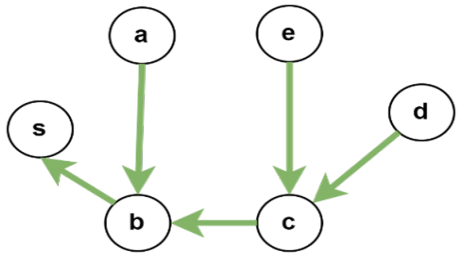


Fig. Minimum Spanning Tree

3 Kruskal's Algorithm

Kruskal's Algorithm is also based upon the Greedy Choice Property, but goes about satisfying this property differently than Prim's:

- Sort edges in non-decreasing order by weight
- Select the lightest existing edge that does not form a cycle with the currently connected nodes
- Repeat until there are $n - 1$ edges in the graph

This algorithm requires a special data structure to detect and avoid cycles: The **disjoint set**.
The Disjoint Set has three operations:

1. Make singleton set: $\{x\}$ - Only one element in set
2. Find set: Identify set that contains x
3. Union: Combine two sets: $\{x\} \cup \{y\}$

At the beginning of the algorithm, all nodes in the graph will be stored in disjoint singleton sets.

- By the end of the algorithm, all nodes will have been joined together with the Union operation.

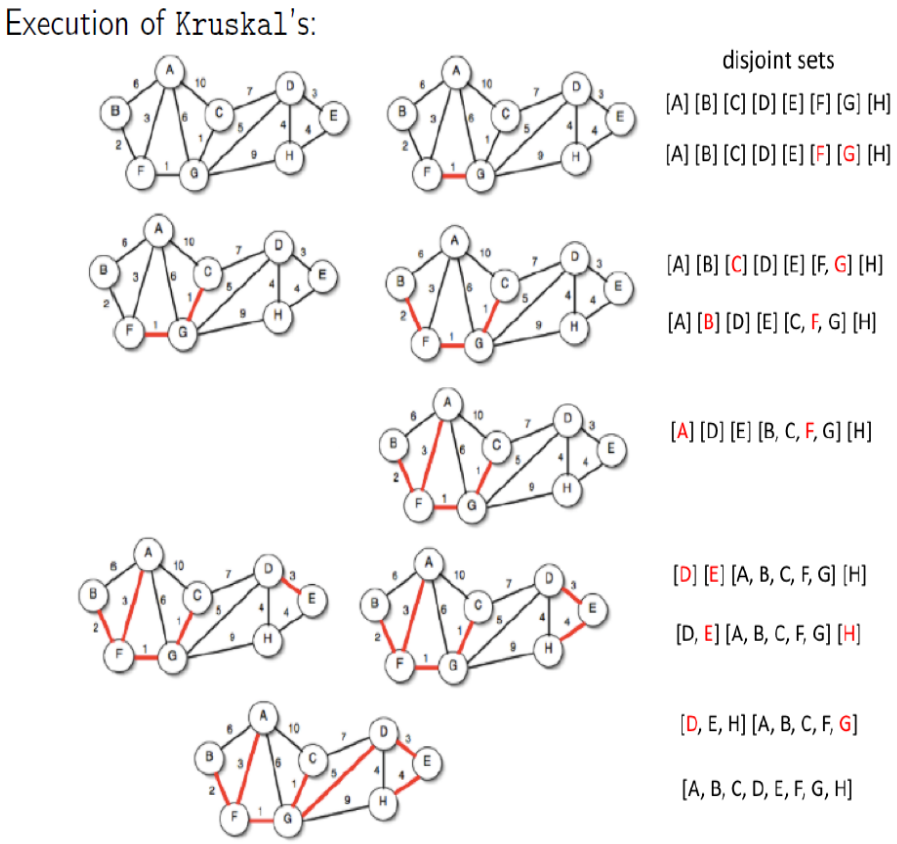
- To avoid cycles, the find() method will make sure two potentially connected nodes are not in the same set.

Note on above: This is accomplished through the find() with complexity $O(\log\, n)$ by using an *auto-balancing tree for each disjoint set*, the root of which is the same for two nodes that are in the same set.

The code implementation of Kruskal's Algorithm for finding the minimum spanning tree of a graph is provided below.
function Kruskal $(G = (V, E),\, w)$

1. Sort edges by weight in the nondecreasing order;
2. forest $F = \emptyset$;
3. for every edge (u, v) in the sorted order;
4. if u and v not belonging to the same tree in F
5. $F = F \cup (u, v)$;
6. update forest F;

Example of Kruskal's Algorithm



Explanation of the example:
The non-decreasing order of the edges given the graph 'G' is,
FG=1; GC=1; FB=2; FA=3; DE=3; DH=4; HE=4; GD=5; GA=6; AB=6; DC=7; GH=9; CA=10;

- Initially all the vertices of the graph are represented as singleton sets.
  `[A] [B] [C] [D] [E] [F] [G] [H]`
- From graph we can observe that FG = 1 is the edge with least cost and since F and G does not belong to same tree, we add them to forest.
  `[A] [B] [C] [D] [E] [F,G] [H]`
- Next, GC = 1 is added to forest because C and [F,G] are in different sets.
  `[A] [B] [C,F,G] [D] [E] [H]`
- FB = 2 is added next to the forest because B and [C,F,G] are in different sets and adding this does not form a cycle.
  `[A] [B,C,F,G] [D] [E] [H]`
- FA = 3 is added next to the forest because A and [B,C,F,G] are in different sets and adding this does not form a cycle.
  `[A,B,C,F,G] [D] [E] [H]`
- Next, DE = 3 is added next to the forest because E and D are in different sets.
  `[A,B,C,F,G] [D,E] [H]`

- DH = 4 is next in the order to be added to the forest. Since H and [D,E] are in different sets we added them to single set by make-set operation.
  `[A,B,C,F,G] [D,E,H]`
- HE = 4 is next in the order to be added to the forest. Since [H,D,E] are in the same sets already, adding this edge would result in a cycle. Hence we continue to next edge by ignoring this edge.
  `[A,B,C,F,G] [D,E,H]`
- GD = 5 is next in the order to be added to the forest. Since [A,B,C,F,G] and [D,E,H] are in different sets we added them to single set by using make-set operation.
  `[A,B,C,F,G,D,E,H]`

Since all the vertices are included in the MST and adding next edges in the list to the MST would lead to cycles, we ignore those edges. The final MST obtained is highlighted above using red lines.
The cost of the minimum spanning tree = 1 + 1 + 2 + 3 + 3 + 4 + 5 = 19

Limitations of Kruskal's Algorithm:
Kruskal's algorithm has a worst-case time complexity of $O(m)$.

- In this worst case, all the light edges chosen form a cycle and all edges must be considered

- In comparison, **Prim's algorithm** has a worst-case time complexity of $O(n)$.

4 While loop and set T relationship in MST by Prim's:

The following is further explanation into the inductive reasoning behind the while loop in Prim's Algorithm. *At each iteration*, it can be said that the current working tree F is part of some MST T spanning graph G.

Theorem

Claim: At every iteration of the while loop in algorithm Prim, working tree F is contained in some MST T.

Proof:

Base Case: $k = 1$; The first edge picked up by node S is necessarily part of MST T, since it is the lightest edge connecting that node to the rest of the tree.

Assumption: At k^{th} iteration the claim is true for $F \subseteq T$ (For some MST T).

Induction: At $(k+1)^{\text{th}}$ iteration, let (u, v) be the current edge that has to be added to the tree. There are two scenarios:

1. if $F \cup (u, v) \subseteq T$, our claim is proved.
2. else, there is an edge (x, y) where $w(x, y) \geq w(u, v)$ which has to be swapped in to obtain a MST T and edge (u,v) will not be included as it forms a cycle (some other edge further on leads to node v with cost equal to or less than $w(u, v)$.
   - Note: Due to the Greedy Choice property for MSTs, if $w(x, y) > w(u, v)$ is included in the tree, then it is *not* a Minimum Spanning Tree

Thereby, at each iteration of Prim's algorithm, the current working tree corresponds to a subset of nodes and edges in *some* MST of graph G. ■

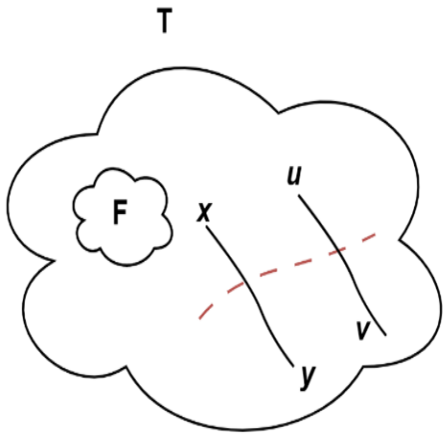


Figure: Two choices at current iteration of Prim's algorithm, where $w(x, y) \geq w(u, v)$.