Topological Sort

CSE 5311: Design and Analysis of Algorithms

Alex Dillhoff

The University of Texas at Arlington

Topological Sort

Topological Sort

A **topological sort** of a directed acyclic graph $G = (V, E)$ is a linear ordering of all its vertices such that for every directed edge $(u, v) \in E$, vertex u comes before vertex v in the ordering.

Topological Sort

The process itself can be described simply:

1. Call DFS(G) to compute the finishing times for each vertex v.
2. As each vertex is finished, insert it onto the front of a linked list.
3. Return the linked list of vertices.

Topological Sort

The entire call takes $\Theta(V + E)$ since DFS(G) takes $\Theta(V + E)$ time.

Inserting each vertex onto the front of the list can be done in constant time.

Topological Sort

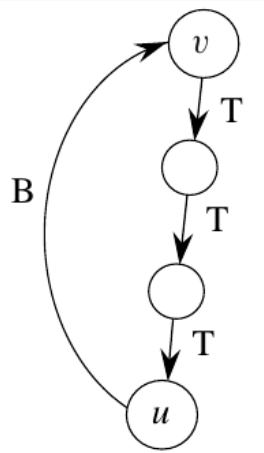
Lemma

A directed graph G is acyclic if and only if a DFS of G yields no *back edges* – an edge (u, v) such that v is an ancestor of u in the DFS forest.

Proof

The proof is by contradiction: if a back edge exists, then there is a cycle in the graph.

Topological Sort



An example of a back edge in a graph.

Proof

- Suppose there is a back edge (u, v).
- In this case, v is an ancestor of u in the depth-first forest.
- There is a path $v \rightsquigarrow u$, so $v \rightsquigarrow u \to v$ is a cycle.

Proof

- In the other direction, suppose that G contains a cycle c.
- Let v be the first vertex discovered in c, and let (u, v) be the preceding edge in c.
- At time $v.d$, vertices of c form a white path $v \rightsquigarrow u$.
- Since u is a descendant of v, (u, v) is a back edge. ■

Theorem

The topological sort algorithm produces a topological sort of a directed acyclic graph.

Topological Sort

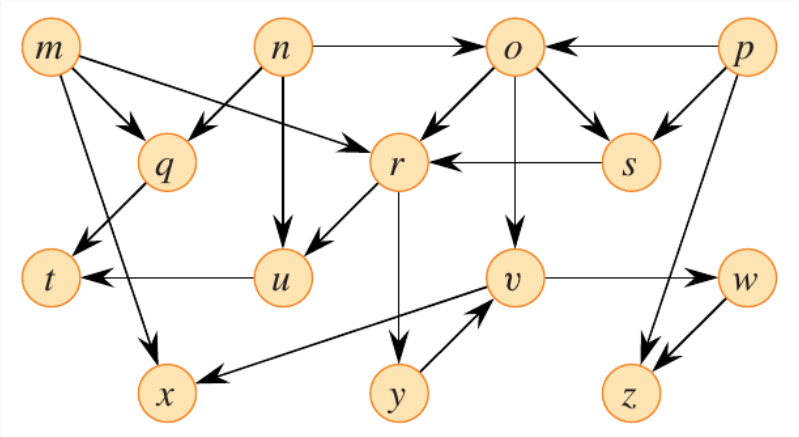
Theorem

The topological sort algorithm produces a topological sort of a directed acyclic graph.

- Run a DFS on the graph G to determine finish times for its vertices.
- For any pair of vertices (u, v), if G contains an edge from u to v, then $v.f < u.f$.

Topological Sort

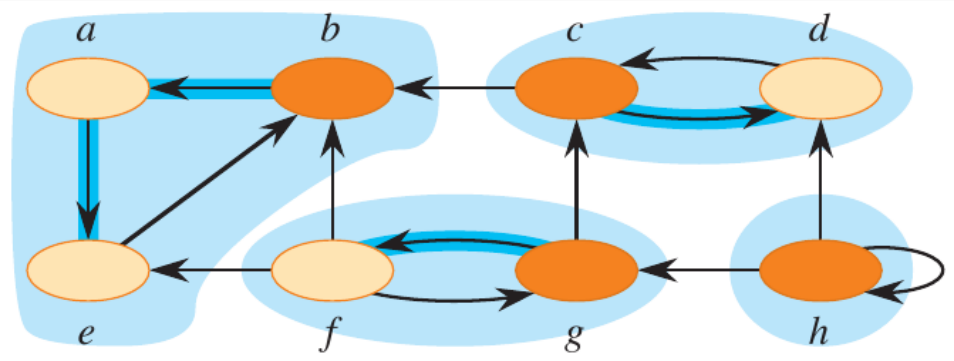
Example: Perform a topological sort on the following graph.



An example of a topological sort.

Strongly Connected Components

Strongly Connected Components



An example of strongly connected components.

Strongly Connected Components

A **strongly connected component** of a directed graph G is a maximal set of vertices such that for every pair of vertices u and v in the set, there is a path from u to v and a path from v to u.

Strongly Connected Components

The algorithm goes as follows:

1. Call DFS(G) to compute the finishing times for each vertex v.
2. Compute the transpose of G.
3. Call DFS(G^T), but in the main loop of DFS, consider the vertices in order of decreasing finishing times.
4. Output the vertices of each tree in the depth-first forest as a separate strongly connected component.

Strongly Connected Components

The transpose of a graph G^T is the graph G with all edges reversed.

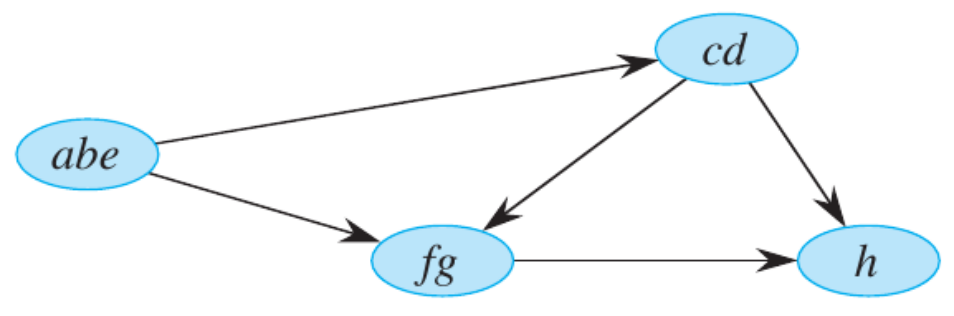
$$G^T = (V, E^T)$$
$$E^T = \{(v, u) \mid (u, v) \in E\}$$

Component Graphs

The resulting **component graph** is a directed graph $G_{SCC} = (V_{SCC}, E_{SCC})$ where each vertex represents a strongly connected component of the original graph G.

There is an edge (C_i, C_j) in G_{SCC} if there is a vertex $u \in C_i$ and a vertex $v \in C_j$ such that $(u, v) \in E$.

Component Graphs



An example of a component graph.

Lemma

The component graph G^{SCC} is a directed acyclic graph. Let C and C' be distinct strongly connected components in G, where $u, v \in C$ and $u', v' \in C'$, and suppose there is a path $u \rightsquigarrow u'$ in G.

Then there cannot also be a path $v' \rightsquigarrow v$ in G.

Proof

- Suppose there is a path $v' \rightsquigarrow v$ in G.
- This implies there are paths $u \rightsquigarrow u' \rightsquigarrow v'$ and $v' \rightsquigarrow v \rightsquigarrow u$.
- If this were possible, then u and v' are reachable from each other, which contradicts the assumption that C and C' are distinct strongly connected components. ■

Component Graphs

Exercise: Compute the component graph...

Finishing Times

The previous and following lemmas establish, given the algorithm presented above, the rules of the finishing times of strongly connected components.

These are used to prove the correctness of the algorithm.

Finishing Times

Lemma

Let C and C' be strongly connected components in a directed graph G.

If there is an edge $(u, v) \in E$ such that $u \in C$ and $v \in C'$, then $f(C) > f(C')$.

Proof: Case 1

- If $d(C) < d(C')$, let x be the first vertex discovered in C. At time $x.d$, the time of discovery, all vertices in C and C' are white. Thus, there exists paths of white vertices from x to all vertices in C and C'.
- By the **white-path theorem**, all vertices in C and C' are descendants of x in the depth-first tree.
- By the parenthesis theorem, $x.f = f(C) > f(C')$.

Proof: Case 2

- If $d(C) > d(C')$, let y be the first vertex discovered in C'. At time $y.d$, all vertices in C and C' are white. Thus, there exists paths of white vertices from y to all vertices in C'. All vertices in C' become descendants of y. Again, $y.f = f(C')$.
- At time $y.d$, all vertices in C are also white.
- Since there is an edge (u, v), where $u \in C$ and $u' \in C'$, we cannot have a path from C' to C.
- No vertex in C is reachable from y.
- Therefore, at time $y.f$, all vertices in C are white.
- Therefore, for all $w \in C, w.f > y.f$, which implies that $f(C) > f(C')$. ■

Finishing Times

Corollary

Let C and C' be distinct strongly connected components in G.

Suppose there is an edge $(u, v) \in E^T$, where $u \in C$ and $v \in C'$.

Then $f(C) < f(C')$.

Proof

- $(u, v) \in E^T \implies (v, u) \in E$
- Since strongly connected components of G and G^T are the same, $f(C') > f(C)$.

Corollary

Let C and C' be distinct strongly connected components in G, and suppose that $f(C) > f(C')$.

Then there cannot be an edge from C to C' in G^T.

Correctness

Proof

- When we perform the second DFS call, on G^T, it starts with the component C such that $f(C)$ is the maximum.
  - This call starts from some $x \in C$ and explores all vertices in C.
  - The corollary says that since $f(C) > f(C')$, there cannot be an edge from C to C' in G^T.
  - Therefore, DFS will visit *only* vertices in C.
  - This means that the depth-first tree rooted at x will contain only vertices in C.

Correctness

Proof

- The next root chosen is in C' such that $f(C')$ is maximum over all strongly connected components **other than** C.
  - DFS visits all vertices in C', but the only edges out of C' go to C, **which have already been visited.**
  - Therefore, the only tree edges will be to vertices in C'.
- Continue the process...
- Each root chosen for the second DFS can reach only
  - vertices in its own strongly connected component, and
  - vertices in strongly connected components /already visited/ in the second DFS.

Application: Recommender Graphs

Recommender Graphs

A recommender graph is a directed graph where each vertex represents an item and each edge represents a transition between the items based on the context of the data.

Recommender Graphs

A recommender graph is a directed graph where each vertex represents an item and each edge represents a transition between the items based on the context of the data.

For example, in a movie recommender graph, each vertex represents a movie, and an edge from u to v indicates that users typically transitioned from watching movie u to watching movie v.

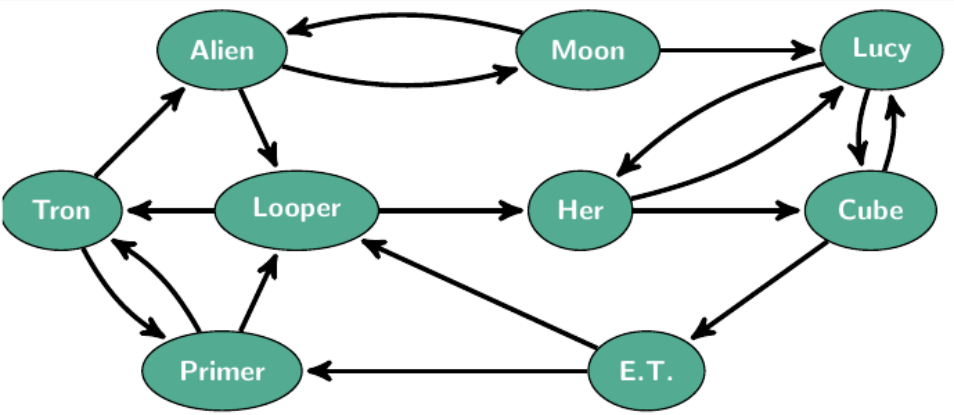
Recommender Graphs

A recommender graph is a directed graph where each vertex represents an item and each edge represents a transition between the items based on the context of the data.

For example, in a movie recommender graph, each vertex represents a movie, and an edge from u to v indicates that users typically transitioned from watching movie u to watching movie v.

The weight of such an edge could be the number of users who made the transition or the average rating improvement when moving from u to v (Lamprecht et al., 2017).

Recommender Graphs



An example of a recommender graph (Lamprecht et al., 2017).

Providing Recommendations

- Strongly connected components can be used to identify groups of items that are closely related.
- A recommender system can suggest items that are similar to the ones a user has already interacted with.
- If the edges contained information such as improvement of ratings, the recommendation system could suggest items that are likely to be enjoyed by the user.

Discovery of New Items

- Given the current recommender graph, it is possible that the strongly connected component related to a particular sub-genre of movies is small, leading to a cycle of recommendations within that sub-genre.
- This discovery would prompt the recommender system to suggest items from other genres to provide a more diverse set of recommendations.