Directed Acyclic Graphs and Topological Sorting
Duke COMPSCI 309s

Siyang Chen

Spring 2014
Introduction

Often when solving problems involving graphs, it’s useful to order the nodes in some way. Some orderings you may have already seen are:
Introduction

Often when solving problems involving graphs, it’s useful to order the nodes in some way. Some orderings you may have already seen are:

- Preorder, postorder, and inorder traversal for trees.
Introduction

Often when solving problems involving graphs, it’s useful to order the nodes in some way. Some orderings you may have already seen are:

- Preorder, postorder, and inorder traversal for trees.
- DFS traversal order
Introduction

Often when solving problems involving graphs, it’s useful to order the nodes in some way. Some orderings you may have already seen are:

- Preorder, postorder, and inorder traversal for trees.
- DFS traversal order
- Ordering by distance from the root (e.g. running BFS or Dijkstra)
Introduction

Often when solving problems involving graphs, it’s useful to order the nodes in some way. Some orderings you may have already seen are:

▸ Preorder, postorder, and inorder traversal for trees.
▸ DFS traversal order
▸ Ordering by distance from the root (e.g. running BFS or Dijkstra)

Today we’re going to look at a different type of ordering, called topological ordering.
Topological Sorting

A topological ordering of a directed graph arranges nodes in order such that edges only go from left to right.
Topological Sorting

A topological ordering of a directed graph arranges nodes in order such that edges only go from left to right.
Topological Sorting

A topological ordering of a directed graph arranges nodes in order such that edges only go from left to right.

This is not always possible:
Topological Sorting

A topological ordering of a directed graph arranges nodes in order such that edges only go from left to right.

This is not always possible:
Topological Sorting

Theorem

A topological ordering exists if and only if the graph is acyclic.
Topological Sorting

Theorem

A topological ordering exists if and only if the graph is acyclic.

Proof.
**Topological Sorting**

**Theorem**

* A topological ordering exists if and only if the graph is acyclic.

**Proof.**

- Suppose there is a cycle \( v_0, v_1, \ldots, v_m \). Then we can’t order the \( v_i \) because the last node will always point to a previous node.
Topological Sorting

Theorem

A topological ordering exists if and only if the graph is acyclic.

Proof.

- Suppose there is a cycle \( v_0, v_1, \ldots, v_m \). Then we can’t order the \( v_i \) because the last node will always point to a previous node.

- Now suppose the graph is acyclic. Then there must be a node \( v' \) with no parents. Put \( v' \) into the list and delete it from the graph. The remaining graph also doesn’t have any cycles. Thus, we can keep on plucking out parent-less nodes and placing them into our list until we’ve ordered the entire graph.

\[ \square \]
Topological Sorting

Theorem

A topological ordering exists if and only if the graph is acyclic.

Proof.

- Suppose there is a cycle \( v_0, v_1, \ldots, v_m \). Then we can’t order the \( v_i \) because the last node will always point to a previous node.

- Now suppose the graph is acyclic. Then there must be a node \( v' \) with no parents. Put \( v' \) into the list and delete it from the graph. The remaining graph also doesn’t have any cycles. Thus, we can keep on plucking out parent-less nodes and placing them into our list until we’ve ordered the entire graph.

The process described in the second part above is an algorithm called topological sort, which places nodes in topological order.
Topological Sorting

Pseudocode for topological sort:

- \( \text{parentCount}_i \leftarrow \) number of parents of node \( i \)
- \( \text{queue} \leftarrow \) all nodes with no parents
- While \( i < \text{length}(\text{queue}) \):
  - For all children \( j \) of \( \text{queue}_i \):
    - \( \text{parentCount}_j \leftarrow \text{parentCount}_j - 1 \)
    - If \( \text{parentCount}_j = 0 \), add \( j \) to \( \text{queue} \)
  - Return \( \text{queue} \)
Topological Sorting

Pseudocode for topological sort:

- \( parentCount_i \leftarrow \text{number of parents of node } i \)
- \( queue \leftarrow \text{all nodes with no parents} \)
- \( \text{While } i < \text{length}(queue): \)
  - \( \text{For all children } j \text{ of } queue_i: \)
    - \( parentCount_j \leftarrow parentCount_j - 1 \)
    - \( \text{If } parentCount_j = 0, \text{ add } j \text{ to } queue \)

- \( \text{Return } queue \)

But what happens if there’s a cycle?
Topological Sorting

Let’s look at an example of what happens when we try to topologically sort a graph with a cycle:

- Initially, parentCount = [0; 2; 1] and queue = [1].
- We simulate deleting 1 from the graph by decrementing parentCount2. Then parentCount = [0; 1; 1] and queue = [1].
- At this point, the algorithm terminates because we’ve reached the end of the queue.

So it looks like what happens is the algorithm terminates without adding all the nodes to the queue.
Topological Sorting

Let’s look at an example of what happens when we try to topologically sort a graph with a cycle:
Topological Sorting

Let’s look at an example of what happens when we try to topologically sort a graph with a cycle:

- Initially \( \text{parentCount} = [0, 2, 1] \) and \( \text{queue} = [1] \)
Topological Sorting

Let’s look at an example of what happens when we try to topologically sort a graph with a cycle:

![Graph Diagram]

- Initially \( \text{parentCount} = [0, 2, 1] \) and \( \text{queue} = [1] \)
- We simulate deleting 1 from the graph by decrementing \( \text{parentCount}_2 \). Then \( \text{parentCount} = [0, 1, 1] \) and \( \text{queue} = [1] \).
Topological Sorting

Let’s look at an example of what happens when we try to topologically sort a graph with a cycle:

- Initially $parentCount = [0, 2, 1]$ and $queue = [1]$
- We simulate deleting 1 from the graph by decrementing $parentCount_2$. Then $parentCount = [0, 1, 1]$ and $queue = [1]$.
- At this point the algorithm terminates because we’ve reached the end of the queue.
Topological Sorting

Let’s look at an example of what happens when we try to topologically sort a graph with a cycle:

![Diagram of a graph with a cycle](image)

Let's look at what happens when we try to topologically sort a graph with a cycle:

- Initially, `parentCount = [0, 2, 1]` and `queue = [1]`.
- We simulate deleting 1 from the graph by decrementing `parentCount_2`. Then `parentCount = [0, 1, 1]` and `queue = [1]`.
- At this point, the algorithm terminates because we've reached the end of the queue.

So it looks like what happens is the algorithm terminates without adding all the nodes to the queue.
Topological Sorting

Pseudocode for topological sort which throws an exception when a cycle is detected:

- `parentCount_i` $\leftarrow$ number of parents of node $i$
- `queue` $\leftarrow$ all nodes with no parents
- While $i < \text{length}(queue)$:
  - For all children $j$ of `queue_i`:
    - `parentCount_j` $\leftarrow$ `parentCount_j` $-$ 1
    - If `parentCount_j` $=$ 0, add $j$ to `queue`
  - If `length(queue)` $<$ $j$, return CYCLE DETECTED
  - Otherwise return `queue`
Topological Sorting

Pseudocode for topological sort which throws an exception when a cycle is detected:

- $parentCount_i \leftarrow$ number of parents of node $i$
- $queue \leftarrow$ all nodes with no parents
- While $i < \text{length}(queue)$:
  - For all children $j$ of $queue_i$:
    - $parentCount_j \leftarrow parentCount_j - 1$
    - If $parentCount_j = 0$, add $j$ to $queue$
  - If $\text{length}(queue) < |V|$, return $CYCLE\_DETECTED$
Topological Sorting

Pseudocode for topological sort which throws an exception when a cycle is detected:

- \( parentCount_i \leftarrow \) number of parents of node \( i \)
- \( queue \leftarrow \) all nodes with no parents
- While \( i < \text{length}(queue) \):
  - For all children \( j \) of \( queue_i \):
    - \( parentCount_j \leftarrow parentCount_j - 1 \)
    - If \( parentCount_j = 0 \), add \( j \) to \( queue \)
  - If \( \text{length}(queue) < |V| \), return \text{CYCLE(DETECTED)}
  - Otherwise return \( queue \)
Topological Sorting

Pseudocode for topological sort which throws an exception when a cycle is detected:

- \( \text{parentCount}_i \leftarrow \) number of parents of node \( i \)
- \( \text{queue} \leftarrow \) all nodes with no parents
- While \( i < \text{length}(\text{queue}) \):
  - For all children \( j \) of \( \text{queue}_i \):
    - \( \text{parentCount}_j \leftarrow \text{parentCount}_j - 1 \)
    - If \( \text{parentCount}_j = 0 \), add \( j \) to \( \text{queue} \)
  - If \( \text{length}(\text{queue}) < |V| \), return CYCLE_DETECTED
  - Otherwise return \( \text{queue} \)

Performance?
Topological Sorting

Pseudocode for topological sort which throws an exception when a cycle is detected:

- `parentCount_i` ← number of parents of node `i`
- `queue` ← all nodes with no parents
- While `i < length(queue)`:
  - For all children `j` of `queue_i`:
    - `parentCount_j` ← `parentCount_j` − 1
    - If `parentCount_j` = 0, add `j` to `queue`
  - If `length(queue) < |V|`, return `CYCLE_DETECTED`
  - Otherwise return `queue`

Performance? $O(|V| + |E|)$
Now let’s look at how we can use topological sorting to solve this problem from last week’s problem set. (You don’t have to use topological sorting, but it’s a useful approach to the problem.)
Now let’s look at how we can use topological sorting to solve this problem from last week’s problem set. (You don’t have to use topological sorting, but it’s a useful approach to the problem.)

Problem link:
http://codeforces.com/problemset/problem/374/C
As with any graph problem, we should first define what our edges and nodes are:
Codeforces: Inna and Dima

As with any graph problem, we should first define what our edges and nodes are:

- Nodes are table cells, and edges are the moves that Inna can make (e.g. from $D$ to $I$, $I$ to $M$, etc...).
As with any graph problem, we should first define what our edges and nodes are:

- Nodes are table cells, and edges are the moves that Inna can make (e.g. from $D$ to $I$, $I$ to $M$, etc...).

How do we detect whether or not Inna will go in a cycle?
As with any graph problem, we should first define what our edges and nodes are:

- Nodes are table cells, and edges are the moves that Inna can make (e.g. from \(D\) to \(I\), \(I\) to \(M\), etc...).

How do we detect whether or not Inna will go in a cycle?

- Topological sort
As with any graph problem, we should first define what our edges and nodes are:

- Nodes are table cells, and edges are the moves that Inna can make (e.g. from $D$ to $I$, $I$ to $M$, etc...).

How do we detect whether or not Inna will go in a cycle?

- Topological sort

So we have a topologically sorted list of cells. What do we do now?
We want to find, for each $D$, the longest path that Inna can take starting from that $D$. 
We want to find, for each $D$, the longest path that Inna can take starting from that $D$. Since we have the edges pointing from left to right, we can just do something like the following:

We want to find, for each $D$, the longest path that Inna can take starting from that $D$. Since we have the edges pointing from left to right, we can just do something like the following:

- For $i$ from $|V| - 1$ to $0$:
  - Let $x, y = queue_i$
  - For $x', y'$ such that $(x, y) \rightarrow (x', y')$ is a valid move:
    - $best_{x, y} \leftarrow \max_{x', y'} \{best_{x, y}, best_{x', y'}\}$
    - $best_{x, y} \leftarrow best_{x, y} + 1$
  - $ans \leftarrow \max\{ans, best_{x, y}\}$
We want to find, for each $D$, the longest path that Inna can take starting from that $D$. Since we have the edges pointing from left to right, we can just do something like the following:

- For $i$ from $|V| - 1$ to $0$:
  - Let $x, y = \text{queue}_i$;
  - For $x', y'$ such that $(x, y) \rightarrow (x', y')$ is a valid move:
    - $\text{best}_{x,y} \leftarrow \max_{x', y'} \{\text{best}_{x,y}, \text{best}_{x',y'}\}$
    - $\text{best}_{x,y} \leftarrow \text{best}_{x,y} + 1$
    - $\text{ans} \leftarrow \max \{\text{ans}, \text{best}_{x,y}\}$