CS483 Design and Analysis of Algorithms

Chapter 3 Decomposition of Graphs

Instructor: Fei Li

lifei@cs.gmu.edu with subject: CS483

Office hours:
Room 5326, Engineering Building, Thursday 4:30pm - 6:30pm or by appointments

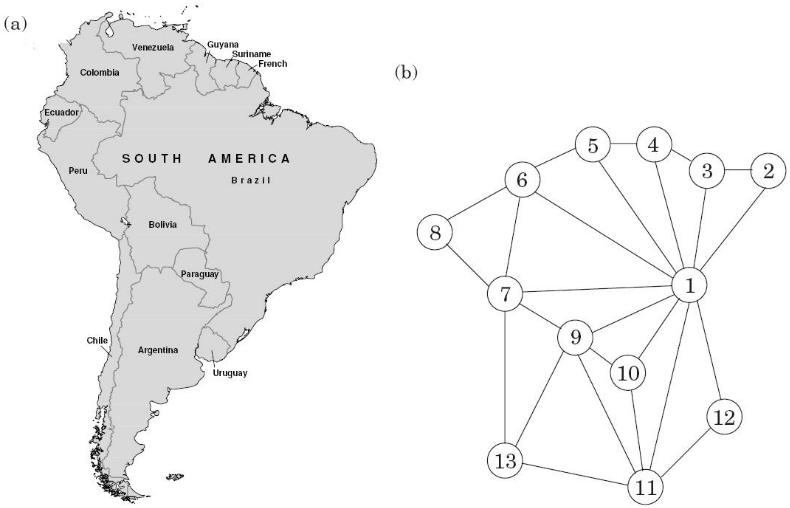
Course web-site:
http://www.cs.gmu.edu/∼lifei/teaching/cs483_fall09

Figures unclaimed are from books "Algorithms" and "Introduction to Algorithms"

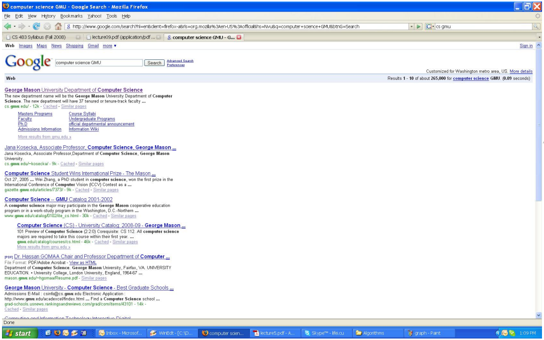
Decomposition of Graphs

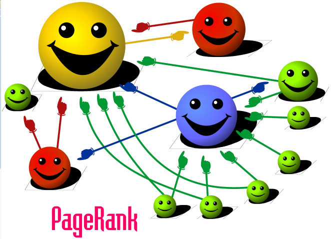
Why Graphs?

Figure 3.1 (a) A map and (b) its graph.



Why Graphs?

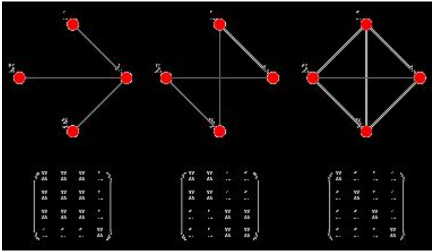

computer science GMU - Google Search - Mozilla Firefox
Google
George Mason University Department of Computer Science
Jana Kosecka, Associate Professor, Computer Science, George Mason ...
Computer Science Student Wins International Prize - The Mason ...
Computer Science -- GMU Catalog 2001-2002
Computer Science (CS) - University Catalog 2008-09 - George Mason ...
Dr. Hassan GOMAA Chair and Professor Department of Computer ...
George Mason University - Computer Science - Best Graduate Schools ...

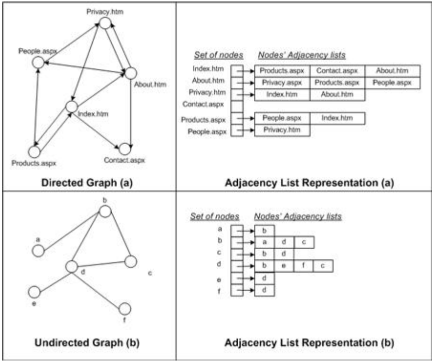

PageRank

Graphs

1. A *graph* $G = (V, E)$ is specified by a set of *vertices (nodes)* V and *edges* E between selected pairs of vertices
2. Edges are symmetric $\rightarrow$ *undirected graph*
3. Directions over edges $\rightarrow$ *directed graph*
4. E.g., political maps, exam conflicts, World Wide Web, etc.

Graph Representation





http://msdn2.microsoft.com/en-us/library/

Graph Traversal

Exploring a graph is rather like navigating a maze
Which parts of the graph are reachable from a given vertex?

http://www.northgacornmaze.com/images/maze_2006.jpg

Depth-First Search

Input: Graph $G = (V, E)$, vertex $s \in V$
Output: All vertices u reachable from s

```
function DFS(G)

    for all v ∈ V
        visited(v) = false;

    for all v ∈ V
        if not visited(v)
            explore(v);
```

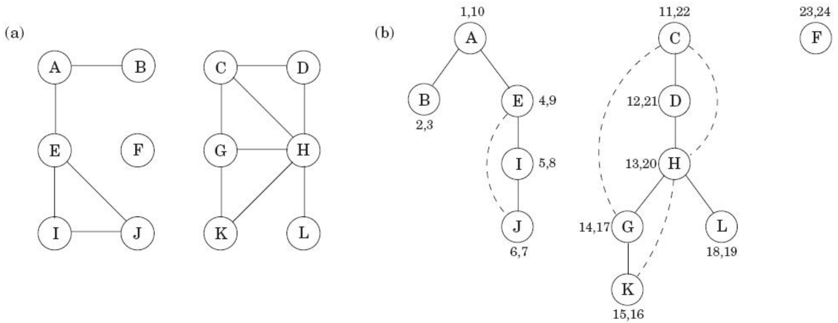
```
function explore(G, v)

    visited(v) = true;

    for each edge (v, u) ∈ E
        if not visited(u)
            explore(u);
```

Depth-First Search

Figure 3.6 (a) A 12-node graph. (b) DFS search forest.



Analysis of DFS

Theorem

All nodes reachable from s can be found via DFS

Proof.

? □

Theorem

The overall running time of DFS *is* $O(|V| + |E|)$

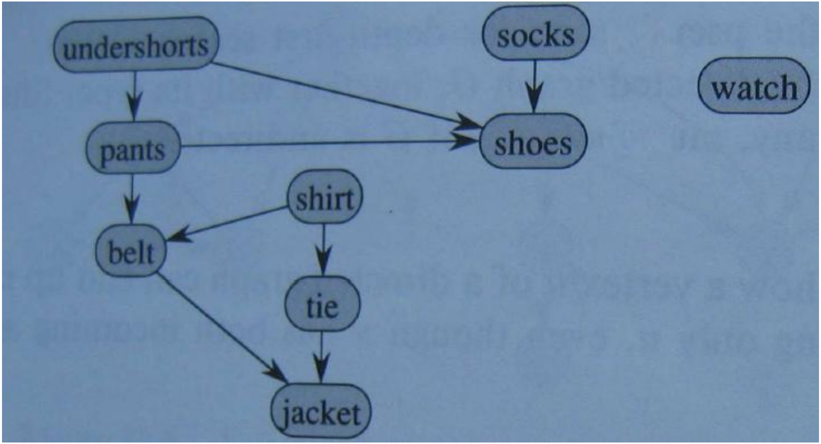
Proof.

1. The time initializing each vertex is $O(|V|)$
2. Each edge $(u, v) \in E$ is examined twice, once exploring u and once exploring v. Therefore takes $O(|E|)$ time

□

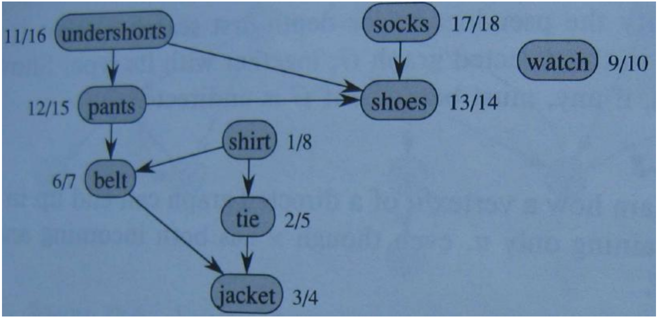
Topological Sorting — An Application of DFS

Input: a directed acyclic graph (DAG) G
output: A linear ordering of all its vertices, such that if G contains an edge (u, v), then, u appears before v in the ordering



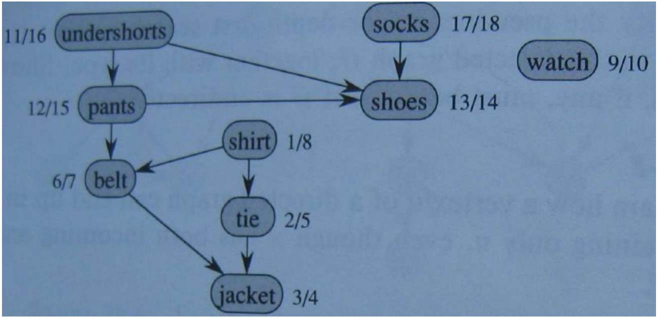
Topological Sorting — An Application of DFS

1. Run DFS to get (startingtime, finishingtime)

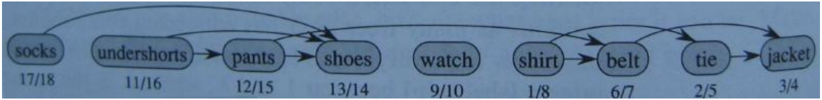


Topological Sorting — An Application of DFS

1. Run DFS to get (`startingtime`, `finishingtime`)



2. List nodes in reverse order of their finishing time



Strongly Connected Components in Directed Graphs

Definition

Two nodes u and v of a directed graph are **connected** if there is a path from u to v and a path from v to u.

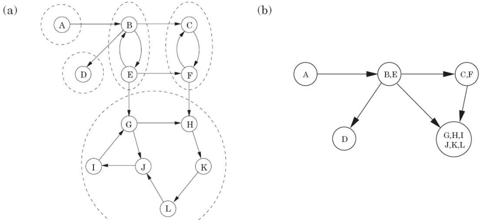
Remark

Connectivity in undirected graphs is straightforward. A graph that is not connected can be decomposed in a natural and obvious manner into several connected components.

Remark

Every directed graph is a directed acyclic graph (DAG) of its strongly connected components (disjoint sets of V). A DAG can be linearized.

Figure 3.9 (a) A directed graph and its strongly connected components. (b) The meta-graph.



How to Find SCC?

Property 1

If the `explore` subroutine starts at node u, then it will terminate precisely when all nodes reachable from u have been visited. If we call `explore` on a node that lies somewhere in a *sink* strongly connected component, then we will retrieve exactly that component

What can we learn?

1. How do we find a node that we know for sure lies in a sink strongly connected component?
2. How do we continue once this first component has been discovered?

Property 2

The node that receives the highest `ending time` (i.e., post number) in a depth-first search must lie in a source strongly connected component.

How to Find SCC?

Property 3

If C and C' are strongly connected components, and there is an edge from a node in C to a node in C', the the highest post number in C is larger than the highest post number in C'

Proof.

If DFS visits component C before component C', then clearly all of C and C' will be traversed before the procedure stuck. Therefore the first node visited in C will have a higher post number than any node of C'. If C' gets visited first, then DFS will get stuck after seeing all of C' before seeing any of C. $\square$

What can we learn?

Remark

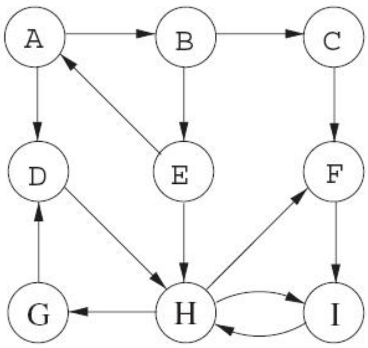
The strongly connected components can be linearized by arranging them in decreasing order of their highest post *numbers.*

Remark

The reverse graph G^R*, the same as* G *but with all edges reversed, has the same SCC as* G.

Strongly Connected Components

1. Run depth-first search on G^R
2. Run the undirected connected components algorithm on G, and during the depth-first search, process the vertices in decreasing order of their *finishing time* from step 1



The algorithm is linear-time, only the constant in the linear term is about twice that of straight DFS.