Lecture 15 : Graphs 4

Single Source Shortest Path

Task: given start node s in a weighted graph with non-negative edge weights, find the shortest path from s to all other vertices (or a specific node)

Dijkstra's Algorithm!

intuition: in BFS, we visited nodes in order based on their layer/depth
in Dijkstra, we visit nodes in order based on their distance

Dijkstra's Pseudocode

1. extract node curr from priority queue
   this makes curr "done"
2. for each not done neighbor of curr:
   - update its distance if we have found a better path.

Seen = added to PQ

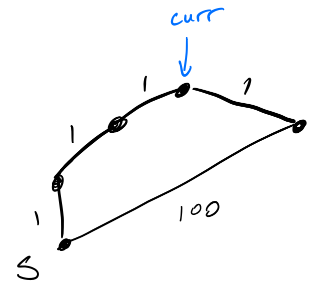
done = removed from PQ

BFS Pseudocode

1. extract node curr
2. for each not visited neighbor of curr:
   add it to queue
   depth of neighbor = depth(curr) + 1

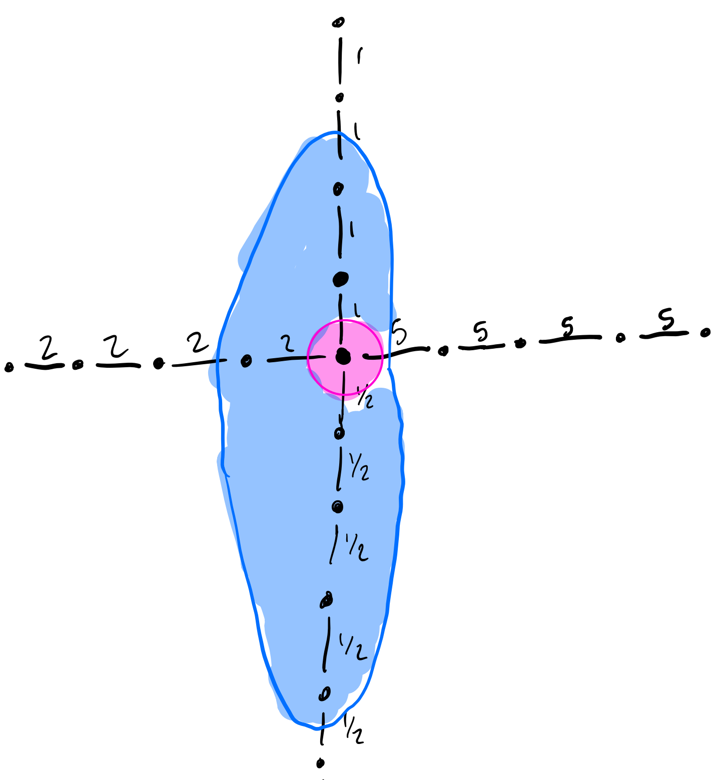
Dijkstra Pseudocode

```
dijkstra(G, s):
    PQ = new priority queue
    add s to PQ with priority 0
    mark s "seen"
    while PQ not empty:
        curr = PQ.extract()
        mark curr as done
        for u in neighbors(curr):
            d = dist to curr + w(curr, u)
            if u not "seen":
                mark u seen
                PQ.add(u, d)
                                   better path!
            if u not done AND d < dist to u:
                distance to u = d
                PQ.decreaseKey(u, d)
```

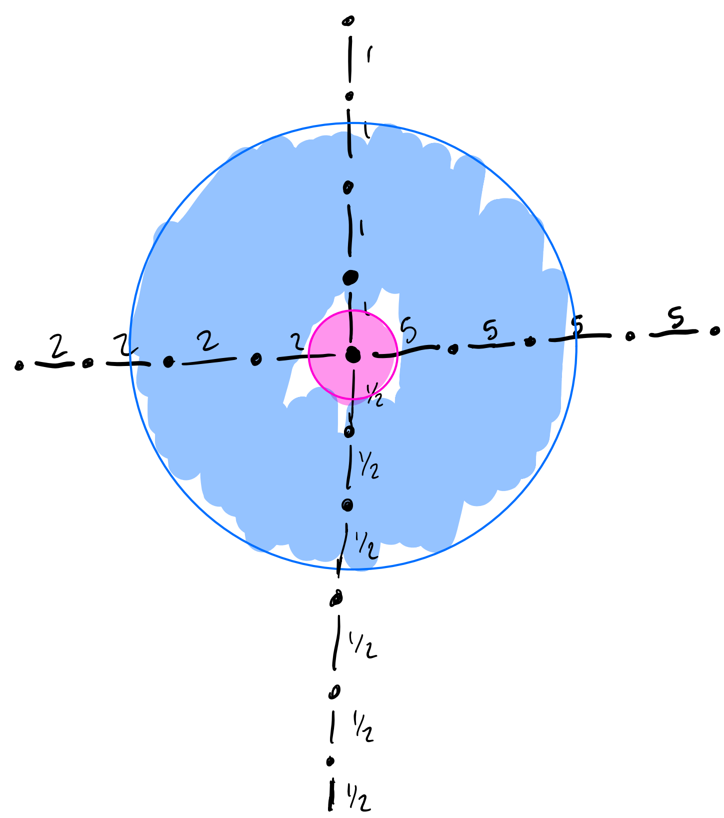


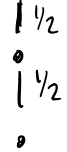
runtime?

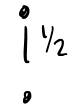
Dijkstra



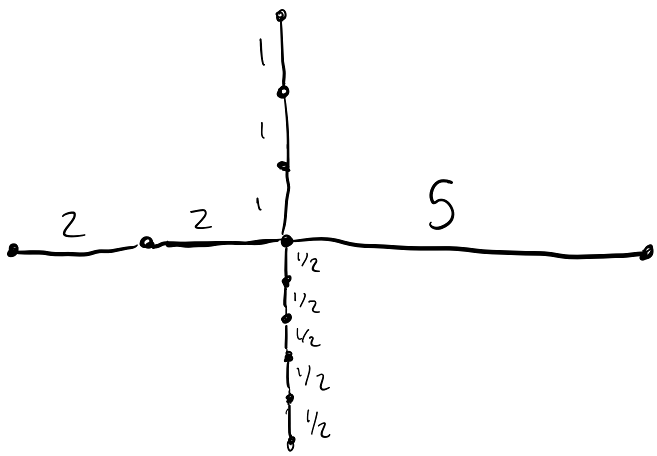
BFS



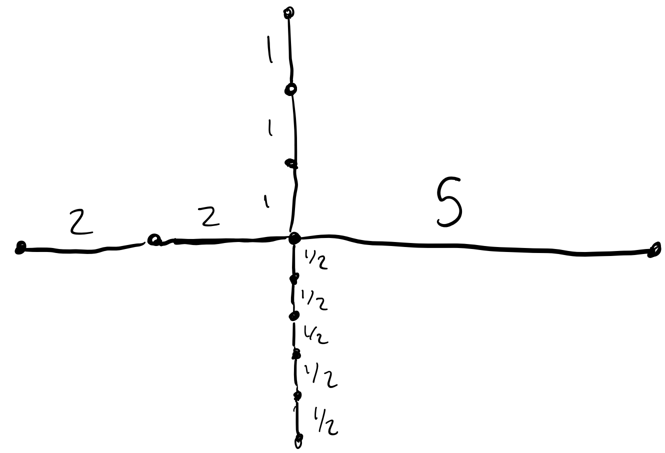




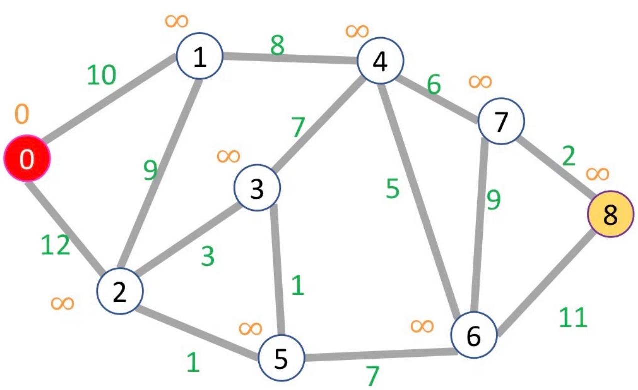
Dijkstra



BFS



Ex 1



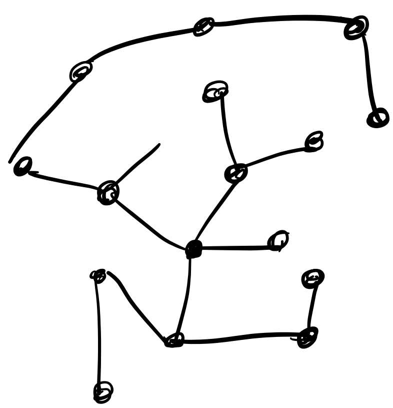
node	seen?	done?	distance
0	True		0
1			∞
2			∞
3			∞
4			∞
5			∞
6			∞
7			∞
8			∞

PQ:

Minimum Spanning Trees

recall **trees** are (connected) graphs with $\boxed{\text{no cycles}}$

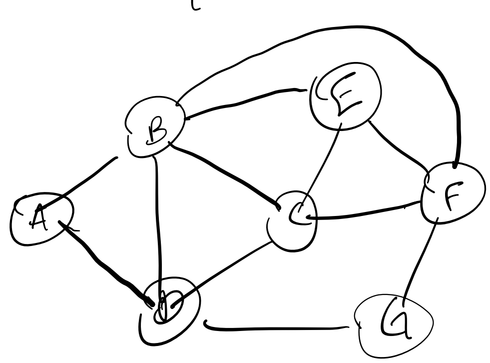
question: how many edges in a tree?



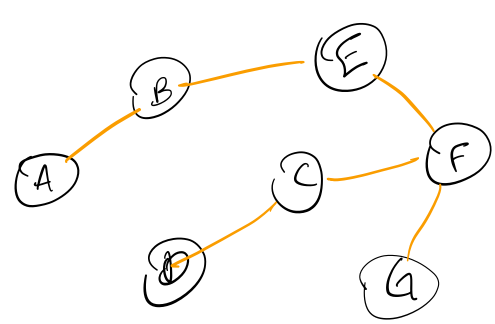
Def (spanning tree) $T = (V_T, E_T)$ is a **spanning tree** for $G = (V, E)$ iff $V_T = V$, $E_T \subseteq E$, and T is a tree.

Ex

G



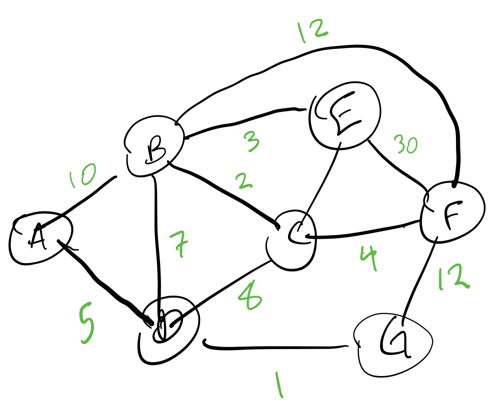
T



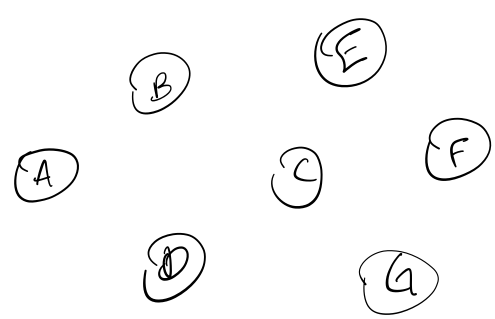
Aside: The number of spanning trees for a complete graph on n vertices is n^{n-2}

Def (Minimum Spanning Tree) Given a (weighted) graph $G=(V,E)$ and weight function $w: E \to \mathbb{R}$, $T=(V_T, E_T)$ is a <u>minimum spanning tree</u> if T is a spanning tree and has minimum cost

$$\text{cost}(T) = \sum_{e \in E_T} w(e)$$



minimum spanning tree T



$\text{cost}(T) =$

intuition: any "trade" only increases cost.

question: is the minimum spanning tree unique?

Applications?

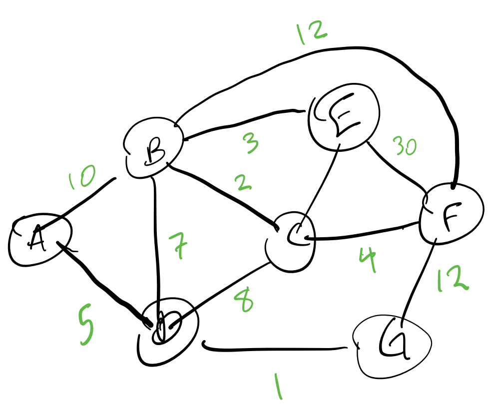
- electric grids
- transportation
-

Prim's Algorithm

- Start with empty tree T
- Pick starting node S, add to T
- Repeat |V|-1 times:

  add the min weight edge with one endpoint in T and one endpoint outside of T

Ex



$S = A$

Prim's Pseudocode

```
Prim(G,s):
  PQ = new priority queue
  add s to PQ with priority 0
  mark s "seen"
  while PQ not empty:
      curr = PQ.extract()
      mark curr as done
      for u in neighbors(curr):
          d = w(curr, u)
          if u not "seen":
              mark u seen
              PQ.add(u,d)
          if u not done AND d < dist to u:
              distance to u = d
              PQ.decreaseKey(u,d)
```

current value of u in PQ

Dijkstra Pseudocode

```
dijkstra(G,s):
  PQ = new priority queue
  add s to PQ with priority 0
  mark s "seen"
  while PQ not empty:
      curr = PQ.extract()
      mark curr as done
      for u in neighbors(curr):
          d = dist to curr + w(curr, u)
          if u not "seen":
              mark u seen
              PQ.add(u,d)
          if u not done AND d < dist to u:
              distance to u = d
              PQ.decreaseKey(u,d)
```

Runtime: still $O(|E| \log |V|)$!

Caveat: Java does not have decreaseKey built in!

Solution 1: add edges to PQ, check if their "destination" is done when extracting

Solution 2: allow PQ to have duplicate vertices

both solutions have runtime $O(|E| \log |E|)$

$= O(|E| \log |V|)$

(unchanged!)

Question: why does Prim's work?

1. Prim's produces a spanning tree (when G connected)

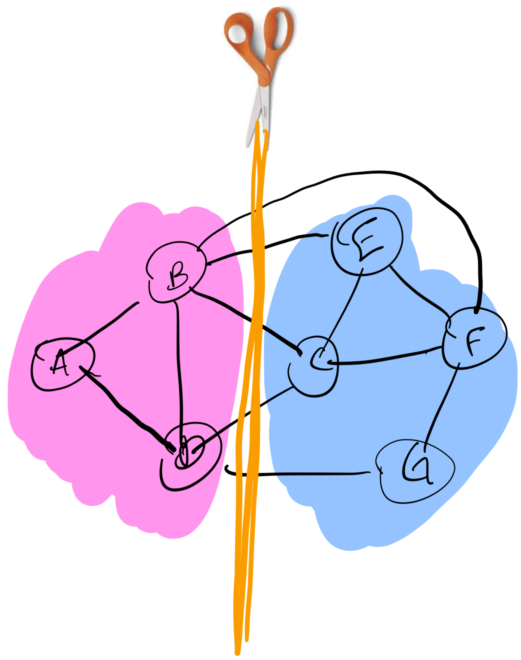
   need to show 2 of:

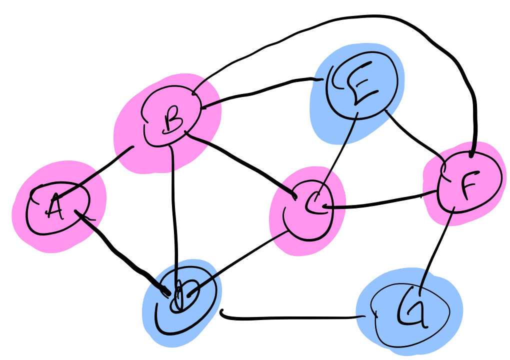
   - connected
   - acyclic
   - $|V|-1$ edges

2. Prim's produces minimum spanning tree

   - uses the "Cut Theorem"

Def'n (Cut) a **cut** of a graph $G = (V, E)$ is a partition of the vertices into two sets, S and $V \setminus S$

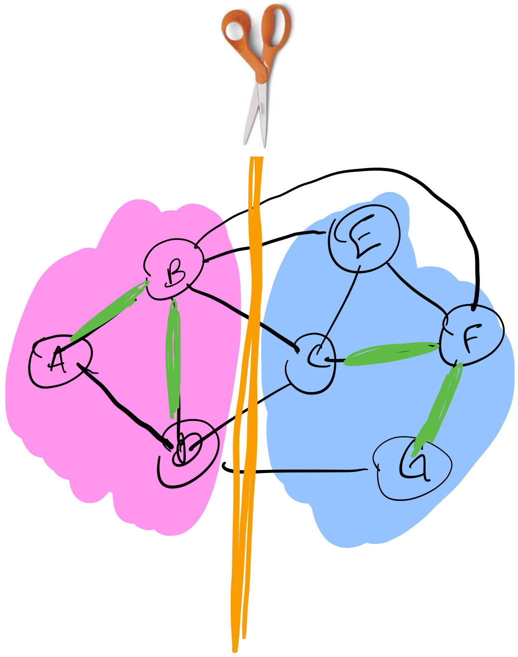




edge $e = (v_1, v_2)$ **crosses** a cut $(S, V \setminus S)$ if $v_1 \in S$ and $v_2 \in V \setminus S$
or $v_1 \in V \setminus S$ and $v_2 \in S$

A set of edges R "**respects a cut**" iff

$\forall e \in R$, e does not cross the cut

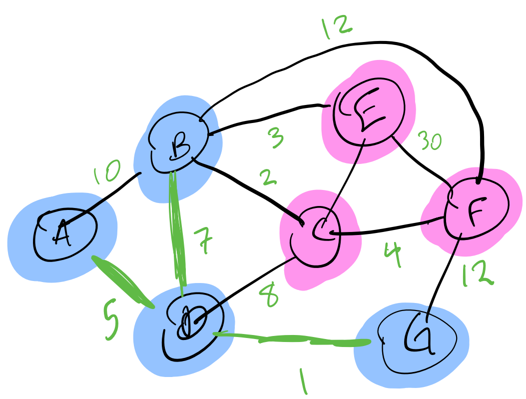


$R = \{(A,B), (B,D), (C,F), (F,G)\}$

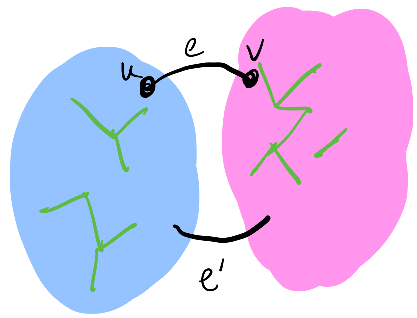
Theorem (Cut Theorem) Suppose $A \subseteq E$ is a subset of a minimum spanning tree T, and let $(S, V\setminus S)$ be any cut which A respects.

If e is a smallest-weight edge crossing $(S, V\setminus S)$, then $e \in T$.

Ex



Proof



any T must have at least one edge crossing $(S, V \setminus S)$

Suppose towards a contradiction that $e = (u,v) \notin T$

There must exist path from u to v in T

That path must involve some e' that crosses $(S, V \setminus S)$

consider $\tilde{T} = T \setminus \{e'\} \cup \{e\}$ (swap e' with e !)

$\tilde{T}$ is a spanning tree

$cost(\tilde{T}) =$

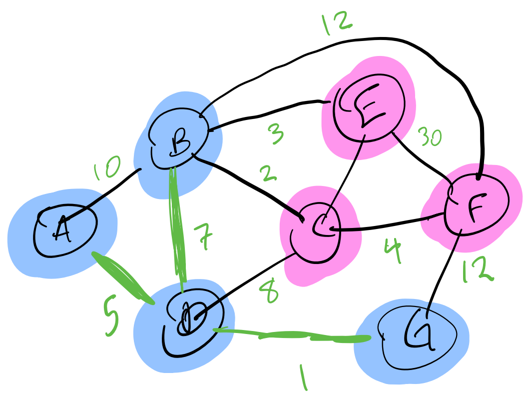
$\boxed{M}$

So... why does Prim's produce a minimum spanning tree?

- Start with empty tree A
- Pick starting node s, add to A
- Repeat $|V|-1$ times:

add the min weight edge with one endpoint in A and one endpoint outside of A

this crosses the cut $(A, V\setminus A)$, which is respected by the current subtree!



General MST Algorithm

- Start with empty tree A
- Repeat $|V|-1$ times:
  - Pick a cut $(S, V\setminus S)$ that A respects and add the min-weight edge crossing that cut.

Kruskal's Algorithm

- Start with empty tree A
- repeatedly add lowest weight e that does not create a cycle.

if $e = (u, v)$ $\quad S$ = connected component of A containing u.