EEcS3101 notes:: Graph algorithms continued

Jerry Wu

2023-03-23

Minimum spanning trees (MST)

MST in an undirected and weighted graph

This will be our main subject of interest today. Say we have a graph $G = (V, E)$ where we have a weight for each edge $w(\epsilon)\forall\epsilon \in E$. We want to find a sub tree (connected and acyclic subgraph) in the graph that covers every vertex $v \in V$ and has the minimum possible total weight. **A good application of this idea** would be to build a road network connecting all cities in a group of n cities while minimizing the cost, or connecting all components of a circuit with the least amount of wiring. More examples would include::

- Cluster analysis
- Approximation algorithms for the travelling salesman problem (can never be exact because it is NP complete).
- The list goes on!

What is a tree?

A tree is an **undirected**, **connected**, and **acyclic** graph that has the following properties::

- A tree with n vertices has exactly $n - 1$ edges.
- Removing any edge from the tree will cause it to become **disconnected** $\implies$ not tree.
- Adding an edge anywhere in the tree will create a cycle $\implies$ not tree.
- $MST(G)$ yields exactly $|V|$ vertices and $|V|-1$ edges by way of the tree property.
- Not necessarily unique. There can be multiple $MST(G)$, but we only need to find one of them.

Our goal with MST is to find a **subset** of the edge set E in the original graph such that we get a tree.

MST algorithms

Start with a $G = (V, E)$, continue to delete edges until we end up with $T = (V, E)$ such that $T \subset G$. It is also tangible to start with no tree and construct a tree by adding edges between vertices. Which one is more efficient? Well, it would depend on the size of the graph.

> *"Don't just say 'it depends'. Keep talking!"-Larry YL Zhang 2023*

- The subtraction based one would take at most $|E| - |V|$ iterations ($|V|^2$ in the worst case, i.e. complete graph), whereas the addition based one would take at most $|V|$ iterations.
- So our best bet is to "grow" the tree from bottom up (NOT DYNAMIC PROGRAMMING!) as a means to save time instead of breaking the graph to form a tree. So this solution would also be more environmentally friendy.

> *"Resist the temptation to solve all problems with dynamic programming"-Larry YL Zhang 2023*

We can now create some pseudocode.

```
GENERIC_MST(G=(V,E,w)):
    T=Null
    while T is not a spanning tree: #(|T|<|V|-1)
        find safe edge e
        T=Union(T,{e})
    return T
```

Safe edge

"Always have the hope to become an MST. When you're ready for it, you will become it!"-Larry YL Zhang 2023

A safe edge is a loop invariant $T \subset someMST$. If we always make sure that T is always a subset of an MST while growing it, it will eventually become an MST. A safe edge keeps the hope of a tree becoming an MST. How can we find it? We can utilize the following algorithms::

- Kruskal's algorithm
- Prim's algorithm

Both are based on the following::

Theorem

- Let $G = (V, E)$ be connected, undirected, and weighted.
- Let $T \subset E$ such that it is included in some $MST(G)$.
- Let C be a connected component (tree) in the forest $G = (V, E)$.
- Let (u, v) be a minimum weighted edge crossing C and some other component in G_T.
- Then the edge (u, v) is safe for T.

This is known as the greedy choice property of MST (always choose the lowest weighted edge/safest edge). It will work no matter which vertex you start to grow the tree at! (example on slide 24).

Things to keep in mind when implementing

- We need to keep track of all the components we have so far.
- How can we find the safe/minimum weighted edge efficiently?

This is where the two algorithms discussed earlier will come into play (animated example on slide 27).

- Prim's algorithm uses a single tree along with isolated vertices to keep track of connected components, and a priority queue (max/min heap) to find the minimum weighted edge. Centered on the root of the tree.
- Kruskal's uses a disjoint set to track connected components and a sorted list of all edges according to their weights to find min edge. More decentralized.

Prim's algorithm (pseudocode on slide 30 with comments)

- Start with some arbitrary $v \in V$ as the root.
- Focus on growing one tree, adding one edge at a time. The current tree is one component, and the isolated vertices are all their own individual components.
- Which edge should we add to the tree? Among all edges that are **incident** to the current tree, pick the **minimum** weighted edge.
- To get the minimum weighted edge, store each weight corresponding to each edge in a min heap, whos key is the weight of said weighted crossing edge.

Runtime analysis

Assuming we use a binary min heap, the worst case runtime would be $V \log(|V|)$. $ExtractMin$ takes $\log(|V|)$ time, and we check all $|E|$ edges.

Kruskal's algorithm (pseudocode on slide 52)

- Sort all weights in ascending order, then start adding to MST from the lowest weight.
- Constraint:: Added edge cannot create a cycle! (must cross 2 components. Cannot be in the same component)
- This process is similar to taking the union of many smaller trees to form a bigger tree.

We need the disjoint set ADT to use Kruskal's, but this is a topic for EECS4101.