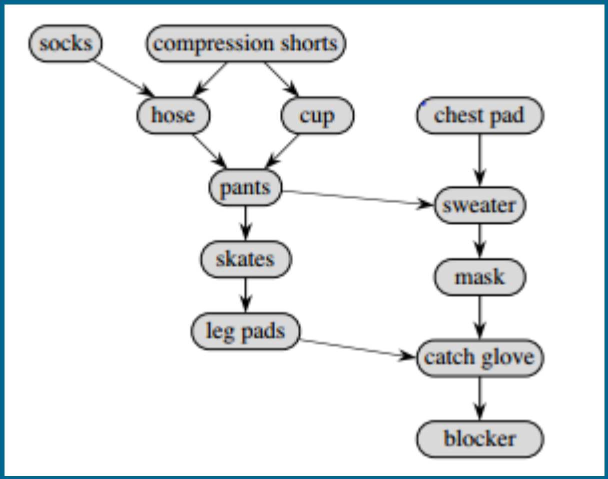
Directed Acyclic Graphs and Topological sort

By Nora Broderick and Hanna Fields

What is a DAG?



Directed Acyclic Graph

Vertices and directed edges

Acyclic - there is no way for a vertex to cycle back to itself

Starting point is vertex with no entering edges

Terms

Transitive - must be put before constraint

Vertices

Directed Edges

Directed Graphs

In-degree number of edges entering a vertex

Usage and Applications

Usage: Task based procedures that can only be done once and have multiple possible starting points potentially

Applications: Recipes, arithmetic operations, revision control

Algorithm Topological Sorting

Single linear order of performing a task

No circular dependencies

Assign numbers to vertices

Uses a stack

O(n+m) worst case

n is all vertices

m is all edges

Procedure TOPOLOGICAL-SORT(G)

Input: G: a directed acyclic graph with vertices numbered 1 to n.

Output: A linear order of the vertices such that u appears before v in the linear order if (u, v) is an edge in the graph.

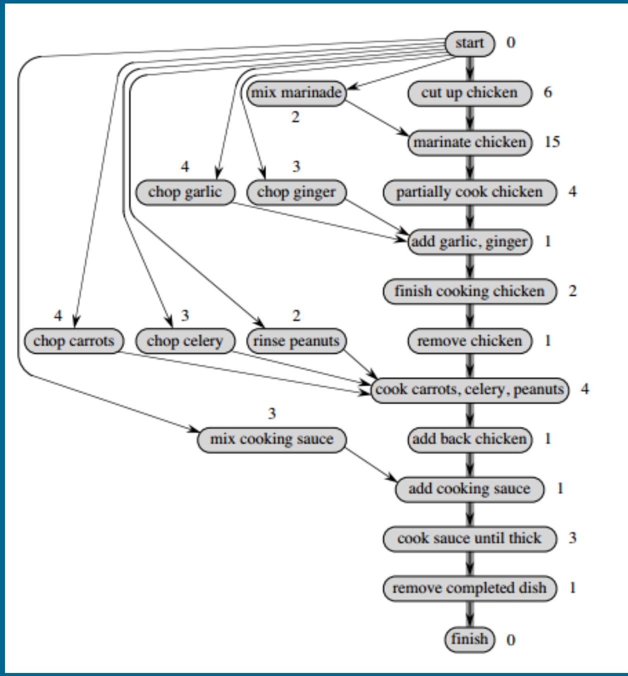
1. Let *in-degree*$[1..n]$ be a new array, and create an empty linear order of vertices.
2. Set all values in *in-degree* to 0.
3. For each vertex u:
   A. For each vertex v adjacent to u:
      i. Increment *in-degree*$[v]$.
4. Make a list *next* consisting of all vertices u such that *in-degree*$[u] = 0$.
5. While *next* is not empty, do the following:
   A. Delete a vertex from *next*, and call it vertex u.
   B. Add u to the end of the linear order.
   C. For each vertex v adjacent to u:
      i. Decrement *in-degree*$[v]$.
      ii. If *in-degree*$[v] = 0$, then insert v into the *next* list.
6. Return the linear order.

PERT Chart

“Program Evaluation and Review Technique”

DAG with time corresponding to tasks

Critical Path: The most efficient amount of time to complete a task given unlimited resources or the minimum sum of time to complete a task



Algorithm Relax

Relaxation steps

Used in DAG shortest paths

Procedure RELAX(u, v)

Inputs: u, v: vertices such that there is an edge (u, v).

Result: The value of *shortest*[v] might decrease, and if it does, then *pred*[v] becomes u.

1. If *shortest*[u] + *weight*(u, v) < *shortest*[v], then set *shortest*[v] to *shortest*[u] + *weight*(u, v) and set *pred*[v] to u.

Adjacency Matrix

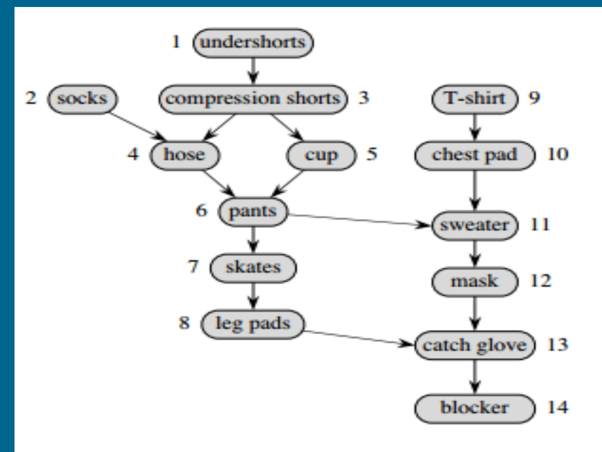
Each row and column correspond to a vertex

Adjacency List representation is an ordered list of the matrix

Fill with 1 if in Adjacency list and fill with 0 if not

Rows correspond to vertexes

Columns correspond to options of vertices to move to



Adjacency matrix

	1	2	3	4	5	6	7	8	9	10	11	12	13	14
1	0	0	1	0	0	0	0	0	0	0	0	0	0	0
2	0	0	0	1	0	0	0	0	0	0	0	0	0	0
3	0	0	0	1	1	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	1	0	0	0	0	0	0	0	0
5	0	0	0	0	0	1	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	1	0	0	0	1	0	0	0
7	0	0	0	0	0	0	0	1	0	0	0	0	0	0
8	0	0	0	0	0	0	0	0	0	0	0	0	1	0
9	0	0	0	0	0	0	0	0	0	1	0	0	0	0
10	0	0	0	0	0	0	0	0	0	0	1	0	0	0
11	0	0	0	0	0	0	0	0	0	0	0	1	0	0
12	0	0	0	0	0	0	0	0	0	0	0	0	1	0
13	0	0	0	0	0	0	0	0	0	0	0	0	0	1
14	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Adjacency lists

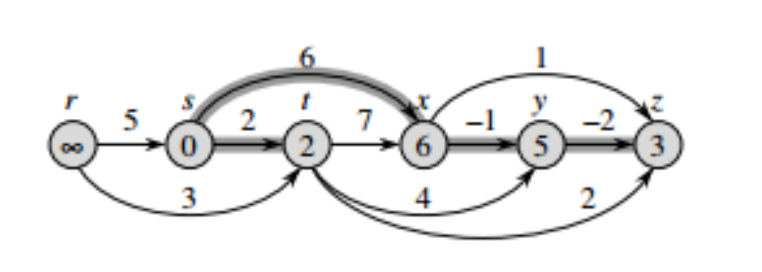
1	3
2	4
3	4, 5
4	6
5	6
6	7, 11
7	8
8	13
9	10
10	11
11	12
12	13
13	14
14	(none)

Algorithm DAG Shortest Path

Source Vertex

Target Vertex

Single source shortest paths



Procedure DAG-SHORTEST-PATHS(G, s)

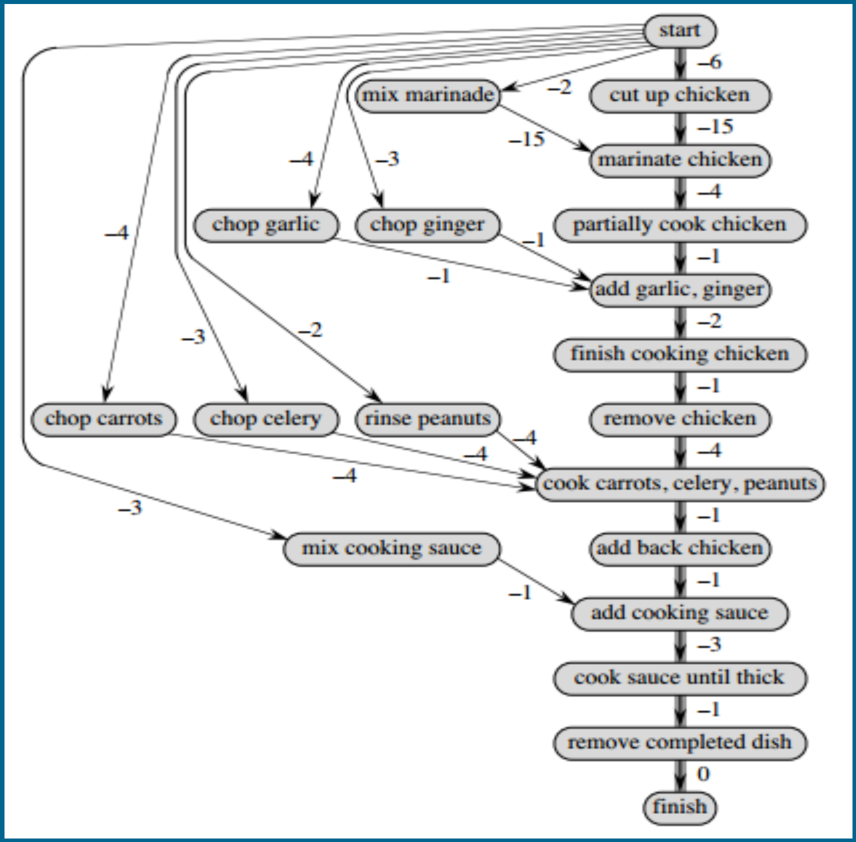
Inputs:

- G: a weighted directed acyclic graph containing a set V of n vertices and a set E of m directed edges.
- s: a source vertex in V.

Result: For each non-source vertex v in V, $shortest[v]$ is the weight $sp(s, v)$ of a shortest path from s to v and $pred[v]$ is the vertex preceding v on some shortest path. For the source vertex s, $shortest[s] = 0$ and $pred[s] =$ NULL. If there is no path from s to v, then $shortest[v] = \infty$ and $pred[v] =$ NULL.

1. Call TOPOLOGICAL-SORT(G) and set l to be the linear order of vertices returned by the call.
2. Set $shortest[v]$ to ∞ for each vertex v except s, set $shortest[s]$ to 0, and set $pred[v]$ to NULL for each vertex v.
3. For each vertex u, taken in the order given by l:
   A. For each vertex v adjacent to u:
      i. Call RELAX(u, v).

DAG Shortest Path Example



Sources

Corman *Algorithms Unlocked* Boston: MIT Press Books, 2013.

Thank you!