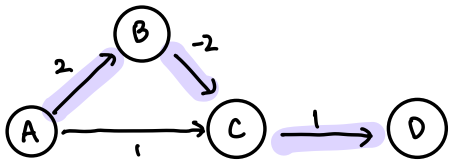
* Dijkstra's key assumption: adding more edges can only increase the length of the path
  - The algorithm doesn't backtrack
  - Once a node is visited, its shortest path is solidified

1 Dijkstra's Algorithm Fails on Negative Edges

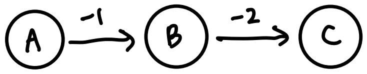
Draw a graph with five vertices or fewer, and indicate the source where Dijkstra's algorithm will be started from.

(a) Draw a graph with no negative cycles for which Dijkstra's algorithm produces the wrong answer.



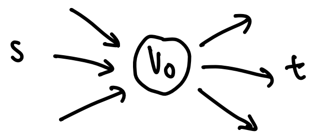
Dijkstra's gives:
A—C—D

(b) Draw a graph with at least two negative weight edge for which Dijkstra's algorithm produces the correct answer.



2 Waypoint

You are given a strongly connected directed graph $G = (V, E)$ with positive edge weights, and there is a special node $v_0 \in V$. Give an efficient algorithm that computes the length of the shortest path from s to t that passes through v_0 for all pairs s, t. Your algorithm should take $O(|V|^2 + |E| \log |V|)$ time.



1. Compute shortest path from V_0 to all other vertices. $(v_0 \to t)$

   $O((V+E)\log V)$

2. Reverse the edges of the graph to get G^R.

   $O(E)$

3. Compute shortest path from V_0 to all other vertices on G^R $(v_0 \to s)$

   $O((V+E)\log V)$

4. For all pairs of vertices s and t, shortest path is $V_0 \to s$ reversed combined with $t \to V_0$.

   $O(V^2)$

3 Dijkstra Tiebreaking

We are given a directed graph G with positive weights on its edges. We wish to find a shortest path from s to t, and, among all shortest paths, we want the one in which the longest edge is as short as possible. How would you modify Dijkstra's algorithm to this end? Just a description of your modification is needed.

(If there are multiple shortest paths where the longest edge is as short as possible, outputting any of them is fine).

Generally for these algorithm questions, there are two approaches:

1. Modify the graph and run an existing algorithm
2. Modify an algorithm

Here we choose to modify Dijkstra's:

* maintain a map that holds $\ell[v]$, the length of the longest edge in the path to vertex v

when exploring the edge (u,v)...

```
if $dist(u) + w(u,v) < dist(v)$:
    $\ell[v] = \max(\ell[u], w(u,v))$
    $edgeTo[v] = u$
if $dist(u) + w(u,v) = dist(v)$ and $\ell[v] > \max(\ell[u], w(u,v))$:
    $\ell[v] = \max(\ell[u], w(u,v))$
    $edgeTo[v] = u$
```

longest edge is either (u,v) or the longest edge in the path to u

update path if a shorter longest edge is found

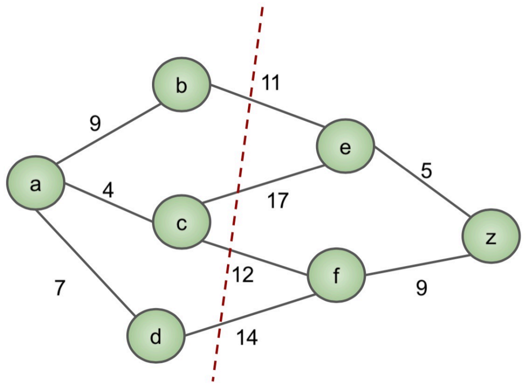
Cut Property

A **cut** of a graph is a partition of the vertices into two disjoint sets. The **edges crossing the cut** are set of edges that separates the vertices of the graph in two.

Cut Property: The lightest edge across a cut is in *some* MST

Reasoning:

- MST must span all vertices, so the two parts of the graph *must* be connected somehow
- Minimize total cost of tree by choosing lightest edge as connection



(b, e) of weight 11 is in the MST

4 Updating a MST

You are given a graph $G = (V, E)$ with positive edge weights, and a minimum spanning tree $T = (V, E')$ with respect to these weights; you may assume G and T are given as adjacency lists. Now suppose the weight of a particular edge $e \in E$ is modified from $w(e)$ to a new value $\hat{w}(e)$. You wish to quickly update the minimum spanning tree T to reflect this change, without recomputing the entire tree from scratch.

There are four cases. In each, give a description of an algorithm for updating T, a proof of correctness, and a runtime analysis for the algorithm. Note that for some of the cases these may be quite brief. For simplicity, you may assume that no two edges have the same weight using both w and $\hat{w}$.

(a) $e \notin E'$ and $\hat{w}(e) > w(e)$

(b) $e \notin E'$ and $\hat{w}(e) < w(e)$

(c) $e \in E'$ and $\hat{w}(e) < w(e)$

(d) $e \in E'$ and $\hat{w}(e) > w(e)$

a) e not in MST and weight of e increased

No change to MST $O(1)$

b) Add e to MST, creating a cycle. Then remove the heaviest edge in the cycle.

$O(V)$

c) Do nothing. $O(1)$

e already in MST, already lightest edge across the cut

d) Delete e from MST. Find lightest edge across the cut to connect the two components.

$O(V+E)$ worst case must scan entire graph

5 MST practice

Let $G = (V, E)$ be an undirected, connected graph.

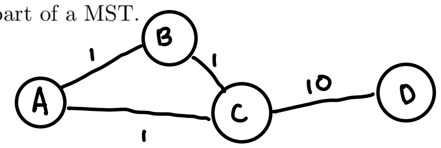
(a) Prove that there is a unique MST if all edge weights are distinct.

Cut property: lightest edge across the cut is in some MST

Here there is one unique lightest edge

(b) True or False? If G has more than $|V| - 1$ edges, and there is a unique heaviest edge, then this edge cannot be part of a MST.

False



(c) True or False? If the lightest edge in G is unique, then it must be a part of every MST.

True

If e is not in MST, adding it would create a cycle. Then removing the heaviest edge in the cycle would produce a more minimum MST.