Advanced Algorithms

Lecture 9: Greedy algorithms (contd.)

Announcements

- HW 2 is out! Due this Friday — ask on Piazza
- HW 1 grading, comments

(contact: Vivek).

Recap: greedy algorithms

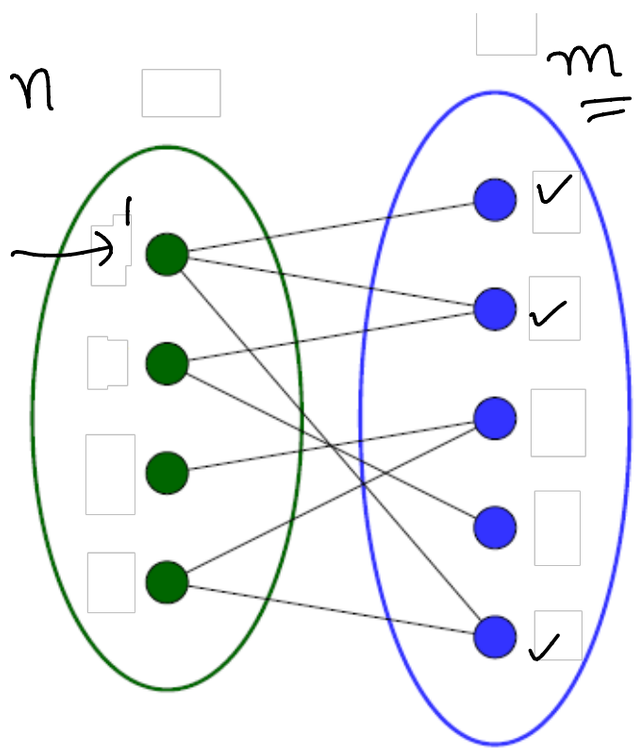
- Similar to DP — sequential decision making
- At each step, make the "best choice" based on some *value* only involving current state
- Choice made is irrevocable — no back-tracking
- **Examples:** coin change (min # of coins), scheduling, "set cover"

Comments

- Greedy algorithms usually “easy” to come up with (we are naturally myopic)
- Usually not optimal — examples, Traveling salesman, set cover, ...
- (Due to this..) analysis is usually tricky

Set cover

Problem: suppose we have *n* people, and *m* "desired skills"; each person has a subset of the skills. Pick the smallest number of people such that every skill is *covered*



Sequential decision making:

- pick one person at a time.

Greedy algorithm

- Maintain list of uncovered skills; call it U (initially $[m]$)
- Iteratively add person with most number of “uncovered” skills, until all skills are covered

Theorem

Conclusion: Greedy is not always optimal, but it is not "terrible". (off by a factor of only $\log m$)

Theorem. Suppose there is an optimum solution that uses k people. Then the greedy algorithm does not use more than $k \log m$.

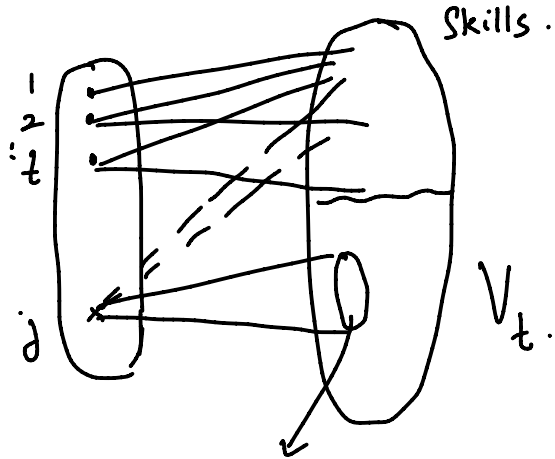
Key idea: many skills are covered at each step! (assuming there is a "small" solution.)

$V_t :=$ set of skills uncovered after picking 't' sets

$u_t = |V_t|$

Claim. $\boxed{u_{t+1} \leq u_t - \frac{u_t}{k}.}$

arguing about what happens in greedy alg using only the size of opt solution.



$\geq \frac{|V_t|}{k}$.

<u>Claim</u>: $\exists j$ such that

$$|S_j \cap V_t| \geq \frac{|V_t|}{k}.$$

proof of this used the opt solution

Completing the proof

Claim: The procedure cannot go on for $k \log m$ iterations.

u_t : # of uncovered skills after t iters.

Proof:

$$u_{t+1} \leq u_t - \frac{u_t}{k} \quad \text{for every } t.$$

$$= u_t\left(1 - \frac{1}{k}\right).$$

$$\leq u_{t-1}\left(1 - \frac{1}{k}\right)^2 \leq \dots \leq u_0\left(1 - \frac{1}{k}\right)^{t+1}$$

What is u_0? $= m$.

$$u_t \leq m\left(1 - \frac{1}{k}\right)^t \rightsquigarrow < 1 \ ?$$

$\log_{(1-\frac{1}{k})} m$

<u>want:</u> $m\left(1-\frac{1}{k}\right)^t < 1.$

$$\left(1-\frac{1}{k}\right)^k \approx \frac{1}{e}.$$

if we were to set $t = k \cdot \alpha$, then

$$m\left(1-\frac{1}{k}\right)^t = m \cdot \left(\frac{1}{e}\right)^\alpha$$

if this is <1, need $\alpha > \log_e m$.

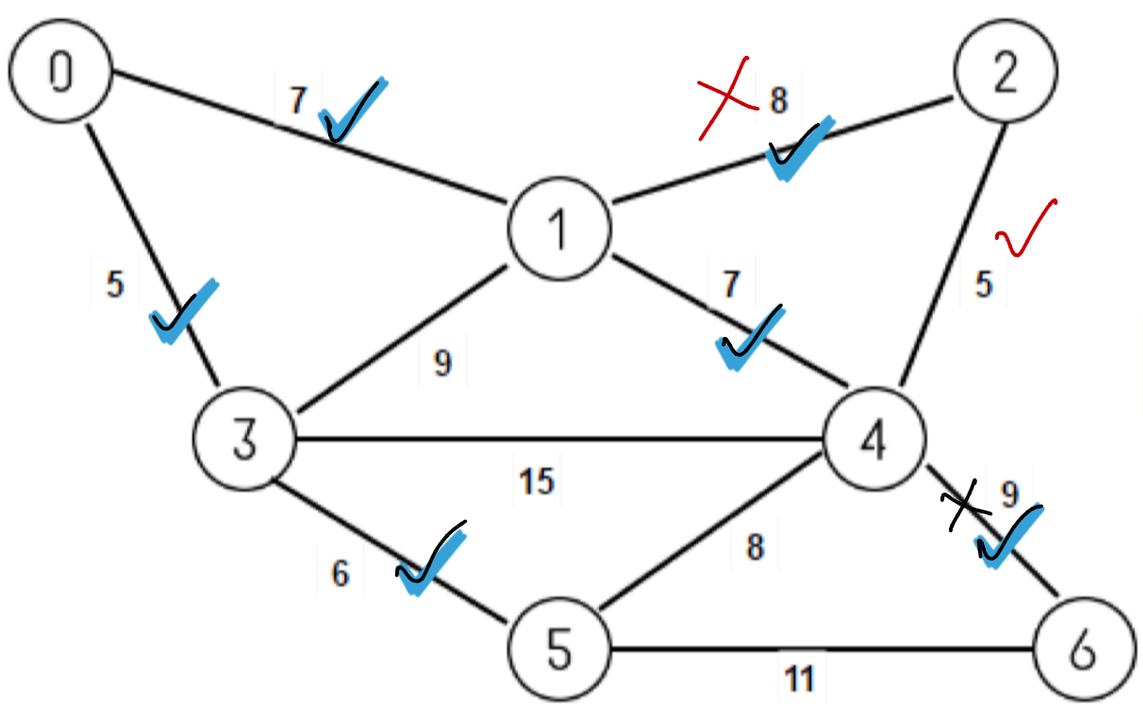
$\therefore$ overall, need to set $t = k \log_e m$.

2, 7, 3, 6

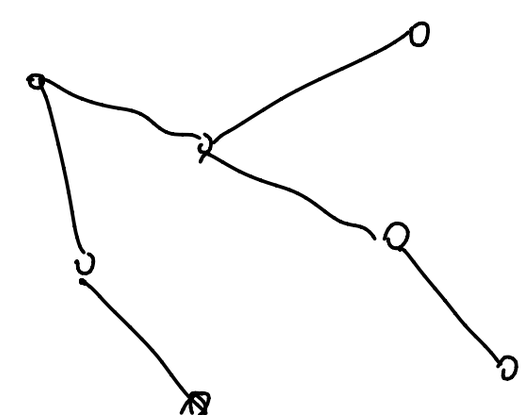
Example 2: spanning trees

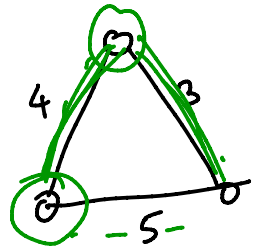
Problem: let G = (V, E) be a (simple, undirected) graph with edge weights $\{w_e\}$ (>0). Pick a subset of the edges, such that (a) all vertices are "connected", (b) total weight of edges picked is minimized

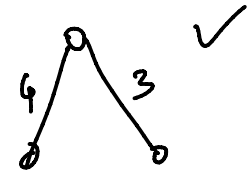
(Communication backbone in a network)

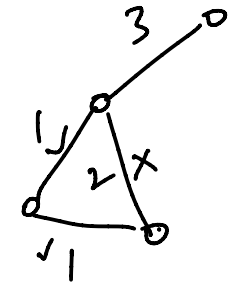


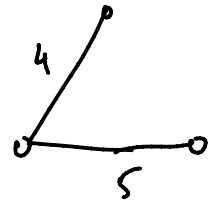
Note: optimal solution is always a tree











<u>Candidate alg:</u> - start with edge of least wt.

- keep adding edges until you've covered all vertices.

(while adding uv, check if u is already reachable from v, add only if it's not).

Greedy strategy

- **Goal:** need to connect all vertices to one another

(Prim's algorithm).

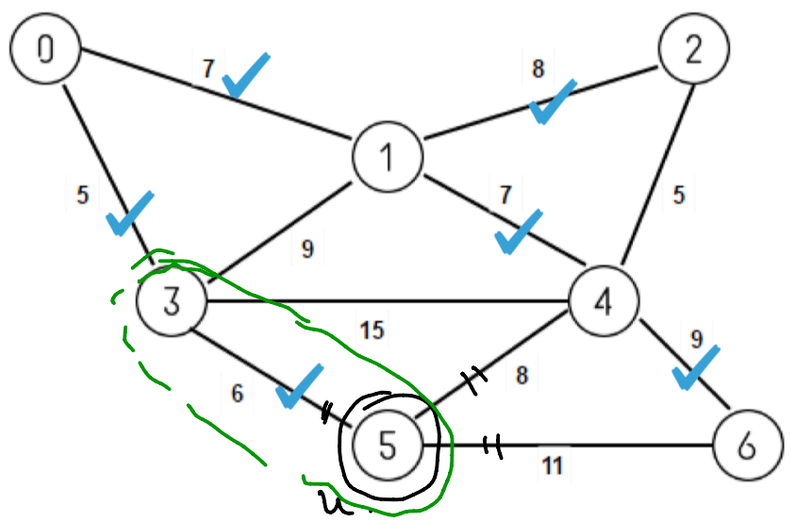
- Greedy 1: Start with one vertex, add a new vertex to connected set each time

- Greedy 2: Add edges one at time, choose min weight edge that isn't "redundant"

(Kruskal's algorithm).

Surprise: both turn out to be optimal!

Prim's algorithm



Building connected component one vertex at a time

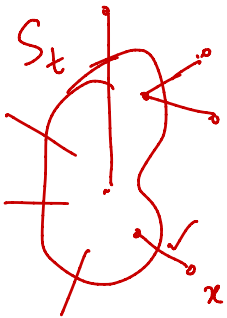
Start with $S_1 = \{u\}$ (any vertex) $= \{5\}$.

Of all edges going out of S_1, add the one with the least weight, & increment S_1; (now $S_2 = \{5, 3\}$.

[Candidate choices for S_3 are $\{3, 5, 6\}$, $\{3, 5, 4\}$, $\{3, 5, 1\}$ & $\{3, 5, 0\}$].

Overall algorithm:

- Start with $S_1 = \{u\}$, for any $u \in V$.
- For $t = 1, \ldots, n-1$:
  - consider all edges going "out" of S_t.
  - pick the one of the least wt & add to solution
  - increment: set S_{t+1} appropriately.



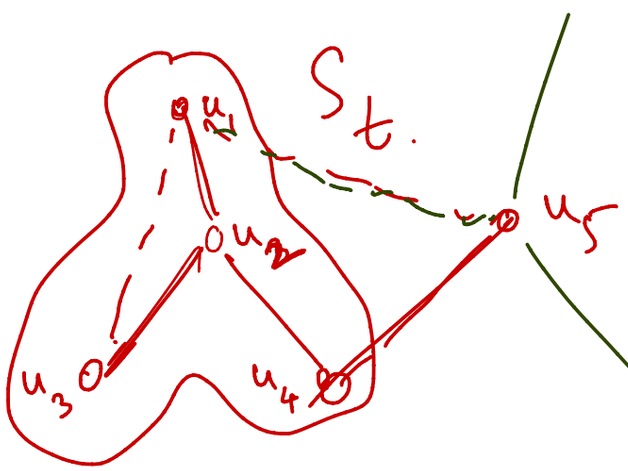
$S_{t+1} = S_t \cup \{x\}$.

Correctness

we don't have a good "structure" for MST of $\{u_1, u_2, u_3, u_4, u_5\}$

- **Observation:** at each iteration, we have a *set of connected vertices* — S_t

- Candidate claim: Set of edges we added so far is a min spanning tree of S_t



- edges chosen so far form MST for $\{u_1, u_2, u_3, u_4\}$.

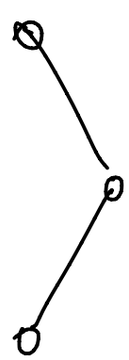
like to

adding edge $u_4 u_5$ gives MST for $\{u_1, u_2, u_3, u_4, u_5\}$.

Correctness

- **Observation:** at each iteration, we have a *set of connected vertices* — S_t

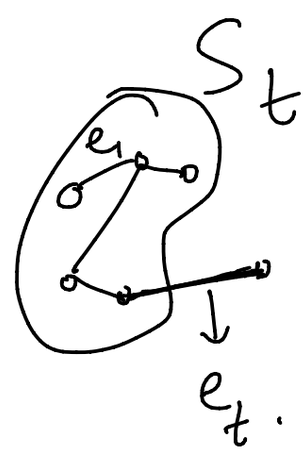
- Will show: There exists a min spanning tree for the *full graph* that contains all edges chosen so far



We will prove this by induction.

Base case: $t = 1$ → no edges → nothing to prove.

Inductive step:



let $e_1, e_2, \ldots, e_{t-1}$ be the edges chosen so far, & let e_t be the new edge added

Proof of "opt prefix" property

Knowing that there is an OPT tree that contains

$\{e_1, \ldots, e_{t-1}\}$, need to prove that

there is an OPT tree that contains

$\{e_1, \ldots, e_t\}$.

Proof of “opt prefix” property

Running time