Graph Representations

$G.V =$ vertices of graph $G$

$G.E =$ edges of graph $G$

$G.Adj[u] =$ all vertices $v$ where $(u, v) \in G.E$

A graph can be represented as an adjacency list or adjacency matrix.

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

(a) (b) (c)

Representations Continued

How efficient are these representations?

Space?

Determining if $(u, v)$ is an edge in the graph?

Visiting each edge once?
**Breadth-First Search**

BFS is $O(V + E)$. 
BFS$(G, s)$ finds shortest paths from $s$.

Algorithm notes:
Search starts from vertex $s$.
WHITE, GRAY, BLACK respectively means undiscovered, discovered, finished.
$v.d =$ distance from $s$
$v.\pi =$ previous vertex on path from $s$ to $v$

---

**Breadth-First Search Algorithm**

**BFS**($G, s$)

for each vertex $v \in G.V$

$v.\text{color} \leftarrow \text{WHITE}$, $v.d \leftarrow \infty$, $v.\pi \leftarrow \text{NIL}$
$s.\text{color} \leftarrow \text{GRAY}$, $s.d \leftarrow 0$, $Q \leftarrow \{s\}$

while $Q \neq \emptyset$

$u \leftarrow \text{DEQUEUE}(Q)$

for each $v \in G.\text{Adj}[u]$

if $v.\text{color} = \text{WHITE}$

then $\text{ENQUEUE}(Q, v)$

$v.\text{color} \leftarrow \text{GRAY}$
$v.d \leftarrow u.d + 1$
$v.\pi \leftarrow u$

$u.\text{color} \leftarrow \text{BLACK}$

---

**Depth-First Search**

DFS is $O(V + E)$.
$v.d =$ discovery "time"
$v.f =$ finishing "time"
v.d and v.f form a parenthesis structure.

**DFS**($G$)

for each vertex $u \in G.V$

$u.\text{color} \leftarrow \text{WHITE}$
$u.\pi \leftarrow \text{NIL}$
time $\leftarrow 0$
for each vertex $u \in G.V$

if $u.\text{color} = \text{WHITE}$

then time $\leftarrow \text{DFS-VISIT}(u, \text{time} + 1)$
DFS Algorithm Continued

DFS-Visit\( (u, \text{time}) \)
\begin{align*}
  u.\text{color} &\leftarrow \text{GRAY} \\
  u.d &\leftarrow \text{time} \\
  \text{for each vertex } v \in G.\text{Adj}[u] \\
  \quad \text{if } v.\text{color} = \text{WHITE} \\
  \quad \quad v.\pi &\leftarrow u \\
  \quad \quad \text{time} &\leftarrow \text{DFS-Visit}(v, \text{time} + 1) \\
  u.\text{color} &\leftarrow \text{BLACK} \\
  \text{return } u.f &\leftarrow \text{time} + 1
\end{align*}

Topological Sort

A topological sort of a directed acyclic graph (dag) orders the vertices. \( u \prec v \) if \( (u, v) \) is an edge.

\text{TOPOLOGICAL-SORT}(G)
Call \text{DFS}(G).
As each vertex is finished, insert the vertex onto the front a linked list.

Topological Sort Analysis

\text{TOPOLOGICAL-SORT} is \( \Theta(V + E) \).
Correct because:
If there is a path from \( u \) to \( v \), then either:

Case 1: \text{DFS-Visit}(u) before \text{DFS-Visit}(v).
\( u.d < v.d < v.f < u.f \) because \text{DFS-Visit}(u) discovers \( v \).

Case 2: \text{DFS-Visit}(v) before \text{DFS-Visit}(u).
\( v.d < v.f < u.d < u.f \) because \text{DFS-Visit}(v) doesn’t discover \( u \).
Strongly Connected Components

A strongly connected component of a directed graph is a maximal set of vertices where every vertex can reach the others.

**STRONGLY-CONNECTED-COMPONENTS(G)**
- Call DFS(G) saving finishing times u.f.
- Compute \( G^T \), i.e., reverse all the edges.
- Call DFS(\( G^T \)), loop order is decreasing u.f.
- Each DFS(\( G^T \)) tree is a SCC.

SCC Example

Let \( C_1 \) and \( C_2 \) be two SCCs in a graph \( G \).
Suppose path from \( C_1 \) to \( C_2 \).
Suppose \( u \) is the first vertex discovered in \( C_1 \).
\( u \)'s finish time will be after other vertices in \( C_1 \).

Case 1: \( C_1 \) is discovered before \( C_2 \).
DFS will explore \( C_2 \) before \( u \) is finished.

Case 2: \( C_2 \) is discovered before \( C_1 \).
DFS will explore \( C_2 \) before \( u \) is discovered.

In both cases, \( u \)'s finish time will be after all vertices in \( C_2 \). When \( G^T \) is explored, \( u \) (and \( C_1 \)) will be discovered and finished before \( C_2 \).