More Graph Algorithms

Announcements
Last time: Graphs

Representations

- Adjacency Lists:
  - \( V_1: V_2, V_3, V_7 \ldots \)
  - \( V_2: V_1, V_5, V_6, V_9 \ldots \)
  - \( \vdots \)
  - \( V_n \)

- Adjacency Matrix
  \[
  \begin{pmatrix}
  V_1 & \cdots & V_n \\
  \vdots & \ddots & \vdots \\
  V_n & \cdots & V_1
  \end{pmatrix}
  \]

\( a_{ij} = 1 \) if \( V_i \) and \( V_j \) have an edge between them

Trade off:

Space vs. Time
**Dos**

- G is connected if for all u and 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.

$\text{disconnected}$ $d(u, v) = \infty$
Algorithms on Graphs

Basic Question: Given 2 vertices, are they connected?

How to solve?

Search 6
Suggestion:
- Suppose we're in a maze, searching for a treasure.
  What do you do?

Pick a direction and go as far as you can (until dead end or repeat)
Depth First Search

Recursive DFS (u):

If u is unmarked:
    mark u
    for each edge (u,v) ∈ E
        Recursive DFS (v)

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

At end, if t is marked, return true.
DFS from 1
Another version of DFS

**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)
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.
Iterative BFS($u$)

$Q \cdot \text{push}(u) \quad \text{O}(1)$

while $Q$ is not empty:

$v \leftarrow Q \cdot \text{pop} \quad \text{O}(1)$

if $v$ is not marked:

mark $(v)$

for each edge $vw$:

$S \cdot \text{push}(w) \quad \text{O}(1)$

$d(v)$
BFS
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)

\[
2 \sum_{v} d(v) = 2 \lfloor 2m \rfloor = O(m)
\]

\( O(m + n) \)
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!