Searching Graphs
Depth-First Search

- Depth-first search (DFS) is a general technique for traversing a graph.
- A DFS traversal of a graph G:
  - Visits all the vertices and edges of G
  - Determines whether G is connected
  - Computes the connected components of G
  - Computes a spanning forest of G

DFS on a graph with \( n \) vertices and \( m \) edges takes \( O(n + m) \) time.

DFS can be further extended to solve other graph problems:
- Find and report a path between two given vertices
- Find a cycle in the graph
DFS Algorithm

- The algorithm uses a mechanism for setting and getting “labels” of vertices and edges.

Algorithm $\text{DFS}(G)$

**Input** graph $G$

**Output** labeling of the edges of $G$ as discovery edges and back edges

for all $u \in G.\text{vertices}()$

setLabel($u$, UNEXPLORED)

for all $e \in G.\text{edges}()$

setLabel($e$, UNEXPLORED)

for all $v \in G.\text{vertices}()$

if $\text{getLabel}(v) = \text{UNEXPLORED}$

$\text{DFS}(G, v)$

else

setLabel($e$, BACK)

Algorithm $\text{DFS}(G, v)$

**Input** graph $G$ and a start vertex $v$ of $G$

**Output** labeling of the edges of $G$ in the connected component of $v$ as discovery edges and back edges

setLabel($v$, VISITED)

for all $e \in G.\text{incidentEdges}(v)$

if $\text{getLabel}(e) = \text{UNEXPLORED}$

$w \leftarrow \text{opposite}(v, e)$

if $\text{getLabel}(w) = \text{UNEXPLORED}$

setLabel($e$, DISCOVERY)

$\text{DFS}(G, w)$

else

setLabel($e$, BACK)
Example

unexplored vertex
visited vertex
unexplored edge
discovery edge
back edge
Example (cont.)

Graphs
DFS and Maze Traversal

- The DFS algorithm is similar to a classic strategy for exploring a maze.
  - We mark each intersection, corner and dead end (vertex) visited.
  - We mark each corridor (edge) traversed.
  - We keep track of the path back to the entrance (start vertex) by means of a rope (recursion stack).
Properties of DFS

Property 1

DFS(G, v) visits all the vertices and edges in the connected component of v

Property 2

The discovery edges labeled by DFS(G, v) form a spanning tree of the connected component of v
Proof of Property 1

Suppose (for contradiction) that at least one vertex $s$ in $v'$'s connected component is not visited.

There is some path from $v$ to $s$ in $G$.

Let $w$ be first unvisited vertex on this path ($w$ may be $s$)

Since $w$ is first unvisited vertex, it has a neighbor $u$ that was visited.
Proof of Property 1

- But when visiting \( u \), we must have considered edge \((u,w)\), thus it cannot be correct that \( w \) is unvisited.

- Thus, there can be no unvisited vertices in the connected component of \( v \).
Proof of Property 2

- Note that we only mark edges as discovery when we go to unvisited vertices.
- Thus we can never form a cycle with discovery edges.
- I.e. the discovery edges form a tree.
- This is a spanning tree because we visit each vertex in the connected component of $v$. 

Spring 2014 Graphs 10
Analysis of DFS

- Setting/getting a vertex/edge label takes $O(1)$ time
- Each vertex is labeled twice
  - once as UNEXPLORED
  - once as VISITED
- Each edge is labeled twice
  - once as UNEXPLORED
  - once as DISCOVERY or BACK
- Method incidentEdges is called once for each vertex
- DFS runs in $O(n + m)$ time provided the graph is represented by the adjacency list structure
  - Recall that $\sum_v \deg(v) = 2m$
Path Finding

- We can specialize the DFS algorithm to find a path between two given vertices $u$ and $z$
- We call $DFS(G, u)$ with $u$ as the start vertex
- We use a stack $S$ to keep track of the path between the start vertex and the current vertex
- As soon as destination vertex $z$ is encountered, we return the path as the contents of the stack

Algorithm $\text{pathDFS}(G, v, z)$

```plaintext
setLabel(v, VISITED)
S.push(v)
if $v = z$
    return $S.elements()$
for all $e \in G.\text{incidentEdges}(v)$
    if $\text{getLabel}(e) = \text{UNEXPLORED}$
        $w \leftarrow \text{opposite}(v, e)$
        if $\text{getLabel}(w) = \text{UNEXPLORED}$
            $\text{setLabel}(e, \text{DISCOVERY})$
            $S.push(e)$
            $\text{pathDFS}(G, w, z)$
            $S.pop(e)$
        else
            $\text{setLabel}(e, \text{BACK})$
            $S.pop(v)$
```
Cycle Finding

- We can specialize the DFS algorithm to find a simple cycle.
- We use a stack $S$ to keep track of the path between the start vertex and the current vertex.
- As soon as a back edge $(v, w)$ is encountered, we return the cycle as the portion of the stack from the top to vertex $w$.

Algorithm `cycleDFS(G, v, z)`

```
setLabel(v, VISITED)
S.push(v)
for all $e \in G.incidentEdges(v)$
    if `getLabel(e) = UNEXPLORED`
        $w \leftarrow $ opposite($v, e$)
        S.push($e$)
        if `getLabel(w) = UNEXPLORED`
            `setLabel(e, DISCOVERY)`
            `pathDFS(G, w, z)`
            S.pop($e$)
        else
            $T \leftarrow $ new empty stack
            repeat
                $o \leftarrow $ S.pop()
                $T.push(o)$
            until $o = w$
            return $T.elements()$
    else
        S.pop($v$)
```

Breadth-First Search
Breadth-First Search

- Breadth-first search (BFS) is a general technique for traversing a graph
- A BFS traversal of a graph G
  - Visits all the vertices and edges of G
  - Determines whether G is connected
  - Computes the connected components of G
  - Computes a spanning forest of G

- BFS on a graph with $n$ vertices and $m$ edges takes $O(n + m)$ time
- BFS can be further extended to solve other graph problems
  - Find and report a path with the minimum number of edges between two given vertices
  - Find a simple cycle, if there is one
BFS Algorithm

- The algorithm uses a mechanism for setting and getting “labels” of vertices and edges.

Algorithm \textit{BFS}(G)

\textbf{Input} graph \( G \)

\textbf{Output} labeling of the edges and partition of the vertices of \( G \)

For all \( u \in G.\text{vertices}() \)

\texttt{setLabel}(u, \text{UNEXPLORED})

For all \( e \in G.\text{edges}() \)

\texttt{setLabel}(e, \text{UNEXPLORED})

For all \( v \in G.\text{vertices}() \)

If \( \text{getLabel}(v) = \text{UNEXPLORED} \)

\( \text{BFS}(G, v) \)

Algorithm \textit{BFS}(G, s)

\( L_0 \leftarrow \text{new empty sequence} \)

\( L_0.\text{insertLast}(s) \)

\texttt{setLabel}(s, \text{VISITED})

\( i \leftarrow 0 \)

While \( \neg L_i.\text{isEmpty}() \)

\( L_{i+1} \leftarrow \text{new empty sequence} \)

For all \( v \in L_i.\text{elements}() \)

For all \( e \in G.\text{incidentEdges}(v) \)

If \( \text{getLabel}(e) = \text{UNEXPLORED} \)

\( w \leftarrow \text{opposite}(v,e) \)

If \( \text{getLabel}(w) = \text{UNEXPLORED} \)

\texttt{setLabel}(e, \text{DISCOVERY})

\texttt{setLabel}(w, \text{VISITED})

\( L_{i+1}.\text{insertLast}(w) \)

Else

\texttt{setLabel}(e, \text{CROSS})

\( i \leftarrow i + 1 \)
Example

unexplored vertex
visited vertex
unexplored edge
discovery edge
cross edge
Example (cont.)

Spring 2014

Graphs
Example (cont.)
Properties

Notation

$G_s$: connected component of $s$

Property 1

$\text{BFS}(G, s)$ visits all the vertices and edges of $G_s$

Property 2

The discovery edges labeled by $\text{BFS}(G, s)$ form a spanning tree $T_s$ of $G_s$

Property 3

For each vertex $v$ in $L_i$

- The path of $T_s$ from $s$ to $v$ has $i$ edges
- Every path from $s$ to $v$ in $G_s$ has at least $i$ edges (i.e. the path above is the shortest possible)
Analysis

- Setting/getting a vertex/edge label takes $O(1)$ time
- Each vertex is labeled twice
  - once as UNEXPLORED
  - once as VISITED
- Each edge is labeled twice
  - once as UNEXPLORED
  - once as DISCOVERY or CROSS
- Each vertex is inserted once into a sequence $L_i$
- Method incidentEdges is called once for each vertex
- BFS runs in $O(n + m)$ time provided the graph is represented by the adjacency list structure
  - Recall that $\sum_v \deg(v) = 2m$
Applications

- We can specialize the BFS traversal of a graph $G$ to solve the following problems in $O(n + m)$ time
  - Compute the connected components of $G$
  - Compute a spanning forest of $G$
  - Find a simple cycle in $G$, or report that $G$ is a forest
  - Given two vertices of $G$, find a path in $G$ between them with the minimum number of edges, or report that no such path exists
DFS vs. BFS

<table>
<thead>
<tr>
<th>Applications</th>
<th>DFS</th>
<th>BFS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Spanning forest, connected components, paths, cycles</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Shortest paths</td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>Biconnected components</td>
<td>✓</td>
<td></td>
</tr>
</tbody>
</table>

Applications for DFS and BFS:
- DFS applications: Spanning forest, connected components, paths, cycles
- BFS applications: Shortest paths, Biconnected components
DFS vs. BFS (cont.)

### Back edge \((v, w)\)
- \(w\) is an ancestor of \(v\) in the tree of discovery edges

---

### Cross edge \((v, w)\)
- \(w\) is in the same level as \(v\) or in the next level in the tree of discovery edges

---

**DFS**

**BFS**