Topological Sort

Superhero Problem

I have a friend who talks about which superheroes are better than others. I have asked them a series of questions regarding which superheroes can beat which. My friend did not give me an answer for all superhero pairs (they were too close to call), but I have enough data to do some analysis.

Superhero Problem

I have a friend who talks about which superheroes are better than others. I have asked them a series of questions regarding which superheroes can beat which. My friend did not give me an answer for all superhero pairs (they were too close to call), but I have enough data to do some analysis.

- Can an ordering of superheroes satisfy my friend's opinions?

Superhero Problem

I have a friend who talks about which superheroes are better than others. I have asked them a series of questions regarding which superheroes can beat which. My friend did not give me an answer for all superhero pairs (they were too close to call), but I have enough data to do some analysis.

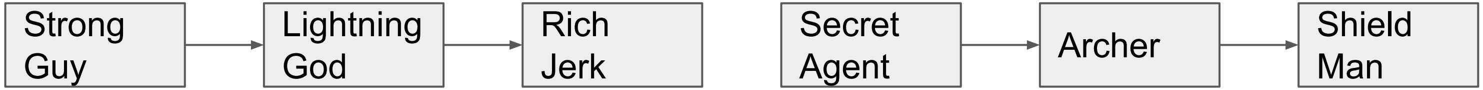
- Can an ordering of superheroes satisfy my friend’s opinions?
- Create at least one of the orderings, if it exists.

Semi-Comparable Values Problem

If we had an ordering, what would make it invalid?

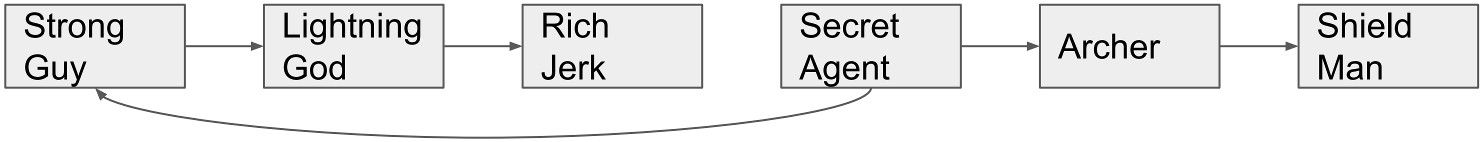
Semi-Comparable Values Problem

If we had an ordering, what would make it invalid?



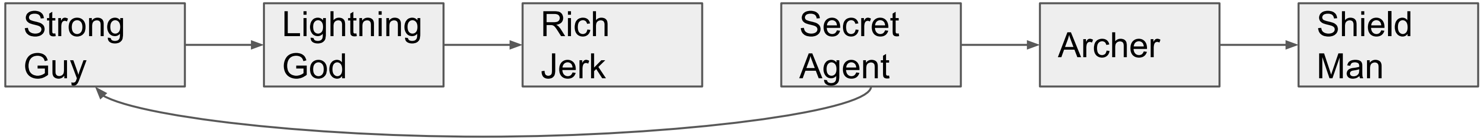
Semi-Comparable Values Problem

If we had an ordering, what would make it invalid?



Semi-Comparable Values Problem

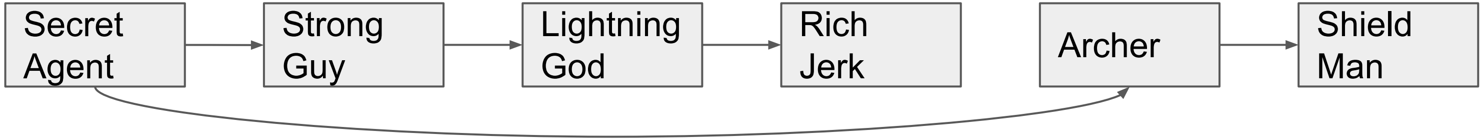
If we had an ordering, what would make it invalid?



Remove inverted values by moving up the better value.

Semi-Comparable Values Problem

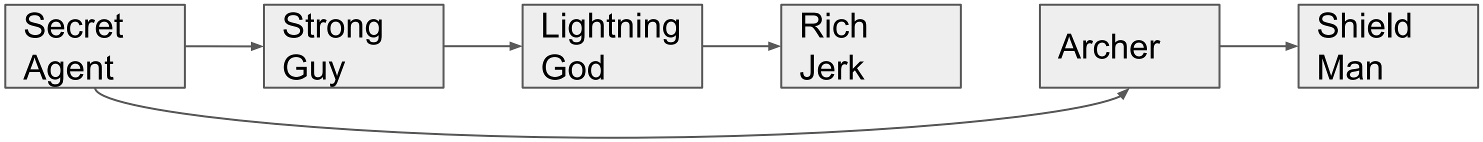
If we had an ordering, what would make it invalid?



Remove inverted values by moving up the better value.

Semi-Comparable Values Problem

If we had an ordering, what would make it invalid?

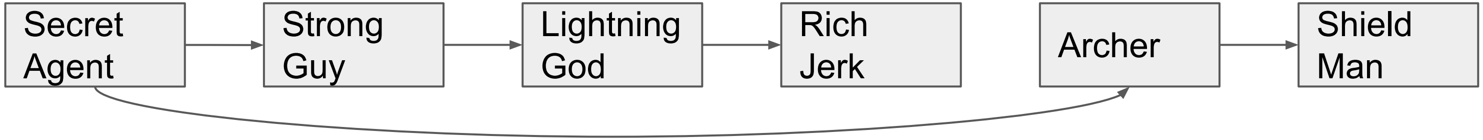


Remove inverted values by moving up the better value.

What would cause a problem in constructing an ordering in general?

Semi-Comparable Values Problem

If we had an ordering, what would make it invalid?



Remove inverted values by moving up the better value.

What would cause a problem in constructing an ordering in general?

A cycle

Topological Orderings

A topological ordering is an ordering of values such that there is no inversion.

Topological Orderings

A topological ordering is an ordering of values such that there is no inversion.

- All arrows will point from earlier values to strictly later ones.

Topological Orderings

A topological ordering is an ordering of values such that there is no inversion.

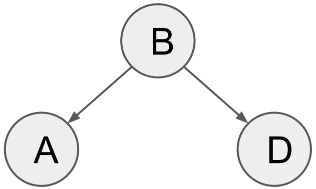
All arrows will point from earlier values to strictly later ones.

Do rooted trees have topological orderings?

Topological Orderings

A topological ordering is an ordering of values such that there is no inversion.

All arrows will point from earlier values to strictly later ones.

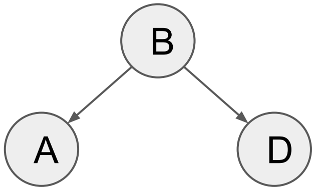


Do rooted trees have topological orderings?

Topological Orderings

A topological ordering is an ordering of values such that there is no inversion.

All arrows will point from earlier values to strictly later ones.



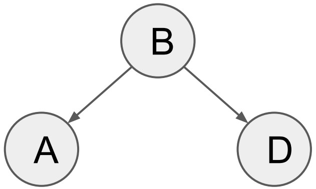
Do rooted trees have topological orderings?

Do topological orderings only exist if the graph is a rooted tree?

Topological Orderings

A topological ordering is an ordering of values such that there is no inversion.

All arrows will point from earlier values to strictly later ones.



Do rooted trees have topological orderings?

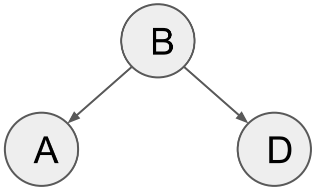
Do topological orderings only exist if the graph is a rooted tree?

A topological ordering is possible if and only if the graph is a DAG.

Topological Orderings

A topological ordering is an ordering of values such that there is no inversion.

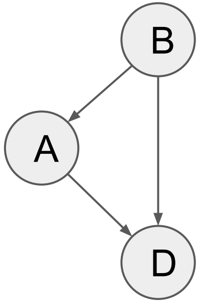
All arrows will point from earlier values to strictly later ones.



Do rooted trees have topological orderings?

Do topological orderings only exist if the graph is a rooted tree?

A topological ordering is possible if and only if the graph is a DAG.



Topo. Ordering Implies DAG

Part 1. If a topological ordering exists, then the Graph is a DAG.

Topo. Ordering Implies DAG

Part 1. If a topological ordering exists, then the Graph is a DAG.

Proof by contradiction. A topological ordering exists, and there is a cycle.

Topo. Ordering Implies DAG

Part 1. If a topological ordering exists, then the Graph is a DAG.

Proof by contradiction. A topological ordering exists, and there is a cycle.

Enumerate the cycle based on their occurrence in the topological ordering

$(a_1, a_2, a_3, \ldots , a_k)$.

Topo. Ordering Implies DAG

Part 1. If a topological ordering exists, then the Graph is a DAG.

Proof by contradiction. A topological ordering exists, and there is a cycle.

Enumerate the cycle based on their occurrence in the topological ordering

$(a_1, a_2, a_3, \dots , a_k)$.

In order for the nodes to form a cycle, there must exists an edge going from the last node (a_k) to an earlier node (a_i where $i < k$).

Topo. Ordering Implies DAG

Part 1. If a topological ordering exists, then the Graph is a DAG.

Proof by contradiction. A topological ordering exists, and there is a cycle.

Enumerate the cycle based on their occurrence in the topological ordering

$(a_1, a_2, a_3, \dots , a_k)$.

In order for the nodes to form a cycle, there must exists an edge going from the last node (a_k) to an earlier node (a_i where $i < k$).

This inversion proves the ordering is not a topological ordering.

Topo. Ordering Implies DAG

Part 1. If a topological ordering exists, then the Graph is a DAG.

Proof by contradiction. A topological ordering exists, and there is a cycle.

Enumerate the cycle based on their occurrence in the topological ordering

$(a_1, a_2, a_3, \ldots, a_k)$.

In order for the nodes to form a cycle, there must exists an edge going from the last node (a_k) to an earlier node (a_i where $i < k$).

This inversion proves the ordering is not a topological ordering. ⊥

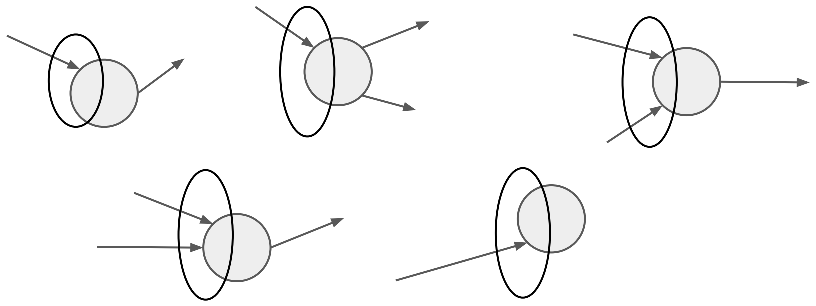
DAG Implies Topo. Ordering (Lemma)

Lemma. If a graph is a DAG, then some node has no incoming edges.

DAG Implies Topo. Ordering (Lemma)

Lemma. If a graph is a DAG, then some node has no incoming edges.

Proof by Contradiction. Suppose some DAG has no nodes with no incoming edges. All nodes has some incoming edge. Suppose the DAG has ***n*** nodes.

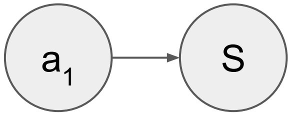


DAG Implies Topo. Ordering (Lemma)

Lemma. If a graph is a DAG, then some node has no incoming edges.

Proof by Contradiction. Suppose some DAG has no nodes with no incoming edges. All nodes has some incoming edge. Suppose the DAG has ***n*** nodes.

Start at an arbitrary node, and “DFS” backwards. Every node should be able to move along an incoming edge.



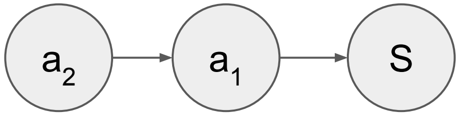
Allow revisiting nodes for this DFS...

DAG Implies Topo. Ordering (Lemma)

Lemma. If a graph is a DAG, then some node has no incoming edges.

Proof by Contradiction. Suppose some DAG has no nodes with no incoming edges. All nodes has some incoming edge. Suppose the DAG has ***n*** nodes.

Start at an arbitrary node, and “DFS” backwards. Every node should be able to move along an incoming edge.



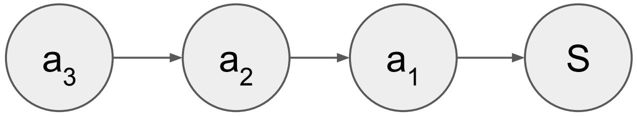
Allow revisiting nodes for this DFS...

DAG Implies Topo. Ordering (Lemma)

Lemma. If a graph is a DAG, then some node has no incoming edges.

Proof by Contradiction. Suppose some DAG has no nodes with no incoming edges. All nodes has some incoming edge. Suppose the DAG has ***n*** nodes.

Start at an arbitrary node, and “DFS” backwards. Every node should be able to move along an incoming edge.



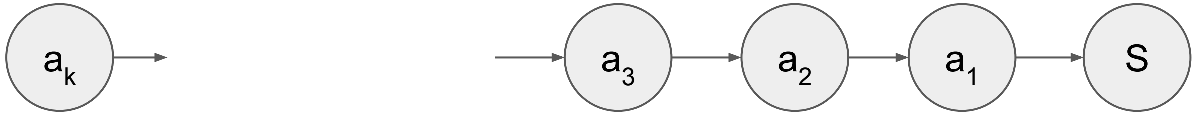
Allow revisiting nodes for this DFS...

DAG Implies Topo. Ordering (Lemma)

Lemma. If a graph is a DAG, then some node has no incoming edges.

Proof by Contradiction. Suppose some DAG has no nodes with no incoming edges. All nodes has some incoming edge. Suppose the DAG has ***n*** nodes.

Start at an arbitrary node, and "DFS" backwards. Every node should be able to move along an incoming edge.



Allow revisiting nodes for this DFS...

DAG Implies Topo. Ordering (Lemma)

Lemma. If a graph is a DAG, then some node has no incoming edges.

Proof by Contradiction. Suppose some DAG has no nodes with no incoming edges. All nodes has some incoming edge. Suppose the DAG has ***n*** nodes.

Start at an arbitrary node, and “DFS” backwards. Every node should be able to move along an incoming edge.

After DFSing ***n*** + 1 times, some node has to have been visited twice by Pigeonhole Principle.

DAG Implies Topo. Ordering (Lemma)

Lemma. If a graph is a DAG, then some node has no incoming edges.

Proof by Contradiction. Suppose some DAG has no nodes with no incoming edges. All nodes has some incoming edge. Suppose the DAG has ***n*** nodes.

Start at an arbitrary node, and “DFS” backwards. Every node should be able to move along an incoming edge.

After DFSing ***n*** + 1 times, some node has to have been visited twice by Pigeonhole Principle.

The two times in which this node has been visited form a cycle.

DAG Implies Topo. Ordering (Lemma)

Lemma. If a graph is a DAG, then some node has no incoming edges.

Proof by Contradiction. Suppose some DAG has no nodes with no incoming edges. All nodes has some incoming edge. Suppose the DAG has ***n*** nodes.

Start at an arbitrary node, and “DFS” backwards. Every node should be able to move along an incoming edge.

After DFSing ***n*** + 1 times, some node has to have been visited twice by Pigeonhole Principle.

The two times in which this node has been visited form a cycle. The graph is not a DAG.

DAG Implies Topo. Ordering (Lemma)

Lemma. If a graph is a DAG, then some node has no incoming edges.

Proof by Contradiction. Suppose some DAG has no nodes with no incoming edges. All nodes has some incoming edge. Suppose the DAG has $\boldsymbol{n}$ nodes.

Start at an arbitrary node, and “DFS” backwards. Every node should be able to move along an incoming edge.

After DFSing $\boldsymbol{n} + 1$ times, some node has to have been visited twice by Pigeonhole Principle.

The two times in which this node has been visited form a cycle. The graph is not a DAG. $\perp$

DAG Implies Topo. Ordering (Proof)

If a graph is a DAG, then it has a topological ordering.

DAG Implies Topo. Ordering (Proof)

If a graph is a DAG, then it has a topological ordering.

Proof by induction.

DAG Implies Topo. Ordering (Proof)

If a graph is a DAG, then it has a topological ordering.

Proof by induction.
Base Case. A DAG with 1 node is has an order by using just that node.

DAG Implies Topo. Ordering (Proof)

If a graph is a DAG, then it has a topological ordering.

Proof by induction.
Base Case. A DAG with 1 node is has an order by using just that node.

Induction Hypothesis. All DAGs with ***n*** nodes have a valid topological Order.

DAG Implies Topo. Ordering (Proof) (cont.)

Induction Step. Let G be a DAG with $\boldsymbol{n}$ + 1 nodes. The DAG (by the lemma) has a node, $\boldsymbol{x}$, with no incoming edges.

DAG Implies Topo. Ordering (Proof) (cont.)

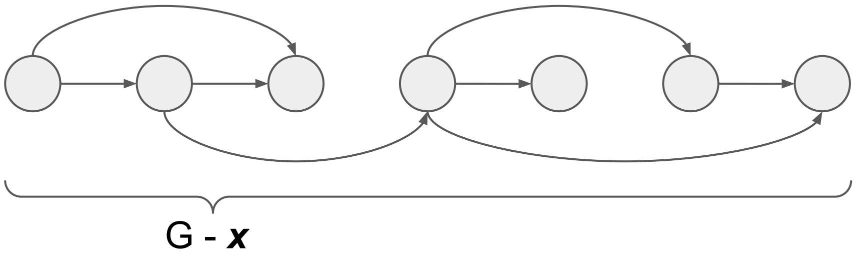
Induction Step. Let G be a DAG with $n + 1$ nodes. The DAG (by the lemma) has a node, x, with no incoming edges.

The graph G without node x is a DAG with n nodes, which has a topological order by the Induction hypothesis.

DAG Implies Topo. Ordering (Proof) (cont.)

Induction Step. Let G be a DAG with ***n*** + 1 nodes. The DAG (by the lemma) has a node, ***x***, with no incoming edges.

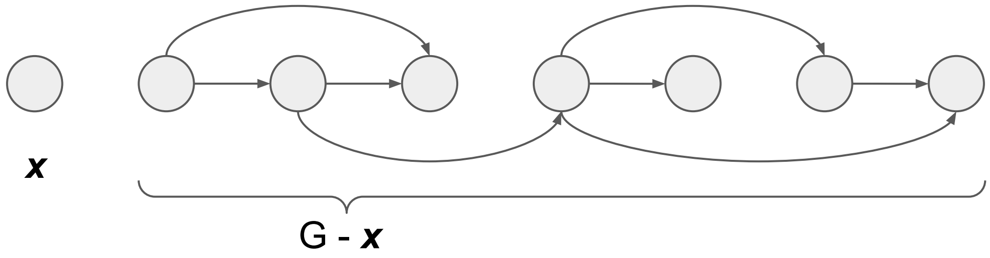
The graph G without node ***x*** is a DAG with ***n*** nodes, which has a topological order by the Induction hypothesis.



DAG Implies Topo. Ordering (Proof) (cont.)

Induction Step. Let G be a DAG with ***n*** + 1 nodes. The DAG (by the lemma) has a node, ***x***, with no incoming edges.

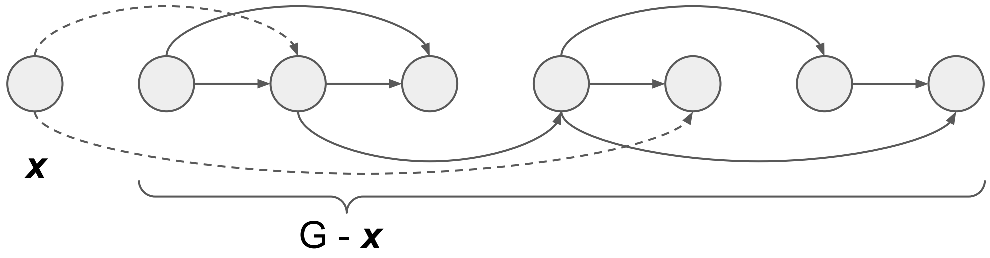
The graph G without node ***x*** is a DAG with ***n*** nodes, which has a topological order by the Induction hypothesis.



DAG Implies Topo. Ordering (Proof) (cont.)

Induction Step. Let G be a DAG with ***n*** + 1 nodes. The DAG (by the lemma) has a node, ***x***, with no incoming edges.

The graph G without node ***x*** is a DAG with ***n*** nodes, which has a topological order by the Induction hypothesis.



No edges added by including ***x*** will create a backwards edge.

DAG Implies Topo. Ordering (Proof) (cont.)

Induction Step. Let G be a DAG with ***n*** + 1 nodes. The DAG (by the lemma) has a node, ***x***, with no incoming edges.

The graph G without node ***x*** is a DAG with ***n*** nodes, which has a topological order by the Induction hypothesis.

No edges added by including ***x*** will create a backwards edge.

DAG Implies Topo. Ordering (Proof) (cont.)

Induction Step. Let G be a DAG with ***n*** + 1 nodes. The DAG (by the lemma) has a node, ***x***, with no incoming edges.

The graph G without node ***x*** is a DAG with ***n*** nodes, which has a topological order by the Induction hypothesis.

No edges added by including ***x*** will create a backwards edge.

The node ***x*** prepended to the ordering of G - ***x*** (i.e. all of G's nodes) will be a valid topological ordering.

DAG Implies Topo. Ordering (Proof) (cont.)

Induction Step. Let G be a DAG with ***n*** + 1 nodes. The DAG (by the lemma) has a node, ***x***, with no incoming edges.

The graph G without node ***x*** is a DAG with ***n*** nodes, which has a topological order by the Induction hypothesis.

No edges added by including ***x*** will create a backwards edge.

The node ***x*** prepended to the ordering of G - ***x*** (i.e. all of G's nodes) will be a valid topological ordering.

By induction a topological ordering will exist for DAGs with any natural number of nodes.

High-Level Kahn’s Algorithm

```
Add to a queue all nodes that have no incoming edges
While the queue has an element
   Let x be the first node of the queue
   Remove x from the queue
   Output x
   Remove x from the graph
   For every node, y, that NOW has no incoming edges
      Add y to the queue
   End For
End While
```

Kahn’s Algorithm Analysis

What would happen if there is no valid topological ordering?

Kahn's Algorithm Analysis

What would happen if there is no valid topological ordering?

How do we handle the inner for loop?

Kahn’s Algorithm Analysis

What would happen if there is no valid topological ordering?

How do we handle the inner for loop?

What is the runtime?

Low-Level Kahn’s Algorithm

We need to know the number of edge entering a node (i.e. in degree)

Low-Level Kahn’s Algorithm

We need to know the number of edge entering a node (i.e. in degree)

The removal + inner for loop should do the following,

Low-Level Kahn’s Algorithm

We need to know the number of edge entering a node (i.e. in degree)

The removal + inner for loop should do the following,

- Loop over the edges (x, y) that are leaving the node x.

Low-Level Kahn’s Algorithm

We need to know the number of edge entering a node (i.e. in degree)

The removal + inner for loop should do the following,

- Loop over the edges (x, y) that are leaving the node x.
  - Decrement the in degree for the y node

Low-Level Kahn’s Algorithm

We need to know the number of edge entering a node (i.e. in degree)

The removal + inner for loop should do the following,

- Loop over the edges (x, y) that are leaving the node x.
  - Decrement the in degree for the y node
  - If the in degree for the y node is 0 add it to the queue

DFS Approach

```
Array on stack is initialized to false
Array visited is initialized to false
DFS from ALL Nodes that have no incoming edges
Function DFS on Node n
   If visited n Then
      Return
   End If
   If n is on stack Then
      BADNESS
   End If
   Let on stack AND visited of n be TRUE
   For all Nodes x that can be reached by an edge leaving n
      DFS on Node x
   Add n to the front of the output list
   Let on stack be FALSE
End Function
```