ECS 122A Lecture 12+13 Jasper Lee

Minimum Spanning Tree

Setting: Given weighted undirected connected graph

compute spanning tree w/ min total edge weight.

Recall: Spanning tree is a tree connecting all vertices, hence w/ n-1 edges.

Two standard greedy algorithms

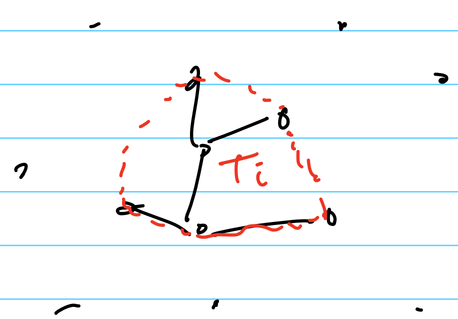
- Prim's alg
- Kruskal's alg

Prim's alg

Idea: Grow a spanning tree from a seed, one edge at a time.

Let's try deriving the alg

At some intermediate stage:



What does any spanning tree need to do?

Connect T_i w/ some vertex outside of T_i

Obs: Might as well take min-weight edge b/w T_i and $G \setminus T_i$

Alg

Connected $= \{s\}$ // u arbitrarily chosen

$T \leftarrow \phi$ // empty tree, no edges.

while $\exists$ edge b/w connected and $V \setminus$ connected

$(u, v) \leftarrow \min\limits_{\substack{(u,v) \in E \\ u \in \text{connected} \\ v \notin \text{connected}}} w(u,v)$

Add (u, v) to T

Add v to connected

return T

Correctness

Induction hypothesis:

At the end of i^{th} iter, $\exists$ an MST T_i^{opt} s.t. T_i^{opt} contains all edges in T.

Base case: $i = 0$, trivial.

Induction step:

Assume IH for iter i, so there is an MST T_i^{opt} containing all of T_i (state of T at end of iter i)

Call (u,v) the new edge added in iter $i+1$
$u \in T_i$ $v \notin T_i$

If $(u,v) \in T_i^{opt}$, then we're done.

Otherwise, $(u,v) \cup T_i^{opt}$ has a cycle.

$\exists\ (a,b) \neq (u,v)$ in cycle s.t.

$a \in T_i$ and $b \notin T_i$

(Why? Walk along cycle from u.
Going to $u \to v$ goes from T
to $V \setminus T$. Eventually end up back at u which is in T.)

$w(a,b) \geq w(u,v)$ by alg.

So swap to get T_{i+1}^{opt}

$w(T_{i+1}^{opt}) \leq w(T_i^{opt})$

T_{i+1}^{opt} still has $n-1$ edges, connects all vertices
hence is a spanning tree,
hence MST //

Wrap up: Upon termination, T connects all vertices reachable from s, and $\exists$ an MST containing all of T. So T is an MST.

Implementation?

Hard part: Find min outgoing edge from T.

Idea: Use heap data structure

Heaps:

(key, value)

- Insert(x), adds x into DS
- Extract Min, returns element w/ min key and deletes from DS
- Delete(pointer to element x), deletes x from DS

Fact: There is a heap implementation s.t. all operations take $O(\log n)$ time, where n is the # of elements in the heap at the time.

How to use a heap here?

Store vertices in heap to decide which one to grow T to

Invariant: At the end of every loop iter, heap H contains all vertices v in $V \setminus \text{connected}$ w/ key $\min_{u \in \text{connected}} w(u,v)$.

Pseudocode:

```
connected ← {s} ; T ← ∅ ; H ← empty heap
pointers ← array indexed by V, init to NULL
for every v ≠ s
    if (s,v) ∈ E then   key(v) ← w(s,v) ;
    else                key(v) ← ∞ ;
    edge(v) ← (s,v) ;
    Create (key(v), (v, edge(v))) w/ pointer pv
    pointers[v] ← pv
    add                                   to H

while H is non-empty
    (key(u), (u, edge(u))) ← ExtractMin(H)
    add u to connected, edge(u) to T
    for each edge (u,v) where v ∉ connected
        if key(v) > w_uv
            delete v from H
            key(v) ← w(u,v) ; edge(v) ← (u,v)
            create ...       w/ pt pv
            pointers[v] ← pv , add ... to H

return T
```

$O((m+n) \log n)$ time

//

Dijkstra's algorithm

Single-source shortest path again

Bellman-Ford : $O(|V||E|)$ time, allows negative weights

Dijkstra's : Greedy $O(|V| \log |V|)$ time, no negative weights

Presentation slightly different from "standard" sources

Def Shortest Path Tree

A (directed) tree T is a shortest path tree w.r.t. directed graph G and source vertex s if 1. T contains all reachable vertices from s
2. T is a subgraph of G

and 3. for all such vertex v, $s \rightsquigarrow_T v$ is a shortest path in G.

Existence

For any vertex v (reachable from s), choose any shortest path $s \rightsquigarrow_G v$ with fewest hops.

Let $prev(v)$ be vertex before v on this path.
$hop(v)$ be # hops on chosen path.

Obs: $hop(prev(v)) = hop(v) - 1$.

Exercise to prove.

n-1 edges → Construct shortest path tree hop-by-hop, adding edges $prev(v) \to v$ for every v.

Exercise to show this is a shortest path tree. (by induction on # hops).

Idea: Greedily grow shortest path tree from source s. (Like Prim's).

<u>Alg</u>

Connected $\leftarrow \{s\}$
$T \leftarrow \emptyset$

while $\exists$ edge from connected to $V \setminus$connected

Find $v = \arg\min_{v \notin \text{Connected}} \min_{u \in \text{Connected}} w_{u \to v} + dist_T(s,u)$

Add $u \to v$ to T
v to connected

return T (and its dist is shortest path dist in G)

Exercise: Figure out how to implement this in $O(|V| \log |V|)$ time.

Correctness:

Induction hypothesis:

At the end of every iter i, T_i is such that

1. $connected_i$ is set of v reachable from s in T_i and # edges in T_i = $|connected| - 1$

2. $\exists$ shortest path tree T_i^{opt} containing all edges in T_i.

Cor: $dist_{T_i}(s, v) = dist_{T_i^{opt}}(s, v)$
$\geq$ since T_i^{opt} is shortest path tree
$\leq$ since T_i^{opt} contains all of T_i

Induction step:

1. is super basic. Every v we add is reachable from s by induction

2. Suppose T_i^{opt} is a shortest path tree containing all of T_i (state of T after ith iter)

let $\underset{\in T_i}{u} \rightarrow \underset{\notin T_i}{v}$ be edge chosen in iter $i+1$.

Suppose $u \rightarrow v \notin T_i^{opt}$, then in T_i^{opt}, the path

$s \rightsquigarrow_{T_i^{opt}} v$ must contain $\underset{\in T_i}{a} \rightarrow \underset{\notin T_i}{b}$ where $a \rightarrow b \neq u \rightarrow v$

So $s \rightsquigarrow_{T_i^{opt}} a \rightarrow b \rightsquigarrow_{T_i^{opt}} prev(v) \rightarrow v$

By alg, $\mathrm{dist}_{T_i}(s, a) + w_{a \to b} \geq \mathrm{dist}_{T_i}(s, u) + w_{u \to v}$

So $\mathrm{dist}_{T_i^{opt}}(s, v) \overset{\text{no neg weight edge}}{\geq} \mathrm{dist}_{T_i^{opt}}(s, a) + w_{a \to b}$

$\overset{\text{alg}}{\geq} \mathrm{dist}_{T_i^{opt}}(s, u) + w_{u \to v}$

So remove $\mathrm{prev}(v) \to v$, add $u \to v$ to get T_{i+1}^{opt}.

Reachability preserved, dist no worse. //

Wrap up: At termination, T^{opt} contains T, has same number of vertices. So edge count the same (|connected| − 1) ⇒ same tree □