Lecture 12

topological sort, BFS

CS 161 Design and Analysis of Algorithms
Ioannis Panageas
DFS Algorithm from a Vertex

Algorithm DFS(G, v):

Input: A graph G and a vertex v in G
Output: A labeling of the edges in the connected component of v as discovery edges and back edges, and the vertices in the connected component of v as explored

Label v as explored
for each edge, e, that is incident to v in G do
  if e is unexplored then
    Let w be the end vertex of e opposite from v
    if w is unexplored then
      Label e as a discovery edge
      DFS(G, w)
    else
      Label e as a back edge
Example

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

\[ \text{DFS}(G, v) \] visits all the vertices and edges in the connected component of \( v \)

Property 2

The discovery edges labeled by \( \text{DFS}(G, v) \) form a spanning tree of the connected component of \( v \)
The General DFS Algorithm

- Perform a DFS from each unexplored vertex:

```
Algorithm DFS(G):
    Input: A graph G
    Output: A labeling of the vertices in each connected component of G as explored
    Initially label each vertex in v as unexplored
    for each vertex, v, in G do
        if v is unexplored then
            DFS(G, v)
```
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 \(\sum_v \deg(v) = 2m\)
Cycle detection

• Graph G has a cycle iff DFS has a back edge

Directed Acyclic Graph = DAG
Topological sort

Topological sort of a DAG $G=(V,E)$

1. Run DFS($G$), compute finishing times of nodes
2. Output the nodes in decreasing order of finishing times
The Graph – relationship between clothing procedures

The Topological sort – a workable sequence of clothing
This is a discov. edge
s < a < b < c
This is a back edge. We don’t follow it because the grey node b is on the stack.
A back edge connects from a grey node to another grey node.
This is a *forward edge*. We don’t follow it because *d* is coloured black.
A forward edge connects a grey node to a black node.
This is a cross edge. It connects between two different subtrees.
Both cross edges and forward edges connect from a grey node to a black one.
This is the DFS tree.
We sort the elements based on their finish times.
1. DFS WITH STACK
DEPTH FIRST SEARCH

Stack Status
DEPTH FIRST SEARCH

Stack Status
DEPTH FIRST SEARCH

Stack Status

A

B

D

C

E

F

G

H

S
DEPTH FIRST SEARCH

Stack Status
DEPTH FIRST SEARCH

Stack Status

A - S - B - C - G - F - H - E - D
DEPTH FIRST SEARCH

Stack Status

A B C D E F G H S
DEPTH FIRST SEARCH

Stack Status

A B C D E F G H

D C S A
DEPTH FIRST SEARCH

Stack Status

A -> B -> C -> D
S -> C -> F
G -> H

CSA
DEPTH FIRST SEARCH

Stack Status

A B C D E S F G H E C S A
DEPTH FIRST SEARCH

Stack Status

A B C D E F G H S

HECSA
DEPTH FIRST SEARCH

Stack Status

A B C D E F G H

G H E C S A
DEPTH FIRST SEARCH

Stack Status

F G H E C S A
DEPTH FIRST SEARCH

Stack Status

A B C D E F G H S F G HE CSA
DEPTH FIRST SEARCH

Stack Status

A B C D
S E G F
G H
DEPTH FIRST SEARCH

Stack Status

A → B → D → C → F → G → H → E

ECSA
DEPTH FIRST SEARCH

Stack Status

A B C D
S C F E
G F H

C S A
DEPTH FIRST SEARCH

Stack Status

A → B → C → D → E → F → G → H

S A
DEPTH FIRST SEARCH

Stack Status
DEPTH FIRST SEARCH

Stack Status
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 “levels” $L_i$ and a mechanism for setting and getting “labels” of vertices and edges.

```
Algorithm BFS($G, s$):
    \textbf{Input:} A graph $G$ and a vertex $s$ of $G$
    \textbf{Output:} A labeling of the edges in the connected component of $s$ as discovery edges and cross edges
    Create an empty list, $L_0$
    Mark $s$ as explored and insert $s$ into $L_0$
    $i \leftarrow 0$
    \textbf{while} $L_i$ is not empty \textbf{do}
        create an empty list, $L_{i+1}$
        \textbf{for} each vertex, $v$, in $L_i$ \textbf{do}
            \textbf{for} each edge, $e = (v, w)$, incident on $v$ in $G$ \textbf{do}
                \textbf{if} edge $e$ is unexplored \textbf{then}
                    \textbf{if} vertex $w$ is unexplored \textbf{then}
                        Label $e$ as a discovery edge
                        Mark $w$ as explored and insert $w$ into $L_{i+1}$
                    \textbf{else}
                        Label $e$ as a cross edge
                $i \leftarrow i + 1$
```
**Example**

- **unexplored vertex**
- **visited vertex**
- **unexplored edge**
- **discovery edge**
- **cross edge**

> L_0

> L_1

© 2015 Goodrich and Tamassia
Example (cont.)
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
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 use the BFS traversal algorithm, for 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
- **DFS**: Spanning forest, connected components, paths, cycles, Biconnected components
- **BFS**: Spanning forest, connected components, paths, cycles, Biconnected components

### Diagrams
- **DFS**
- **BFS**
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

DFS

BFS
2. BFS WITH QUEUE
BREADTH FIRST SEARCH

Queue Status
BREADTH FIRST SEARCH

Queue Status

B S
BREADTH FIRST SEARCH

Queue Status
BREADTH FIRST SEARCH

Queue Status

C
BREADTH FIRST SEARCH

Queue Status

G
BREADTH FIRST SEARCH

Queue Status

A B C D E
G S C F H

G D E
BREADTH FIRST SEARCH

Queue Status

A, B, C, D, E, F, G, H
BREADTH FIRST SEARCH

Queue Status

A  B  C  D  E  F  G  H

S  D  F  H
BREADTH FIRST SEARCH

Queue Status

E  F  H
BREADTH FIRST SEARCH
BREADTH FIRST SEARCH

Queue Status

H

Diagram showing a graph with vertices A, B, C, D, E, F, G, H, S, and connections between them.