Mon, March 8

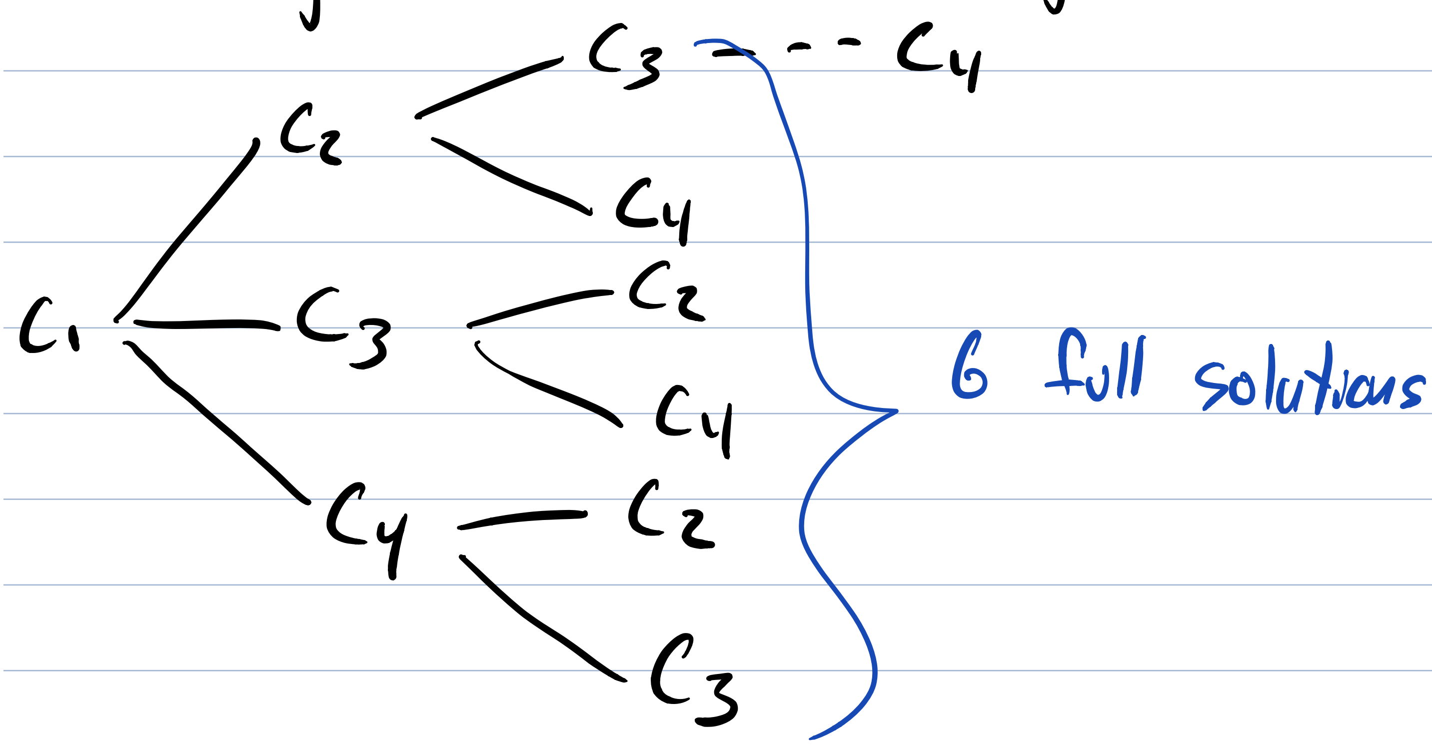
Announcements:

* Tuesday OH moved to 10am - 11am
* No lecture on Wednesday
* HW 3 due next Mon

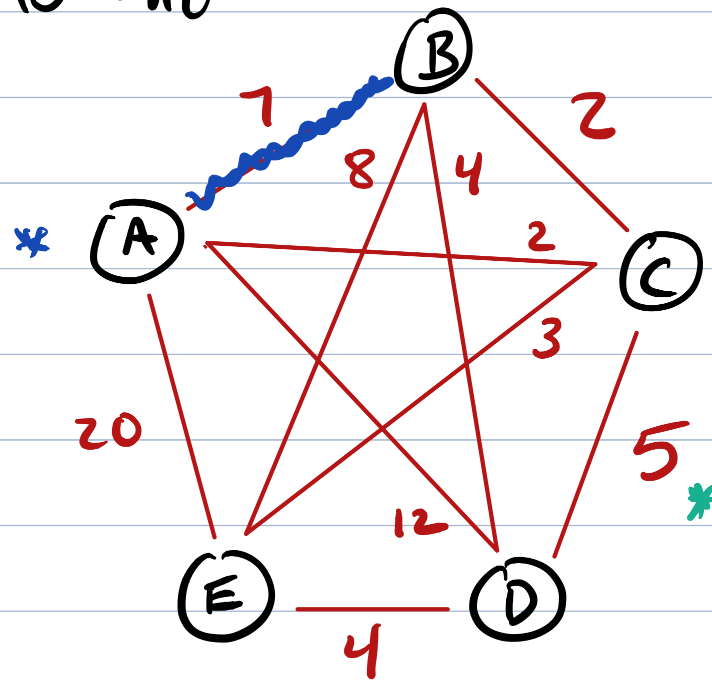
Last class: TSP.

- 4 cities, C_1, C_2, C_3, C_4

Branching: Pick a start city: C_1



Bound:



If we start at A, then go to B, what is a lower bound on the cheapest we can finish?

* We're going to have to exit B, which will cost at least 2.

* We will have to enter + exit C: cost at least 2+2=4
* D: 4+4=8
* E: 4+3=7
* A: 2

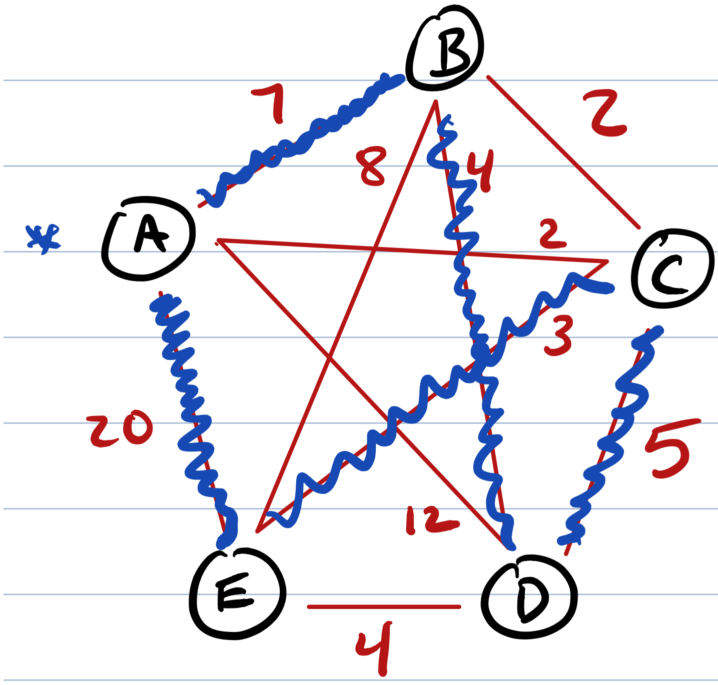
Wrong LB: 7+2+4+8+7+2 = 30.

This double counts the cost. When we exit B, we enter some other node.

One solution: only think about exiting.

7+2+2+4+3=18

(18.5)



Let T be a given tour (a solution).

If you add up the cost of going into and out of every city, you get <u>double</u> the total cost.

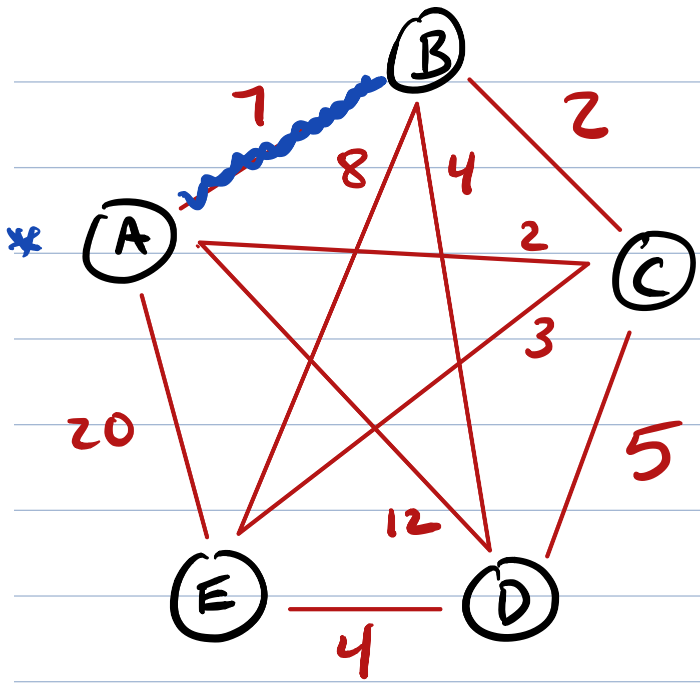
$$\text{cost}(T) = \frac{1}{2} \cdot \underbrace{\sum_{v \in V}}_{\text{sum over every vertex}} \left([\text{cost to enter}] + [\text{cost to exit}] \right)$$

Now, suppose we're in a subspace S, and we want a LB on the cost of any tour in S. Given any arbitrary $T \in S$:

$$\text{cost}(T) = \frac{1}{2} \cdot \left(\underbrace{[\text{enter } v_1] + [\text{exit } v_1]}_{\geq \text{ sum of two cheapest edges attached to } v_1} + \cdots + \underbrace{[\text{enter } v_n] + [\text{exit } v_n]}_{\geq \text{ sum of two ch. edges att. to } v_n} \right)$$

$\geq$ sum of two cheapest edges attached to v_1 $+ \cdots +$ $\geq$ sum of two ch. edges att. to v_n

$\geq \frac{1}{2} \cdot \Big($sum of: for each vertex, use any edges you already decided on, together with the cheapest remaining edges, to get two in total$\Big)$



LB:

$$\frac{1}{2}\cdot\Big(\overbrace{7+2}^{A} + \overbrace{7+2}^{B} + \overbrace{2+2}^{C} + \underbrace{4+4}_{D} + \underbrace{3+4}_{E}\Big)$$

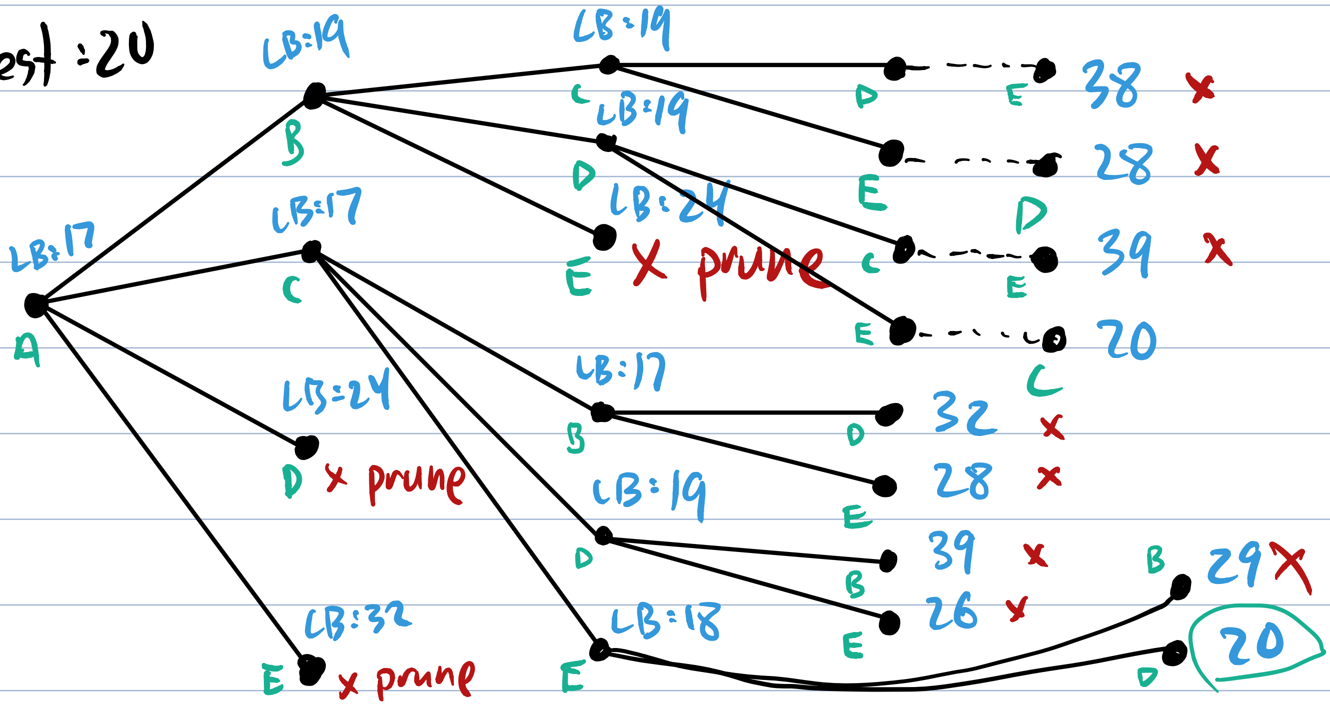
$$= \frac{1}{2}(37) = 18.5$$

$$\Rightarrow 19$$

1) Greedy solution:

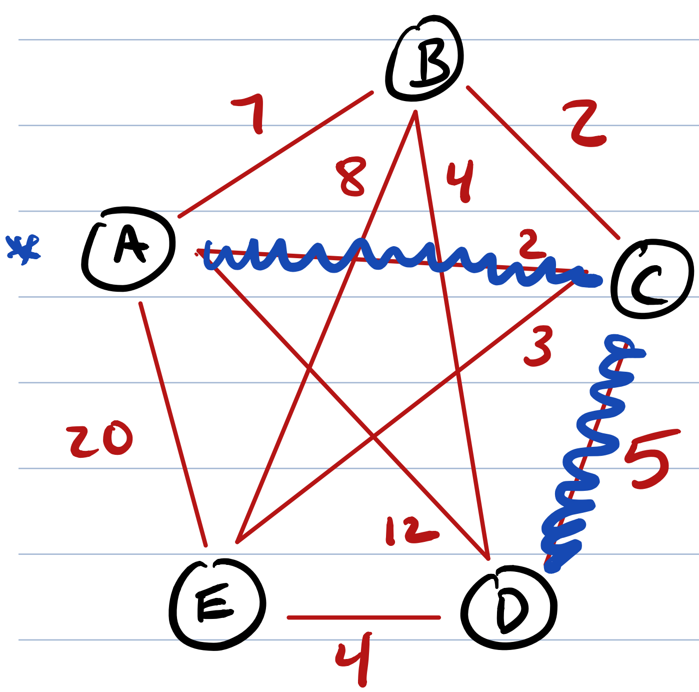
C → A → B → D → E → C : 20

Best: 20



* Would picking a diff. start node have helped? Probably!

Slight improvement:



A: 2,7

B: ~~2~~, 4, 7

C: 2, 5

D: 4, 5

E: ~~3~~, 4, 8

+5

+5

$\frac{+10}{2} = +5$

Topic #9: Dynamic Programming

<u>Idea</u>: Find the optimal solution by solving <u>subproblems</u>, building up bit-by-bit.

kind of like the partial solutions of branching

Weighted Interval Scheduling

Brute Force: $O(2^n)$

Greedy: fast, not optimal

Backtracking: vastly better, ~200, still $O(2^n)$

B+B: even better, still $O(2^n)$

Dynamic Progr.: $O(n)$ linear time

Suppose we have requests $R = \{r_1, \ldots, r_n\}$ with start s_i, finish f_i, value v_i. Assume these are sorted by finish time:

$$f_1 \leq f_2 \leq \ldots \leq f_n$$

Given any set of requests S, define $\mathcal{O}(S)$ to be the score of the optimal solution using the requests in S. (We want $\mathcal{O}(R)$.)

$R_k = \{r_1, r_2, \ldots, r_k\}$.

Question: If we know $O(R_1)$, $O(R_2)$, $O(R_3)$, $\ldots$, $O(R_{\ell-1})$, can we use this to compute $O(R_\ell)$?

If so:

- $\rightarrow$ $O(R_1)$ is easy
- $\rightarrow$ Use $O(R_1)$ to compute $O(R_2)$.
- $\rightarrow$ Use $O(R_1)$ and $O(R_2)$ $\cdots\cdots$ $O(R_3)$

  $\vdots$

- $\rightarrow$ Use $O(R_1), \ldots, O(R_{n-1})$ to compute $O(R_n)$

$\underset{\uparrow}{R_n}$ R.