CS180 – Graphs & graph algorithms

Note Title 12/9/2011

Announcements

- Lab tomorrow
- HW due Sunday
- Next HW will be due last day of class

Graphs

(V, E)

vertices edges

Representations: Vertex list : space $O(m+n)$

checking adjacency: $O(n)$

Adjacency matrix

space: $O(n^2)$

checking adjacency: $O(1)$

RecursiveDFS(u):

extra O(n) space

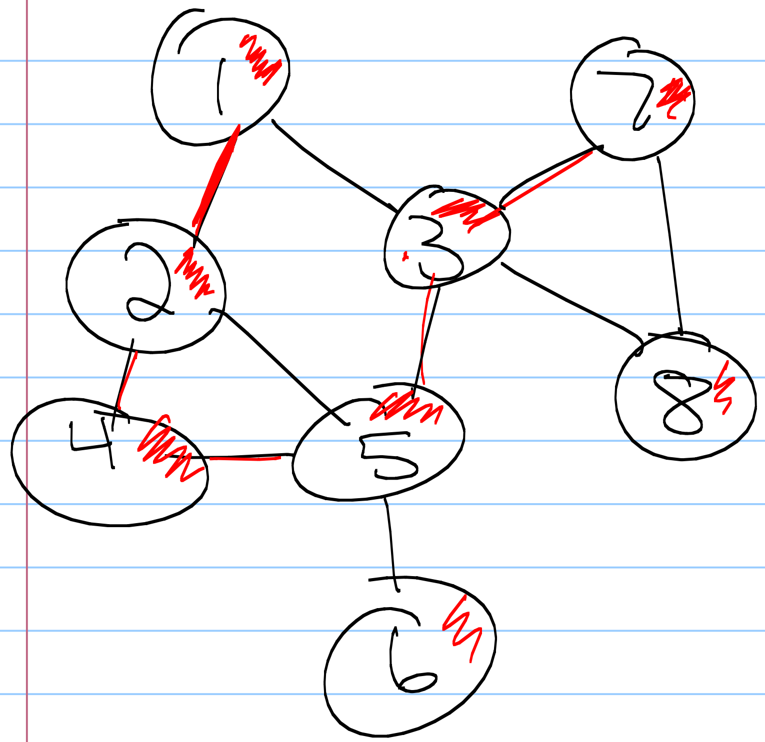
If u is unmarked:
- mark u
- for each edge $\{u,v\} \in E$
  - RecursiveDFS(v)

(depth-first search)

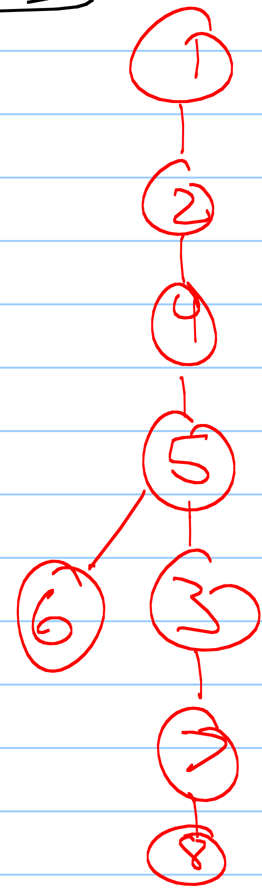
To check if s & t are connected,
Call DFS(s).

At end, if t is marked, return true

DFS "tree":



DFS (1):



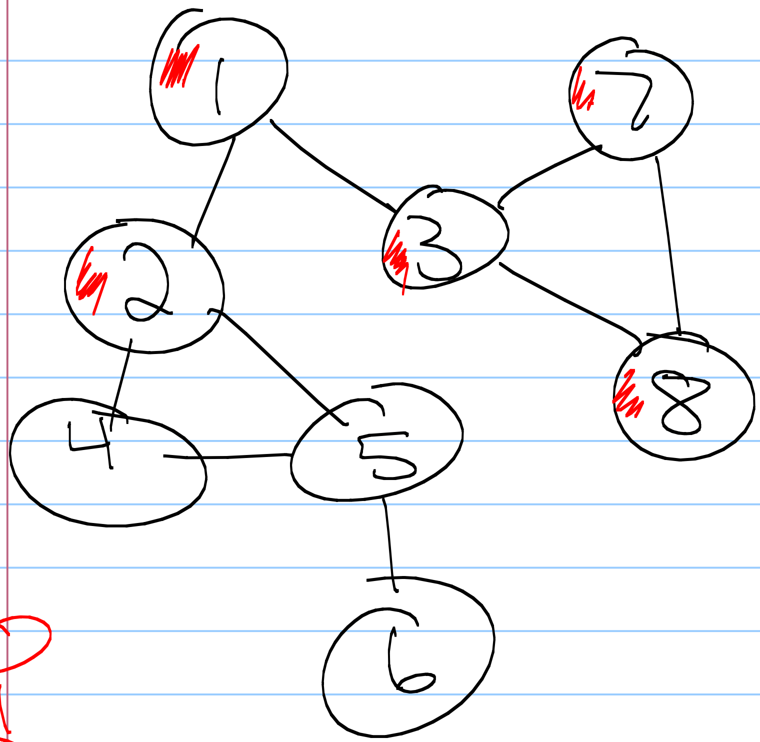
Another version of DFS: not recursive

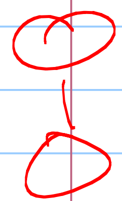
Iterative DFS(u):

```
    create empty stack S
    S.push(u)

    while S is not empty:
        v ← S.pop
        if v is not marked
            mark(v)
            for each edge vw
                S.push(w)
```

Iterative DFS (1):



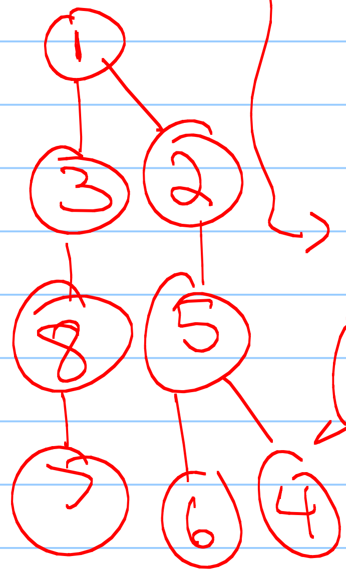


Stack :

~~1~~ (1-2) ~~(1-3)~~ ~~(3-7)~~ ~~(3-8)~~

~~(3-1)~~ ~~(8-3)~~ ~~(8-7)~~

~~(7-8)~~ ~~(7-3)~~



~~(1-2)~~ (2-1) (2-4)

~~(2-5)~~

Idea: replace stack with a queue!

Iterative BFS(u)

What happens?

$O(1) \rightarrow$ Q.push(u)

while Q is not empty:

$O(1) \longrightarrow$ $v \leftarrow$ Q.pop

$O(1) \rightarrow$ if v is not marked

$O(1) \rightarrow$ mark(v)

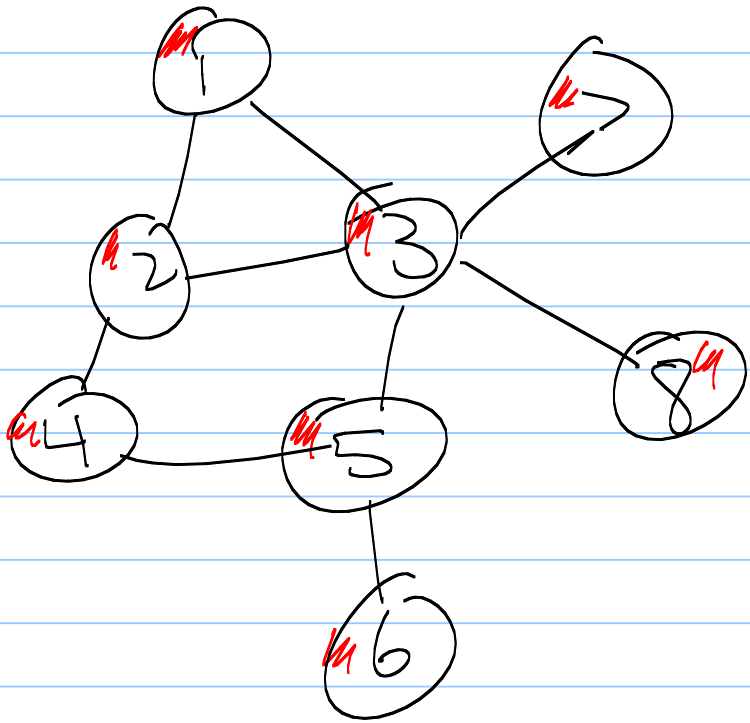
$O(d(v))$ for each edge vw

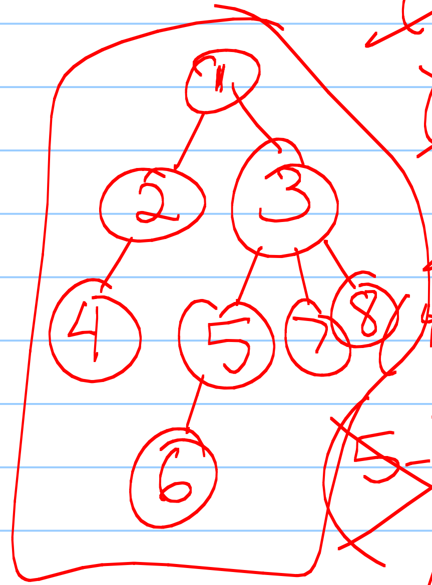
S.push(w)

$O(1)$

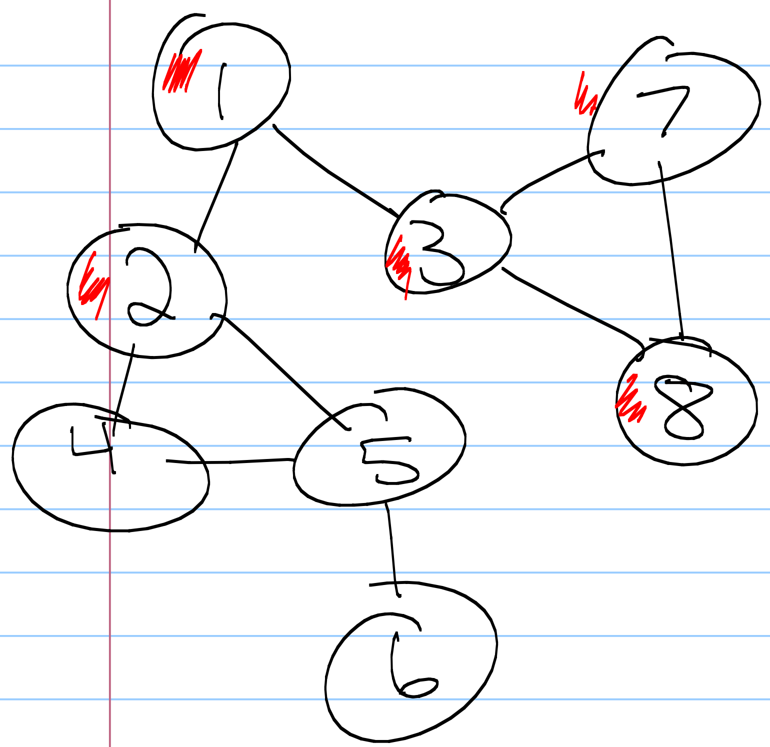
$O(d(v))$

Breadth-First Search (1):

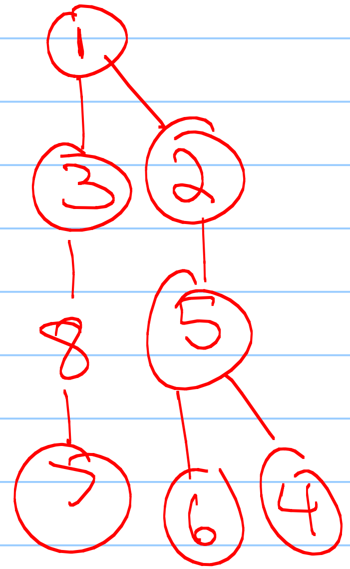




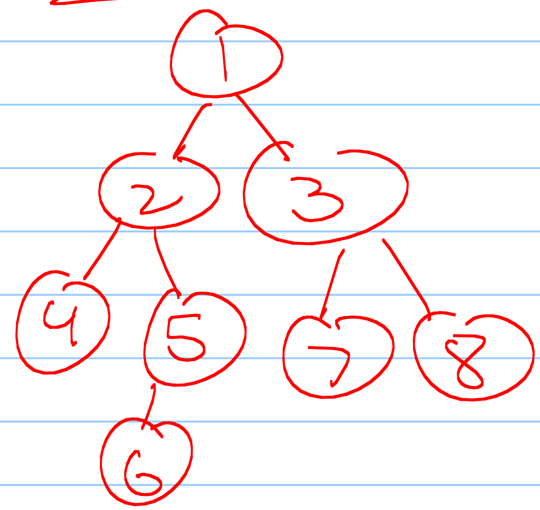
Q: ~~1~~ ~~(1-2)~~ ~~(1-3)~~ ~~(2-1)~~ ~~(2-3)~~ ~~(2-4)~~ ~~(3-1)~~ ~~(3-2)~~ ~~(3-5)~~ ~~(3-7)~~ ~~(3-8)~~ ~~(4-2)~~ ~~(4-5)~~ ~~(5-3)~~ ~~(5-4)~~ ~~(5-6)~~ ~~(7-3)~~ ~~(8-3)~~ ~~(6-5)~~



DFS:



BFS:



BFS versus DFS

- Both can tell if 2 vertices are connected
- Both can be used to detect cycles.
  How?
  Any edge in addition to the tree will create a cycle.
- Difference is structure of trees

Running Times

Each edge is put on stack/queue 2 times.

$\Rightarrow O(m)$

First time vertex is visited, spend $O(d(v))$ time. (next time(s) are $O(1)$)

$$\sum_{v \in V} d(v) = 2m = O(m)$$

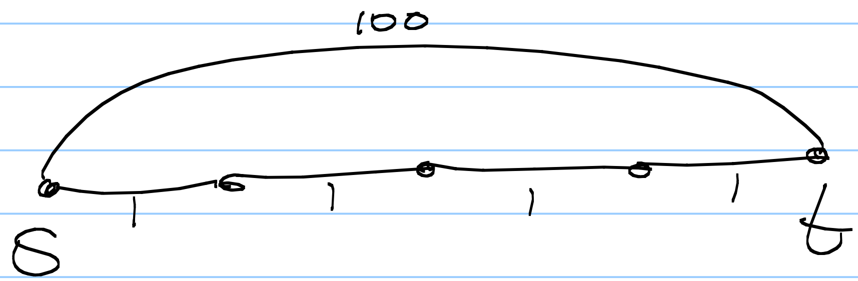
Total: $O(m+n)$

Other graph algorithms

- BFS returns a "short" s-t path, in some sense.

But won't work if graph has weights on the edges.

Why?



Which s-t path will be in BFS tree?

Shortest path trees

Given a weighted graph, find shortest path from s to t.

Uses?

Algorithms to solve this actually solve
a more general problem:
Find shortest path from s to
every other vertex.

Called the shortest path tree
rooted at s.

Can be computed in polynomial time.

Another question:

Given G, find a tree containing every vertex with minimum total weight.

Uses?

This is called the minimum spanning tree of G.

Note: Not the same as shortest path tree!