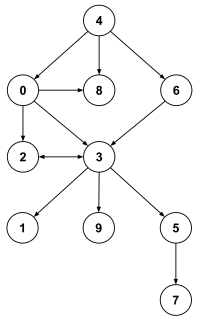
CS 340 - Analysis of Algorithms Week 2 Lab

Dianna Xu

Trace

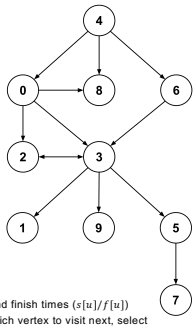
```
BFS(G, s) {
  for each (u in V) {
    mark[u] = false
    d[u] = infinity
    pred[u] = null
  }
  mark[s] = true, d[s] = 0, Q = {s}
  while (Q is not empty) {
    u = dequeue of Q
    for each (v in Adj[u]) {
      if (!mark[v]){
        mark[v] = true
        d[v] = d[u]+1
        pred[v] = u
        append v to Q
    }
  }
}
```



2

Trace

```
DFSG(G) {
  time = 0
  for each (u in V) {
    mark[u] = unseen
  }
  for each (u in V) {
    if (mark[u] == unseen)
      DFS(u)
  }
}
DFS(u) {
  mark[u] = seen
  s[u] = time++
  for each (v in Adj[u]) {
    if (mark[v] == unseen){
      pred[v] = u
      DFS(v)
    }
  }
  mark[u] = finished
  f[u] = time++
}
```



Label each vertex u with its start and finish times ($s[u]/f[u]$)
Whenever you have a choice of which vertex to visit next, select the lowest vertex in numerical order.

3

Modifying Standard Algorithms

- You may take anything straight from the book or the slides without having to justify time or correctness
- If you modify, then you must justify your changes
  - how the changes do or do not add time
  - correctness proof often has to be completely redone because the algorithm may have been changed to achieve completely different goals

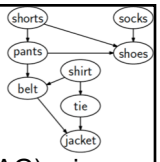
4

Graph Assumptions

- Unless otherwise stated
  - Simple graphs – no parallel edges or loops
  - Graphs may or may not be connected
  - Graphs may or may not be directed
- 3.2
  - problem says “undirected”
  - cycle should have more than one edge

5

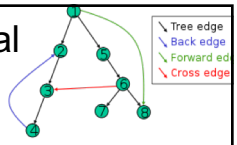
Topological Sort



- A *directed acyclic graph* (DAG), is a digraph (directed graph) that has no cycles.
- A *topological sort* (topological ordering) of a DAG is a linear ordering of the vertices of the DAG such that for each edge (u, v), u appears before v in the ordering.
- Any graph with a topological ordering is a DAG

6

DFS and Topological Sort

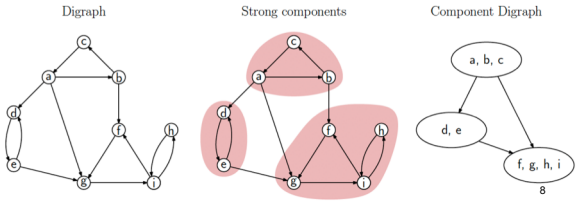


- In general, topological orderings are not unique
- A DAG has no back edges.
- Tree, forward and cross edges on a DFS has an ordering in reverse of what we want
- Combine DFS with a stack and output in reverse order

7

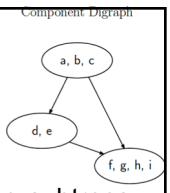
Strongly-Connected Components

- A digraph $G = (V, E)$ is *strongly connected* if for every pair of vertices u and v there is a path from u to v and from v to u

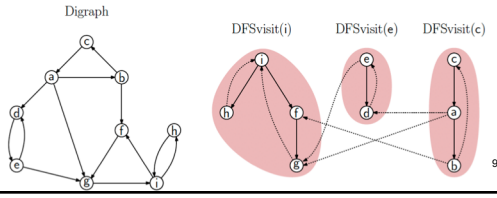


8

DFS and SCC



Starting a DFS at any of the component Digraph vertices in reverse topological order, then the subtrees of the DFS forest are the SCCs of $\mathcal{G}$



9