# Recap

- HW due next Wed.
- Next HW: over graphs
  - ↳ due last Fri of classes
- 1 more lab (?)
- 2 books reading for next Wed.
- Midterm2 grades are in blackboard (please double check)

# Graphs

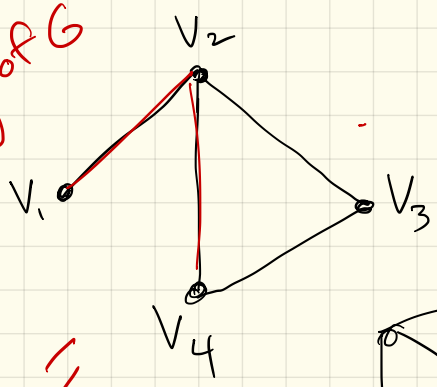
A graph $G = (V, E)$ is an ordered pair of 2 sets:

$V$ = vertices = $\{v_1, v_2, v_3, v_4\}$

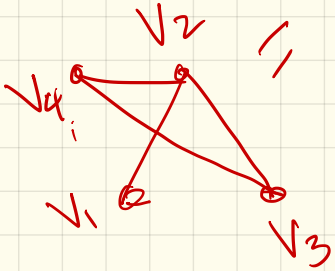
$E$ = edges = $\{\{v_1, v_2\}, \{v_2, v_4\}, \ldots\}$

View:

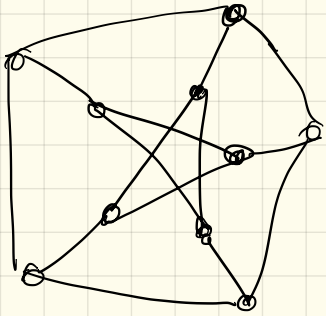
drawing of G




=




=



Which is better?

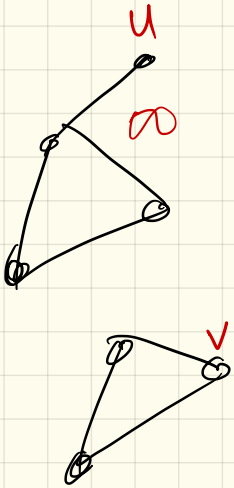
Depends!

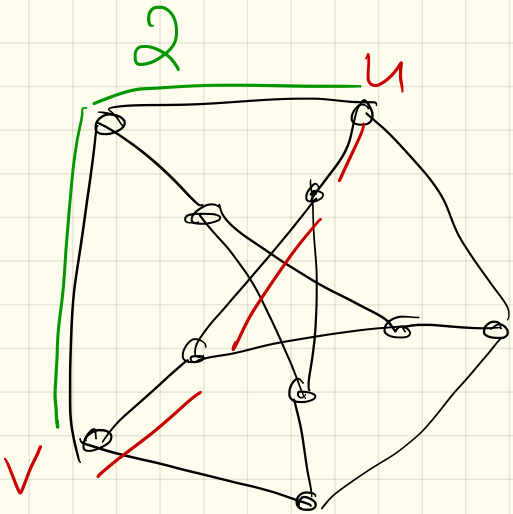
(Representations)

| | Adjacency matrix | Standard adjacency list (linked lists) | Adjacency list (hash tables) |
|---|---|---|---|
| Space | $\Theta(V^2)$ | $\Theta(V+E)$ | $\Theta(V+E)$ |
| Time to test if $uv \in E$ | $O(1)$ | $O(1+\min\{\deg(u),\deg(v)\}) = O(V)$ | $O(1)$ |
| Time to test if $u \to v \in E$ | $O(1)$ | $O(1+\deg(u)) = O(V)$ | $O(1)$ |
| Time to list the neighbors of $v$ | $O(V)$ | $O(1+\deg(v))$ | $O(1+\deg(v))$ |
| Time to list all edges | $\Theta(V^2)$ | $\Theta(V+E)$ | $\Theta(V+E)$ |
| Time to add edge $uv$ | $O(1)$ | $O(1)$ | $O(1)^*$ |
| Time to delete edge $uv$ | $O(1)$ | $O(\deg(u)+\deg(v)) = O(V)$ | $O(1)^*$ |

Dfn:

- $G$ is connected if $\forall u, v$, there $\exists$ path from $u$ to $v$.
- The distance from $u$ to $v$, $d(u,v)$, is equal to the # of edges on the minimum $u,v$-path



# Algorithms on graphs

## Basic 1st question:

Given any 2 vertices, are they connected?

Also: what is their distance?

↳ minimum path

How to solve?

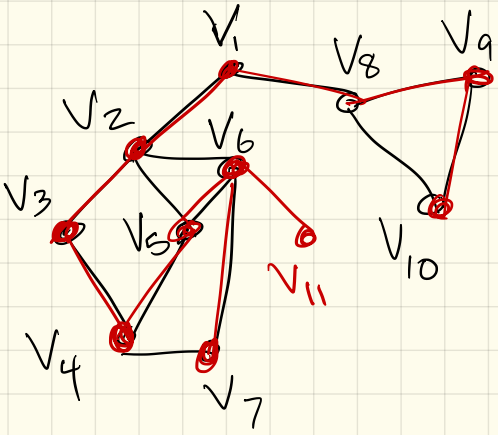
# Pseudocode : two versions

```
RecursiveDFS(v):
  if v is unmarked
      mark v
      for each edge vw
          RecursiveDFS(w)
```

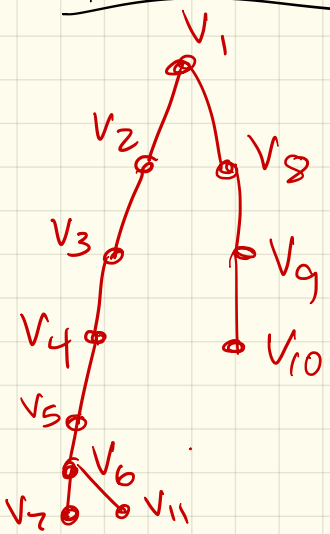
```
IterativeDFS(s):
  Push(s)
  while the stack is not empty
      v ← Pop
      if v is unmarked
          mark v
          for each edge vw
              Push(w)
```

↑ runtime!

Really, building a "tree":

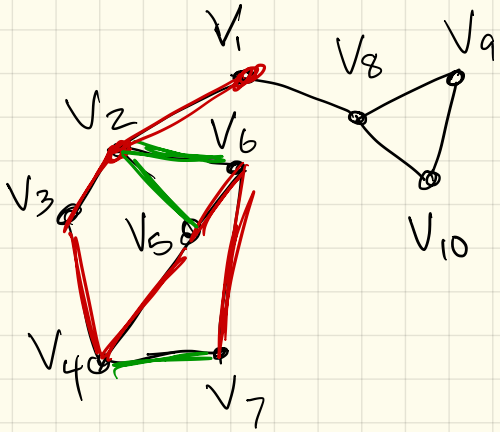


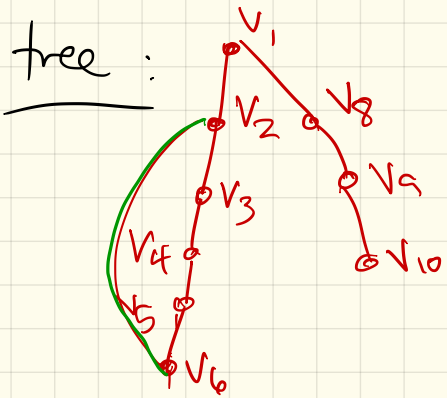

DFS tree:

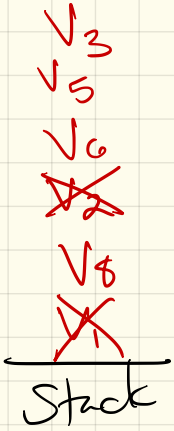


```
RecursiveDFS(v):
    if v is unmarked
        mark v
        for each edge vw
            RecursiveDFS(w)
```

```
IterativeDFS(s):
    Push(s)
    while the stack is not empty
        v ← Pop
        if v is unmarked
            mark v
            for each edge vw
                Push(w)
```



# General traversal strategy:

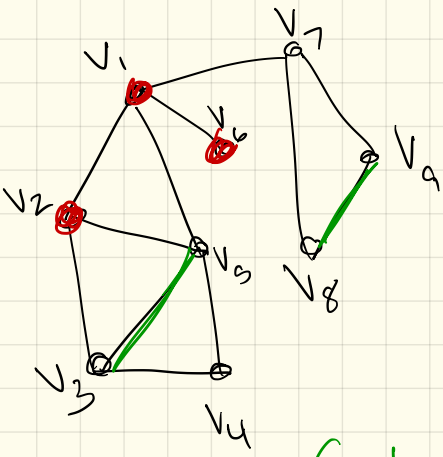
```
TRAVERSE(s):
  put s into the bag
  while the bag is not empty
      take v from the bag
      if v is unmarked
          mark v
          for each edge vw
              put w into the bag
```

Q: Can we use a different "bag"?

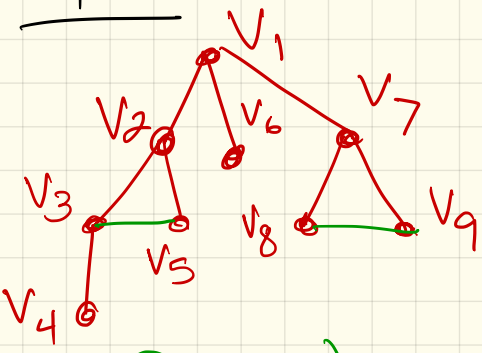
queue!
(also O(1) for everything)

# BFS: use a queue

```
TRAVERSE(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
```




tree:




(not in BFS tree)

queue: ~~v1~~ ~~v2~~ ~~v6~~ ~~v5~~ v1 v3 v5 v1 v8 v9

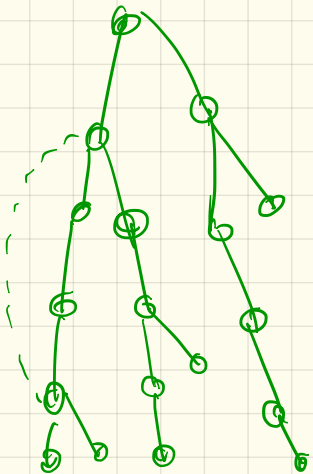
# BFS vs. DFS:

- Both can tell if 2 vertices are connected
- Both can be used to detect cycles in the graph.

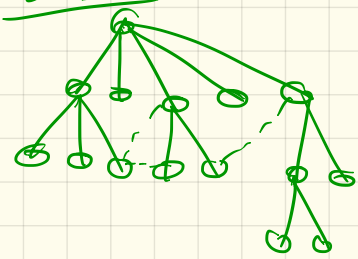
How? If any edges of G are not in tree, must have a cycle.

- Difference

DFS



BFS:



Short, "Bushy"

# Runtimes:

```
TRAVERSE(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
```

stack or queue

O(1)

$\sum_v d(v)$


## DFS:

each edge puts vertex in bag

$O(E)$

$O(V)$ → each vertex gets marked once

$\sum_v d(v) = 2|E|$ ↳ add all nbrs ($O(d(v))$)

## BFS:

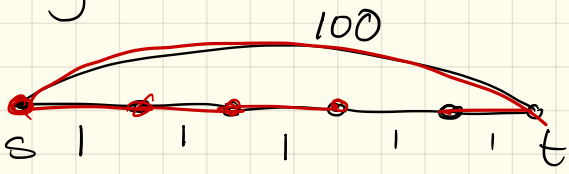
$\Rightarrow O(m+n) \Leftarrow$

Next time:

In some sense, BFS trees are "short".

But – what if graph is weighted?

Ex:




BFS tree:

ignores weights entirely

Need to examine how to get minimum things when graph is weighted.