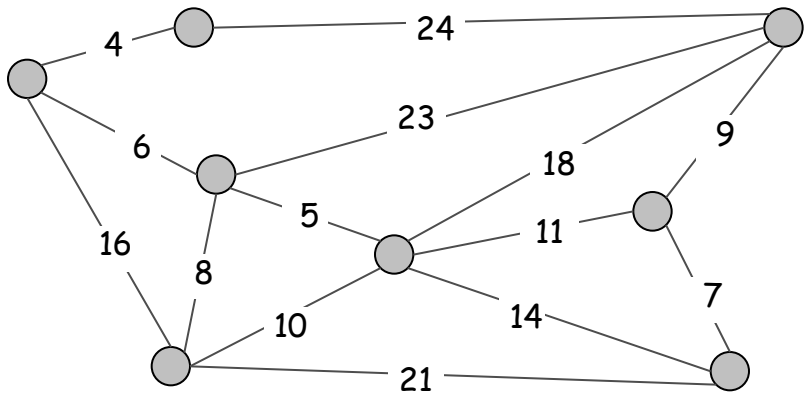
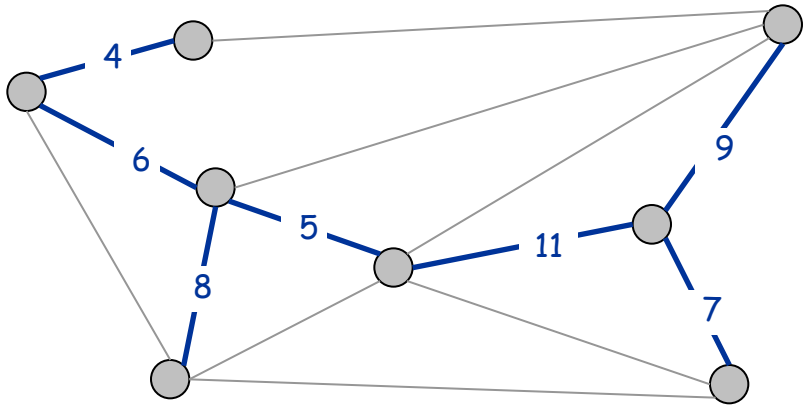
Minimum Spanning Trees
Shortest Paths

Minimum Spanning Tree

Given a set of locations, with positive distances to each other, we want to create a network that connects all nodes to each other with minimal sum of distances.



$G = (V, E)$



$\Sigma_{e \in T} c_e = 50$

Then that graph is a tree.

WHY?

Applications

MST is fundamental problem with diverse applications.

- Network design.
  - telephone, electrical, hydraulic, TV cable, computer, road
- Approximation algorithms for NP-complete problems.
  - traveling salesperson problem
- Cluster analysis.

Minimal vs minimum.

Minimum (noun) is always the smallest possible or allowable amount.

Minimal (adjective) implies that the amount is (relatively) small. But English allows the use of nouns as adjectives, and also nouns as verbs. (I Googled it ☺).

"There was minimal residue in the wine bottle." (compared with other wines, this one didn't throw much sediment).

"You must leave a minimum sum of $50 in the account." (whether you think $50 is a small amount of money or not).

Greedy Algorithms

Kruskal's algorithm. Start with $T = \phi$. Consider edges in ascending order of cost. Add edge e to T unless doing so would create a cycle.

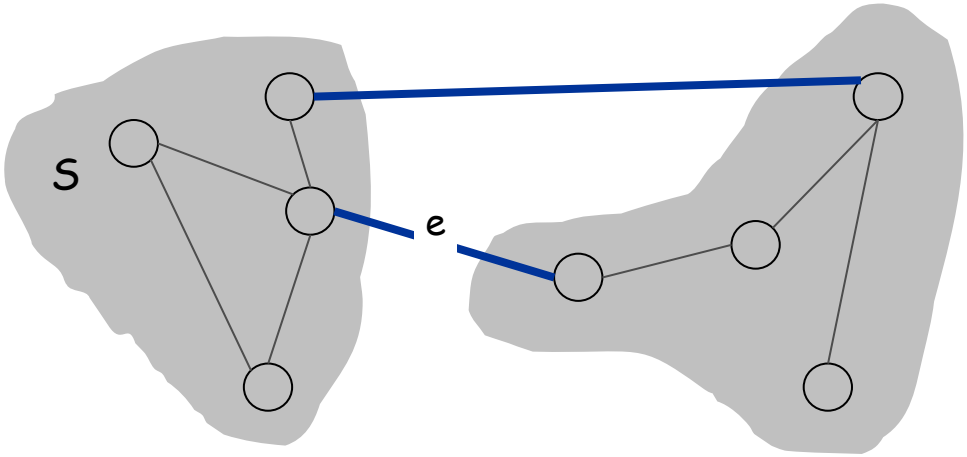
Reverse-Delete algorithm. Start with T = E. Consider edges in descending order of cost. Delete edge e from T unless doing so would disconnect T.

Prim's algorithm. Start with some node s and greedily grow a tree T from s. At each step, add the cheapest edge e to T that has exactly one endpoint in T, ie without creating a cycle.

The cut property

Simplifying assumption. All edge costs are distinct.
Cut property. Let S be a subset of nodes, S neither empty nor equal V, and let e be the minimum cost edge with exactly one endpoint in S.
Then the MST contains e.

The cut property establishes the correctness of Prim's algorithm.



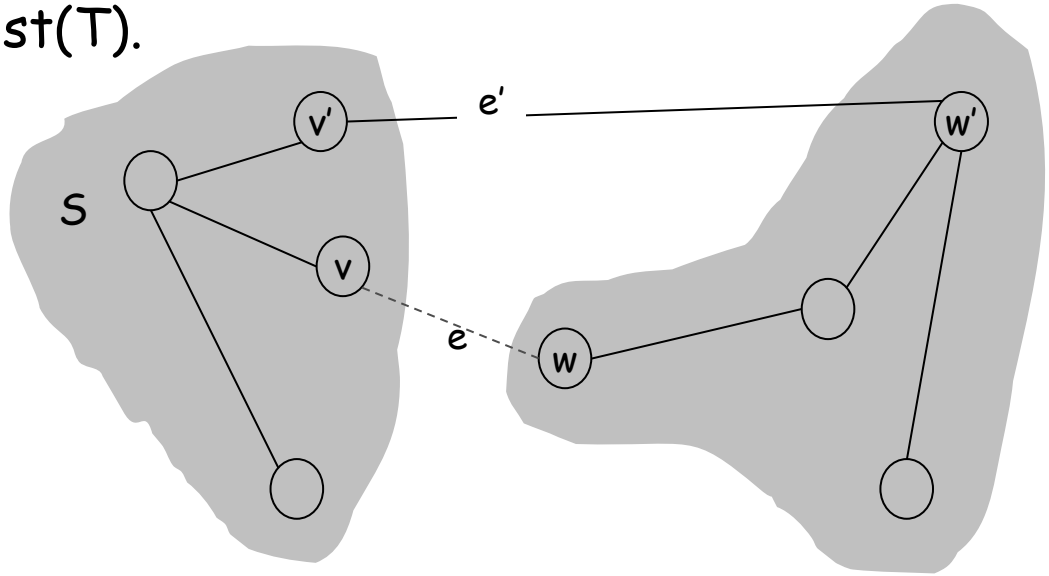
e is in the MST

The cut property

Cut property. Let S be a subset of nodes, and let e be the min cost edge with exactly one endpoint in S. Then the MST T contains e.

Proof. (exchange argument)

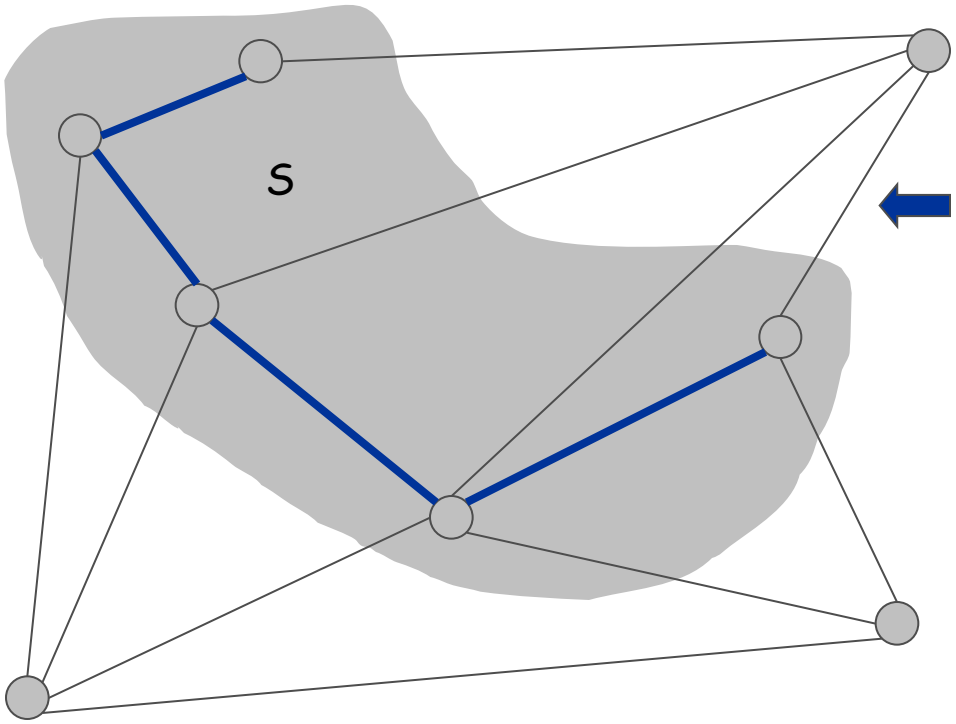
- If e =(v,w) is the only edge connecting S and V-S it must be in T, else e is on a cycle in the graph (not the MST). Now suppose e does not belong to T.
- Let e'= (v',w') be the first edge between S and V-S on the path from v'. T' = T $\cup$ { e } - { e' } is also a spanning tree.
- Since $c_e < c_{e'}$, cost(T') < cost(T).
- This is a contradiction. ▪



Prim's Algorithm

Prim's algorithm. [Jarník 1930, Prim 1957, Dijkstra 1959]

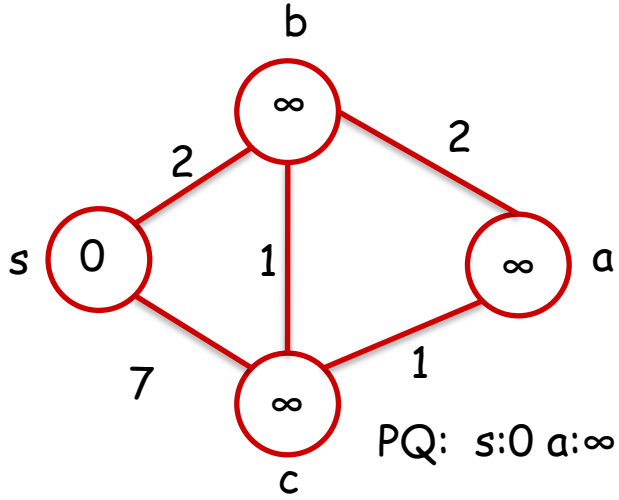
- Initialize S = any node.
- Apply cut property to S: add min cost edge (v, w) where v is in S and w is in V-S, and add w to S.
- Repeat until S = V.



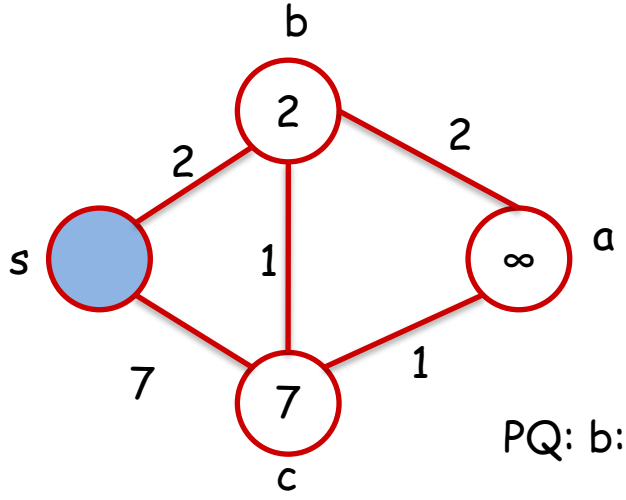
Prim's algorithm: Implementation

- Maintain set of explored nodes S.
- For each **unexplored** node v, maintain attachment cost a[v] = cost of cheapest edge v to a node in S.

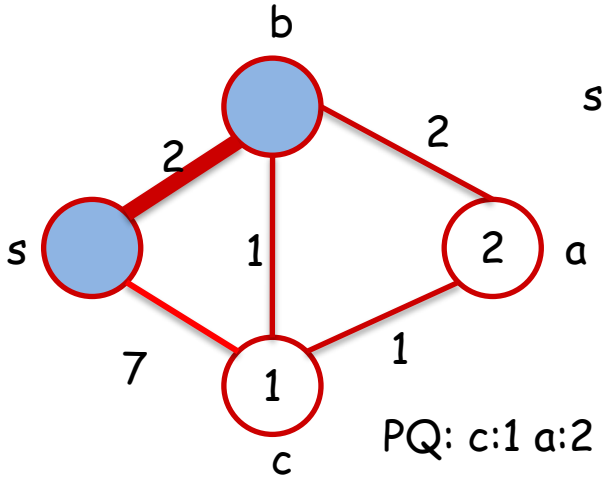
```
Prim(G,s)
    foreach (v ∈ V)
        a[v] ← ∞
    a[s]= 0
    priority queue Q = {}
    foreach (v ∈ V) insert v onto Q (key: a value)
    set S ← {}
    while (Q is not empty) {
       u ← delete min element from Q
       S ← S ∪ { u }
       foreach (edge e = (u, v) incident to u)
           if ((v ∉ S) and (c_e < a[v]))
              decrease priority a[v] to c_e
```



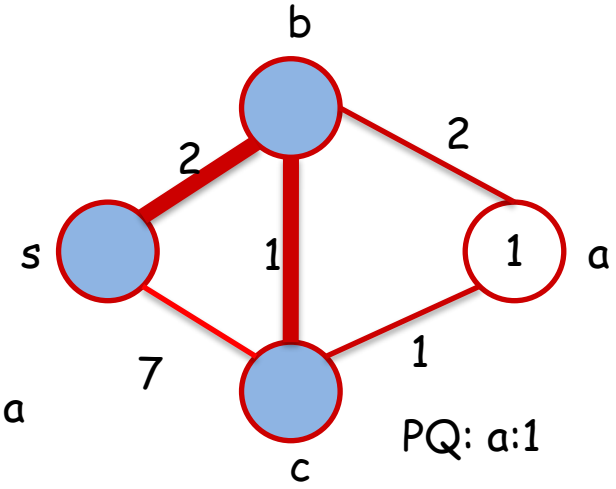
PQ: s:0 a:∞ b:∞ c:∞



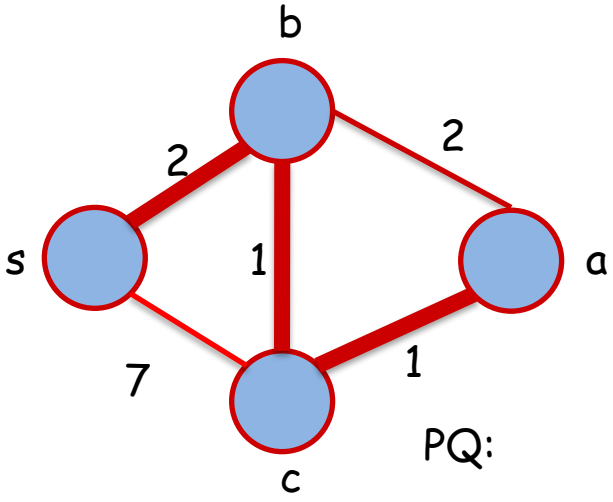
PQ: b:2 c:7 a:∞



PQ: c:1 a:2



PQ: a:1



PQ:

Kruskal produces an MST

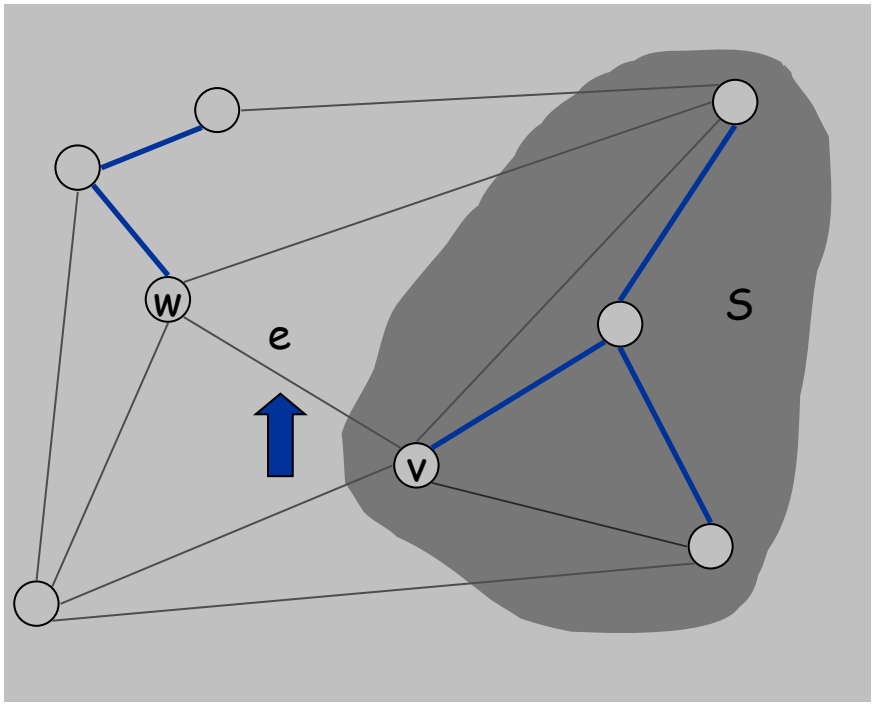
Kruskal's algorithm. [Kruskal, 1956]

- Consider edges in ascending order of weight. Add edge unless doing so would create a cycle.

- Kruskal keeps adding edges until all nodes are connected, and does not create cycles, so produces a spanning tree.

Kruskal produces an MST

- Consider e=(v, w) added by Kruskal. S is the set of nodes connected to v just before e is added; v is in S and w is not (otherwise we created a cycle). Therefore e is the cheapest edge connecting S to a node in V-S, and hence, e is in any MST (cut property).



Reverse-Delete algorithm

Start with T = E. Consider edges in descending order of cost. Delete edge e from T unless doing so would disconnect T.

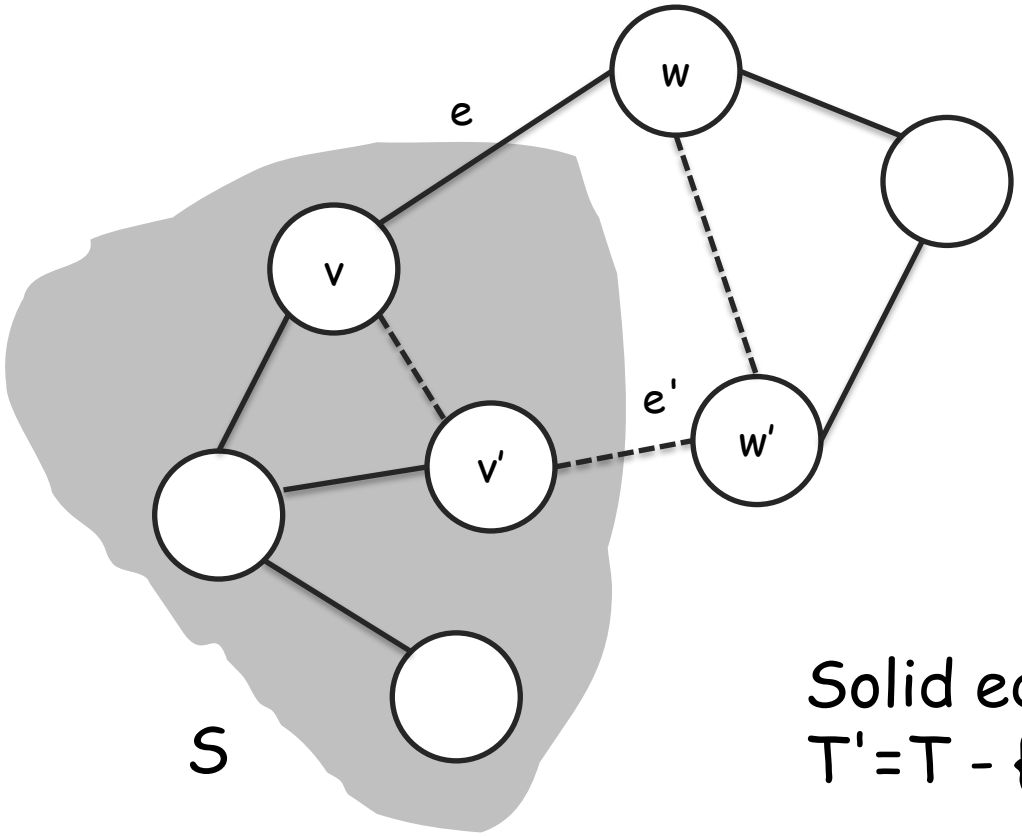
The result is a spanning tree.

Is it optimal?

When is it safe to remove an edge?

Safely removing edges

Cycle property. Let C be any cycle in G, and let e be the max cost edge belonging to C. Then e doesn't belong to any MST of G.

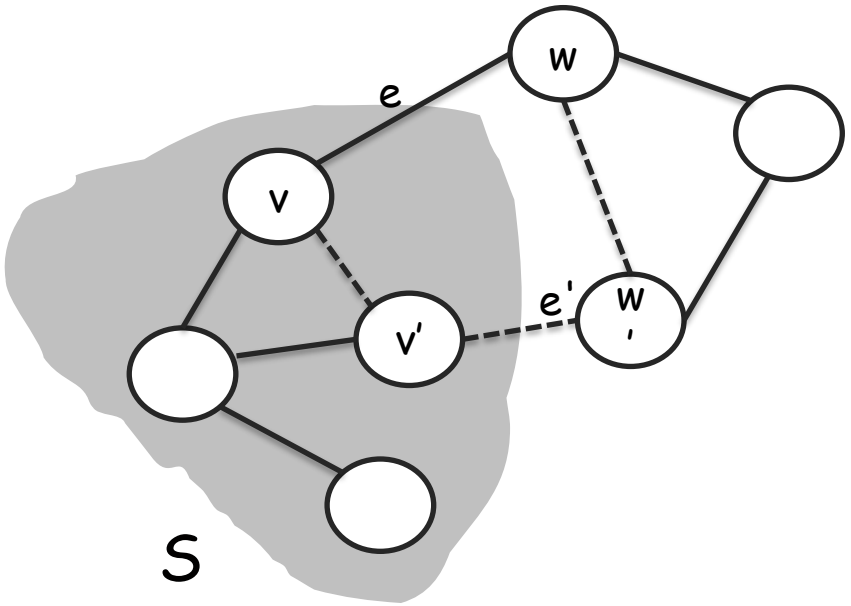


Solid edges: T
T' = T - {e} + {e'}

Safely removing edges

Cycle property. Let C be any cycle in G, and let e be the max cost edge belonging to C. Then e doesn't belong to any MST of G.

Let T be a spanning tree that contains e=(v,w). Remove e; this will disconnect T, creating S containing v, and V-S containing w. C–{e} is a path P. Following P from v will at some stage cross S into V-S by edge e'.
T - {e} + {e'} is again a spanning tree and its cost is lower than T, so T is not an MST.



Shortest Paths Problems

Given a **weighted directed** graph G=(V,E)

find the shortest path

- path length is the sum of its edge weights.

The shortest path from u to v is ∞ if there is no path from u to v.

Variations:

1) **SSSP** (Single source SP): find the SP from some node s to all nodes in the graph.

2) **SPSP** (single pair SP): find the SP from some u to some v.

We can use 1) to solve 2), also there is no asymptotically faster algorithm for 2) than that for 1).

3) **SDSP** (single destination SP) can use 1) by reversing its edges.

4) **APSP** (all pair SPs) could be solved by |V| applications of 1), but can be solved faster (cs420).

Dijkstra SSSP

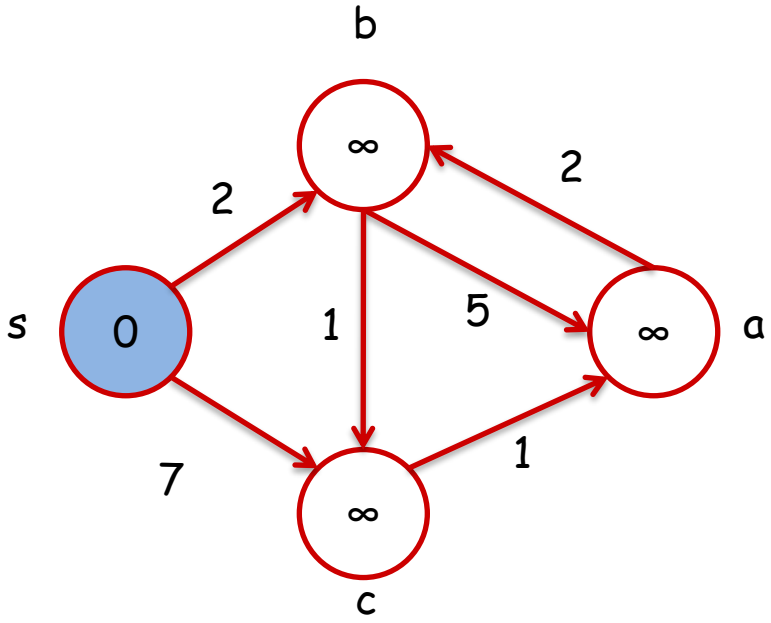
Dijkstra's (Greedy) SSSP algorithm only works for graphs with only positive edge weights.

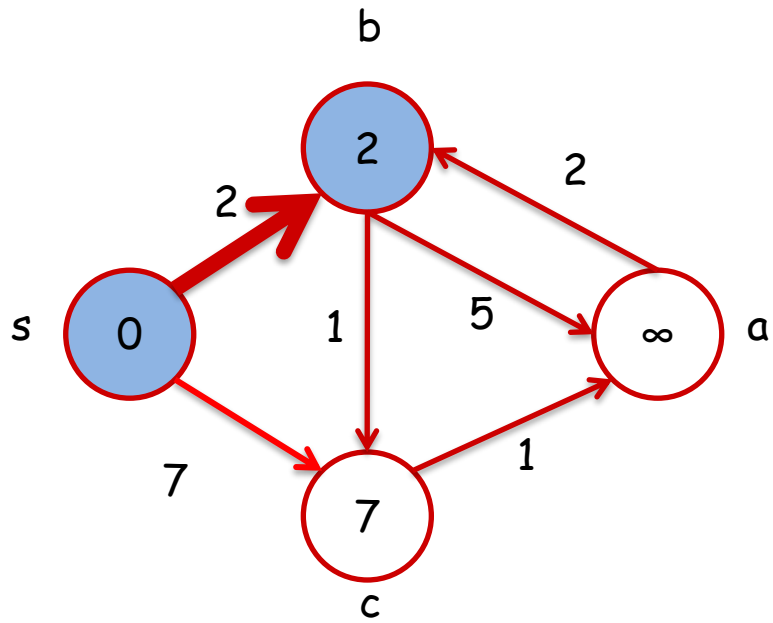
S is the set of explored nodes
For each u in S, d[u] is a distance
Init: S = {s} the source, and d[s]=0
while S≠V:
  select a node v in V-S with at least one edge
  from S, for which $d'[v]=\min_{e=(u,v),u \text{ in } S} d[u]+w_e$
  add v to S (S=S+v)
  d[v]=d'[v]

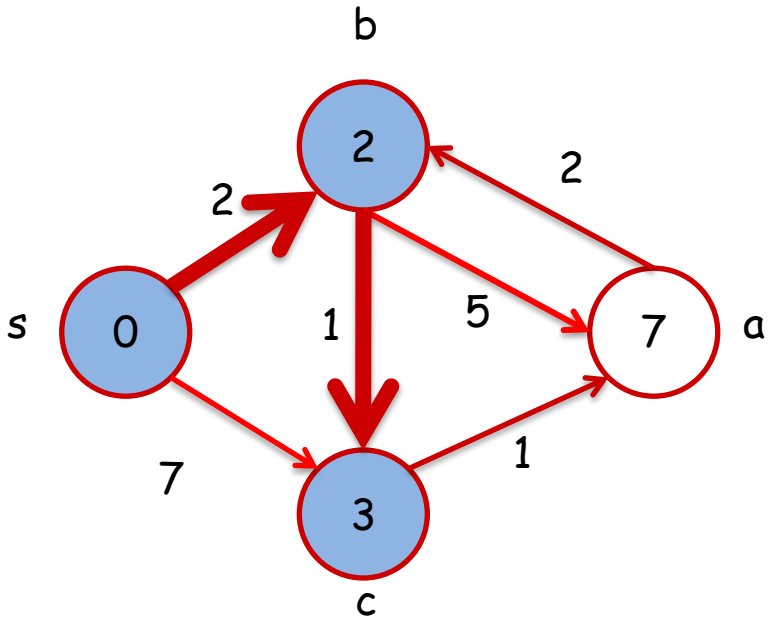
the minimum **path** extending out of S

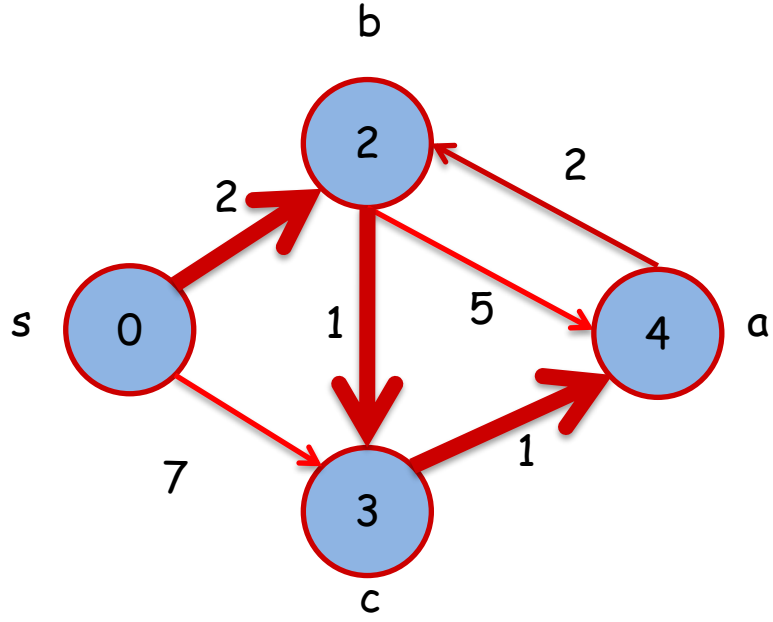
To compute the actual minimum paths, maintain an array p[v] of predecessors.

Notice: Dijkstra is very similar to Primm's MST algorithm


b
∞
2
2
s
0
1
5
∞
a
7
1
∞
c


b
2
2
2
s
0
1
5
∞
a
7
1
7
c


b
2
2
2
s
0
1
5
7
a
7
1
3
c


b
2
2
2
s
0
1
5
4
a
7
1
3
c

Dijkstra works

For each u in S, the path $P_{s,u}$ is the shortest (s,u) path

Proof by induction on the size of S

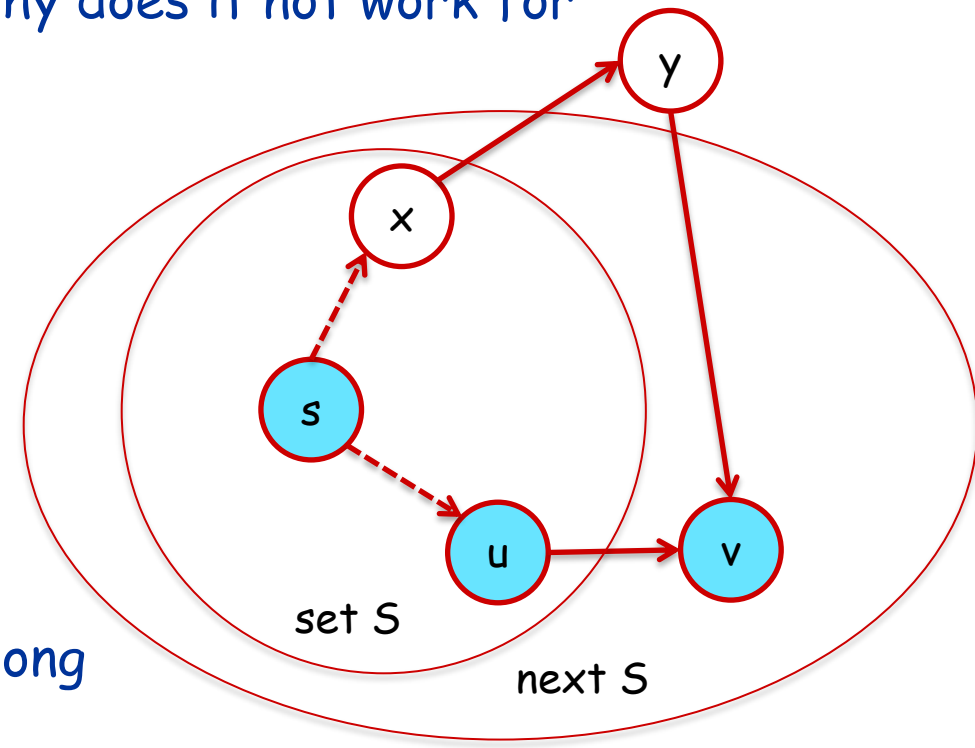
Base: |S| = 1 d[s]=0 OK

Step: Suppose it holds for |S|=k>=1, then grow S by 1 adding node v using edge (u,v) (u already in S) to create the next S.
Then path $P_{s,u,v}$ is path $P_{s,u}$+(u,v), and is the shortest path to v

WHY? What are the "ingredients" of an exchange argument?
What are the inequalities?

Greedy exchange argument

Assume there is **another path P from s to v**.
P leaves S somewhere with edge (x,y).
Then the path P goes from s to x to y to v.
What can you say about P: s →* x → y compared to $P_{s,u,v}$? How does the algorithm pick $P_{s,u,v}$? Why does it not work for negative edges?



P from s to y is at least as long as $P_{s,u,v}$ because the algorithm picks the shortest extension out of S.

Hence the path
P: s →* x → y →* v is at least as long as
$P_{s,u,v}$: s →* u → v
Would not work if w(y,v) <0