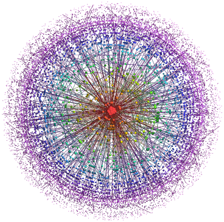

THE UNIVERSITY OF
AUCKLAND
Te Whare Wānanga o Tāmaki Makaurau
NEW ZEALAND

Algorithms and Data Structures

Lecture 13 Directed Acyclic Graph

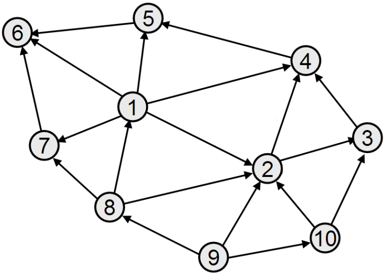
Jiamou Liu
The University of Auckland



Directed Acyclic Graphs

Definition [DAG]

A directed acyclic graph (dag) is a digraph that does not contain a cycle.

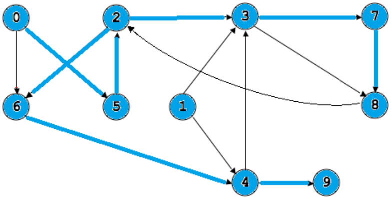


Acyclicity Problem:

- INPUT: A digraph
- OUTPUT: decide if the digraph is a dag.

Let T be the DFS forest in G. There are four types of edges in G:

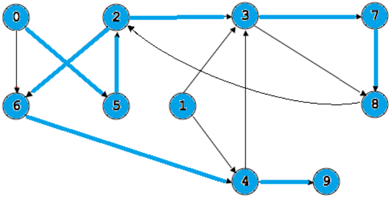
1. If (u, v) belongs to the search forest, (u, v) is a tree edge
2. Otherwise if u is an ancestor of v in T, (u, v) is a forward edge
3. Otherwise if v is an ancestor of u in T, (u, v) is a back edge
4. Otherwise (u, v) is a cross edge



Tree edges: (0,5)(5,2),(2,6),(2,3),(3,7),(7,8),(6,4),(4,9)

Let T be the DFS forest in G. There are four types of edges in G:

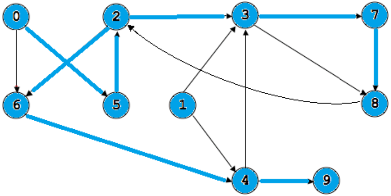
1. If (u, v) belongs to the search forest, (u, v) is a tree edge
2. Otherwise if u is an ancestor of v in T, (u, v) is a forward edge
3. Otherwise if v is an ancestor of u in T, (u, v) is a back edge
4. Otherwise (u, v) is a cross edge



Tree edges: (0,5)(5,2),(2,6),(2,3),(3,7),(7,8),(6,4),(4,9)
Forward edges: (0,6),(3,8)

Let T be the DFS forest in G. There are four types of edges in G:

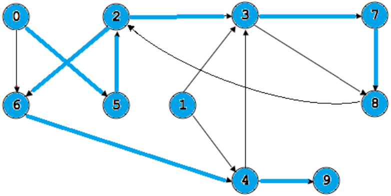
1. If (u,v) belongs to the search forest, (u,v) is a tree edge
2. Otherwise if u is an ancestor of v in T, (u,v) is a forward edge
3. Otherwise if v is an ancestor of u in T, (u,v) is a back edge
4. Otherwise (u,v) is a cross edge



Tree edges: (0,5)(5,2),(2,6),(2,3),(3,7),(7,8),(6,4),(4,9)
Forward edges: (0,6),(3,8) Back edges: (8,2)

Let T be the DFS forest in G. There are four types of edges in G:

1. If (u, v) belongs to the search forest, (u, v) is a tree edge
2. Otherwise if u is an ancestor of v in T, (u, v) is a forward edge
3. Otherwise if v is an ancestor of u in T, (u, v) is a back edge
4. Otherwise (u, v) is a cross edge



Tree edges: (0,5)(5,2),(2,6),(2,3),(3,7),(7,8),(6,4),(4,9)
Forward edges: (0,6),(3,8) Back edges: (8,2)
Cross edges: (4,3),(1,3),(1,4)

Fact.

Let G be a digraph. Then the following are equivalent:

- (1). G is a DAG
- (2). the DFS forest has no back edge.

Fact.

Let G be a digraph. Then the following are equivalent:

- (1). G is a DAG
- (2). the DFS forest has no back edge.

Proof.

$\Rightarrow$ Suppose G is a dag, then the search forest doesn't have a back edge as otherwise, there will be a cycle.

$\Leftarrow$ Suppose G is not a dag, then there is a cycle C in G.
Let v be the first node discovered by the DFS in C.
Let (u, v) be the edge in C that goes into v.
Then in the search tree v is an ancestor of u.
Then (u, v) is a back edge. $\square$

Fact

The following algorithm runs in time $O(n + m)$ and decides whether any given digraph G is a dag.

Fact

The following algorithm runs in time $O(n+m)$ and decides whether any given digraph G is a dag.

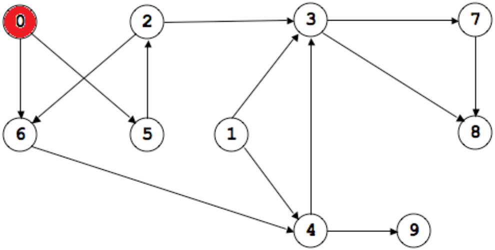
Algorithm: acyclic(G)

INPUT: A digraph G
OUTPUT: Return if G is a dag
Run DFS(G) with the following modification:
  Whenever discover a node u, do
    for every edge (u, v) out of u
      if $pre(v) < pre(u)$ and $post(v)$ is undefined
        Declare G has a cycle and return
Declare that G is a dag.

DFS and Linearisations

Definition [Linearisations]

A linearization or (topological sort) of a digraph G is a list of all nodes in G such that if G contains an edge (u, v) then u appears before v in the list.

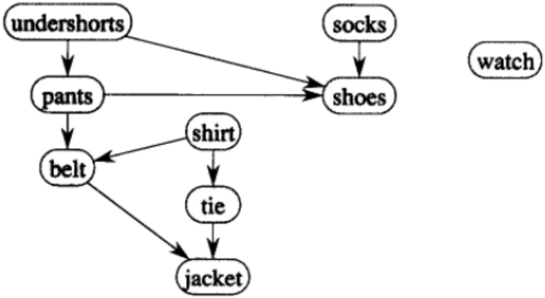


Topological Sorts:

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

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

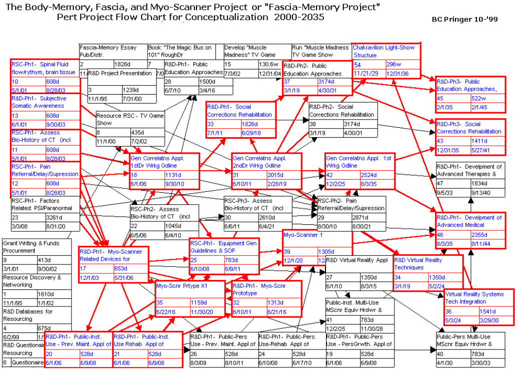
In what order should I put on my cloths?



Possible orderings are linearisations of the dependency graph:
Possible order 1: Shirt, Socks, Undershorts, Watch, Pants, Tie, Belts, Jacket, Shoes
Possible order 2: Watch, Undershorts, Socks, Pants, Shoes, Shirt, Belt, Tie, Jacket

Application of Linearisation

- Job/Task/Instruction scheduling
- Project Evaluation and Review Technique (PERT)
- `makefiles` in Unix / `APT` in Ubuntu Linux
- Class/Package dependency in a software project



- **Question 1.** Is there a digraph that can not be linearised?

- **Question 1.** Is there a digraph that can not be linearised?

  Answer: Yes! Digraphs with cycles.

- **Question 1.** Is there a digraph that can not be linearised?

  Answer: Yes! Digraphs with cycles.

- **Question.** What dags can be linearised?

- **Question 1.** Is there a digraph that can not be linearised?

  Answer: Yes! Digraphs with cycles.

- **Question.** What dags can be linearised?

  Answer: All of them! We are now going to present algorithms to linearise a DAG.

First Try: Zero In-degree Algorithm

The Zero In-degree algorithm finds a linearisation for a dag:

Algorithm: ZeroInDegree(G)

INPUT: a DAG G
OUTPUT: a linearisation of G
list $\leftarrow$ an empty list
while G is not empty **do**
  for each u in V
    if $inDegree(u) = 0$ **then**
      Add u to the end of *list*
      Delete u from G
return *list*

First Try: Zero In-degree Algorithm

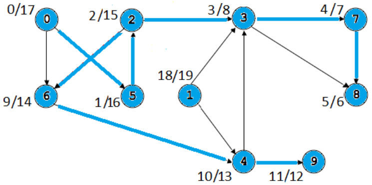
The Zero In-degree algorithm finds a linearisation for a dag:

Algorithm: ZeroInDegree(G)

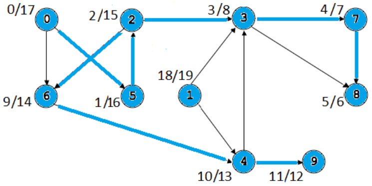
INPUT: a DAG G
OUTPUT: a linearisation of G
list $\leftarrow$ an empty list
while G is not empty **do**
  for each u in V
    if $inDegree(u) = 0$ **then**
      Add u to the end of *list*
      Delete u from G
return *list*

Running time: The algorithm runs in time $O((n+m)n)$.

Second Try: DFS-based Linearisations



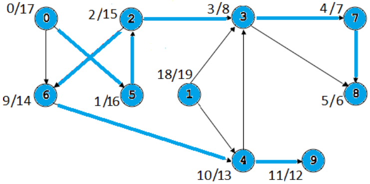
Second Try: DFS-based Linearisations



Fact

Let G be a dag. If (u, v) is an edge in G, then $post(v) < post(u)$.

Second Try: DFS-based Linearisations



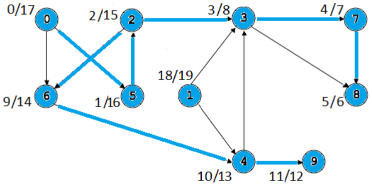
Fact

Let G be a dag. If (u, v) is an edge in G, then $post(v) < post(u)$.

Proof

There are two cases:

Second Try: DFS-based Linearisations



Fact

Let G be a dag. If (u, v) is an edge in G, then $post(v) < post(u)$.

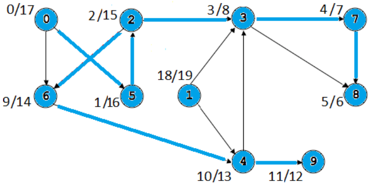
Proof

There are two cases:

Case 1. u is discovered earlier than v is.

Then v must be finished before u is finished.

Second Try: DFS-based Linearisations



Fact

Let G be a dag. If (u, v) is an edge in G, then $post(v) < post(u)$.

Proof

There are two cases:

Case 1. u is discovered earlier than v is.

Then v must be finished before u is finished.

Case 2. v is discovered earlier than u is.

Since G is acyclic, there is no path that goes from v to u.

Hence v is again finished earlier than u is finished. $\square$

We obtain an easy algorithm for graph linearisation in time $O(m + n)$:
Output the list of nodes in decreasing finishing order.

Algorithm: DFS-Linearise(G)

INPUT: a dag G
OUTPUT: a linearisation of G
stack $\leftarrow$ an empty stack
Run DFS, in addition:
    When a node is finished, push it to *stack*.
return elements in *stack* in the same order as they are popped out

We obtain an easy algorithm for graph linearisation in time $O(m + n)$: Output the list of nodes in decreasing finishing order.

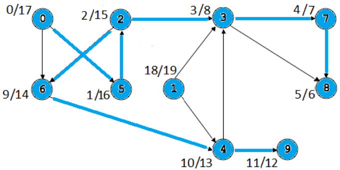
Algorithm: DFS-Linearise(G)

INPUT: a dag G
OUTPUT: a linearisation of G
stack ← an empty stack
Run DFS, in addition:
    When a node is finished, push it to *stack*.
return elements in *stack* in the same order as they are popped out

We obtain an easy algorithm for graph linearisation in time $O(m+n)$:
Output the list of nodes in decreasing finishing order.

Algorithm: DFS-Linearise(G)

INPUT: a dag G
OUTPUT: a linearisation of G
stack $\leftarrow$ an empty stack
Run DFS, in addition:
  When a node is finished, push it to *stack*.
return elements in *stack* in the same order as they are popped out



DFS-Linearize(G):

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

Further Comments

Acyclicity and Linearizability

- We established two characterizations of linearisability of a digraph:

  A digraph is linearizable if and only if

  - it is acyclic
  - the DFS forest has no back edge

- In other words

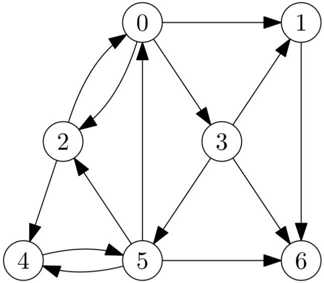
$$\text{Linearizable} \equiv \text{Acyclicity} \equiv \text{No-Back-edgeness}$$

- With this understanding, we are able to design algorithms for deciding these properties.

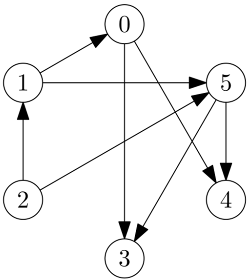
Summary

- Directed acyclic graph (DAG)
- Acyclicity problem: DFS-based algorithm (no back edge)
- Linearisation problem:
  - Zero in-degree algorithm: $O(n(m+n))$
  - DFS-based algorithm (decreasing finishing order): $O(m+n)$

Question 1. For the following digraph, perform DFS starting from 0. Find all tree edges, forward edges, backward edges, and cross edges.



Question 2. For the following digraph, perform the algorithm taught above and find a topological sort (linearisation).




THE UNIVERSITY OF
AUCKLAND
Te Whare Wānanga o Tāmaki Makaurau
NEW ZEALAND