More Graph Algorithms

Note Title 12/3/2013

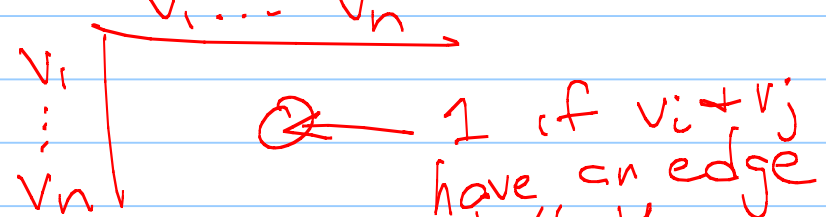
Announcements

Last time: Graphs

Representations

- Adjacency Lists: v_1: $v_2, v_5, v_7 \ldots$
  v_2: $v_1, v_5, v_6, v_9 \ldots$
  ⋮

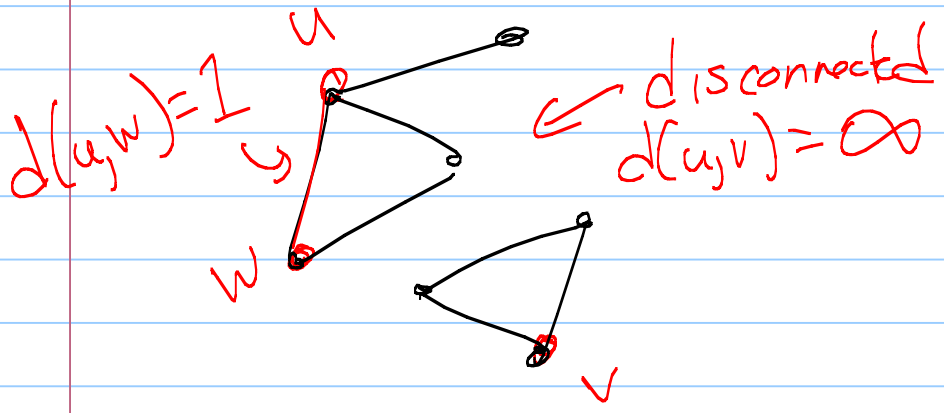
- Adjacency Matrix

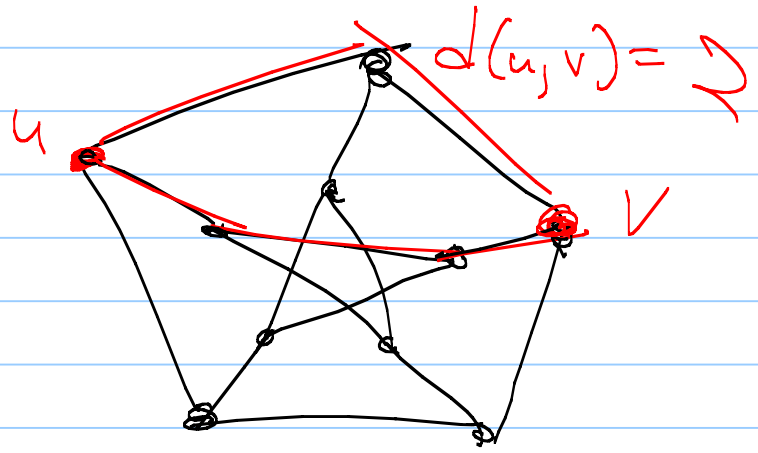


Trade off:
Space vs. time

Dfs

- G is connected if for all $u \neq v$, there is a path from u to v.
- The distance from u to v, $d(u,v)$, is equal to the length of the minimum u,v-path.





Algorithms on Graphs

Basic Question: Given 2 vertices, are they connected?

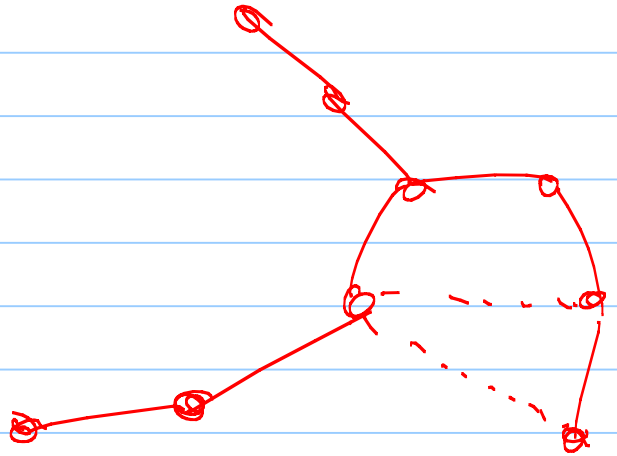
How to solve?

Search G

Suggestion:

- Suppose we're in a maze, searching for a treasure.

What do you do?



Pick a direction & go as far as you can (until dead end or repeat)

↖ depth first search

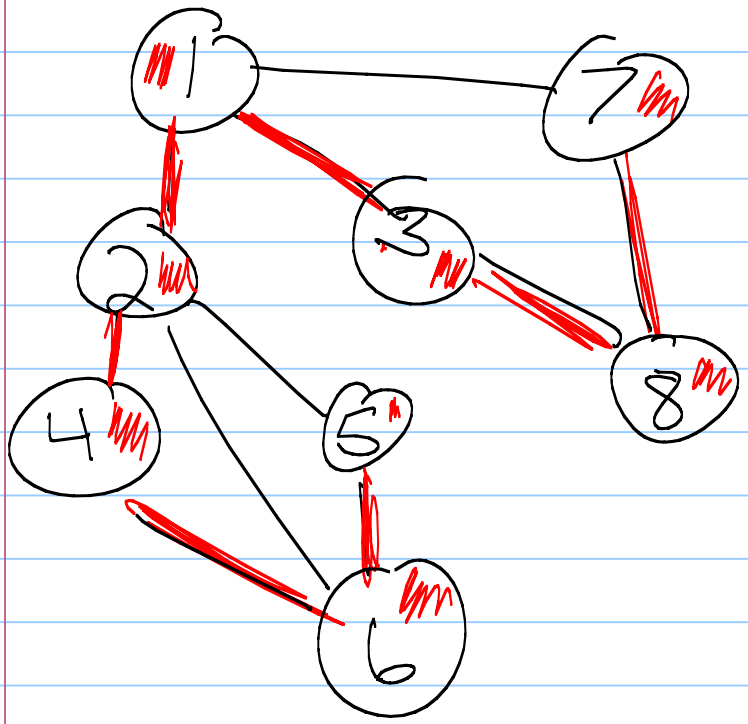
RecursiveDFS(u):

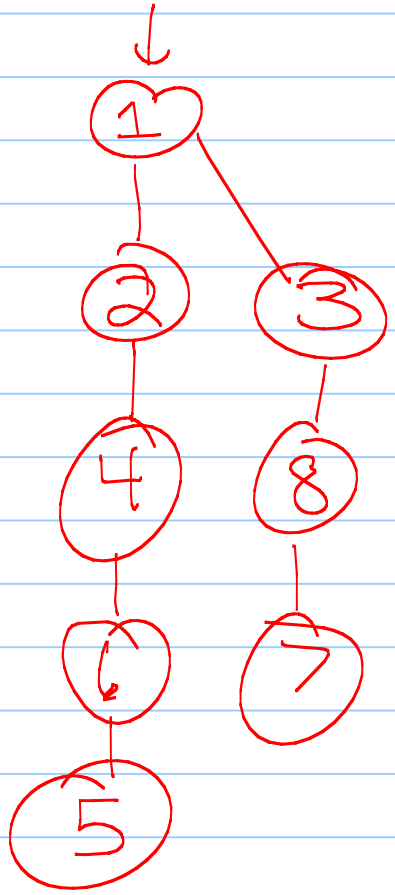
If u is unmarked:
- mark u
- for each edge $\{u, v\} \in E$
  RecursiveDFS(v)

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

At end, if t is marked, return true

DFS from 1

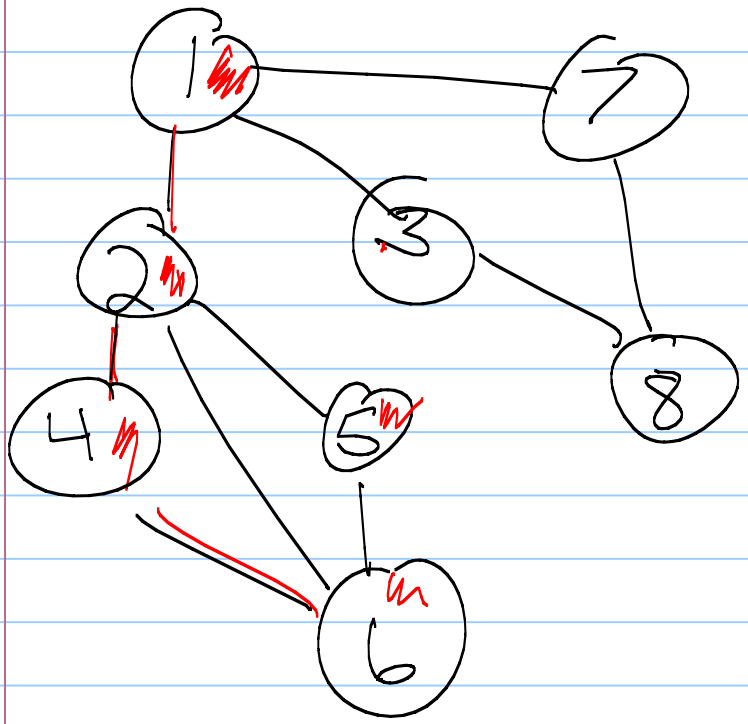


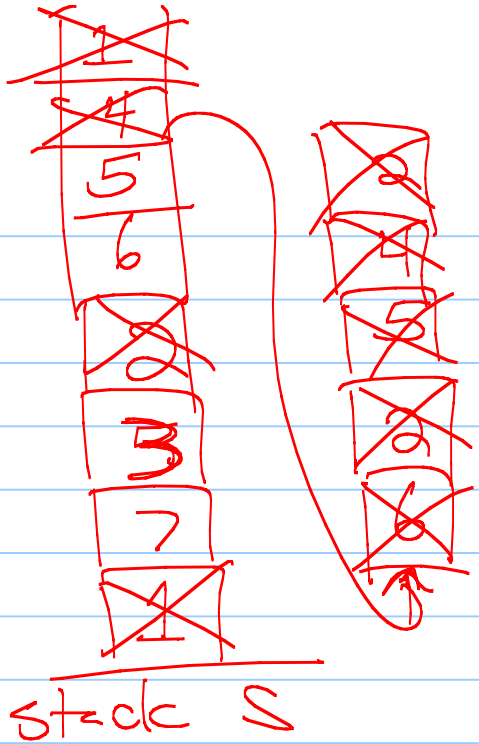


Another version of DFS

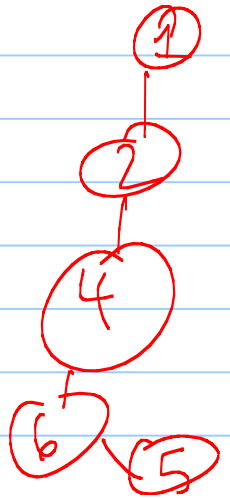
```
IterativeDFS(u):
    create empty stack S
    S.push(u)

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

DFS from 1





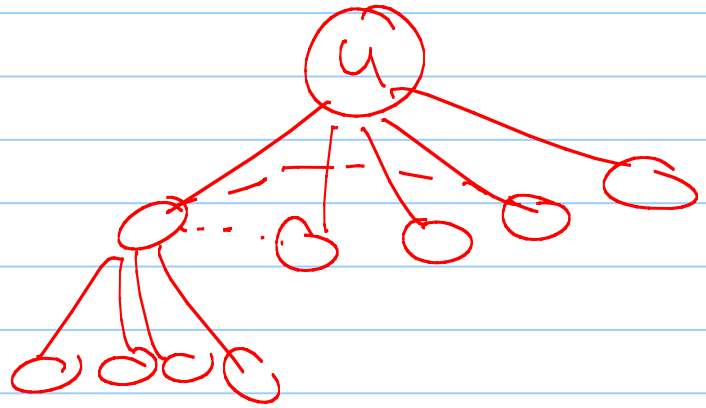
stack S



BFS:

Instead of a stack, could push all the neighbors on a queue!

So from S all of S's neighbors will connect to it.

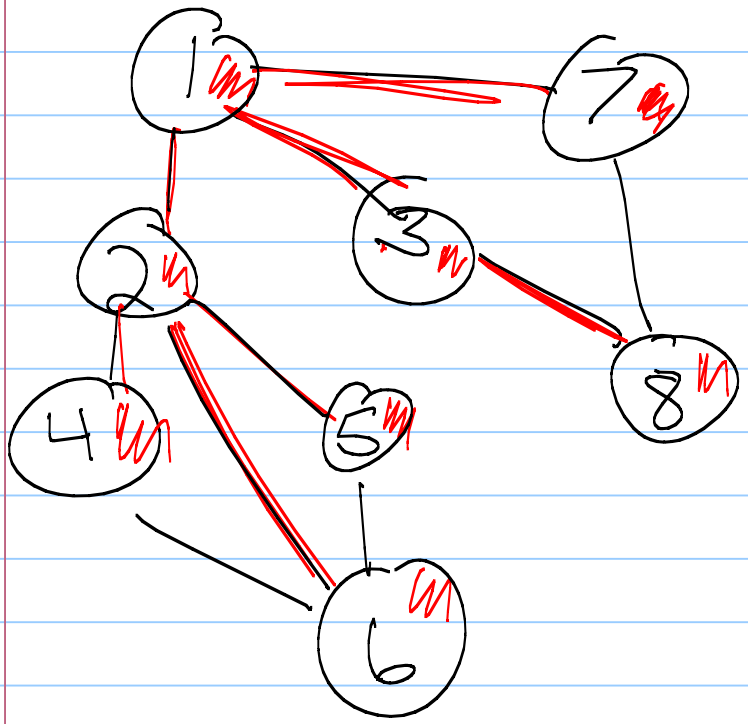


Breadth First Search

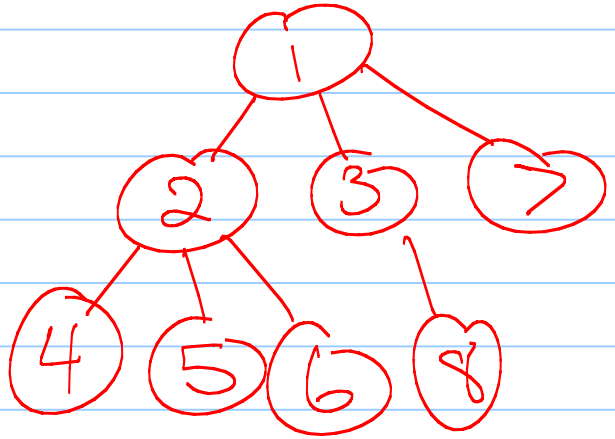
Iterative BFS(u)

```
Q.push(u)                          <-- O(1)
while Q is not empty:              <-- bound repetitions
    v ← Q.pop                      <-- O(1)
    if v is not marked:
        mark(v)
        for each edge vw           ]
            S.push(w) O(1)         ] d(v)
```

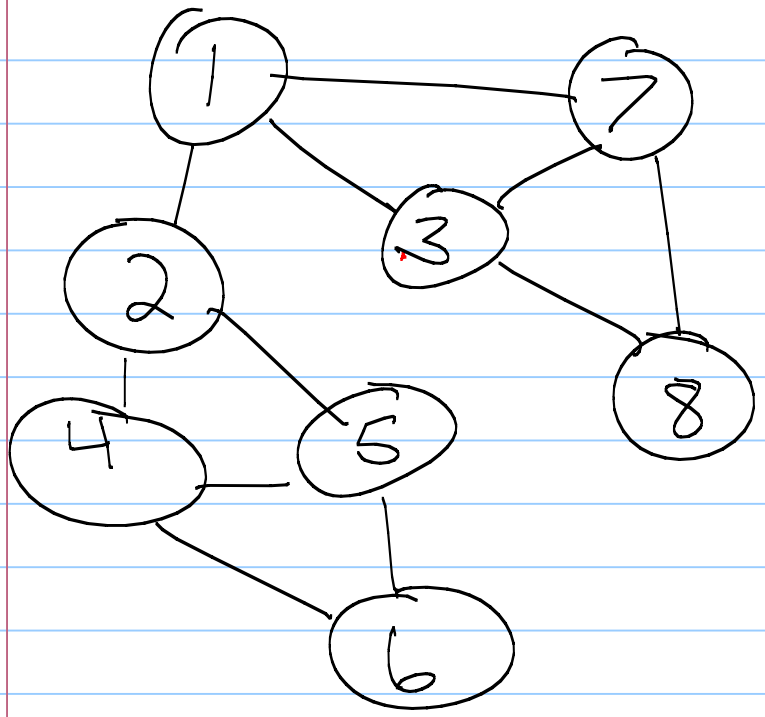
BFS



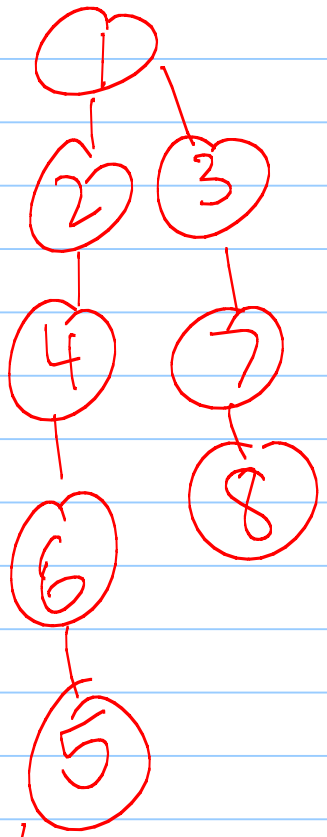
Q: 1, 2, 3, 7, 4, 5, 6, 8, 8, 8, 8, 8



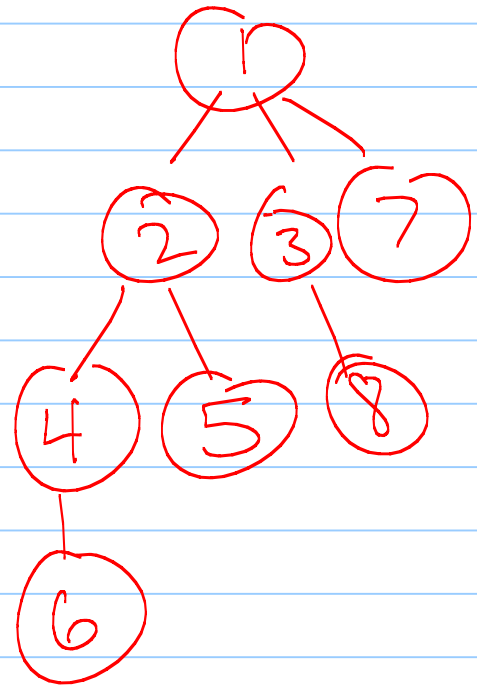
Spanning Trees:



DFS



BFS:



either one captures connectivity

BFS versus DFS

- Both can tell if 2 vertices are connected
- Both can be used to detect cycles.
  How?
  Take spanning tree. If G has any edge not in tree, G has a cycle.
- Difference is structure of trees

Runtimes:

Every vertex (v) gets pushed and popped $d(v)$ times.

(At each vertex, push on $d(v)$ other vertices)

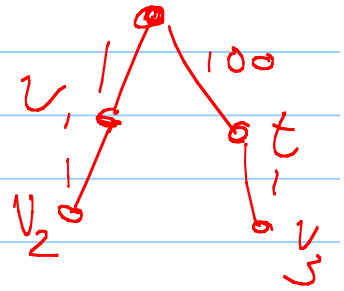
$$2\left[\sum_{v} d(v)\right] = 2[2m] = O(m)$$

$O(m+n)$

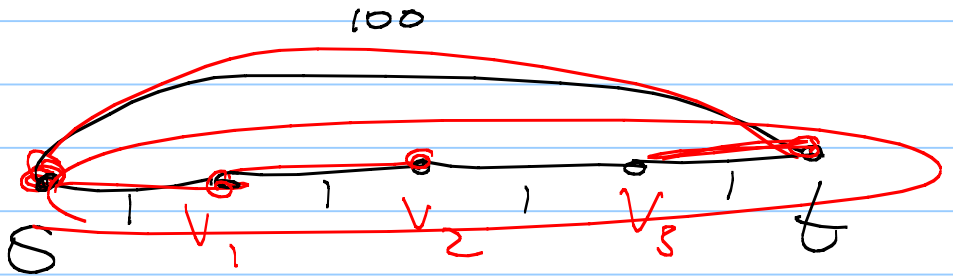
Other graph algorithms

- 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?

- Maps

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:

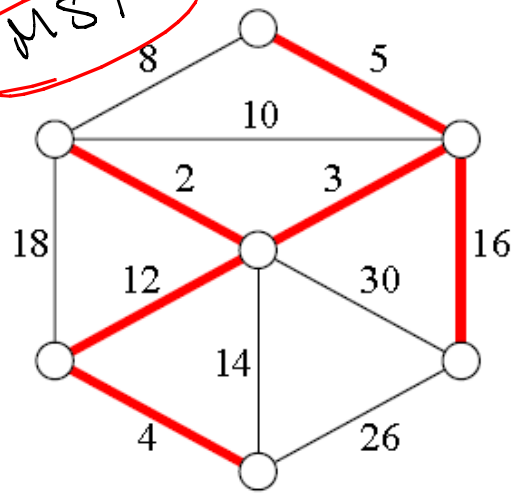
Given G, find a tree containing every vertex with minimum total weight.

Uses?

This is called the minimum spanning tree of G.

Note: Not the same as shortest path tree!

MST



SP tree