Traversals of Directed Graphs

- To solve many problems dealing with digraphs, we need to visit vertexes and arcs in a systematic way

- Depth-first search (DFS)
  - A generalization of preorder traversal of a tree
  - Visit a start vertex \( v \) – it is marked *visited*
  - Each unvisited vertex adjacent to \( v \) is searched in turn, using depth-first-search recursively
Analysis of Depth-First Search (DFS)

- DFS on $G = (V, E)$
  - $|V| = n$, $|E| = e$, $n \leq e$
  - DFS time complexity: $O(e)$

Procedure $dfs\_visit\ (G: \text{graph})$;

```
for v := 1 to n do
    mark[v] := unvisited;
for v := 1 to n do
    if mark[v] = unvisited then
        dfs(v)
```

Procedure $dfs\ (v: \text{vertex})$;

```
begin
    var w: \text{vertex};
    mark[v]: = visited;
    for each vertex w on L[v] do
        if mark[w] = unvisited then
            dfs(w)
end; \{ dfs \}
```
DFS - example

- `dfs_visit(graph)`
  - Assume adjacent list has adjacent nodes in alphabetical order
  - Visit order: `A,`
Depth-First Spanning Forest

- During depth-first traversal, certain arcs lead to unvisited vertexes
  - Arcs leading to new vertexes are called *tree arcs*
  - Tree arcs forms a *depth-first spanning forest*
    (forest = set of trees)

- Other types of arcs
  - Back arc – leads to one of its ancestors (including itself) in DFSF
  - Forward arc – leads to one of its proper descendant in DFSF
  - Cross arc – leads to a vertex which is neither ancestor nor descendant
Depth-First Spanning Forest - example

- Tree arcs: A→B, B→C, B→D, E→F, E→G (solid arcs)
- Back arcs:
- Forward arcs:
- Cross arcs:
Depth-First Numbering

- Assign numbers to vertexes as we visit in DFS
  - Number the vertexes of digraph in the order we first mark them as visited during DFS
  - Insert the following after line (1) of $dfs(v)$
    
    
    \[
    \text{dfnumber}[v] := \text{count}; \\
    \text{count} := \text{count} + 1;
    \]

- Arcs
  - Back arcs: high number -&gt; low number
  - Forward arcs: low number -&gt; high number
  - Cross arcs: high number -&gt; low number
Directed Acyclic Graphs

- DAG (directed acyclic graph)
  - Directed graph with no cycle
DAGs

- Useful for following representations
  - Arithmetic expressions with common subexpressions
    \[((a+b)*c + ((a+b)+e) * (e+f)) * ((a+b)*c)\]
  - A binary relation: partial order \(R\) on a set \(S\)
    - For all \(a \in S\), \(a \, R \, a\) is true (reflexive)
    - For all \(a, b \in S\), if \(a \, R \, b\) and \(b \, R \, a\) are both true, \(a = b\) (antisymmetric)
    - For all \(a, b, c \in S\), if \(a \, R \, b\) and \(b \, R \, c\) are both true, \(a \, R \, c\) is true (transitive)

- E.g. \(R\) can be \(\leq\) relation on integers, or \(\subseteq\) on sets
DAGs - example

- Arithmetic expression
  - \(((a+b)\cdot c + ((a+b)+e) \cdot (e+f)) \cdot ((a+b)\cdot c)\)
DAGs - example

- Containment relation $\subseteq$ on sets
Test for Acyclicity

- Determine $G$ is acyclic
  - $G$ has no cycles
  - Depth-first search can be used

- A back arc in DFS on $G \equiv G$ is acyclic
  - If a back arc is encountered during DFS, then the graph has a cycle
  - Conversely, if a directed graph has a cycle, a back arc will always be encountered in any DFS
Topological Sort

- A large project is often divided into a collection of smaller tasks, some of which have to be performed in certain specific orders
  - Easily represented with DAGs
  - E.g. prerequisites in university curriculum

- Topological sort
  - A process of assigning a linear ordering to the vertexes on a dag, s.t. if $i \rightarrow j \in E$, $i$ appears before $j$ in the linear ordering
Topological Sort - example

- Linear ordering
  - $C_1, C_2, C_3, C_4, C_5$
Topological Sort - algorithm

- Reverse topological ordering based on \textit{dfs()}
- Add print statement at the end

\begin{verbatim}
procedure topsort ( v: vertex );
    {print vertexes accessible from v in reverse topological order } 
    var
        w: vertex;
    begin
        mark[v] := visited;
        for each vertex w on L[v] do 
            if mark[w] = unvisited then 
                topsort(w);
        writeln(v); 
    end; { topsort }
\end{verbatim}
Strongly Connected Components

- A strongly connected component of a digraph
  - A maximal set of vertexes in which there is a path from any one vertex in the set to any other vertex in the set

- Equivalence classes
  - A set $S$ is divided into subsets where elements in a subset has a equivalence relation
  - Equivalence relation – a binary relation on a set $S$
    - For all $a \in S$, $a R a$ is true (reflexive)
    - For all $a, b \in S$, if $a R b$ is true, $b R a$ is also true (symetric)
    - For all $a, b, c \in S$, if $a R b$ and $b R c$ are both true, $a R c$ is true (transitive)
Strongly Connected Components

- For a digraph \( G = (V, E) \), \( G_i = (V_i, E_i) \) are called **strongly connected components (SCC)**

- Partition \( V \) into equivalence classes, \( V_i \) \((1 \leq i \leq r)\)
  - Vertexes \( v \) and \( w \) are equivalent if and only if there is a directed path from \( v \) to \( w \) and a directed path from \( w \) to \( v \)

- Let \( E_i \) \((1 \leq i \leq r)\) be the set of arcs
  - Their head and tail are in \( V_i \)

- Also called **strong components**
Strongly Connected Components

- Cross-component arcs
  - Arcs not in any components

- Reduced graph
  - Strong components and cross-component arcs
  - Always DAG

Diagram:
- Directed graph
- Strongly connected components
- Reduced graph
Strongly Connected Components - algorithm

Steps

1. DFS on $G$ and number the vertexes in order of completion of the recursive call – assign a number to vertex $v$ after line (4) of $dfs(v)$ (different from dfnumber)

2. Construct $G_r$ by reversing the direction of every arc

3. DFS on $G_r$ starting from the highest-numbered vertex. If DFS does not reach all vertexes, start DFS from the highest-numbered remaining vertex

4. Each tree in the resulting spanning forest is a strongly connected component of $G$
Strongly Connected Components – Example

After step 1

$G_r$ in step 2

Depth-first spanning forest for $G_r$ in step 4
Strongly Connected Components – Example