TUTORIAL 8.

Recap: Dijkstra's Algorithm.

Suppose we want to find the shortest path in a weighted graph to every other vertex from some starting vertex. How can we do this?

The basic premise of Dijkstra's algorithm is to take the most optimal decision at the current moment (i.e. the presently-available shortest path) and continually revise the known shortest distances to every other vertex in the graph wrt some starting vertex. When every vertex has been visited, we will have all shortest paths.

Assume $G = (V, E)$, $v_0 \in V$ and $|V| = n$. Then Dijkstra's algorithm is:

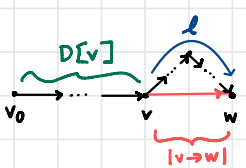
Initialisation

1. Create an array of distances, say $D[0, \ldots, n-1]$ (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 $P[0, \ldots, n-1]$, 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. Lowe 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

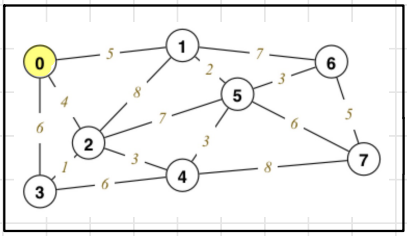
   i. Compute $d = D[v] + |v \to w|$ = weight of the edge $v \to w$
   ii. If $d < D[w]$:
   - set $D[w] = d$
   - set $P[w] = v$
   - Add (w, d) to Q.



update distance if

$D[v] + |v \to w| < D[v] + l$ ($= D[w]$)

4. To keep things easy we will just find the closest edge by inspection and keep track of vertices to consider with a set, vSet.



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	X	✓	✓	✓	✓	✓	✓	✓
Dist	0	5	4	6	∞	∞	∞	∞
Pred	0	0	0	0	?	?	?	?

Iteration 2:

Vertex	0	1	2	3	4	5	6	7
vSet	X	✓	X	✓	✓	✓	✓	✓
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	X	X	X	✓	✓	✓	✓	✓
Dist	0	5	4	5	7	7	12	∞
Pred	0	0	0	2	2	1	1	?

Iteration 4:

Vertex	0	1	2	3	4	5	6	7
vSet	X	X	X	X	✓	✓	✓	✓
Dist	0	5	4	5	7	7	12	∞
Pred	0	0	0	2	2	1	1	?

Iteration 5:

Vertex	0	1	2	3	4	5	6	7
vSet	X	X	X	X	X	✓	✓	✓
Dist	0	5	4	5	7	7	12	15
Pred	0	0	0	2	2	1	1	4

Iteration 6:

Vertex	0	1	2	3	4	5	6	7
vSet	X	X	X	X	X	X	✓	✓
Dist	0	5	4	5	7	7	10	13
Pred	0	0	0	2	2	1	5	5

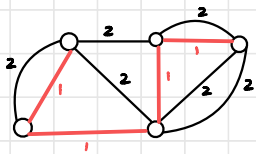
Iteration 7:

Vertex	0	1	2	3	4	5	6	7
vSet	X	X	X	X	X	X	X	✓
Dist	0	5	4	5	7	7	10	13
Pred	0	0	0	2	2	1	5	5

(Next iteration doesn't change)

Recap: Minimal spanning trees

Suppose we want to find a minimal spanning tree in a graph, i.e. a connected, acyclic subgraph of a directed graph with least edge weight sum that touches every vertex. How can we do this?



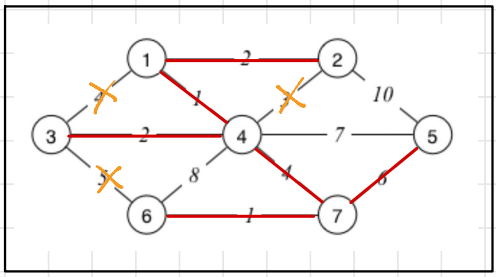
There are 2 main algorithms for finding one: Kruskal's and Prim's. For exact implementations, see the tutorial. The basic idea though is that

- Kruskal's algorithm chooses the least weight edge that doesn't create a cycle, and repeats until there are V-1 edges chosen.
  (We can prove that this is indeed correct: beyond the scope of 2521)
  Important feature: partial constructions not guaranteed to be connected!

- Prim's algorithm chooses the edge with least weight with one endpoint in the partial MST, and one not. We repeat until V-1 edges chosen.
  Important features: partial constructions are connected, and don't have to explicitly check if adding an edge creates a cycle.

Can prove by induction that any tree has V-1 edges. NOT assessable!

5.

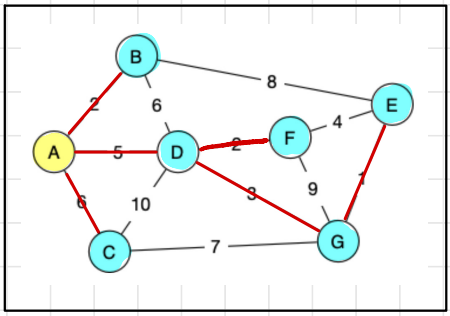


Kruskal's algorithm:

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

(If you have multiple feasible options, pick whatever you want.)

6.



Prim's algorithm, starting at A:

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