11.0

Last class started discussing algs for Minimum Spanning Trees

Input: undirected connected $G=(V,E)$, $w: E \to \mathbb{R}$

Output: T s.t. $w(T) := \sum_{e \in T} w(e)$ is minimal over T a spanning tree

Greedy: at each step we add to our slowly growing tree an edge (u,v) s.t its wt is minimal among all "allowed" edges

We call such edge a *safe* edge.

~~Thm~~ An edge is safe if when we add it to our set we still hold a subset of ~~an~~ MST

We also proved.

Thm: (23.1) $G=(V,E)$ connected undirected, $w: E \to \mathbb{R}$. $A \subseteq E$ is included in some MST. $(S, V \setminus S)$ a cut respecting A (no edge of A crosses the cut). (u,v) light edge for the cut. Then (u,v) safe for A.

Corolary: Let $C = (V_c, E_c)$ be a connected component in $G_A = (V,A)$. If (u,v) ~~is~~ a light edge in the $(V_c, S \setminus V_c)$ cut then it is safe for A.

This is the basic idea behind MST algs!

Today: MST algs.

Single source shortest path

11.1

We start by giving Kruskal's alg for MST. Then Prim's. The only difference is the rule according to which we find a safe edge.

Kruskal's alg

The idea: Each vertex forms a connected component in G_A when $A = \emptyset$.

We then gradually merge components using lightest available edge.

The alg. uses a data structure that manages disjoint sets – think of a c.c. as a set of vertices. For each set S there will be a representative vertex.

Allowed operations are:

Make-set(v): creates a set $\{v\}$.

Find-set(u): given u it finds the rep. ~~vertex~~ element in u's set.

Union(u,v): Merges the sets that contain u,v.

Not difficult to implement this D.S. with $O(\log|V|)$ cost for basic ops.

Kruskal's alg

MST-Kruskal(G, w)

1. $A = \emptyset$
2. For each $v \in G.V$
3. ~~the~~ Make-set (v)
4. Sort edges in E in nondecreasing order according to w.
5. For each $(u,v) \in E$ (taken according to the sorted order from light to heavy)
6. If Find-set(u) $\neq$ Find-set(v)
7. $A = A \cup \{(u,v)\}$
8. Union(u,v)
9. Return(A)

Run time: lines 2-3 $O(V)$

line 4 $O(E\lg E)$

lines 5-8 $O(E\lg V)$

$\Rightarrow$ total $O(E\lg V)$

Correctness: by Corollary. We unite two trees by a lightest edge. ∎

Prim's alg

Main difference: instead of ~~connect~~ starting from many connected component we grow a tree (so only one component). Each step adds a light edge to some isolated vertex.
The idea is to keep for each v not in our tree the wt of the lightest edge connecting it to the tree and then choose the vertex with min. value.
We implement this with a min priority queue.

MST-Prim (G, w, r) - r is the root.

```
1.  For each u ∈ G.V
2.        u.key = ∞
3.        u.π = NIL
4.  r.key = 0
5.  Q = G.V       (min-priority-queue)
6.  While Q ≠ ∅
7.        u = extract-min(Q)
8.        For each v ∈ G.Adj[u]
9.            If v ∈ Q and w(u,v) < v.key
10.               v.π = u
11.               v.key = w(u,v)
```

11.3

<u>Run time:</u> 1-4 $O(V)$

5 $O(V)$ (we ~~did~~ showed it after ...)

6 $O(1)$

7 $O(\lg |V|)$ done $|V|$ times so $O(V \lg |V|)$

8-11 We go over all edges eventually

and step 11 requires fixing Q so it takes $O(\lg |V|)$

so overall $O(E \lg |V|)$

<u>Correctness:</u> At each step A is a tree and v.key for $v \in Q$ is lightest edge to A from v. (loop inv.) By cos. we get correctness.

<u>Note:</u> if we use a data structure called Fibonacci-heap then run-time improves to $O(E + V \lg V)$ (in Prim's alg.)

11.4

Single source shortest path.

Similar to BFS just now edges have weights (that can be negative...) but later on that

An instance of the problem: A map with road lengths and we wish to find all distances from, say, NYC to all other cities.

As before $G=(V,E)$ directed, $w: E \to \mathbb{R}$

for a path $p=(v_0, v_1, \ldots, v_k)$, $wt(p) = \sum_{i=1}^{k} wt(v_{i-1}, v_i)$

shortest-path-weight $\delta(u,v)$ from u,v is

$$\delta(u,v) = \begin{cases} \min w(p) & p \text{ path from } u \text{ to } v \\ \infty & \text{no such } p \end{cases}$$

<u>Important variants</u>

<u>Single destination shortest path</u>: Find shortest paths to a given destination from all other vertices

(Same as S.S.S.P by reversing edge directions)

<u>Single pair shortest path</u>: Find s.p. between u and v. Simpler but all known algs. have same run-time as s.p.

<u>All pairs s.p.</u>: Find all s.p for all pairs. Can run S.S.S.P n times but faster alg. known.

11.5

Optimal substructure

For greedy/dynamic we need to understand this.

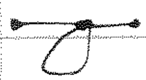
Claim: if $p = \langle v_0, v_1, \ldots, v_k \rangle$ s.p. from v_0 to v_k then for any $i < j$ $\langle v_i, v_{i+1} \ldots, v_j \rangle$ s.p. between v_i to v_j.

Pf: o.w. can get a shorter path... ∎

Negative weights

Doesn't come up for distances but an important case for some applications.

Notice that if G contains a cycle whose total wt is negative then

 has wt $-\infty$ ...

so cycles of negative wt ruin the problem.

What about cycles with positive wt? no s.path will contain any such cycle.

Representing s.s.s.p

By the optimal sub structure we can represent it by a tree. (prove to yourself)

Thus our alg will maintain a tree rooted at u.

How do we build the tree?

First it is empty then we add edges. An important step is modifying the tree when a shorter path is found. This is called relaxation.

11.6

First:

Initialize-Single-Source (G, s)

```
1. For each v ∈ G.V
2.     v.d = ∞
3.     v.π = NIL
4. s.d = 0
```

Relax (u, v, w)

```
1. If v.d > u.d + w(u,v)
2.     v.d = u.d + w(u,v)
3.     v.π = u
```

If it is shorter to first get to u and from it to v then we do it.

We now give the Bellman-Ford alg. It also says whether there is a Negative cycle (outputs False)

Bellman-Ford (G, w, s)

```
1. Initialize-Single-Source (G, s)
2. For i = 1 to |G.V| - 1
3.     For each edge (u,v) ∈ G.E
4.         Relax (u, v, w)
5. For each edge (u,v) ∈ G.E
6.     If v.d > u.d + w(u,v)
7.         Return False
8. Return True
```

$\left.\right\}$ |V|-1 steps of relaxing according to all edges

$\left.\right\}$ checking for a negative cycle.

<u>Run time:</u> $O(V \cdot E)$ by lines 2-4

11.7

Important lemma:

Lemma (Path-Relaxation-Property):

If $p = \langle v_0, \_\_, v_k \rangle$ is a s.p from v_0 to v_k and we relax edges of p in the order $(v_0, v_1), (v_1, v_2), \ldots$ then after these k relaxations $v_k.d = \delta(v_0, v_k)$. This remains true even if other relaxation steps occur.

Pf: By induction on k. Clear for $k=0$. General case follows easily from optimal structure. ∎

Thus, since any path is of length $\leq n-1$ correctness follows, assuming no negative cycle.

If there is a negative cycle the alg. cannot return true!

If cycle is $s = v_0 \_\_, v_k = s$ then

$$\sum_{i=1}^{k} (v_{i-1}, v_i) < 0.$$

But, if alg. answers true then

$$v_i.d \leq v_{i-1}.d + w(v_{i-1}, v_i) \quad \forall \quad i = 1 \ldots k$$

$$\text{Thus} \quad \underbrace{\sum_{i=1}^{k} v_i.d \leq \sum_{i=1}^{k} v_{i-1}.d}_{\text{equal}} + \underbrace{\sum_{i=1}^{k} w(v_{i-1}, v_i)}_{\text{Negative}}$$

in contradiction. ∎