Digraphs: Depth First Search

Given $G = (V, E)$ and all $v$ in $V$ are marked unvisited, a depth-first search (dfs) (generalisation of a pre-order traversal of tree) is one way of navigating through the graph

- select one $v$ in $V$ and mark as visited
- select each unvisited vertex $w$ adjacent to $v$ - dfs($w$) (recursive!)
- if all vertices marked => search complete
- otherwise select an unmarked node and apply dfs

Implementation: adjacency list
DFS: Example

Start: A
A, B, C, D, E, F, G
in a dfs of a directed graph, certain edges, when visited, lead to unvisited vertices such edges are called TREE EDGES and form a DEPTH FIRST SPANNING FOREST for the given digraph
Depth First Spanning Forest

- Other edges are
  - back edge
    - vertex to an ancestor
  - forward edge
    - non-tree edge from a vertex to a proper descendant (in the tree)
  - cross edge
    - edge from $V_1$ to $V_2$ - neither an ancestor nor descendant
Depth First Spanning Forest

- Nota Bene (NB)
  - all cross edges go from right to left assuming that
  - children added to tree in order visited (l to r)
  - new trees added to forest in left to right order
- vertices can be numbered (dfn) in depth first order
  A B C D E F G
  1 2 3 4 5 6 7
Depth First Spanning Forest

- **All descendants** of $v$ have $\text{dfn} \geq \text{dfn}(v)$
- **forward edges**
  low $\text{dfn}$ to high $\text{dfn}$
- **back edges**
  high $\text{dfn}$ to low $\text{dfn}$
- **cross edges**
  high $\text{dfn}$ to low $\text{dfn}$
- **back edge** $\implies$ cycle

- $w$ is a descendant of $v$ iff
  \[ \text{dfn}(v) \leq \text{dfn}(w) \leq \text{dfn}(v) + \text{number of descendants of } v \]
Digraphs: Breadth First Search

Given $G = (V, E)$ and all $v$ in $V$ are marked unvisited, a breadth-first search (bfs) is another way of navigating through the graph.

select one $v$ in $V$ and mark as visited; enqueue $v$ in $Q$

while not is_empty($Q$) {
    $x = \text{front}(Q)$; dequeue($Q$);
    for each $y$ in adjacent ($x$) if unvisited ($y$) {
        $\text{mark}(y)$; enqueue $y$ in $Q$; process ($x,y$)
        // (e.g. add to tree);
    }
}
BFS: Example

Start: E
E, F, G, B, D, C, A
Breadth First Spanning Forest

in a bfs of a directed graph, certain edges, when visited, lead to unvisited vertices- such edges are called TREE EDGES

and form a BREADTH FIRST SPANNING FOREST for the given digraph

NB only tree & non-tree (cross) edges
Directed Acyclic Graphs (DAGs)

- DAG - digraph with no cycles
- compare: tree, DAG, digraph with cycle
DAG: use

- Syntactic structure of arithmetic expressions with common sub-expressions
  
e.g. $((a+b)*c + ((a+b)+e)*(e+f)) * ((a+b)*c)$
DAG: use

- To represent **partial orders**
- A partial order R on a set S is a binary relation such that
  - for all a in S a R a is false (irreflexive)
  - for all a, b, c in S if a R b and b R c then a R c (transitive)
- examples: “less than” (<) and proper containment on sets

- S =\{1, 2, 3\}
- P(S) - power set of S (set of all subsets)
DAG: use

To model course prerequisites or dependent tasks

Year 1
- Data & Prog.
- Discrete Math
- PUMA

Year 2
- Op Systems
- Data Comm 1
- DS&A

Year 3
- Real time systems
- Data Comm 2
- Prog. Languages

Year 4
- Distributed Systems
- Compiler construction
Topological sort

- Given a DAG of prerequisites for courses, a topological sort can be used to determine an order in which to take the courses.
- (TS: DAG => sequence) (modified dfs)
- prints reverse topological order of a DAG from v

```c
int tsort(int v) {
    mark v visited
    for each w adjacent to v if w unvisited tsort(w)
    display(v)
}
```
Topological sort: example

start: A

tsort(A) => G K H D E C A B

reverse => B A C E D H K G
Detecting Cycles

- Use Warshall

- Use depth first search
Connectivity - & Reachability (Warshall)

Strongly Connected Components (SCCs)

- **Strongly connected component** of a digraph - set of vertices in which there is a **path** from any one vertex in the set to any other vertex in the set.

- Partition $V$ into equivalence classes $V_i$, $1 \leq i \leq r$ such that $v$ and $w$ are equivalent iff there is a path from $v$ to $w$ and from $w$ to $v$.

- Let $E_i$ be the set of edges with head and tail in $V_i$.

- The graphs $G_i = (V_i, E_i)$ are called **STRONGLY CONNECTED COMPONENTS (SCCs)** of $G$.

- A **STRONGLY CONNECTED GRAPH** has only one SCC.
SCC: example

- a digraph and its strongly connected components

- every vertex of G is in some SCC
- **NOT** every edge of G is in some SCC
- SCC = Strongly Connected Component
In a reduced graph (RG), the vertices are the **strongly connected components** of $G$.

- edge from vertex $C$ to $C'$ in RG if there is an edge from some vertex in $C$ to some vertex in $C'$.
- RG is always a DAG since if there were a cycle, all components in the cycle would be one strong component.
SCCs: algorithm

1. Perform a dfs and assign a number to each vertex
   
   \[
   \text{dfs}(v) \{ \text{mark } v \text{ visited} \\
   \text{for each } w \text{ adjacent to } v \text{ if } w \text{ unvisited } \text{dfs}(w) \\
   \text{number } v
   \}
   \]

2. construct digraph \( G_r \) by reversing every edge in \( G \)

3. perform a dfs on \( G_r \) starting at highest numbered vertex
   (repeat on next highest if all vertices not reached)

4. each tree in resulting spanning forest is an SCC of \( G \)
SCCs: example

Graph \[ G_r \]

```plaintext
dfs(v) {
    mark v visited
    for each w adjacent to v if w unvisited dfs(w)
    number v
}
```

After step 1

\[ df \text{ spanning forest for } G_r \]
Graphs: terminology

- \( G = (V,E) \)  
  \( V \) = set of vertices, \( E \) = set of edges \((v,w)\)

- \((v,w)\) ordered = **digraph** (directed graph)

- \((v,w)\) non-ordered = **undirected graph**

- digraph: \( w \) is **adjacent** to \( v \) if there is an edge from \( v \) to \( w \)

- **DAG**: directed acyclic graph

- **path**: sequence of vertices \( v_1..v_n \) where \((v_1,v_2)\)...\((v_{n-1},v_n)\) are edges

- **path length**: number of edges in a path

- **simple path**: all vertices are distinct (except possibly the first and last)

- **simple cycle**: simple path, length \( \geq 1 \), begin/end on same vertex
Graphs: terminology

- **Strongly Connected Component**: set of vertices in which there is a path from any vertex in the set to any other vertex in the set.

- **Reduced Graph**: vertices are strongly connected components of G.

- **Strongly Connected Digraph**: a path from every vertex to every other vertex.

- **Complete graph**: if there is an edge between every pair of vertices.

- **Implementation**: adjacency matrix or adjacency list.
Graphs: algorithms

- **Dijkstra**: single source shortest path
- **Floyd**: all pairs shortest path
- **Warshall**: transitive closure (determines if a path exists from v to w)
- **Depth First Search**:  
  - used to derive the depth first spanning forest for the graph  
  - used in cycle detection  
  - used to derive the strong components
- **Breadth First Search**:  
  - used to derive the breadth first spanning forest for the graph
- **Topological Sort**: DAG => sequence