Dijkstra's Algorithm

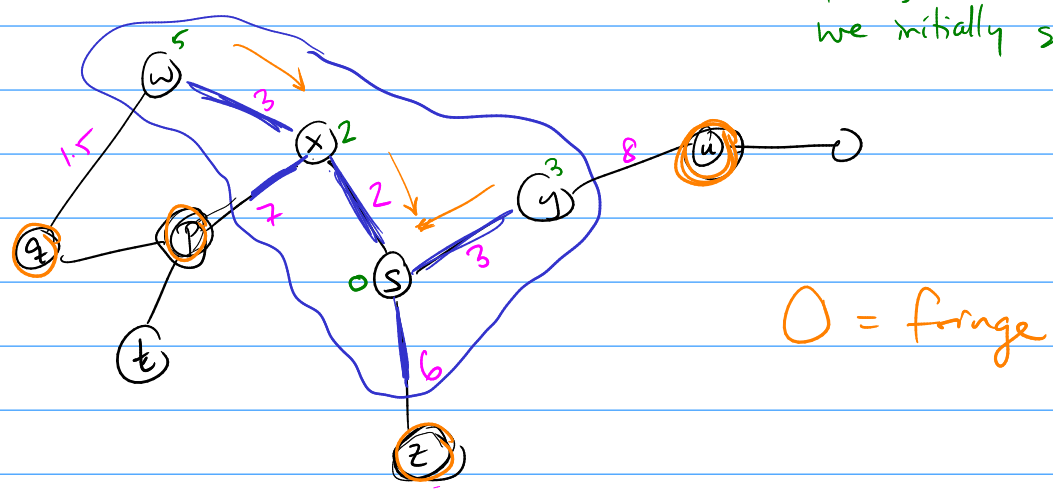
- Weighted, directed graphs.
- We will use weights in $\mathbb{R}^+$ (non-negative real #'s).
- The weight of edge $u \to v$ will be written w_{uv}.

Problem 1: Given vertices s, t, find a shortest (weighted, directed) path from s to t.

Problem 2: (Single-source shortest paths, SSSP): Given a start vertex s, find shortest paths from s to every other vertex.

Dijkstra's algorithm generalizes BFS.

What did BFS keep track of?	(How will it generalize to Dijkstra?)
– Set of visited vertices	– Same
– Parent map	– Same
– Layer map	– keep track of a distance map instead.
– Queue of vertices to process	– Needs to change – won't process vertices in the same order as we initially see them.



BasicDijkstra(G, s):

Mark s visited
$d[s] \leftarrow 0$ } $\Theta(1)$. — shortest distance to s is 0.

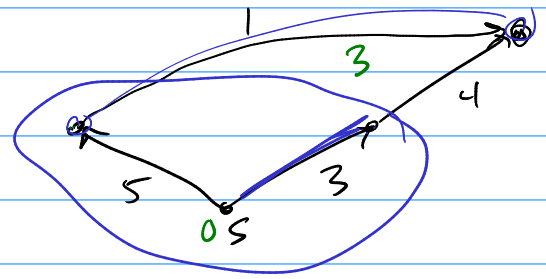
while not all vertices are visited: ← $\Theta(V)$.

* Pick a visited u, unvisited v such that $d[u] + w_{uv}$ is as small as possible. — $O(E)$ — v is when water would reach next.

  parent[v] ← u
  $d[v] \leftarrow d[u] + w_{uv}$
  Mark v visited. } $\Theta(1)$.

return d, parent.

How do we do * ? Brute force — loop over all u in visited, loop over all unvisited neighbors of each, compute $d[u]+w_{uv}$ for each, pick smallest. Worst case — we look @ every edge. Total: $O(VE)$. $|E|$ is $O(V^2)$, so $O(V^3)$.



- d[v] is now going to keep track of the shortest known distance to v so far.
- Likewise, parent[v] will be the parent along the current shortest known distance.
- We need a priority queue to keep track of vertices we could potentially reach next; each vertex v will be stored with priority equal to d[v].

  Need 3 operations: (1) add vertex, (2) remove smallest, (3) decrease priority.

DIJKSTRA(G, s):

$\Theta(V)$ [$d[s] \leftarrow 0$
$d[v] \leftarrow \infty$ for all other vertices.

$\Theta(1)$ - Create priority queue Q containing s.

while Q is not empty:

$V \cdot T_{rem}(V)$ — $u \leftarrow$ Q.removeMin

for each outgoing edge $u \to v$:

total, once per edge.

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

$O(E)$ — $d[v] \leftarrow d[u] + w_{uv}$

if v in Q:

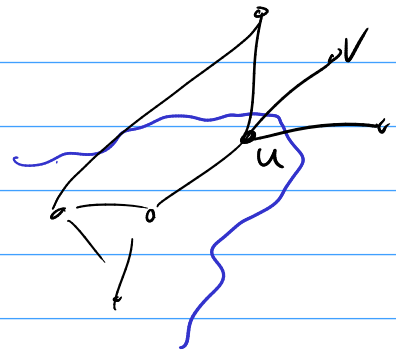
$E \cdot T_{upd}(v)$ — Q.decreasePrio(v, d[v])

else:

$E \cdot T_{add}(v)$ — Add v to Q w/ priority d[v].

$O(E)$ — parent[v] $\leftarrow$ u.

return d, parent.



$T_{rem}(n)$ = time to remove min from prio. queue of size n

$T_{upd}(n)$ = time to update priority

Total time is $\Theta(V) + \Theta(1) + O(V \cdot T_{rem}(V)) + O(E) + O(E \cdot T_{upd}(V))$

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

Prio. Q. implementations :	DirtArray	Sorted array	Binary heap	Fibonacci heap
T_{rem} :	$O(n)$	$O(1)$	$\lg(n)$	$\lg(n)$
T_{upd} :	$O(1)$	$O(n)$	$\lg(n)$	$O(1)$