Design and Analysis of Algorithms

Hoeteck Wee · hoeteck@cs.qc.cuny.edu
http://www.cs.qc.edu/~hoeteck/f09/
Directed graphs

BASICS. \( n \) nodes/vertices, \( m \) directed edges
  ▶ e.g. nodes = web pages, edges = hyperlinks

REPRESENTATION. adjacency list, each node has two associated lists:
  nodes to which it has edges and nodes from which it has edges

DIRECTED REACHABILITY.
  ▶ given a node \( s \), find all nodes reachable from \( s \).
  ▶ e.g. in web crawler, find all web pages linked from \( s \).

DIRECTED S-T SHORTEST PATH PROBLEM.
  ▶ given two nodes \( s \) and \( t \), what is the shortest path from \( s \) to \( t \)?

BREADTH-FIRST SEARCH. layer-by-layer outward search from \( s \)
  ▶ layer \( L_i \) : nodes where the shortest path from \( s \) has exactly \( i \) edges
  ▶ running time \( O(m + n) \)
DEFINITIONS.

- nodes $u$ and $v$ are mutually reachable if $\exists$ a path from $u$ to $v$ and from $v$ to $u$
- a directed graph is strongly connected if every pair of nodes is mutually reachable
- the strong component of a node $s$ is the set of nodes $v$ s.t. $s$ and $v$ are mutually reachable

FACT. for any two nodes $s$ and $t$ in a directed graph, their strong components are either identical or disjoint.

- mutual reachability is an equivalence relation

ALGORITHM. can find the strong component of a node in $O(m + n)$ time.

- Run BFS in $G$ and $G^{REV}$, compute the intersection.
- Can find all strong components in $O(m + n)$ time.
- Can determine if $G$ is strongly connected in $O(m + n)$ time.
**Definition.** A DAG is a directed graph that contains no (directed) cycles.

- typically precedence relations or dependencies, e.g. courses & prerequisites

**Definition.** A topological ordering of a DAG is an ordering $v_1, \ldots, v_n$ of its nodes so that for every edge $(v_i, v_j)$, we have $i < j$.

- i.e. all edges point from left to right
- e.g. scheduling lectures so that all dependencies are respected

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

- converse is also true! – can we compute the ordering efficiently?
PROBLEM. given a DAG, compute a topological ordering.

- getting started: node $v_1$ and incident edges.

FACT. in every DAG, there exists a node $v$ with no incoming edges.

- suppose otherwise: every node has in-degree $\geq 1$
- fix any node $v$, and keep walking backwards
- must visit some node $w$ twice
- yields a cycle, a contradiction!

ALGORITHM.

- find a node $v$ with in-degree 0
- delete $v$ and recurse on $G - \{v\}$
Topological sort

(a) Diagram of a directed graph with vertices labeled $v_1$ to $v_7$ and edges indicating dependencies.

(b) Another diagram of a directed graph with vertices labeled $v_2$ to $v_5$ and edges indicating dependencies.

(c) A simplified diagram with vertices labeled $v_3$ and $v_4$ with dependencies.

(d) Diagram with vertices $v_6$ to $v_7$ and dependencies.

(e) Diagram with vertices $v_6$ to $v_5$ and dependencies.

(f) A simplified diagram with single vertex $v_6$ and dependency.
Topological sort

ALGORITHM.

- find a node $v$ with in-degree 0
- delete $v$ and recurse on $G - \{v\}$

IMPLEMENTATION.

- maintain $\text{indegree}[w]$ and set of undeleted nodes
- initializing $\text{indegree}[w]$: $O(m + n)$ time
- deleting $v$ and updating indegree takes $O(\text{outdeg}(v))$ time
- finding a new node with in-degree 0? (maintain a queue of in-degree 0 nodes)
- overall running time $O(m + n)$
GREEDY ALGORITHMS
First representative problem

INTERVAL SCHEDULING.

- Input: set of jobs with start and finish times.
- Find **max cardinality** subset of compatible (i.e. non-overlapping) jobs.

GREEDY TEMPLATE. use a simple rule to pick the first job and repeat, ignoring incompatible jobs
Interval scheduling

**GREEDY TEMPLATE.** use a simple rule to pick the first job and repeat, ignoring incompatible jobs

- attempt 1: earliest start time
- attempt 2: shortest interval
- attempt 3: fewest conflicts
- “correct” approach: earliest finish time
Interval scheduling

Intervals numbered in order

Selecting interval 1

Selecting interval 3

Selecting interval 5

Selecting interval 8
**THEOREM.** Greedy algorithm is optimal.

- **GREEDY:** jobs $i_1, i_2, \ldots, i_k$
- **OPT:** jobs $j_1, j_2, \ldots, j_m$

**CLAIM.** **GREEDY** stays head, i.e.: for all $r$, $f(i_r) \leq f(j_r)$

- **inductive step:** suppose $f(i_{r-1}) \leq f(j_{r-1})$
- **observation:** if $f(i_r) > f(j_r)$, then **GREEDY** will pick $j_r$ as $r$'th interval
- **follows from claim that** $k \geq m$
THE END