Algorithm Design and Analysis

Lecture 6

- Topological ordering
- Greedy Algorithms

Adam Smith

9/5/2008

A. Smith; based on slides by E. Demaine, C. Leiserson, S. Raskhodnikova, K. Wayne
Notes

• Read on your own
  – Graph bipartiteness
  – DFS implementation

• Homework notation:
  – $\log^a(n) = (\log n)^a$
  – $\lceil x \rceil = \text{smallest integer } \geq x$
    = “ceiling”
  – Useful property: $x \leq \lceil x \rceil < x + 1$

9/5/2008

A. Smith; based on slides by E. Demaine, C. Leiserson, S. Raskhodnikova, K. Wayne
Last lecture: BFS

• Recall: Digraph G is strongly connected if for every pair of vertices, \( s \sim t \) and \( s \sim t \)

• Question: Give an algorithm for determining if a graph is connected. What is the running time?
Strong Connectivity: Algorithm

**Lemma:** $G$ is strongly connected if and only if for any node $s$, every other node $t$ has paths to and from $s$.

**Theorem.** Can determine if $G$ is strongly connected in $O(m + n)$ time.

**Pf.**
- Pick any node $s$.
- Run BFS from $s$ in $G$.
- Run BFS from $s$ in $G^{rev}$.
- Return true iff all nodes reached in both BFS executions.
- Correctness follows immediately from the previous lemma.

![Diagram showing strongly connected and not strongly connected graphs](image)
Directed Acyclic Graphs

**Def.** An DAG is a directed graph that contains no directed cycles.

**Ex.** Precedence constraints: edge \((v_i, v_j)\) means \(v_i\) must precede \(v_j\).

**Def.** A topological order of a directed graph \(G = (V, E)\) is an ordering of its nodes as \(v_1, v_2, ..., v_n\) so that for every edge \((v_i, v_j)\) we have \(i < j\).
Precedence Constraints

Precedence constraints. Edge \((v_i, v_j)\) means task \(v_i\) must occur before \(v_j\).

Applications.

- Course prerequisite graph: course \(v_i\) must be taken before \(v_j\).
- Compilation: module \(v_i\) must be compiled before \(v_j\). Pipeline of computing jobs: output of job \(v_i\) needed to determine input of job \(v_j\).
**Directed Acyclic Graphs**

**Lemma.** If $G$ has a topological order, then $G$ is a DAG.

**Pf.** (by contradiction)
- Suppose that $G$ has a topological order $v_1, \ldots, v_n$ and that $G$ also has a directed cycle $C$. Let's see what happens.
- Let $v_i$ be the lowest-indexed node in $C$, and let $v_j$ be the node just before $v_i$; thus $(v_j, v_i)$ is an edge.
- By our choice of $i$, we have $i < j$.
- On the other hand, since $(v_j, v_i)$ is an edge and $v_1, \ldots, v_n$ is a topological order, we must have $j < i$, a contradiction. □
Directed Acyclic Graphs

**Lemma.** If $G$ has a topological order, then $G$ is a DAG.

**Q.** Does every DAG have a topological ordering?

**Q.** If so, how do we compute one?
**Directed Acyclic Graphs**

**Lemma.** If $G$ is a DAG, then $G$ has a node with no incoming edges.

**Pf.** (by contradiction)

- Suppose that $G$ is a DAG and every node has at least one incoming edge. Let's see what happens.
- Pick any node $v$, and begin following edges backward from $v$. Since $v$ has at least one incoming edge $(u, v)$ we can walk backward to $u$.
- Then, since $u$ has at least one incoming edge $(x, u)$, we can walk backward to $x$.
- Repeat until we visit a node, say $w$, twice.
- Let $C$ denote the sequence of nodes encountered between successive visits to $w$. $C$ is a cycle. □
Directed Acyclic Graphs

Lemma. If $G$ is a DAG, then $G$ has a topological ordering.

Pf. (by induction on $n$)

- **Base case:** true if $n = 1$.
- **Given DAG on** $n > 1$ **nodes, find a node** $v$ **with no incoming edges.**
- $G - \{ v \}$ **is a DAG**, since deleting $v$ cannot create cycles.
- **By inductive hypothesis**, $G - \{ v \}$ has a topological ordering.
- Place $v$ first in topological ordering; then append nodes of $G - \{ v \}$ in topological order. This is valid since $v$ has no incoming edges.

---

To compute a topological ordering of $G$:
Find a node $v$ with no incoming edges and order it first
Delete $v$ from $G$
Recursively compute a topological ordering of $G - \{ v \}$
and append this order after $v$
Topological Sorting Algorithm: Running Time

Theorem. Algorithm finds a topological order in $O(m + n)$ time.

Pf.

- Maintain the following information:
  - $\text{count}[w] = \text{remaining number of incoming edges}$
  - $S = \text{set of remaining nodes with no incoming edges}$
- Initialization: $O(m + n)$ via single scan through graph.
- Update: to delete $v$
  - remove $v$ from $S$
  - decrement $\text{count}[w]$ for all edges from $v$ to $w$, and add $w$ to $S$ if $c_{\text{count}[w]}$ hits 0
  - this is $O(1)$ per edge

Alternative algorithm: Modification of DFS (exercise?)
Design technique #1: Greedy Algorithms
Greedy Algorithms

• Build up a solution to an optimization problem at each step shortsightedly choosing the option that currently seems the best.
  – Sometimes good
  – Often does not work
Interval Scheduling Problem

- Job \(j\) starts at \(s_j\) and finishes at \(f_j\).
- Two jobs are **compatible** if they do not overlap.
- **Find**: maximum subset of mutually compatible jobs.

![Diagram of intervals](image-url)
Possible Greedy Strategies

Consider jobs in some natural order. Take next job if it is compatible with the ones already taken.

- **Earliest start time:** ascending order of $s_j$.
- **Earliest finish time:** ascending order of $f_j$.
- **Shortest interval:** ascending order of $(f_j - s_j)$.
- **Fewest conflicts:** For each job $j$, count the number of conflicting jobs $c_j$. Schedule in ascending order of $c_j$. 

A. Smith; based on slides by E. Demaine, C. Leiserson, S. Raskhodnikova, K. Wayne
Greedy: Counterexamples

for earliest start time

for shortest interval

for fewest conflicts
Next lecture

- We will see that adding jobs greedily in order of earliest finishing time gives an optimal solution.