Algorithms - Fall 2020

Graphs:
BFS + DFS

Recap

- HW3 - due today (GROUPS!)
- HW4 - up tonight, due next week
- Reading as usual

Last time: Graphs!

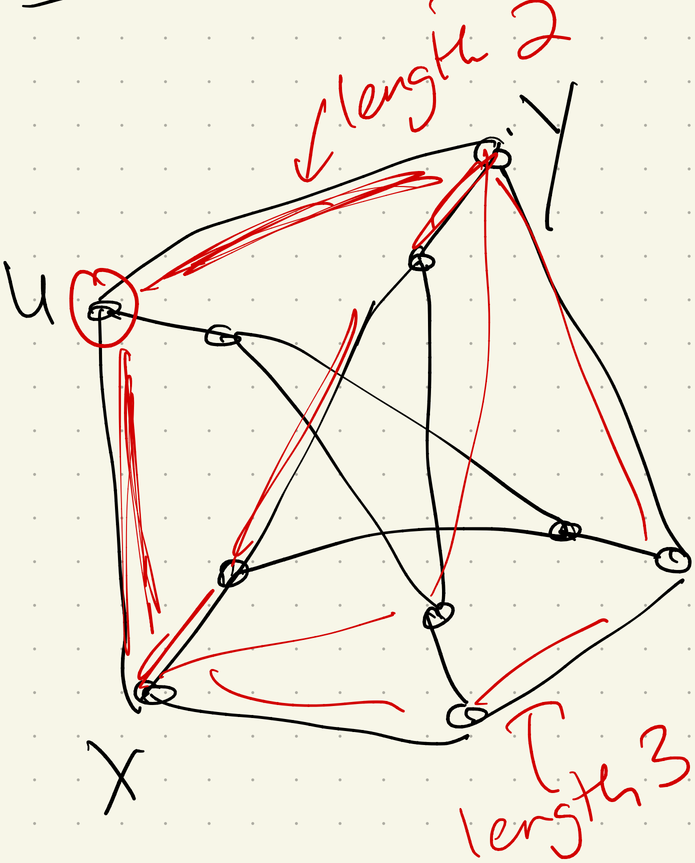
Lots of notation, definitions, + initial properties.

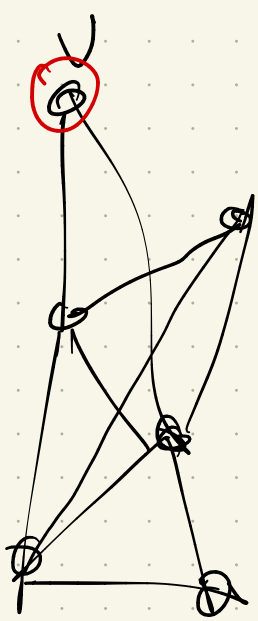
keep this section handy

Today: Searching

Q: are u + v connected in G? No

Are x + y? Yes!



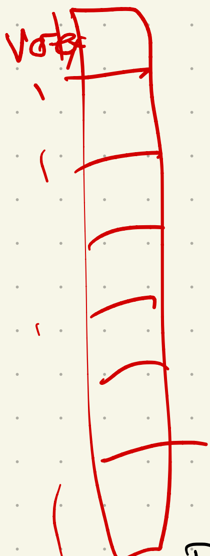


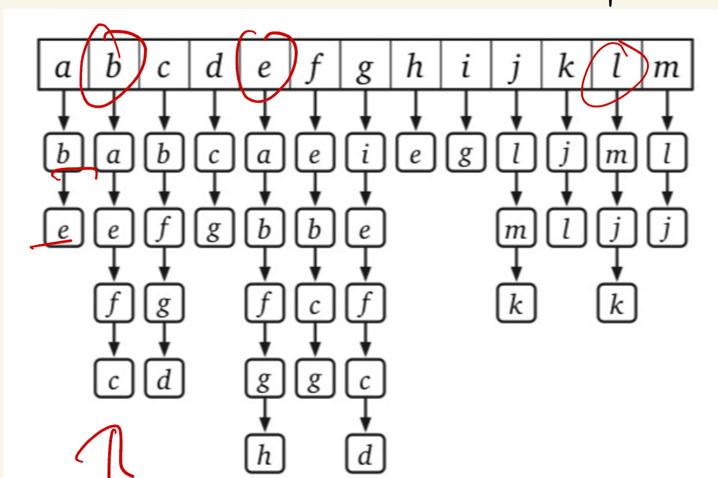
Graph Searching

How can we tell if 2 vertices are connected?

Remember, the computer only has:

parent





adj lists

is a connected to l?

Bigger question: can we tell if all the vertices are in a single connected component?

Possibly you saw depth first search (DFS) or breadth first search (BFS) in data structures:

```
WhateverFirstSearch(s):
    put s into the bag
    while the bag is not empty
        take v from the bag
        if v is unmarked
            mark v
            for each edge vw
                put w into the bag
```

data structure: stack + queue

These are essentially just search strategies!

How can we decide if u + v are connected?

First question: bag?!

→ DFS: stack

→ BFS: queue

heaps ⟷ weighted edges — later chapter

↑ PQ

Can use this to build a spanning tree:

↑ acyclic graph touching all edge

parent list

$s \to \phi$

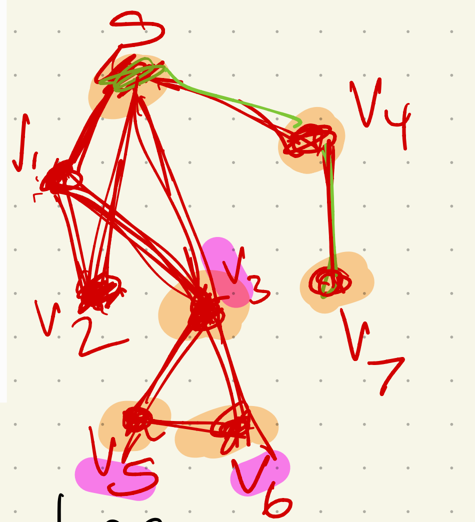
$v_1 \to s$

$v_2 \to s$

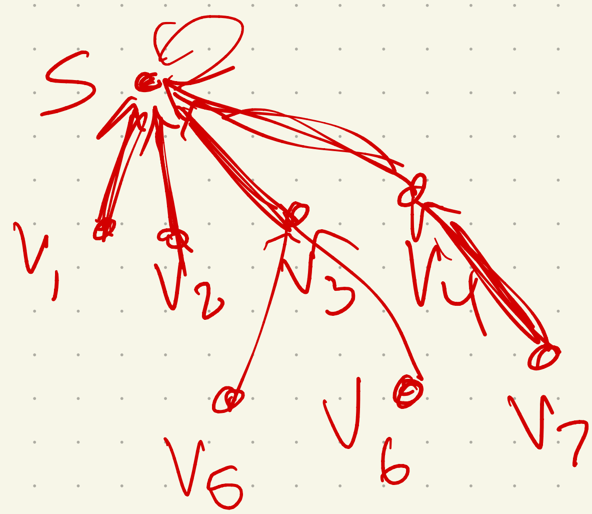
⋮

$v_7 \to v_4$

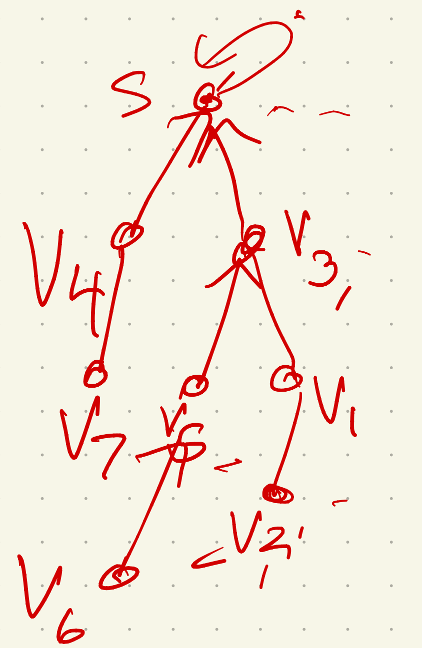
```
WhateverFirstSearch(s):
    put (∅, s) in bag
    while the bag is not empty
        take (p, v) from the bag                 (⋆)
        if v is unmarked
            mark v
            parent(v) ← p
            for each edge vw                     (†)
                put (v, w) into the bag          (⋆⋆)
```



BFS tree:



DFS tree



~~(s, ϕ)~~ → ~~(s, v1)~~ → ~~(s, v2)~~ → ~~(s, v3)~~ → ~~(s, v4)~~ → ~~(v1, v2)~~ → ~~(v1, v3)~~ → ~~(v2, s)~~ → ~~(v2, v1)~~ → ~~(v3, v5)~~ → v_3 v_6 → $v_4 v_3$

Queue

p v

~~(s, ϕ)~~

Stack

Just remember: different!

other edges

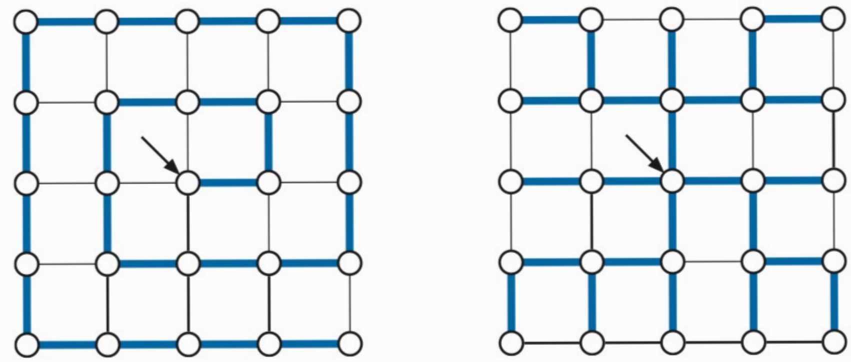
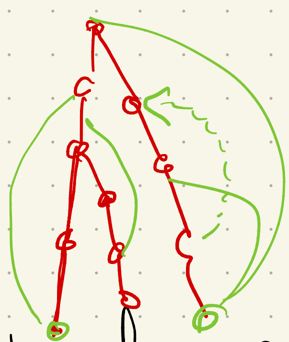


Figure 5.12. A depth-first spanning tree and a breadth-first spanning tree of the same graph, both starting at the center vertex.

DFS:

long &

"skinny"



BFS:

short

&

"bushy"

distance 1

distance

2

3

Which one?

depends!

Runtime:

```
WHATEVERFIRSTSEARCH(s):
    put s into the bag
    while the bag is not empty
        take v from the bag
        if v is unmarked
            mark v
            for each edge vw
                put w into the bag
```

(p, v)

ugly for recursion

→ Each vertex: visited at most once

when visited: first time, add its edges to DS "bag"

+ mark vertex

other time: $O(1)$ check

$$\underbrace{\sum_{v} d(v) \cdot [\text{add to bag}]}_{} \quad [O(1)$$

$+ V$

$O(V+E)$ ~~$E \leq V^2$~~

$\left(\text{cost to add to DS}\right) \cdot 2E + V$

stack or queue $O(1)$

Correctness:

Claim: WFS will mark all reachable vertices.

Pf: induction on distance to the source:

$d=0$: S!

it is marked ✓

$d>0$: Consider v at distance d, so $S \xrightarrow{e_1} v_1 \xrightarrow{e_2} v_2 \cdots \rightarrow v_{d-1} \xrightarrow{e_d} v$ in G.

$\underbrace{\qquad\qquad\qquad\qquad}_{d \text{ edges}}$

By IH: v_{d-1} is marked correctly,

That means v_{d-1} was marked!

```
WHATEVERFIRSTSEARCH(s):
    put s into the bag
    while the bag is not empty
        take v from the bag
        if v is unmarked
            mark v
            for each edge vw
                put w into the bag
```

↳ all its edges were added to "bag"

so edge $v_{d-1} \to v$ (distance d) was added.

↳ at some point, removed + marked.

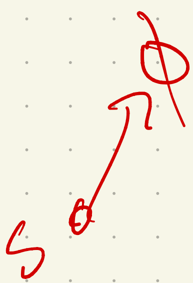
$\blacksquare$

Claim: marked v's + parents form a spanning tree. (See demo's...)

proof:

```
WHATEVERFIRSTSEARCH(s):
  put (∅, s) in bag
  while the bag is not empty
      take (p, v) from the bag          (⋆)
      if v is unmarked
          mark v
          parent(v) ← p
          for each edge vw              (†)
              put (v, w) into the bag   (⋆⋆)
```

For each marked vertex:



someone added it!

(p, v) pair

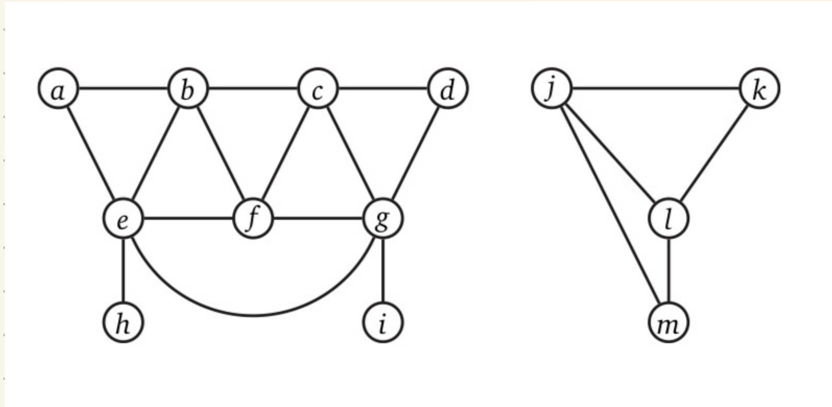
next (u, v) edge
↳ already marked

added n-1 edges, no cycles
⇒ tree

In a disconnected graph:

Often want to count or label the <u>components</u> of the graph!

(WFS(v) will only visit the piece that v belongs to.)



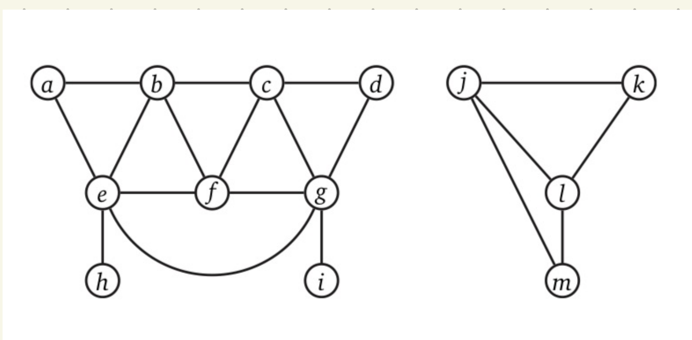
Solution: Call it more than one time!

<u>un</u>mark all vertices

For all vertices v:

Might want to count the # of components:

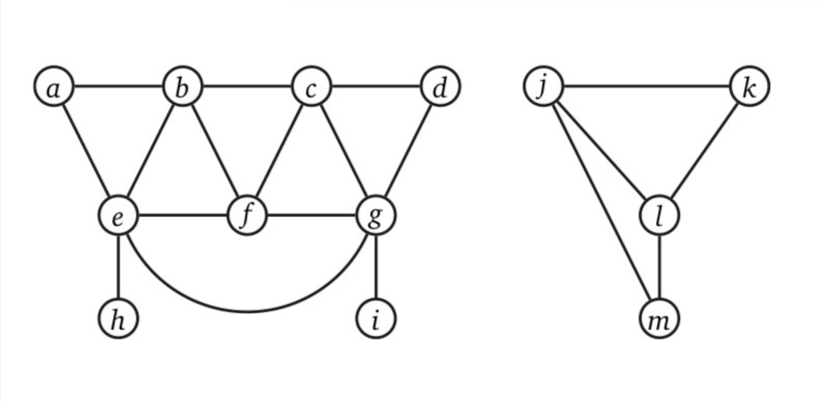
```
COUNTCOMPONENTS(G):
  count ← 0
  for all vertices v
      unmark v
  for all vertices v
      if v is unmarked
          count ← count + 1
          WHATEVERFIRSTSEARCH(v)
  return count
```



Finally, can even record which component each vertex belongs to:

```
COUNTANDLABEL(G):
  count ← 0
  for all vertices v
      unmark v
  for all vertices v
      if v is unmarked
          count ← count + 1
          LABELONE(v, count)
  return count
```

```
⟨⟨Label one component⟩⟩
LABELONE(v, count):
  while the bag is not empty
      take v from the bag
      if v is unmarked
          mark v
          comp(v) ← count
          for each edge vw
              put w into the bag
```



Next time!

Reductions & applications.