CS 314 - Depth First Search

Note Title 1/27/2010

Announcements

- HW due Friday
- Next HW out Friday
  due Thursday (orally graded)

Depth-First Search (DFS)

Idea: What if your "graph" is modeling a maze?

BFS is not a good search strategy.
Why?

Backtracking

What would you do instead?

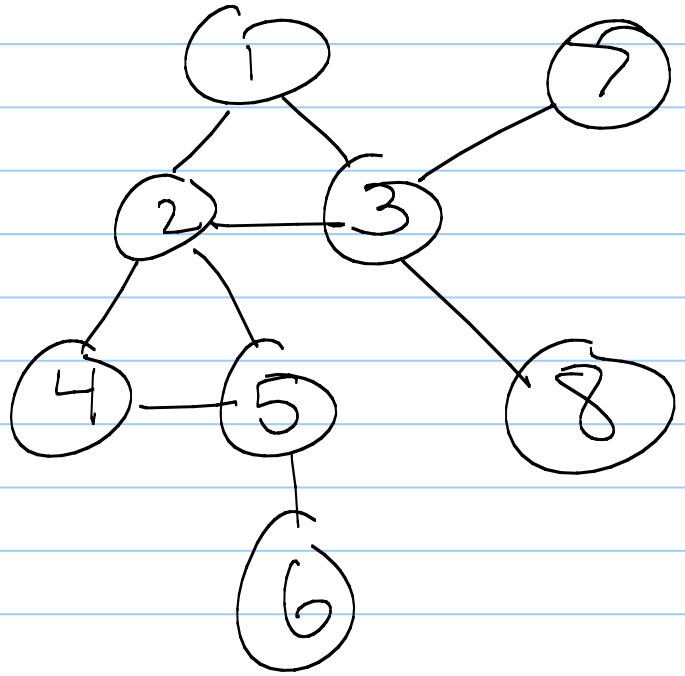
Go until hit a dead end.

RecursiveDFS(u):

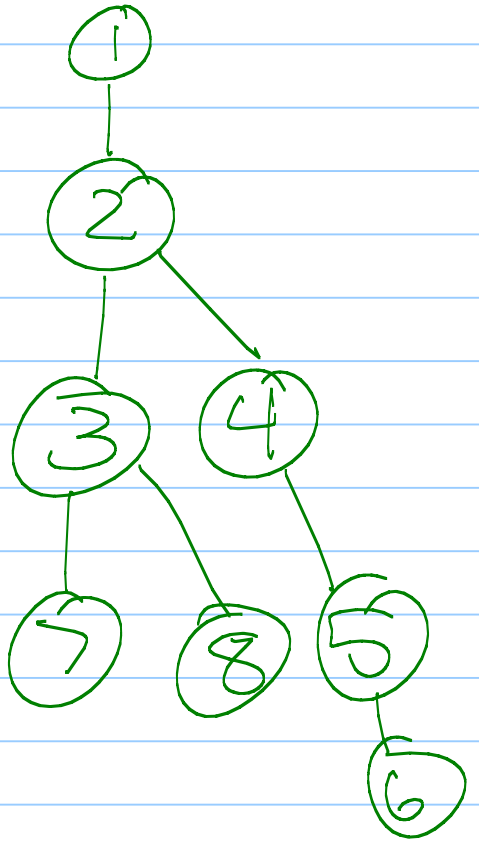
```
if u is unmarked
    mark u
    for each edge {u,v} ∈ E(G)
        RecursiveDFS(v)
    endfor
end if
```

For s-t connectivity, call DFS(s), & if t is ever marked then they are connected.

Example

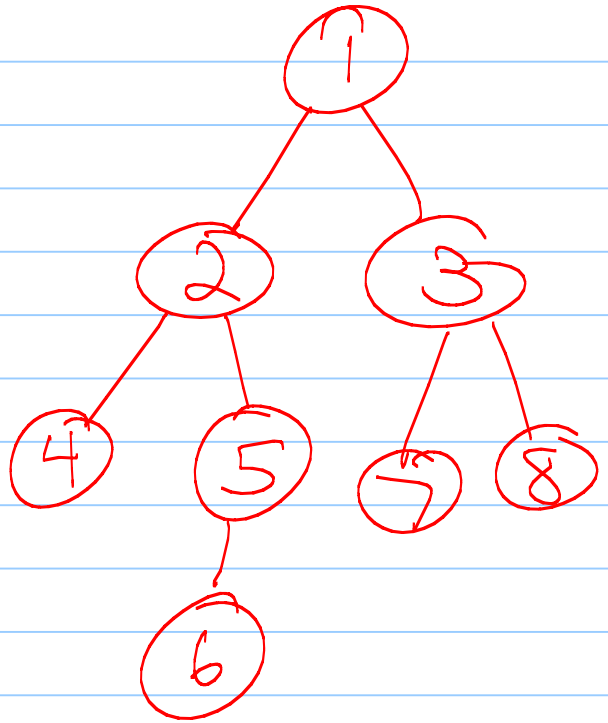


DFS tree:

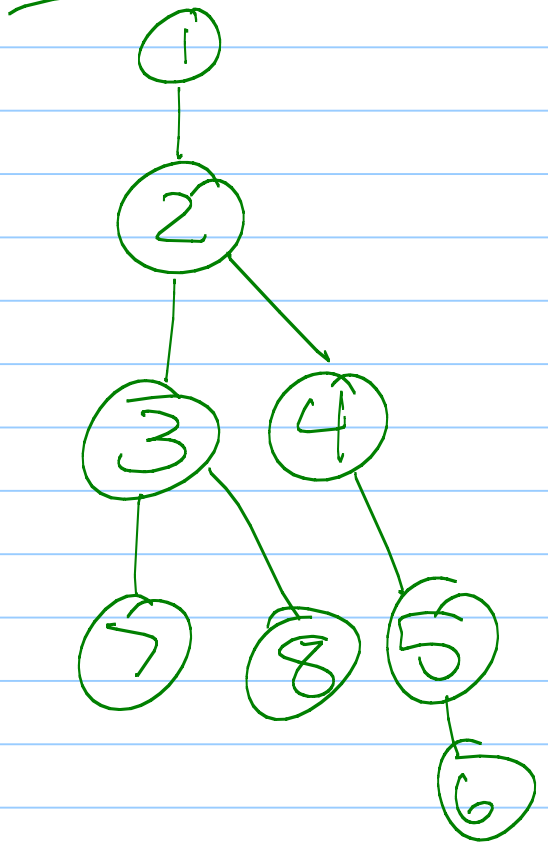


Compare

BFS:



DFS:



One problem with our code...

What is the tree?

No tree was saved

What do we need to keep track of the tree?

```
RecursiveDFS(u):
    if u is unmarked
        mark u
        for each edge {u,v} ∈ E(G)
            RecursiveDFS(v)
        endfor
    end if
```

↳ need to know what edge led to vertex u.

Code to give DFS tree:

```
DFS(s):
    Initialize empty tree T
    Initialize array Explored to be false for all nodes
    Initialize S to be stack with 1 element, (ϕ, s)
    While S is not empty
        (p, u) ← S.pop()
        If Explored[u] = false
            Explored[u] ← true
            Add (p, u) to T
            For each edge {u, v} ∈ E(G)
                put {u, v} into S
            endfor
        endif
    end while
```

Runtime?

What is put in the stack?

edges — m

actually, each edge is added exactly twice.

Each time we remove item from stack, how much time do we spend?

Total runtime:

$$O(n) + O(m) + \sum_{v \in V} O(d(v)+1) = O(m) + O(m) + O(n)$$

$$= \boxed{O(m+n)}$$

$O(n)$: initialization; $O(m)$: stack operations; $O(d(v)+1)$: $\downarrow$

Prop: Let T be a DFS tree, & $x \neq y$ be nodes in T s.t. $\{x,y\} \in E(G)$ but $\{x,y\} \notin E(T)$.

Then one of x or y is an ancestor of the other. (in the tree)

proof is by contradiction (p. 85)

Next time: Divide & Conquer (Ch. 5)

But we've already seen this!

Examples: Merge sort ←

Quicksort

Binary Search ←

$T(n) = O(1) + T(n/2)$

$= O(\log n)$

(Recurrences for running time)