Dijkstra's Algorithm

More generally: greedy algorithms

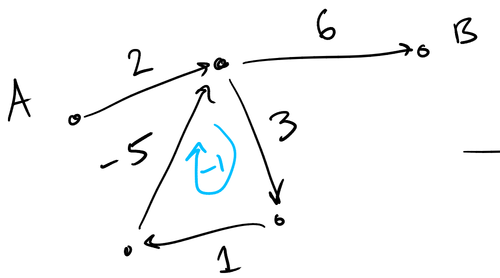
repeated locally best choices → globally best result.

Def'n A *weighted* (directed or undirected) graph is a graph where each edge is assigned a weight. We will denote the weight of edge (u,v) by w_{uv}.

The weight of a path is the sum of all the edge weights.

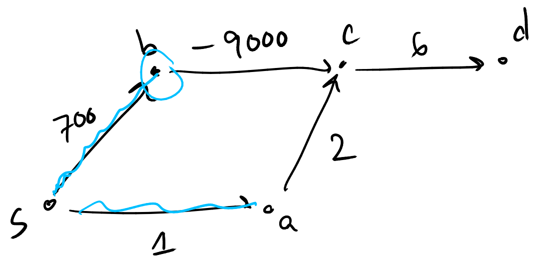
For now we will stick to weights in $\boxed{\mathbb{R}_{\geq 0}}$, i.e. nonnegative real numbers.
Later we will consider $\mathbb{R}$.

Aside: negative weights.



If we have negative cycles, "shortest" path might not even make sense.

Even w/o neg. cycles, Dijkstra does not work w/ negative edges.



Recall, BFS kept track of:

- visited vertices — stay same
- parents — stay same.
- layer of each vertex — we will store (shortest) distance to each vertex.
- queue of vertices to visit

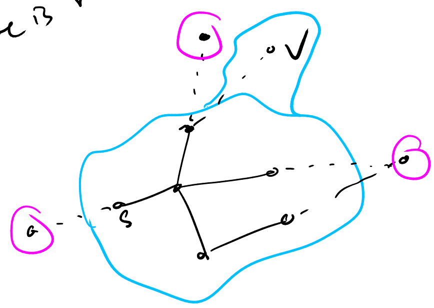
Dijkstra

Basic Dijkstra(G, s):

Mark all UNVISITED
Mark s VISITED
parent ← empty dict
d ← empty dict
d[s] ← 0.

Θ(1) or Θ(V)

Assume $W_{uv} = \infty$ if there is no edge $u \to v$.



Θ(V) while not all vertices are VISITED:

Θ(E) → [Pick visited u, UNVISITED v such that $\underline{d[u] + w_{uv}}$ is as small as possible]

↑ ∞ if no edge $u \to v$.

Mark v VISITED
parent[v] ← u
d[v] ← $\underline{d[u] + W_{uv}}$

Θ(1)

return parent, d

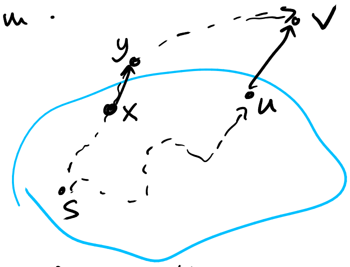
Whole thing is Θ(VE).
i.e. V^2 to V^3.

Thm Dijkstra's Alg. correctly solves the single-source shortest path problem (SSSP), i.e. after running Dijkstra(G, s), d[v] will store the shortest distance $s \to v$ for all vertices.

Proof. As a loop invariant, we will show d[v] is correct for all VISITED v.

(base case 0 loops) → • Before the loop starts, only s is VISITED, and d[s] = 0, which is indeed the shortest distance $s \to s$.

(true for k loops → true for k+1 loops) • Now suppose the loop invariant is true at some point; we will show it still holds after one more loop. Suppose u, v are the vertices picked by the algorithm.



We want to show that $d[u] + w_{uv}$ is in fact the shortest distance $s \to v$. Consider any other path $s \to v$. It must cross from visited → unvisited at some point, call them $x \to y$.
By assumption, d[x] is the shortest possible distance to x.
Also, $d[x] + w_{xy} \geq d[u] + w_{uv}$, because of how u, v were chosen. The extra part of the path $y \to v$ can only make it longer, since there are no negative edges. Hence $d[u] + w_{uv}$ is in fact the shortest possible distance to v.

- We're going to generalize $d[v]$ to store, for <u>each</u> vertex, the shortest <u>currently known</u> distance $s \to v$.
- Likewise, $parent[v]$ will store the parent of v along the shortest <u>currently known</u> path $s \to v$.
- We will store unvisited vertices in a <u>priority queue</u> with priority given by $d[v]$.

DIJKSTRA(G, s):

$\Theta(V)$
- $d[s] \leftarrow 0$, $d[v] \leftarrow \infty$ for all other vertices.
- parent $\leftarrow$ empty map
- $Q \leftarrow$ priority queue containing all vertices, keyed by $d[v]$.

while Q is not empty:

    $u \leftarrow$ Q.removeMin — $V \cdot T_{rem}(V)$ — i.e. $V \times$ time to remove from PQ of size V.

    // thing the "water" will reach next — we know $d[u]$ must be correct.

    for each outgoing edge (u,v):

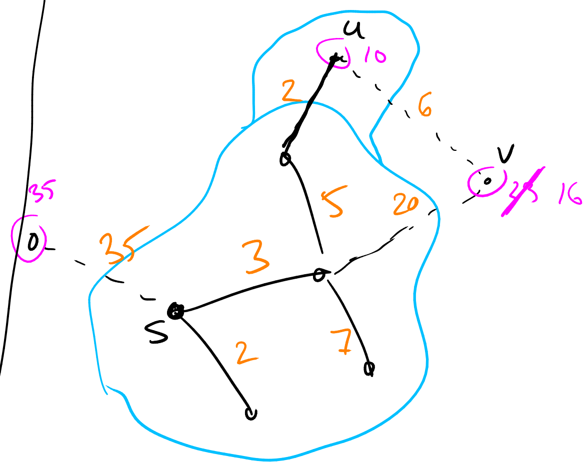
        if $d[u] + w_{uv} < d[v]$:

            $d[v] \leftarrow d[u] + w_{uv}$

            Q.updateKey(v, d[v]) — $E \cdot T_{upd}(V)$

            $parent[v] \leftarrow u$

return d, parent.



$E \cdot T_{upd}(V)$

Overall: $O(V \cdot T_{rem}(V) + E \cdot T_{upd}(V) + V + E)$

$= O(\underline{V\, T_{rem}(V)} + \underline{E\, T_{upd}(V)})$

PQ	T_{rem}	T_{upd}	Dijkstra
Dict	$O(V)$	$O(1)$	$O(V^2 + E) = O(V^2)$
Sorted list	$O(1)$	$O(V)$	$O(V + EV) = O(VE)$
Binary heap	$O(\lg V)$	$O(\lg V)$	$O(V \lg V + E \lg V)$ $= O(E \lg V)$
Fibonacci Heap	$O(\lg V)$	$O(1)$	$= O(V \lg V)$