Shortest paths in graphs

SHORTEST PATH NETWORK.

- (weighted) directed graph $G = (V, E)$
- each edge has a length $\ell_e \geq 0$
- length of a (directed) path = sum of lengths of all edges on the path
- examples: commute time in transportation networks, latencies in communication networks

SHORTEST PATH PROBLEM.

- Input: shortest path network $G$, start node $s$, destination node $t$.
- Find the shortest directed path from $s$ to $t$. 
Dijkstra's algorithm

**Algorithm.** finds shortest paths from $s$ to every other node

$S = \{s\}$, $d[s] = 0$

while len($S$) != $n$:

  for all nodes $v$ one hop away from $S$:
    $d'[v] = \text{length of shortest path from } s \text{ to } v$
    passing through only nodes in $S$

  append $v$ with smallest $d'[v]$ to $S$, set $d[v] = d'[v]$

0. $S = \{s\}$, $d[s] = 0$
1. add $a$ to $S$, $d[a] = 5$, $p[a] = s$
2. add $f$ to $S$, $d[f] = 6$, $p[f] = s$
3. $d'[b] = 7$, $d'[e] = 14$, $d'[g] = 17$
   add $b$ to $S$, $d[b] = 7$, $p[b] = a$
Dijkstra’s algorithm

**Algorithm.** finds shortest paths from $s$ to every other node

$S = [s]$, $d[s] = 0$

while len($S$) != $n$:

  for all nodes $v$ one hop away from $S$:
    
    $d'[v] = \text{length of shortest path from } s \text{ to } v$
    
    passing through only nodes in $S$

  append $v$ with smallest $d'[v]$ to $S$, set $d[v] = d'[v]$

---

0. $S = [s]$, $d[s] = 0$

1. add $a$ to $S$, $d[a] = 5$, $p[a] = s$

2. add $f$ to $S$, $d[f] = 6$, $p[f] = s$

3. add $b$ to $S$, $d[b] = 7$, $p[b] = a$

4. $d'[e] = 13$, $d'[g] = 17$, $d'[c] = 10$

   add $c$ to $S$, $d[c] = 10$, $p[c] = b$
**Dijkstra’s algorithm**

**CLAIM.** for each $u \in S$, $d(u)$ is the length of the shortest $s$-$u$ path.

**PROOF.** by induction on $|S|

- **base case:** $|S| = 1$ is trivial; suppose true for $|S| = k \geq 1$
- **algorithm adds node $v$** with $P_v = P_u \circ (u, v)$
- **take any other $s$-$v$ path $P$ leaving $S$ with edge $(x, y)$

$$\ell(P) \geq \ell(P') + \ell_{(x,y)}$$ non-negative weights
$$\geq d(x) + \ell_{(x,y)}$$ inductive hypothesis, since $x \in S$
$$\geq d'(y)$$ definition of $d'(y)$
$$\geq d'(v)$$ algorithm picked $v$ over $y$
Dijkstra’s algorithm

**NAIVE IMPLEMENTATION.**
- $n - 1$ iterations of while loop
- each $v \notin S$, computing $d'(v)$: $O(\text{indeg}(v))$ time
- each iteration of while loop: $O(m)$ time
- total running time: $O(mn)$

**BETTER IMPLEMENTATION.**
- key observation: $d'(v)$ only changes when a neighbor gets added to $S$
- computing $d'(v)$ $O(n)$ times $\rightarrow$ update $d'(v)$ $O(\text{indeg}(v))$ times
- store $d'(v)$ using a priority queue with \texttt{CHANGEKEY} operation
- running time: $O(m + n)$ plus $n$ \texttt{EXTRACTMIN} and $m$ \texttt{CHANGEKEY} operations
- total running time: $O(m \log n)$ (using a min-heap)
Minimum spanning tree

**PROBLEM.**
- **input:** a set of locations $V = \{v_1, \ldots, v_n\}$, with costs for building a (undirected) link between some pairs of locations
- **goal:** build the cheapest communication network with constraint any two locations are connected (assume graph is connected)
- **observation:** in an optimal solution, the edges form a tree
- **this is called the minimum spanning tree (MST) problem**
**ALGORITHM.** incrementally add least cost edges that do not make a cycle

Sort edges by cost

\[ T = [] \]

for e in edges:

    if adding e to T does not create a cycle:

        T.append(e)

return T
**ALGORITHM.** build a single tree incrementally by adding least cost edges

\[
S, T = [s], [] \quad \# \text{pick any start vertex } s
\]

\[
\text{while len}(S) \neq n:
\]

\[
\text{let } e = (u,v) \text{ be the min cost edge amongst those}
\]

\[
\text{connecting a vertex in } S \text{ to one outside } S
\]

\[
T.\text{append}(e)
\]

\[
S.\text{append}(v)
\]

\[
\text{return } T
\]
THE END