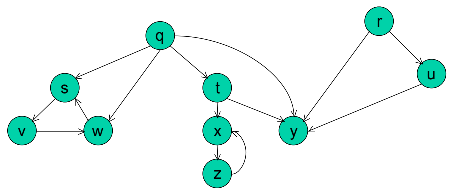
Elementary Graph Algorithms

Chapter 22

Your Turn

- Solve exercise 22.3-2 on page 547



Topological Sort

- A topological sort is performed on a directed acyclic graph
- A topological sort is a linear ordering of all vertices of a graph such that if G contains an edge (u, v), then u appears before v in the ordering

Topological Sort

- A topological sort of a graph can be viewed as an ordering of its vertices along a horizontal line so that all directed edges go from left to right
- Directed Acyclic Graphs (DAG) are used in many applications to indicate precedences among events
- What is a DAG?

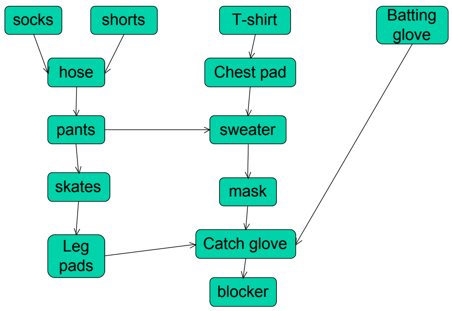
Topological Sort

- Good for modeling processes and structures that have a partial order:
  - a > b and b > c implies that a > c
  - But may have and b such that neither a > b nor b > c

TOPOLOGICAL-SORT(G)

- Call DFS(G) to compute finishing times f[v] for each vertex v
- As each vertex is finished, insert it onto the front of a linked list
- Return the linked list of vertices

Example



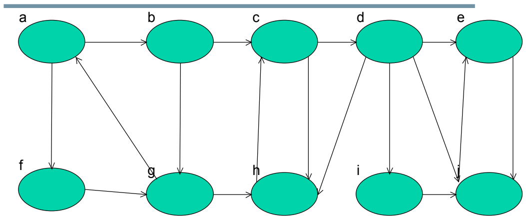
Topological Sort

- Running time for topological sort is:

Strongly Connected Components

- Given a directed graph G = (V, E)
- A strongly connected component (SCC) of G is a maximal set of vertices $C \subseteq V$
- Such that for all $u, v \in C$ both u -> v and v->u

Example



- Identify the strongly connected components

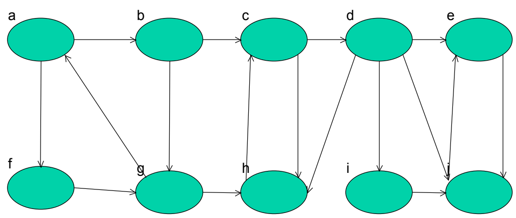
Transpose

- Algorithm uses G^T = transpose of G
  - G^T
- How long does it take to create G^T if using adjacency lists?
- Observation: G and G^T have the same SCC's.

SCC(G)

- Call DFS(G) to compute finishing times f[u] for all u
- Compute G^T
- Call DFS(G^T), but in the main loop, consider vertices in order of decreasing f[u] (as computed in first DFS)
- Output the vertices in each tree of the depth-first forest formed in second DFS as a separate SCC

Example



Class Problem - Bicoloring

- In 1976 the ``Four Color Map Theorem" was proven with the assistance of a computer. This theorem states that every map can be colored using only four colors, in such a way that no region is colored using the same color as a neighbor region. Here you are asked to solve a simpler similar problem. You have to decide whether a given arbitrary connected graph can be bicolored. That is, if one can assign colors (from a palette of two) to the nodes in such a way that no two adjacent nodes have the same color. To simplify the problem you can assume:
  - no node will have an edge to itself.
  - the graph is nondirected. That is, if a node *a* is said to be connected to a node *b*, then you must assume that *b* is connected to *a*.
  - the graph will be strongly connected. That is, there will be at least one path from any node to any other node.

Bicoloring

- Input
  - The input consists of several test cases. Each test case starts with a line containing the number n ($1 < n < 200$) of different nodes. The second line contains the number of edges l. After this, l lines will follow, each containing two numbers that specify an edge between the two nodes that they represent. A node in the graph will be labeled using a number a. An input with $n = 0$ will mark the end of the input and is not to be processed.
- Output
  - You have to decide whether the input graph can be bicolored or not

Bicoloring

Input:

```
3
3
0 1
1 2
2 0
9
8
0 1
0 2
0 3
0 4
0 5
0 6
0 7
0 8
0
```

Output:

```
NOT BICOLORABLE
BICOLORABLE
```