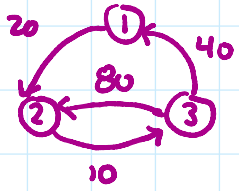
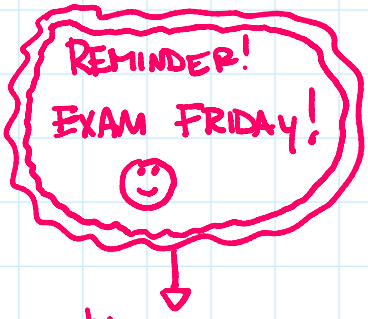
CMSC351-2023-11-13

Monday, November 13, 2023 8:13 AM

Reminder: We were applying Floyd to:





Counting sort
→ Floyd

The algorithm created (at the end):

$$d = \begin{bmatrix} 0 & 20 & 30 \\ 50 & 0 & 10 \\ 40 & 60 & 0 \end{bmatrix} \qquad p = \begin{bmatrix} 1 & 1 & 2 \\ 3 & 2 & 2 \\ 3 & 1 & 3 \end{bmatrix}$$

↳ distances! ↳ predecessors!

Path Reconstruction

Recall: $p[i,j]$ = pred. of j along a shortest path from i to j.

How do we get the path from this?

Consider the shortest path from 3 → 2. → start w/ path = [2]

- first observe $P[3,2] = 1$
  So the pred. of 2 is 1.
  So prepend 1. → now path = [1,2]
- next observe $P[3,1] = 3$
  So the pred. of 1 is 3
  So prepend 3. → now path = [3,1,2]
- done!

More generally, to construct the path from u to v

Start w/ path = [v]

- then suppose $p[u,v] = v_1$, so now path = $[v_1, v]$
- then suppose $p[u,v_1] = v_2$, so now path = $[v_2, v_1, v]$
- etc
- until $p[u,v_2] = u$, so now path = $[u, v_2, \ldots, v_2, v_1, v]$

DONE!

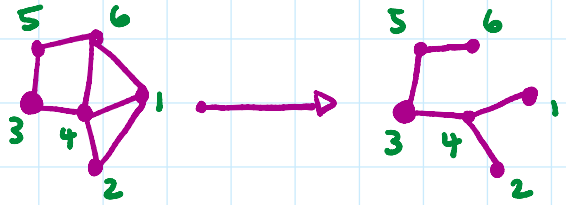
exam stops here!

exam stops here!

Minimal Spanning trees, Prim's Algorithm, Kruskal's Algorithm

① defn: Sps we have a graph G (simple, connected, undirected).
A spanning tree is a subgraph of G which contains all vertices, and is a tree

ex



← not the only one!

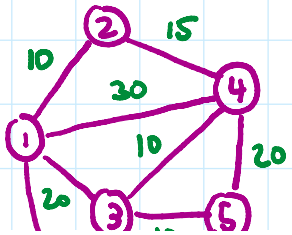
defn: If G is weighted, a minimal weight spanning tree is a spanning tree s.t. the total of all the edge weights is as small as possible.
note May or may not be unique!

Q: How might we find such a thing?

② Prim's Algorithm:

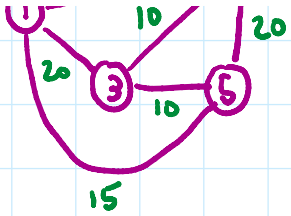
(A) Pick any vertex, and put it in our tree T.
ie. T = just that vertex to start!
(B) From amongst all edges (u,v) such that $u \in T$ and $v \notin T$, pick one of minimal weight and add it, and v, to T.
(C) Repeat (B) until we have all vertices.

ex Consider G:



Let's start w/ 5.
So to start, we have T:

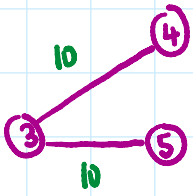
Then note that the edge (3,5) joins 5 to a vertex not in T, has min weight. So:
T:



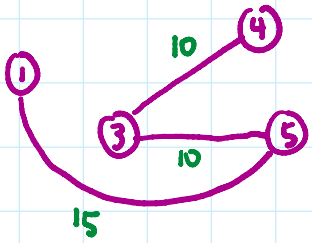
has min weight. So:

T:

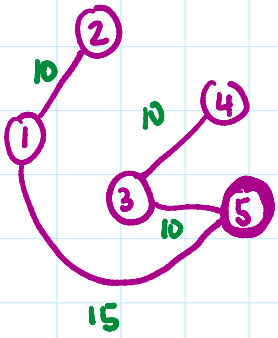
Next now (3,4) is best:



next: now (1,5) and (2,4) tie.
Let's pick (1,5)



Last: (1,2) is best!



Done!

thinking of 5 as the root but doesn't matter!

Note: There is (in text notes) a PROOF that actually constructs a MWST! not responsible for it!

Pseudocode

T = graph consisting of one vertex from G.
while vert in T ≠ vert in G: ← execute V times
    select (u,v), min weight edge w/ u∈T, v∉T ← HMM!
    T = T + (u,v) + v
end while

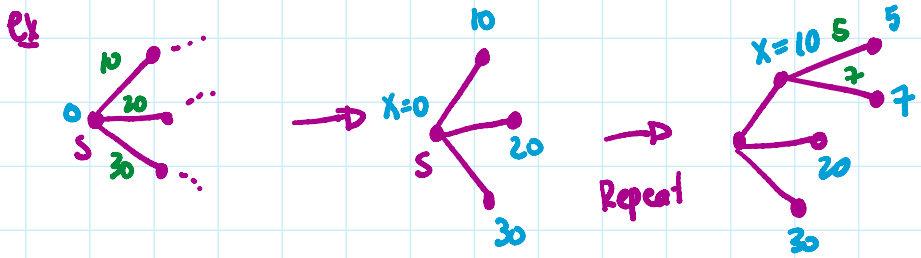
Time Complexity

Issue is how to perform the selection step!
Here are two options plus one comment!

(A) Sps G is stored as an adj. matrix.
Have a boolean list INT of length V which indicates if a vertex is in T or not. Then:
Scan the adj matrix. For each entry $[i,j]$ see if $i \in T, j \notin T$ (or reverse) and of all those, store the $[i,j]$ w/ min weight.
At the end, that's the edge which gets included.
Scanning is $\Theta(V^2)$, each check is $\Theta(1)$.
Since $\Theta(V^2)$ happens $\Theta(V)$ times, we get $\Theta(V^3)$.

(B) Sps G is still an adj matrix. Still have INT.
We'll also have a list DIST of length V which will contain distances from T to vertices which are adjacent to vertices in T.
Start w/ DIST all ∞ except DIST[s] = 0 where s = starting vertex (the one we put in to begin with).
To select, we choose the vertex $x \in T$ s.t. DIST[x] is minimal.
For all y adj to x, update DIST[y] w/ the edge weight $w(x,y)$ if that value is less that DIST[y].

Ex



TO BE CONTINUED AFTER BREAK!