CMPSC 250

Analysis of algorithms

Dr. Aravind Mohan

Allegheny College
Topological Sort Algorithm

- We have a set of tasks and a set of dependencies (precedence constraints) of form “task A must be done before task B”
- Topological sort: An ordering of the tasks that conforms with the given dependencies
- Goal: Find a topological sort of the tasks or decide that there is no such ordering
Topological Sort Applications

- Scheduling: When scheduling task graphs in distributed systems, usually we first need to sort the tasks topologically and then assign them to resources (the most efficient scheduling is an NP-complete problem)
- Or during compilation to order modules/libraries
Topological Sort Applications

- Resolving dependencies: apt-get uses topological sorting to obtain the admissible sequence in which a set of Debian packages can be installed/removed
Topological Sort More Formally

- Suppose that in a directed graph $G = (V, E)$ vertices $V$ represent tasks, and each edge $(u, v) \in E$ means that task $u$ must be done before task $v$.
- What is an ordering of vertices $1, \cdots, |V|$ such that for every edge $(u, v)$, $u$ appears before $v$ in the ordering?
- Such an ordering is called a topological sort of $G$.

Note: there can be multiple topological sorts of $G$. 
Is it possible to execute all the tasks in G in an order that respects all the precedence requirements given by the graph edges?

The answer is "yes" if and only if the directed graph G has no cycle! (otherwise we have a deadlock)

Such a G is called a Directed Acyclic Graph, or just a DAG.
Topological Sort Algorithm

TOPOLOGICAL-SORT(G):

- call DFS(G) to compute finishing times $f[v]$ for each vertex $v$
- as each vertex is finished, insert it onto the front of a linked list
- return the linked list of vertices

Note that the result is just a list of vertices in order of decreasing finish times $f[]$
Topological Sort Example
Topological Sort Example

- Topological order [3, 0, 4, 1, 2, 6, 5, 7]
Topological Sort Running Time

- Same as DFS running time $O(|V| + |E|)$
Strong components

- Vertices $v$ and $w$ are strongly connected if there is a directed path from $v$ to $w$ and a directed path from $w$ to $v$
- A strong component is the maximal subset of strongly-connected vertices
Connected vs. strongly-connected components

- **Connected components**: v and w are connected if there is a path between v and w.
- **Strongly-connected components**: v and w are strongly connected if there is a directed path from v to w and a directed path from w to v.

### Connected component id (easy to compute with DFS)

- cc[] = 0 0 0 0 0 1 1 1 2 2 2 2

### Strongly-connected component id (how to compute?)

- scc[] = 1 0 1 1 1 1 3 4 3 2 2 2

### Code Examples

- Public int connected(int v, int w) {
  return cc[v] == cc[w];
}

- Public int stronglyConnected(int v, int w) {
  return scc[v] == scc[w];
}

**Constant-time client connectivity query**

**Constant-time client strong-connectivity query**
Strong component application: ecological food webs

- **Food web graph:** Vertex = species, edge = from producer to consumer
- **Strong component:** Subset of species with common energy flow
Kosaraju Algorithm Example
Kosaraju Algorithm

- Initialize counter $c := 0$
- While not all nodes are labeled:
  - Choose an arbitrary unlabeled node $v$
  - Start DFS from $v$
    - Check the current node $x$ as visited
    - Recurse on all unvisited neighbors
    - After the DFS calls are finished, increment $c$ and set the label of $x$ as $c$
- Reverse the direction of all the edges
- For node $v$ with label $n, n-1, \ldots, 1$:
  - Find all reachable nodes from $v$ and group them as an SCC

Running Time: Same as DFS running time $O(|V| + |E|)$
Chapter 4: pages 578 - 594
Class activity, post your solutions in slack

Given graph above, apply kosaraju algorithm.
Class activity, post your solutions in slack

Given graph above, apply topological sort algorithm.