CSC 162
DATA STRUCTURES
Traversing graphs

Depth-First Search
  like a post-order traversal of a tree
Breadth-First Search
  Less like tree traversal
Exploring a Maze

A depth-first search (DFS) in an undirected graph G is like wandering in a maze with a string and a can of paint – you can prevent yourself from getting lost.
DFS

1. Start at vertex s
   Tie the end of the string to s and mark “visited” on s
   Make s the current vertex u

2. Travel along an arbitrary edge (u,v)
   unrolling string

3. If edge(u,v) leads to an already visited vertex v
   then return to u
   else mark v as “visited”, set v as current u, repeat @ step 2

4. When all edges lead to visited vertices, backtrack to
   previous vertex (roll up string) and repeat @ step 2

5. When we backtrack to s and explore all it’s edges we
   are done
DFS Pseudocode (labels edges)

DFS( Vertex v)
   for each edge incident on v do:
      if edge e is unexplored then
         let w be the other endpoint of e
         if vertex w is unexplored then
            label e as a discovery edge
            recursively call DFS(w)
         else
            label e as a backedge
      
   

Example
Example
Example
Example
Example
Example

A — B — C — D

E — F — G — H

I — J — K — L

M — N — O — P
Example
Example
Example
Example
Example
Example

Diagram:

- A connected to E, B, D, and J
- B connected to A, F, C, and G
- C connected to B, G, and D
- D connected to C, H, and A
- E connected to A, F, and I
- F connected to E, B, and J
- G connected to B, C, and K
- H connected to D, G, and I
- I connected to E, H, J, and O
- J connected to F, I, and K
- K connected to G, J, and L
- L connected to K, H, and N
- M connected to P, N, O, and I
- N connected to M, J, L, and O
- O connected to M, N, K, and P
- P connected to O, M, and N
Example

A --> B --> C --> D
E --> F --> G --> H
I --> J --> K --> L
M --> N --> O --> P
Example
DFS Properties

Starting at $s$

- The traversal visits all the vertices in the connected component of $s$
- The discovery edges form a spanning tree of the connected component of $s$
DFS Runtime

DFS is called on each vertex exactly once
Every edge is examined exactly twice (once from each of its vertices)
So, for $n_s$ vertices and $m_s$ edges in the connected component of the vertex $s$, the DFS runs in $O(n_s + m_s)$ if:
- The graph data structure methods take constant time
- Marking takes constant time
- There is a systematic way to examine edges (avoiding redundancy)
Marking Verticies

Extend vertex structure to support variable for marking

Use a hash table mechanism to log marked vertices
Breadth-First Search

Starting vertex has level 0 (anchor vertex)

Visit (mark) all vertices that are only one edge away

mark each vertex with its “level”

One edge away from level 0 is level 1
One edge away from level 1 is level 2
Etc. . . .
Example
Example
Example
Example
Example
Example
Example
Example
Example
Example
Example

0 1 2 3

A → B → C → D

E → F → G → H

I → J → K → L

M → N → O → P
BFS Tree

A — B — C — D

E — F — G — H

I — J — K — L

M — N — O — P
BFS Pseudocode (1 of 2)

BSF(Vertex s)
initialize container $L_0$ to contain vertex s
i $\leftarrow$ 0
while $L_i$ is not empty do
    create container $L_{i+1}$ to initially be empty
    for each vertex v in $L_i$ do
        // next slide
    i $\leftarrow$ i+1
BFS Pseudocode (2 of 2)

// for each vertex v in L_i do
if edge e incident on v do
  let w be the other endpoint of e
  if w is unexplored then
    label e as a discovery edge
    insert w into L_{i+1}
  else
    label e as a cross edge
  //i ← i+1
BSF Properties

The traversal visits all vertices in the connected component of s

The discover edges form a spanning tree of the cc

For each vertex v at level I, the path of the BSF tree T between s and v has I edges and any other path of G between s and v has at least I edges

If (u,v) is an edge that is not in the BSF tree, then the level number of u and v differ by at most one
Run Time

A BFS traversal takes $O(n+m)$ time
Also, there exist $O(n+m)$ time algorithms base on BFS which test for
  Connectivity of graph
  Spanning tree of G
  Connected component
  Minimum number of edges path between s and v