Depth First Search

**Goal:** Visit all vertices of the graph

**Idea:**
- start at $s$
- keep going “deeper” into the graph whenever possible
Depth First Search

Graph-DFS(G)
1  unmark all vertices
2  for each v ∈ V
3   if v not marked
4    DFS(G,v)

DFS(G,s)
1  mark s
2  for each v ∈ Adj[s]
3   if v not marked yet
4    DFS(G,v)
5    DFS-Tree ← DFS-Tree U {(s,v)}
Data Structure for DFS

Last In First Out (LIFO) Stack

push  pop
Example

Graph-DFS(G)
  | DFS(G, s)
  |   | DFS(G, r)
  |   |   | DFS(G, v)
  |   |   | DFS(G, w)
  |   |   | DFS(G, t)
  |   |   |   | DFS(G, u)
  |   |   |   | DFS(G, y)
  |   |   |   | DFS(G, x)
Time Stamping

**Graph-DFS(G)**

1. \( \text{time} \leftarrow 0 \)
2. for each \( v \) in \( V \) : \( d[v] \leftarrow \text{unseen} \)
3. for each \( v \) in \( V \)
4. \( \text{if } d[v] = \text{unseen} \)
5. \( \text{DFS}(G, v) \)

**DFS(G,s)**

1. \( d[s] \leftarrow \text{time} \leftarrow \text{time+1} \) \( \text{discovery time} \)
2. visit(s)
3. for each \( v \) in \( \text{Adj}[s] \)
4. \( \text{if } d[v] = \text{unseen} \)
5. \( \text{DFS}(G, v) \)
6. \( f[s] \leftarrow \text{time} \leftarrow \text{time+1} \) \( \text{finishing time} \)
Example

```
8/1  7/2  12/9
s--->r----t
5/4  6/3  11/10
v--->w--->x

Intro to Data Structures and Algorithms ©
Depth-first search, slide 6
### BFS vs. DFS

**BFS(G)**

1. list \( L \leftarrow \phi \)
2. tree \( T \leftarrow \phi \)
3. choose start vertex \( s \)
4. \( \text{search}(s) \)
5. while \( L \neq \phi \)
6. remove \((v,w)\) from \( \text{front} \) of \( L \)
7. if \( w \) not yet visited
8. \( T \leftarrow T \cup \{(v,w)\} \)
9. \( \text{search}(w) \)

**DFS(G)**

1. list \( L \leftarrow \phi \)
2. tree \( T \leftarrow \phi \)
3. choose start vertex \( s \)
4. \( \text{search}(s) \)
5. while \( L \neq \phi \)
6. remove \((v,w)\) from \( \text{back} \) of \( L \)
7. if \( w \) not yet visited
8. \( T \leftarrow T \cup \{(v,w)\} \)
9. \( \text{search}(w) \)

**Search(v)**

1. visit(v)
2. for each edge \((v,w)\)
3. add \((v,w)\) to \text{back} of \( L \)
Classification of Edges in DFS

Directed Graph

- **Tree edges** → in the DFS forest
- **Back edges** ← from vertex to an ancestor
- **Forward edges** → from a vertex to a descendant
- **Cross edges** → all other edges
Classification of Edges
Undirected Graph

tree edge

back edge

back edge
DFS - Running Time

Adjacency list representation

$\Theta(V)$ – visit each vertex

$\Theta(E)$ – check all edges, going out from each node

=>$\textbf{Total: } \Theta(V+E) = \Theta(\max\{V,E\})$
Applications of DFS

In $\Theta(V+E)$ time, we can:

- Find connected components of $G$
- determine if $G$ has a cycle
- determine if removing an edge / vertex disconnects $G$
- determine if $G$ is planar
Cycles in Directed Graphs

Theorem: DiGraph G has a cycle

↔ DFS forest has a back edge

Proof:
back edge => cycle: obvious
cycle => back edge:

\[ d[u] < d[v] \]
Topological Sort

Topological Sort:

an ordering "<" of \( V[G] \) such that \( (u, v) \in E[G] \Rightarrow u < v \)
Topological Sort

**Lemma:**
G can be topologically sorted $\iff$ G is a DAG

**Proof:**
cycle $\Rightarrow$ G can't be sorted - obvious:

no cycle $\Rightarrow$ G can be sorted:
we will show an algorithm
A Simple TopSort Algorithm

**Source** = a vertex with indegree = 0

```
Simple-Topological-Sort(G)
1  k = 0
2  while ∃s∈V s.t. s is a source
3    N[s] ← k
4    k ← k + 1
5    remove s & all its outgoing edges
6  if  k = n then successful
7    else  failed
```
A Simple TopSort Algorithm

Source = a vertex with indegree = 0

Simple-Topological-Sort(G)
1 $k = 0$
2 while $\exists s \in V$ s.t. $s$ is a source
3 $\text{Sorted}[k] \leftarrow s$
4 $k \leftarrow k + 1$
5 remove $s$ & all its outgoing edges
6 if $k = n$ then successful
7 else failed
Adjacency Matrix Implementation

Running Time:
• $O(V)$ - look at column $v$ to see if $v$ is a source
  => $O(V^2)$ - to find a source
• $O(V)$ - to remove source & update Adj matrix
  => Total: $O(V^3)$

Improvement:
• keep $\text{InDeg}[v] = \#$ incoming edges at $v$
  => Total: $O(V^2)$
Adjacency List Implementation

Running Time:
- $\Theta(E)$ - look at all adjacency lists
to see if $v$ is a source *(why not $\Theta(V)$?)*
  $\Rightarrow \Theta(EV)$ - to find a source
- $\Theta(1)$ - to remove source
  $\Rightarrow$ **Total:** $\Theta(EV^2)$

Improvement:
- keep $\text{InDeg}[v] = \# \text{ incoming edges at } v$
  $\Rightarrow$ **Total:** $\Theta(V^2+E)$
Topological-Sort

Topological-Sort(G)
1. for all v in V
2. unmark v
3. for each vertex v in V
4. if v is unmarked TopSort(G,v)

TopSort(G,s)
1. visit(s)
2. for each v in Adj[s]
3. if v not marked
4. TopSort(G,v)
5. add s to front of TopSort list
Strongly Connected Components

$G$ strongly connected

$\forall u, v \in E \ \exists u \sim v \text{ and } \exists v \sim u$
Simple SCC Algorithms

• Transitive Closure
  \Rightarrow \text{ matrix multiplication}

  \textbf{Running Time: } \Theta(|V|^3)

• DFS from every u, v

  \textbf{Running Time: } \Theta(|V| \times |E|) = \Theta(|V|^3)
Transpose of a DiGraph

\[ G = (V, E) \]

\[ G^T = (V, E^T) \]

\[ E^T = \{ (v, u) \mid (u, v) \in E \} \]
SCC’s via DFS

Strongly-Connected-Components(G)
1 Graph-DFS(G) to compute $f[v]$ (finishing times) for all $v \in V$
2 Compute $G^T$
3 Graph-DFS($G^T$), with vertices chosen in decreasing $f[v]$ order

Theorem:
Each tree in the DFS-forest of line 3 is a SCC of $G$
SCC’s – Running Time

Adjacency List Representation

$\Theta (V+E)$ – Graph-DFS ($G$)

$\Theta (V+E)$ – Compute $G^T$ from $G$

$\Theta (V+E)$ – Graph-DFS ($G^T$)

$\Rightarrow$ **Total:** $\Theta (V+E)$