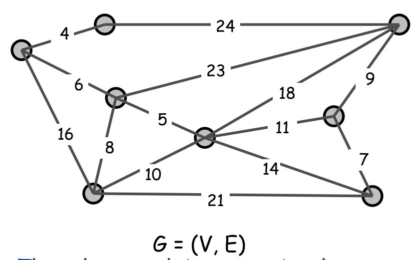
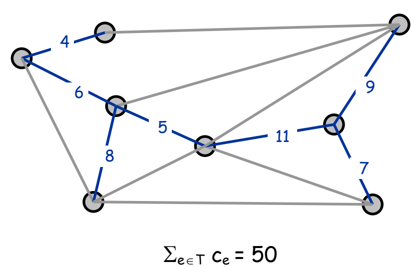
Minimum Spanning Trees
Shortest Paths

Cormen et. al. VI 23,24

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.





Then that graph is a tree, i.e., has no cycles.

WHY?

If there is a cycle, we can take one edge out of the cycle and still connect all nodes. (Repeat if there are more cycles.)

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 or Minimum Spanning Tree?

Minimum is the smallest possible or allowable amount.
Minimal implies that the amount is (relatively) small.
Hence Minimum Spanning Tree.

Optimum is often used to denote the unique best, and
Optimal denotes one of (possibly multiple) best values

3

A generic MST algorithm

Loop invariant: Prior to each iteration, A is a subset of some minimum spanning tree.

```
GENERIC-MST(G, w)
A = ∅
while A does not form a spanning tree
    find an edge (u, v) that is safe for A
    A = A ∪ {(u, v)}
return A
```

How to determine a 'safe edge'?

*Introduction to Algorithms by Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein.

4

5

Three Greedy Algorithms for MST

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.

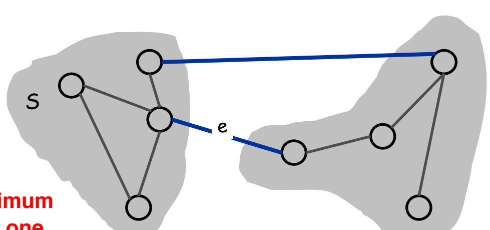
Prim's algorithm. Start with some node s and greedily grow a tree T from s. At each step, add the cheapest edge e to T that has exactly one endpoint in T, i.e., without creating a cycle.

The cut property

Simplifying assumption. All edge costs are distinct. In this case the MST is unique. In general it is not.

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 MST algorithm.



If multiple equal minimum cost edges, just pick one

e is a safe edge

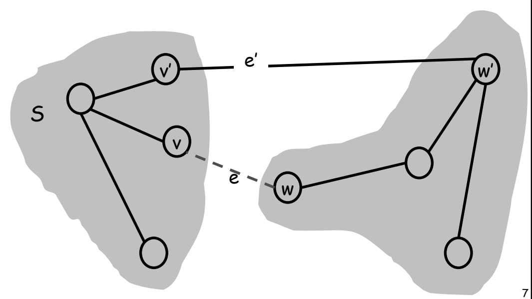
6

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, there is another edge e'= (v',w') with $c_{e'} > c_e$ connecting S and V-S. Assume e' is in the MST, and not e. Adding e to the spanning tree creates a cycle, then taking out e' out removes the cycle creating a new spanning tree with lower cost. **C**ontradiction.

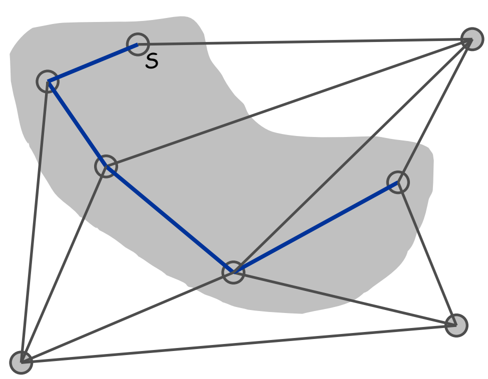
Remember CS220:
if we add an edge to a tree
we get a cycle,
if we take any edge out of that
cycle we get a tree again.



7

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, greedily growing the MST.



8

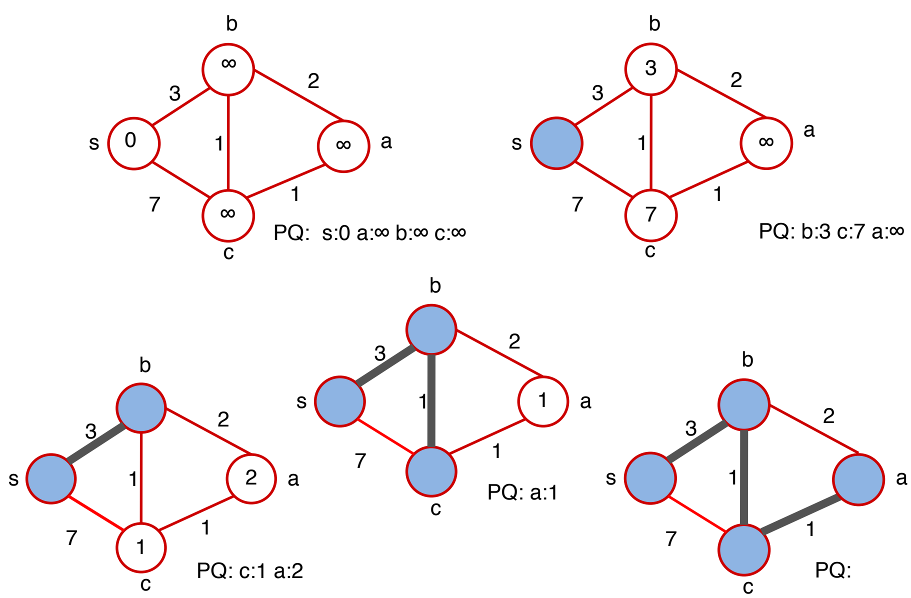
Prim's algorithm: Implementation

- 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.

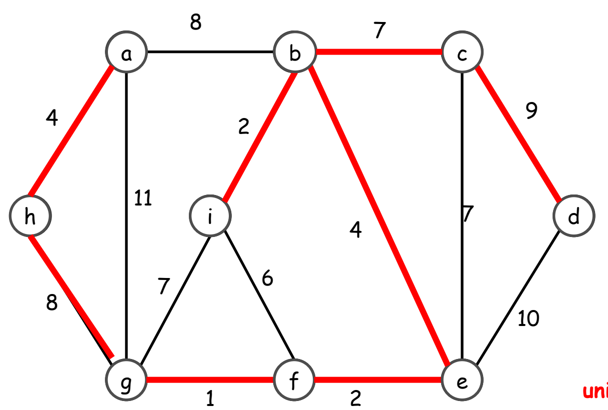
```
Prim(G,s)
   foreach (v ∈ V)
       priority a[v] ← ∞
   a[s]= 0
   priority queue Q = {}
   foreach (v ∈ V) insert v onto Q (key: a[v] )
   set S ← {}
   while (Q is not empty) {
      u ← delete min element from Q
      S ← S ∪ { u }
      foreach (edge e = (u, v) incident to u)
          if ((v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

9

Prim: DO IT, DO IT!



Let's do the Prim, starting at d



unique?

{(d,c),(c,b), (b,i), (b,e), (e,f), (f,g), (g,h), (h,a) }

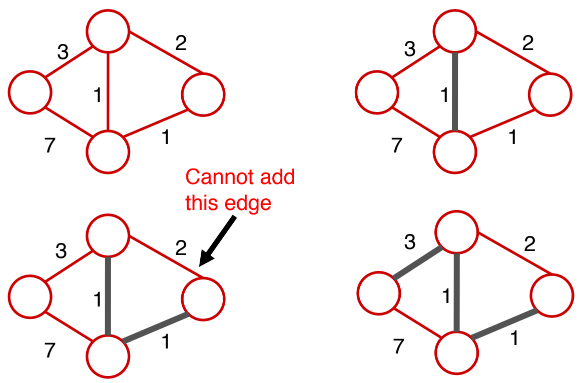
Kruskal's Algorithm

```
MST-KRUSKAL(G, w)
A = ∅
for each vertex v ∈ G.V
    MAKE-SET(v)
sort the edges of G.E into nondecreasing order by weight w
for each edge (u, v) ∈ G.E, taken in nondecreasing order by weight
    if FIND-SET(u) ≠ FIND-SET(v)
        A = A ∪ {(u, v)}
        UNION(u, v)
return A
```

*Introduction to Algorithms by Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein.

Kruskal's algorithm [Kruskal, 1956]

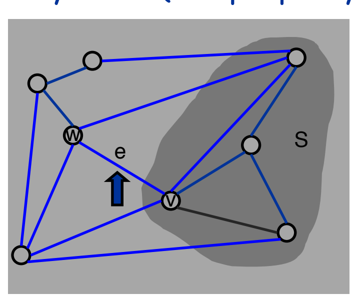
Kruskal:
Consider edges in ascending order of weight. Add edge unless doing so would create a cycle.



Kruskal works

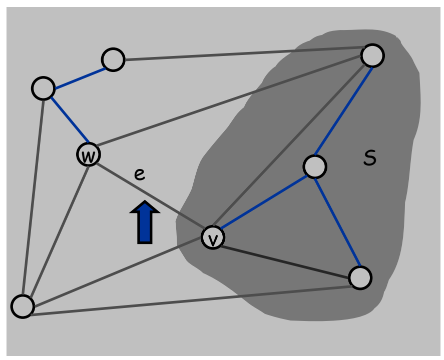
1 **Spanning Tree:** Kruskal keeps adding edges until all nodes are connected, and does not create cycles, so produces a spanning tree.

2. **Minimum Spanning Tree:** 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).



Kruskal produces an MST

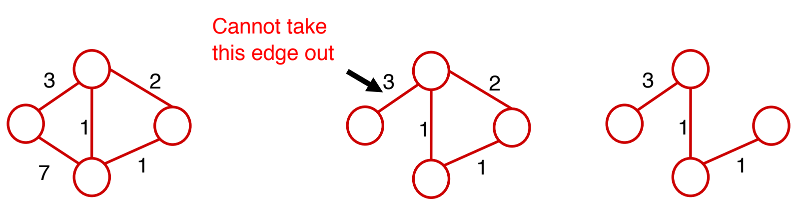
- Consider e=(v, w) added by Kruskal. S is the set of nodes connected to v just before e is added; v is in S and w is not (otherwise we created a cycle). Therefore e is the cheapest edge connecting S to a node in V-S, and hence, e is in any MST (cut property).



16

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.



Is it always safe to remove e, i.e. could e be in an MST?

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.

Is it safe to remove e, i.e. could e be in an MST?

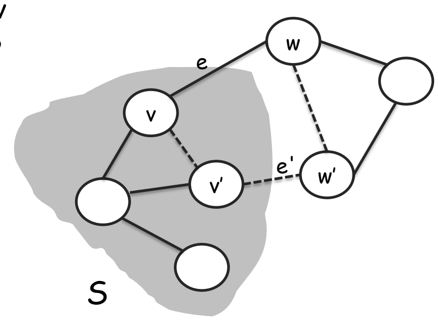
Cycle property. Let C be any cycle in G, and let e be the max cost edge belonging to C. Then e doesn't belong to any MST of G.

17

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', with lower cost than e, so T - {e} + {e'} is again a spanning tree and its cost is lower than T, so T is not an MST.



18

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 of the shortest path problem:

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 more efficient 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 there are other approaches (cs420).

Dijkstra SSSP

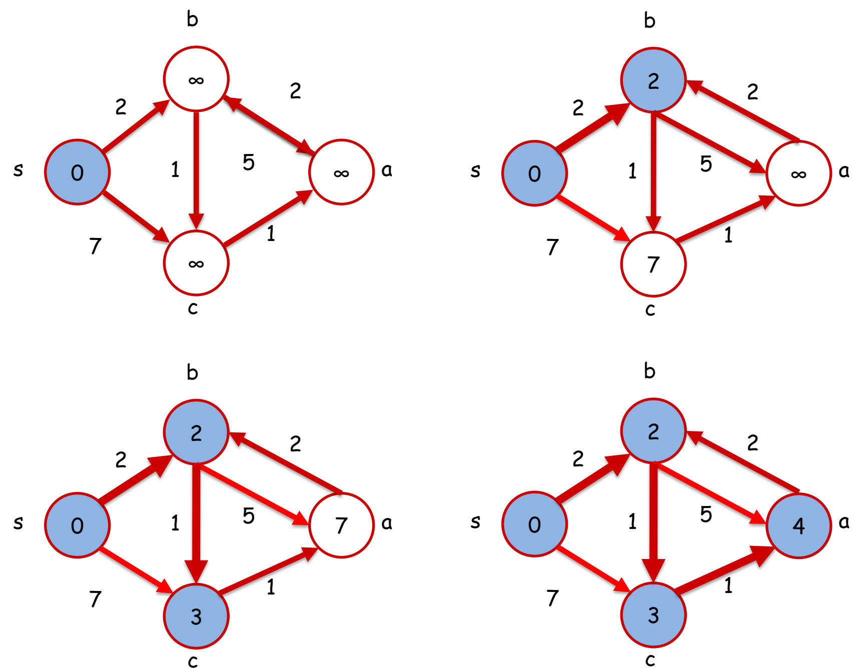
Dijkstra's (Greedy) SSSP algorithm only works for graphs with only positive edge weights.

```
DIJKSTRA(G, w, s)
INITIALIZE-SINGLE-SOURCE(G, s)
S = Ø
Q = G.V
while Q ≠ Ø
    u = EXTRACT-MIN(Q)
    S = S ∪ {u}
    for each vertex v ∈ G.Adj[u]
        RELAX(u, v, w)
```

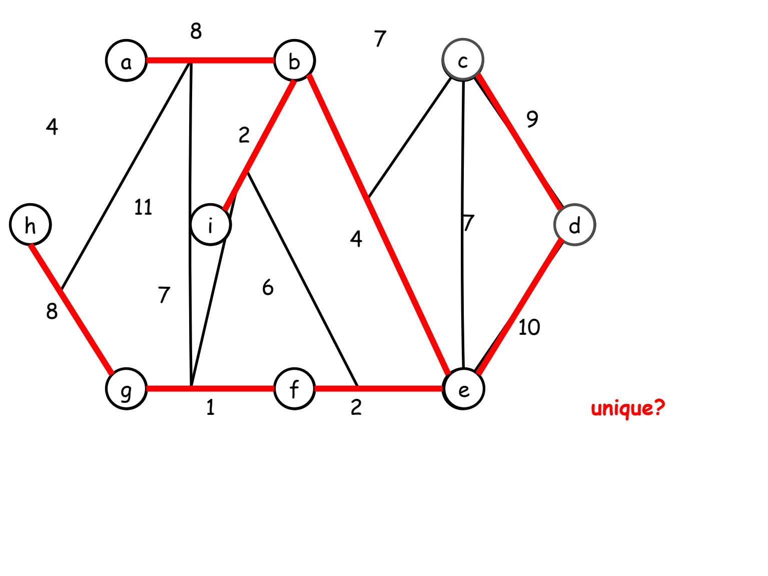
To compute the actual minimum paths, maintain an array p[v] of predecessors. **WHY predecessors?**

Notice: Dijkstra is very similar to Prim's MST algorithm. Where Dijkstra minimizes path lengths, Prim minimizes sum of edge lengths.

*Introduction to Algorithms by Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein.



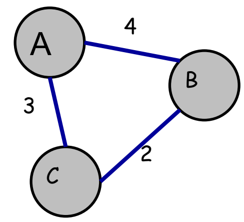
Let's do Dijkstra, starting at d



Does Dijkstra's algorithm lead to a Minimum Spanning Tree?

No.

Create a counter example: (s=A)



Shortest paths from A?
Minimum Spanning Tree?

Formulate the difference between Prim and Dijkstra

23

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

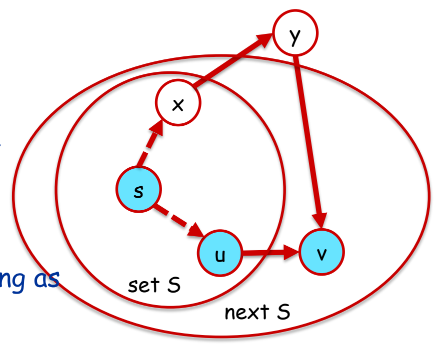
Base: |S| = 1 d[s]=0 OK

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 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

This would not work if w(y,v) <0

Bellman-Ford algorithm

Worse than Dijkstra but works with negative-weight edges
- returns True iff graph does not contain negative-weight cycles

```
BELLMAN-FORD(G, w, s)
INITIALIZE-SINGLE-SOURCE(G, s)
for i = 1 to |G.V| - 1
    for each edge (u, v) ∈ G.E
        RELAX(u, v, w)
for each edge (u, v) ∈ G.E
    if v.d > u.d + w(u, v)
        return FALSE
return TRUE
```

Each edge is relaxed |V-1| times
- Dijkstra's algorithm relaxes each edge exactly once

*Introduction to Algorithms by Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein.