
15.082 and 6.855J
February 11, 2003

Lecture 3. Graph Search

Breadth First Search

Depth First Search

Topological Sort

Overview

Today: Different ways of searching a graph

- **a generic approach**
 - **breadth first search**
 - **depth first search**
 - **data structures to support network search**
- ◆ **Fundamental for most algorithms considered in this subject**

Searching a Directed Graph

ALGORITHM SEARCH

INPUT: A directed network G , and node s

OUTPUT: The set of nodes $\{j : \text{there is a directed path from } s \text{ to } j \text{ in } G\}$. These are the nodes reachable from s .

A node is either *marked* or *unmarked*. Initially only node s is marked. If a node is marked, it is reachable from node s

An arc $(i,j) \in A$ is *admissible* if node i is marked and j is not.

pred(i) is the predecessor of i on the path from s to i .

Algorithm Search

Begin

Initialize.

while LIST $\neq \emptyset$ **do**

begin

select a node i in LIST;

if node i is incident to an admissible arc (i,j) **then**

begin

mark node j ;

pred(j) := i ;

next := next + 1

order(j) := next;

add node j to LIST;

end

else delete node i from LIST

end;

end

Initialize

Initialize

begin

unmark all nodes in N;

mark node s;

pred(s) = 0; {that is, it has no predecessor}

next := 1; (next is a counter)

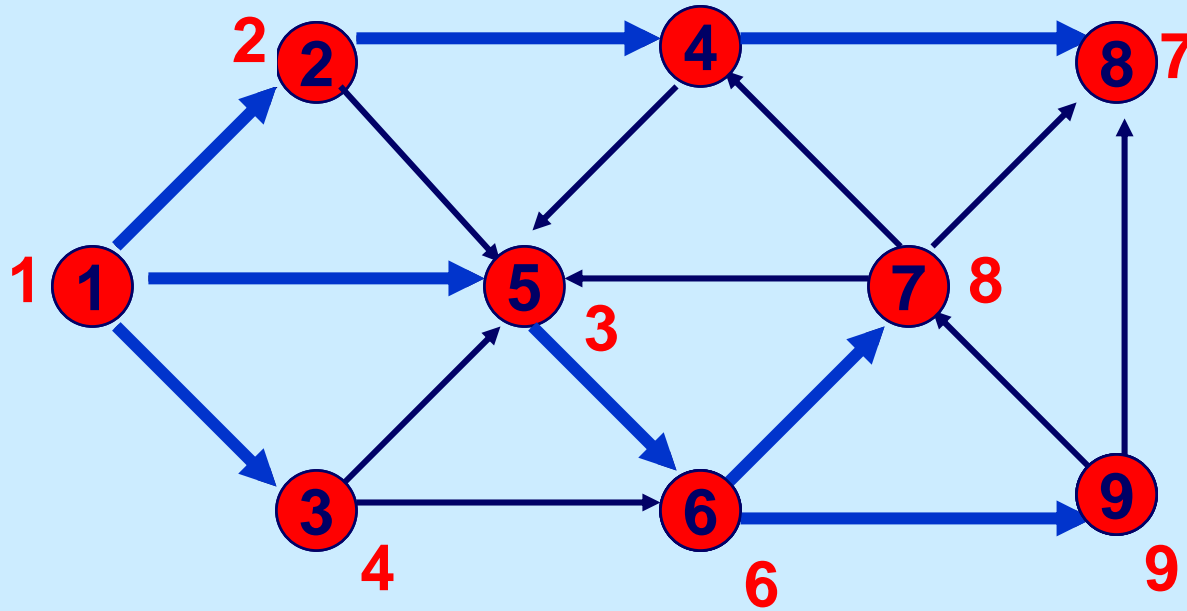
order(s) := next;

LIST = {s}

end

**Breadth First Search
Animation**

Breadth First Search



Final output from breadth first search

It provides a label of the nodes that is called a **breadth first search labeling**. We say that a labeling of the nodes is a **breadth first search labeling** if the nodes are labeled in non-decreasing order of “distance” from the origin, where distance is the min number of arcs on the path from the origin.

Next: how to implement this so that it runs in $O(n+m)$ time.

Algorithm Search

Begin

Initialize.

while LIST $\neq \emptyset$ **do**

begin

select a node i in LIST;

if node i is incident to an admissible arc (i,j) **then**

begin

mark node j ;

pred(j) := i ;

next := next + 1

order(j) := next;

add node j to LIST;

end

else delete node i from LIST

end;

end

See next slide

n iterations of the while loop

select $O(1)$

The key step

$O(1)$ for
these steps

$O(1)$

Initialize

Initialize

begin

unmark all nodes in N ;

mark node s ;

$\text{pred}(s) = 0$; {that is, it has no predecessor}

$\text{next} := 1$; (next is a counter)

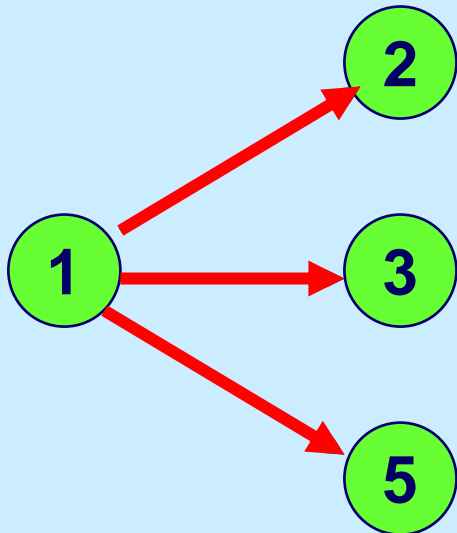
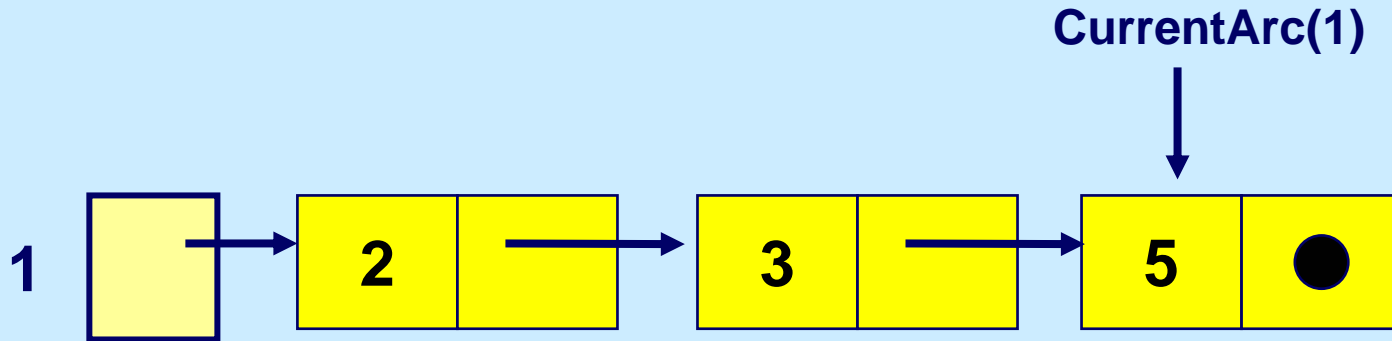
$\text{order}(s) := s$;

$\text{LIST} = \{s\}$

end

**Unmarking takes $O(n)$
All else takes $O(1)$**

Finding an admissible arc



Scan through the arc list for the selected node, and keep track using current arc.

Next Steps

Prove a “cutset theorem” showing how the algorithm will terminate.

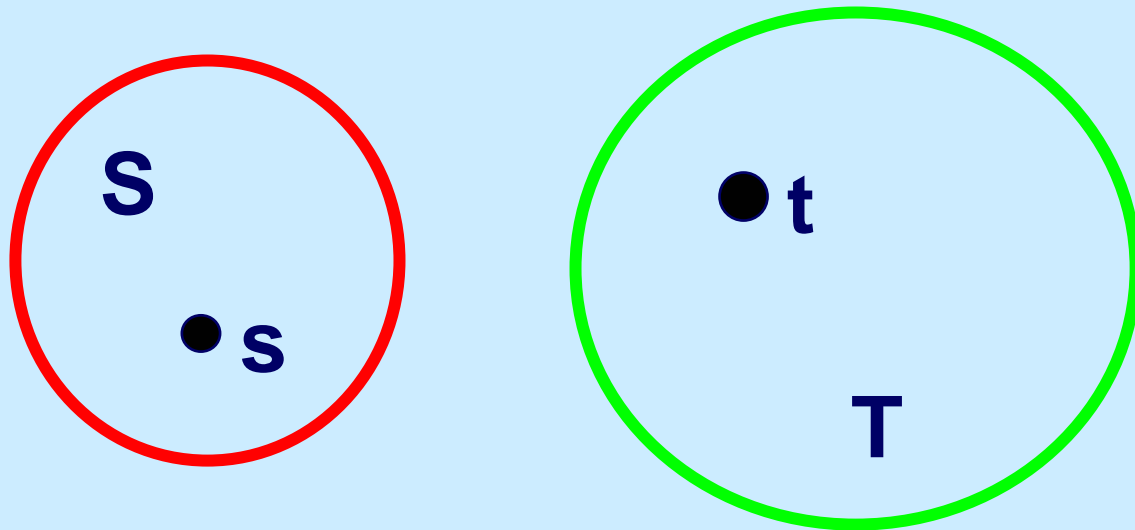
A cutset is a partition of the nodes into two parts S and $T = N \setminus S$.

Show that it ends with the set of nodes reachable from the origin.

Show that the distances from the origin node in a breadth first tree are all minimum.

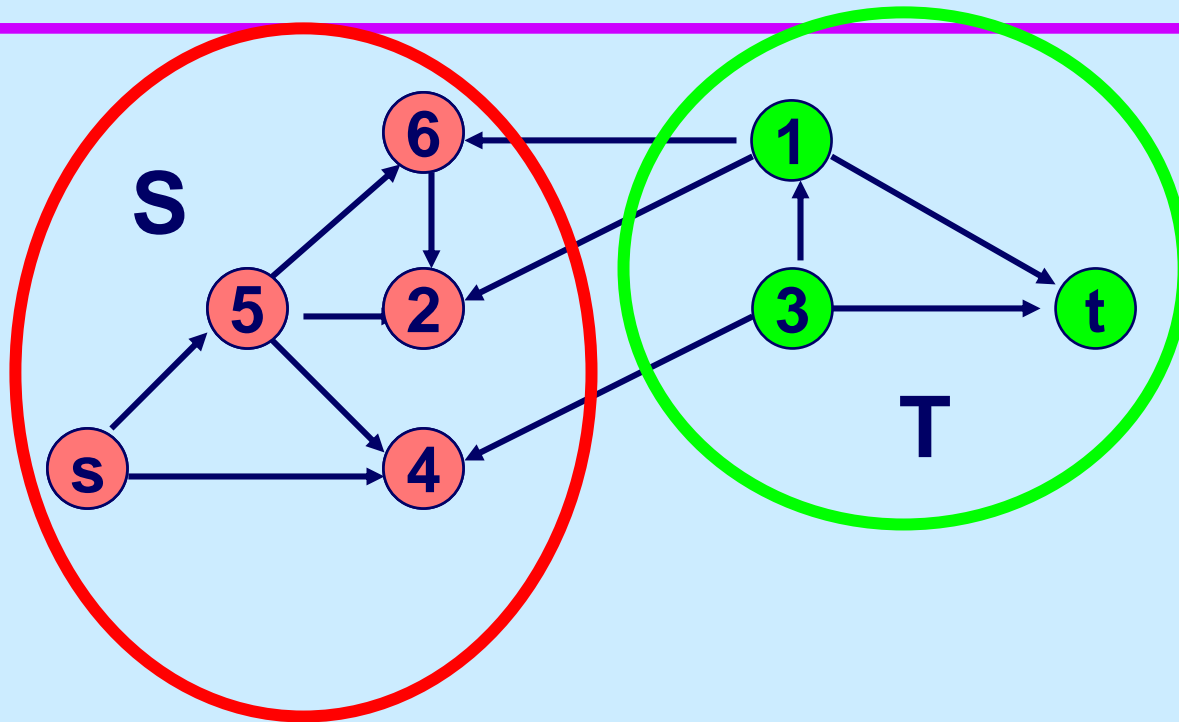
Cutset Theorem

Theorem. There is no directed path from s to t if and only if the following is true:
there is a partition of the node set N into subsets S and $T = N - S$ such that there is no arc directed from a node in S to a node in T .



Proof. If such a cutset exists, then there is no path from s to t .

Proof of “only if” part of the Cutset Theorem



Let S be the set of nodes reachable from s .

Let T be the remaining nodes

Then no arc (i,j) directed from a node in S to a node in T . Otherwise, j would be reachable.

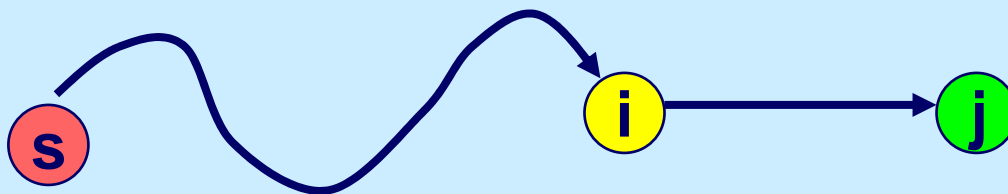
Theorem.

Theorem. Algorithm search determines all nodes reachable from node s in $O(n + m)$ time.

Proof. We already did this.

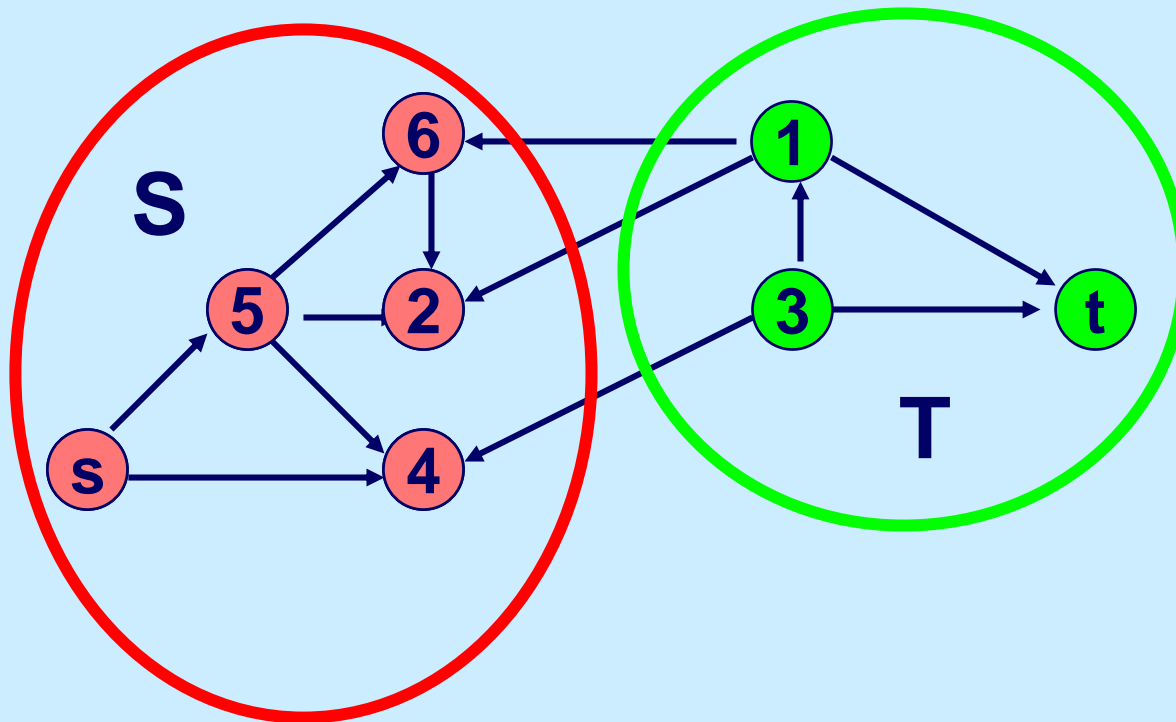
Claim: If node j is marked, then it is reachable from node s .

Proof of Claim: Assume true for all nodes marked before j , and look at the iteration in which node j was marked. Then i was marked. And so there was a path from s to i . So there is a path from s to j .



Proof (continued)

The algorithm ends when there are no admissible arcs.



Then there is no arc, and hence no path, from a marked node to an unmarked node.

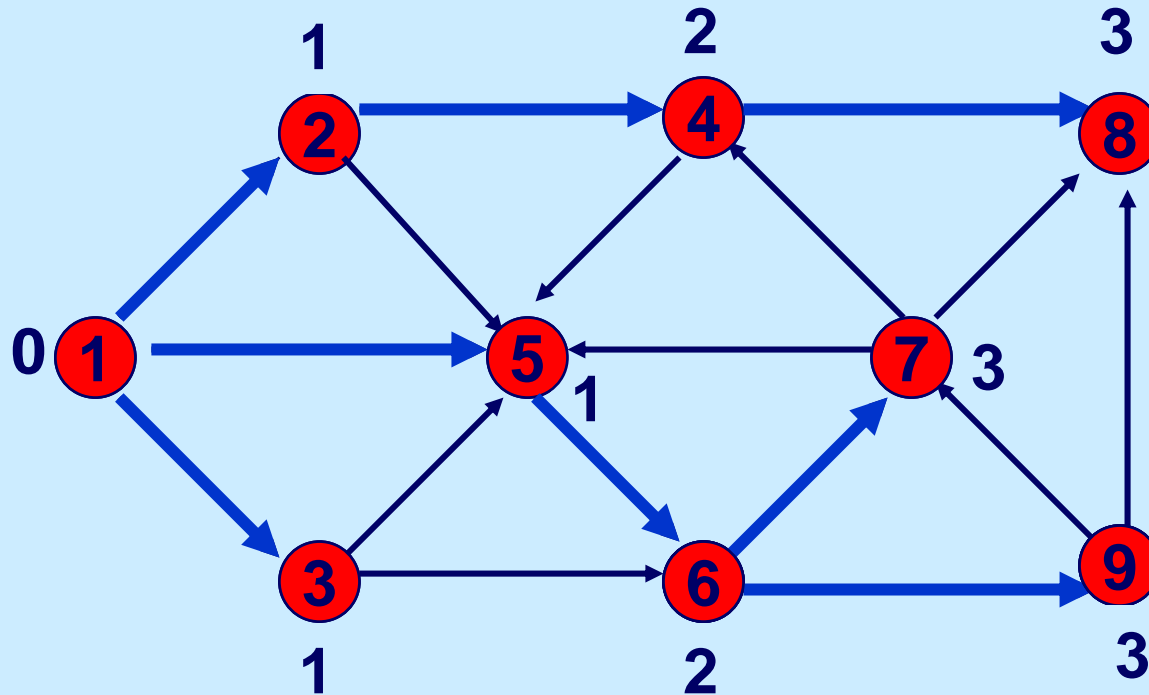
Finding Connected Components

Breadth first search will find a connected component of an undirected graph in $O(n+m)$ time. How can it find all connected components in $O(n+m)$ time?

Maintain a set S of unmarked nodes. After each component is searched, select a node of S . This set is maintained in $O(1)$ time per node.

More on Breadth First Search

Theorem. *The breadth first search tree is the “shortest path tree”, that is, the path from s to j in the tree has the fewest possible number of arcs.*



Proof of Breadth First Search Ordering

Proof. Let $d(i)$ be the fewest number of arcs on the path from s to i . Let T be the tree of predecessors obtained in the breadth first search.

Let $d^T(j)$ be the number of arcs on the path from s to j in T .

Claim:

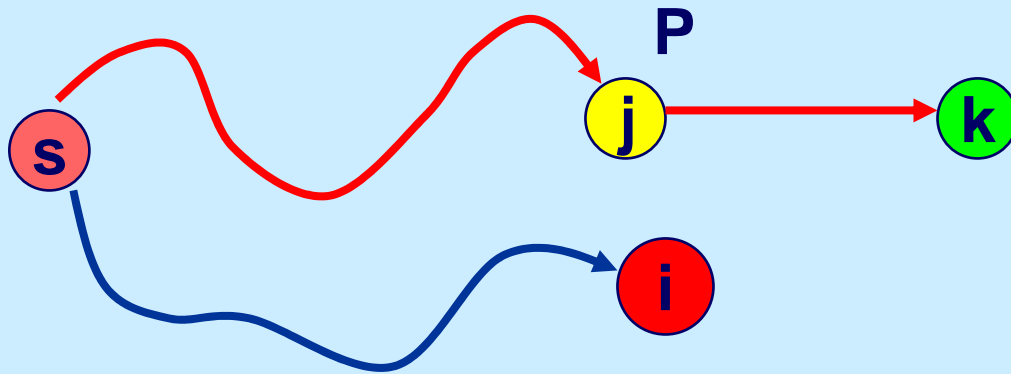
1. If node i is marked before node j , then $d(i) \leq d(j)$.
2. If node i is scanned before node j , then $d(i) \leq d(j)$.
3. $d^T(j) = d(j)$.

Proof of claims. 1 \Leftrightarrow 2 because LIST is a queue.

Assume that the claims are all true for nodes marked before node k .

Proof continued.

$$d^T(k) = d(k)$$



Let P be a path from s to k with $d(k)$ arcs.

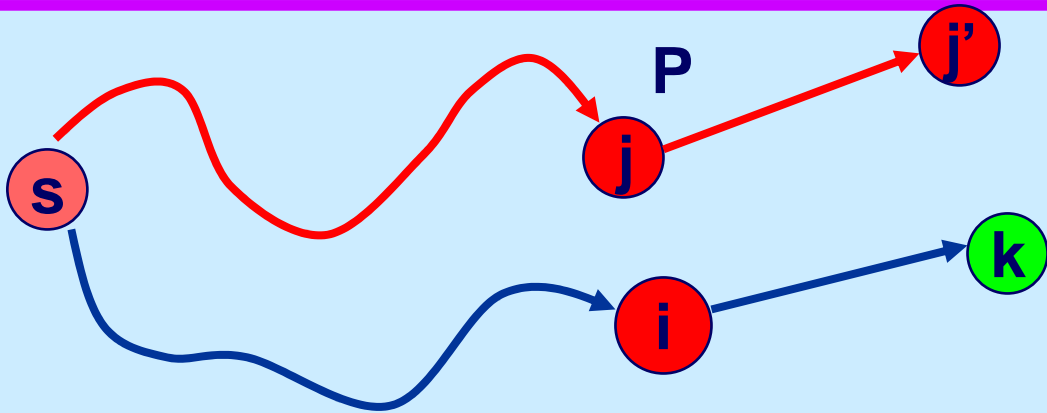
Suppose that $\text{pred}(k) = i$ in T

Then i was scanned before j . By induction, $d(i) \leq d(j)$, and $d^T(i) = d(i)$.

Then $d^T(k) = d^T(i) + 1 = d(i) + 1 \leq d(j) + 1 = d(k)$

But $d^T(k) \geq d(k)$. So $d^T(k) = d(k)$.

Proof continued.



Let P be a path from s to k with $d(k)$ arcs.

Suppose that $\text{pred}(k) = i$ in the tree.

Suppose j' was scanned before k . We claim that $d(j') \leq d(k)$. Suppose $\text{pred}(j') = j$.

If j is scanned before i , then by induction $d(j) \leq d(i)$, and thus $d(j') = d(j) + 1 \leq d(i) + 1 = d(k)$.

If j were scanned after i , then j' would be marked after k , contrary to assumption.

So, $d(j') \leq d(k)$. This completes the proof.

Depth first search

This is the search technique, where LIST is treated as a stack.

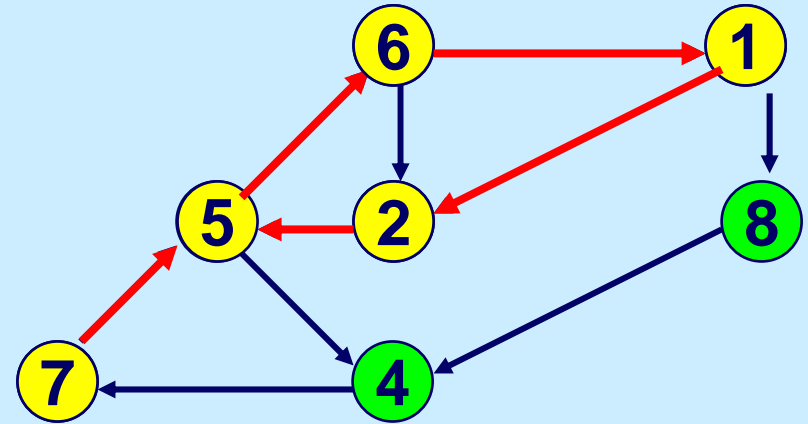
It is great for finding a directed cycle quickly.

It has other surprising applications in graph search.

Depth First Search Animation

Preliminary to Topological Sorting

LEMMA. If each node has at least one arc going out, then the first inadmissible arc of a depth first search determines a directed cycle.



COROLLARY 1. If G has no directed cycle, then there is a node in G with no arcs going. And there is at least one node in G with no arcs coming in.

COROLLARY 2. If G has no directed cycle, then one can relabel the nodes so that for each arc (i,j) , $i < j$.

Topological ordering

begin

INITIALIZE

while LIST $\neq \emptyset$ **do**

begin

select a node i from LIST and delete it;

next := next + 1;

order(i) := next;

for all (i, j) $\in A(i)$ **do**

begin

indegree(j) := indegree(j) - 1;

if indegree(j) = 0 **then** LIST := LIST \cup { j };

end

end

if next < n **then** the network contains a directed cycle

else the network is acyclic and the order is topological

end

Topological Ordering

INITIALIZE

begin

for all $i \in N$ **do** $\text{indegree}(i) := 0$;

for all $(i,j) \in A$ **do** $\text{indegree}(j) := \text{indegree}(j) + 1$;

LIST := \emptyset ;

next := 0;

for all $i \in N$ **do**

if $\text{indegree}(i) = 0$, **then** **LIST** := **LIST** \cup { i };

end

Topological Sorting

More on Topological Sorting

Very useful for algorithms on acyclic graphs.

Runs in $O(n+m)$ time.

Consider the iteration in which node i is selected.

Because $\text{indegree}(i) = 0$, for any arc (k,i) , node k was selected at a previous iteration.

Therefore, if (i,j) is an arc, then $\text{order}(i) < \text{order}(j)$.

If the algorithm ends before labeling all nodes, then there is a directed cycle in the unlabeled nodes.

Corollary. Topological Sort computes a topological order in $O(n+m)$ steps.

Summary on Graph Search

- ◆ Find the shortest path from s to each other node
path length = number of arcs on path;
- ◆ Determine the connected components of a network;
- ◆ Determine breadth first search;
- ◆ Determine depth first search;
- ◆ Shows up in other algorithms as well.

Summary on Graph Search

- ◆ **Determine topological sort;**
 - **Running time is $O(n+m)$ using simple data structures and algorithms.**
 - **Very important for preprocessing.**