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: \( O(1) \)
Recursive DFS (u):

If u is unmarked:
  mark u
  for each edge (u, v) ∈ E
    RecursiveDFS(v)  
  (depth-first search)

To check if s and 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:
  $v \leftarrow S$.pop
  if $v$ is not marked
    mark $(v)$
    for each edge $vw$
      $S$.push($w$)
Iterative DFS (1):

Stack:

\[ (-2, -3, 3, -3, 2, 3) \]

\[ (-1, 2, 3, -1, 2, 4) \]
Idea: replace stack with a queue!

Iterative BFS($u$)

\[ \text{O}(1) \rightarrow Q \cdot \text{push}(u) \]

while Q is not empty:

\[ \text{O}(1) \rightarrow v \leftarrow Q \cdot \text{pop} \]

\[ \text{O}(1) \rightarrow \text{if } v \text{ is not marked} \]

\[ \text{O}(1) \rightarrow \text{mark}(v) \]

\[ \text{O}(1) \rightarrow \text{for each edge } vw \]

\[ \text{O}(d(v)) \rightarrow S \cdot \text{push}(w) \]

\[ \text{O}(d(v)) \]

What happens?
Breath-First Search(1):

0:

\[ \begin{align*}
1 & \quad 2 \quad 3 \quad 4 \quad 5 \quad 6 \\
\end{align*} \]
DFT:
```
1
 / \
2   3
 / \
4   5
 / \
6   7
```

BFS:
```
1
 / \
2   3
 / \
4   5
 / \
6   7
```
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. Note: Not the same as shortest path tree.