TUTORIAL 8

SHORTEST PATHS: DIJKSTRA'S ALGORITHM

Problem: Given a weighted graph and two vertices v, w, find a path $v \xrightarrow{*} w$ of least cumulative weight (i.e. shortest).

Actually, it turns out to be easier to find shortest paths from v to all other vertices $\Rightarrow$ Dijkstra's algorithm.

Idea: Always visit the closest known vertex at each step, and continually revise distances as you traverse the graph and discover new edges. Otherwise behaves a bit like BFS.

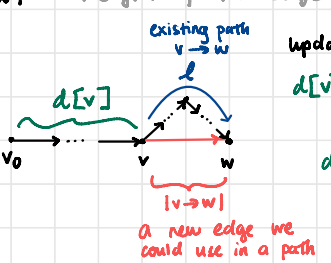
Initialisation

0. Suppose v_0 is our starting vertex.
1. Create an array of distances, say $d[0, \dots, n-1]$ (n elements), where $d[v]$ is the shortest currently known distance along some path from v_0 to v. Set $d[v_0] = 0$, and $d[v] = \infty$ if $v \neq v_0$.
2. Create a predecessor array, say $p[0, \dots, n-1]$, where $p[v]$ is the vertex that precedes v in the shortest path from v_0 to v. Set $p[v_0] = v_0$.
3. Create a new priority queue, say Q. Lower priority values are given higher priority, i.e. they leave Q first. (Priority = distance here)

4. Add $(v_0, d[v_0])$ to Q. (i.e. add vertex v_0 with priority $d[v_0] = 0$.)
5. While Q is not empty:
   a. Take the top vertex v from Q and remove it.
   b. For each vertex w connected to v:

Relaxation

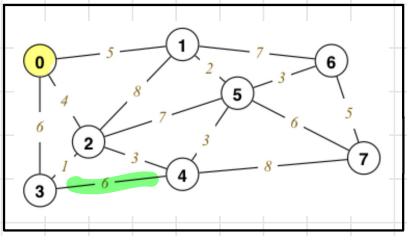
   i. Compute $d' = d[v] + |v \to w|$ = weight of the edge $v \to w$
   ii. If $d' < d[w]$:
   - set $d[w] = d'$
   - set $p[w] = v$
   - Add (w, d') to Q.



Update distance if

$d[v] + |v \to w| < d[v] + \ell$ (where $d[v] + \ell = d[w]$)

3. Find shortest paths from 0 to all other nodes:



Caveats: we will use the version in the lecture slides with a set instead of a priority queue, and find the closest vertex by inspection

Initial state:

Vertex	0	1	2	3	4	5	6	7
vSet	✓	✓	✓	✓	✓	✓	✓	✓
Dist	0	∞	∞	∞	∞	∞	∞	∞
Pred	0	?	?	?	?	?	?	?

Iteration 1:

Vertex	0	1	2	3	4	5	6	7
vSet	✗	✓	✓	✓	✓	✓	✓	✓
Dist	0	5	4	6	∞	∞	∞	∞
Pred	0	0	0	0	?	?	?	?

Iteration 2:

Vertex	0	1	2	3	4	5	6	7
vSet	✗	✓	✗	✓	✓	✓	✓	✓
Dist	0	5	4	5	7	11	∞	∞
Pred	0	0	0	2	2	2	?	?

Iteration 3:

Vertex	0	1	2	3	4	5	6	7
vSet	✗	✓	✗	✗	✓	✓	✓	✓
Dist	0	5	4	5	7	11	∞	∞
Pred	0	0	0	2	2	2	?	?

Iteration 4:

Iteration 5:

Iteration 6:

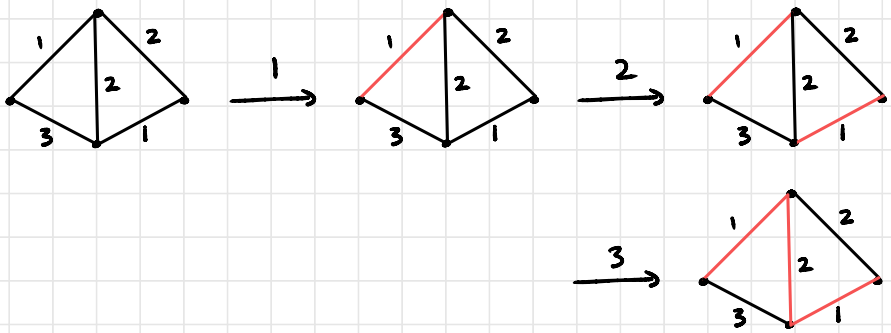
Iteration 7:

MINIMAL SPANNING TREES: KRUSKAL'S AND PRIM'S ALGORITHMS

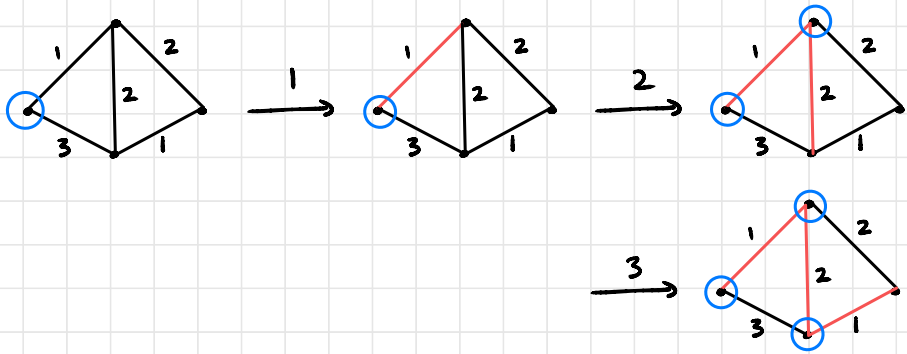
Problem: Given a weighted, connected graph, find a minimal spanning tree, i.e. a connected, acyclic subgraph with least cumulative edge weight and which includes each vertex.

Solution: We have two different algorithms for finding one:

① **Kruskal's algorithm:** Choose the edge of least weight which doesn't create a cycle in the subgraph, and repeat until $|V|-1$ edges are chosen.



② **Prim's algorithm:** Start with any vertex, then in each step choose the edge with least weight with one edge in the subgraph and one edge not, and repeat until $|V|-1$ edges are chosen.



i.e. The subgraphs created by considering the currently-chosen edges

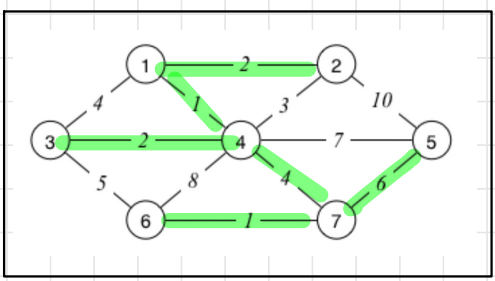
Remarks:

① In Kruskal's, the partial constructions may not be connected (see above). OTOH, the partial constructions in Prim's algorithm are always connected.

② In Prim's algorithm, there's no need to detect cycles, so the implementation is "simple" in a way.

③ Kruskal's is generally better for sparse graphs, Prim's for dense graphs (see lectures for the complexity analysis).

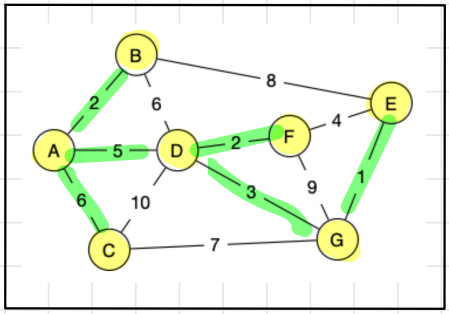
4.



Run Kruskal's on this graph:

Iteration	Choice	Edge candidates (which have shortest length?)
1	1→4	1→4 (1), 6→7 (1)
2	6→7	
3	3→4	1→2 (2), 3→4 (2)
4	1→2	
5	4→7	~~2→4 (3)~~, ~~1→3 (4)~~, 4→7 (4)
6	5→7	~~3→6 (5)~~, 5→7 (6)

5. Run Prim's on this graph, starting at A:



Iteration	Choice	Edge candidates
1	AB	AB(2), AD(5), AC(6)
2	AD	AD(5), AC(6), BD(6), BE(8)
3	DF	AC(6), BD(6), ~~BE(8)~~, DF(2), DG(3), DC(10)
4	DG	AC(6), BD(6), DG(3), DC(10), EF(4), FG(9)
5	EG	AC(6), BD(6), EG(1), DC(10), EF(4), CG(7)
6	AC	AC(6), ~~BD(6)~~, ~~EG(1)~~, DC(10), ~~EF(4)~~, CG(7)