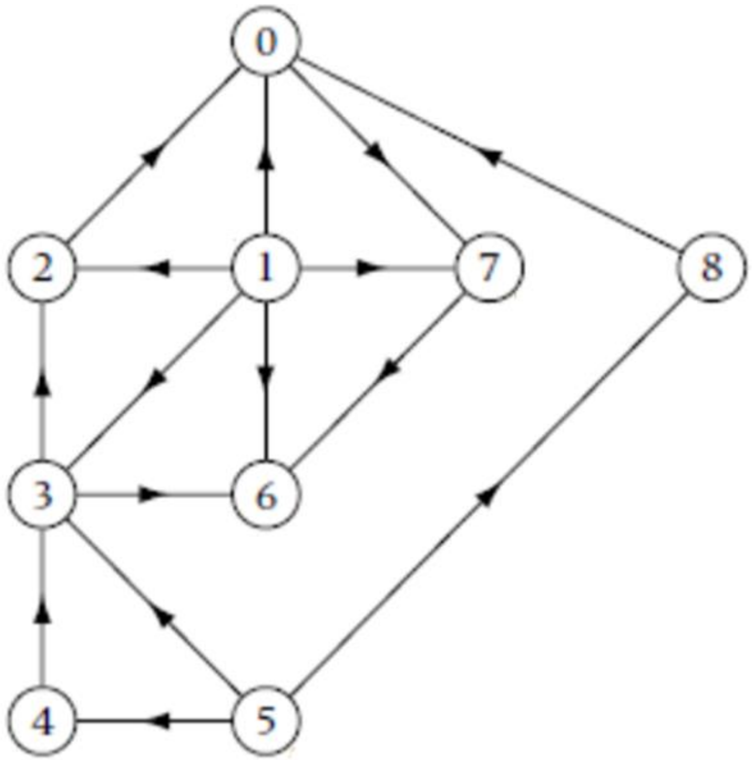
DAGs and Topological Sorting

Reading from textbook Algorithms: Foundations and Design Strategies:

Chapter 5, Section 5.5, pp. 231-234

DAG

A **Directed Acyclic Graph** or **DAG** is a digraph without any directed cycles.



Sources and Sinks

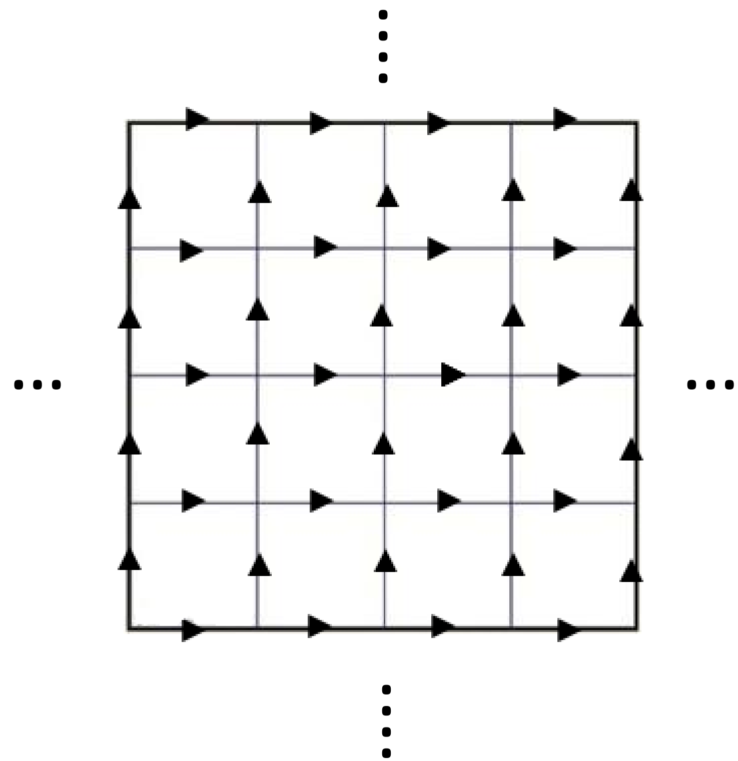
Source – a vertex *v* where all edges incident with v are directed out of *v*, i.e., in-degree(*v*) = 0

Sink – a vertex *v* where all edges incident with v are directed into *v*, i.e., out-degree(*v*) = 0

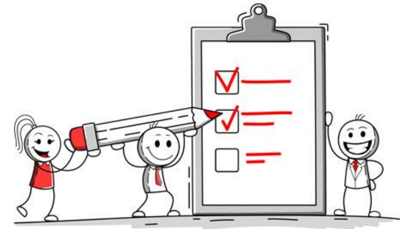
PSN. Show that a (finite) DAG must always contain a source and a sink.

Result not true for infinite graphs

Infinite grid with the edges oriented to the right and up is acyclic but has no source nor sink.



Ordering Tasks



- *n* tasks to be performed
- Certain tasks must be performed before others.
- For example, if we are building a house, the task of pouring the foundation must precede the task of laying down the first floor. However, another pair of tasks might not need to be done in a particular order, such as painting the kitchen and painting the bathroom.
- The problem is to obtain a linear ordering of the tasks in such a way that if task *u* must be done before task *v*, then *u* occurs before *v* in the linear ordering.

Modelling with a DAG

Construct a digraph $D = (V, E)$ where V is the set of tasks and $(u, v) \in E$ whenever task *u* must precede, i.e., be performed before, task *v*.

PSN. Prove D is a DAG.

Modelling with a DAG cont'd

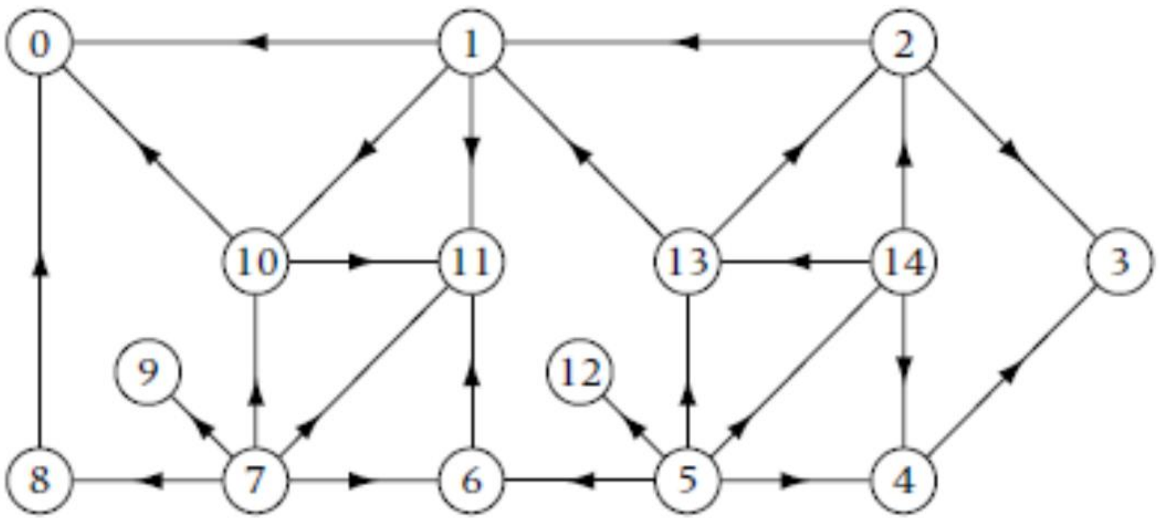
- The vertices of the DAG D correspond to the tasks, and a directed edge from u to v is in D iff task u must precede task v.
- A **topological sorting** *of D* is a listing of the vertices such that if uv is an edge of D, then u precedes v in the list.
- A **topological-sort labeling** of D is a labeling of the vertices in D with the labels $0, . . . , n - 1$ such that for any edge uv in D, the label of u is smaller than the label of v.

Topological Sort – Straightforward Algorithm

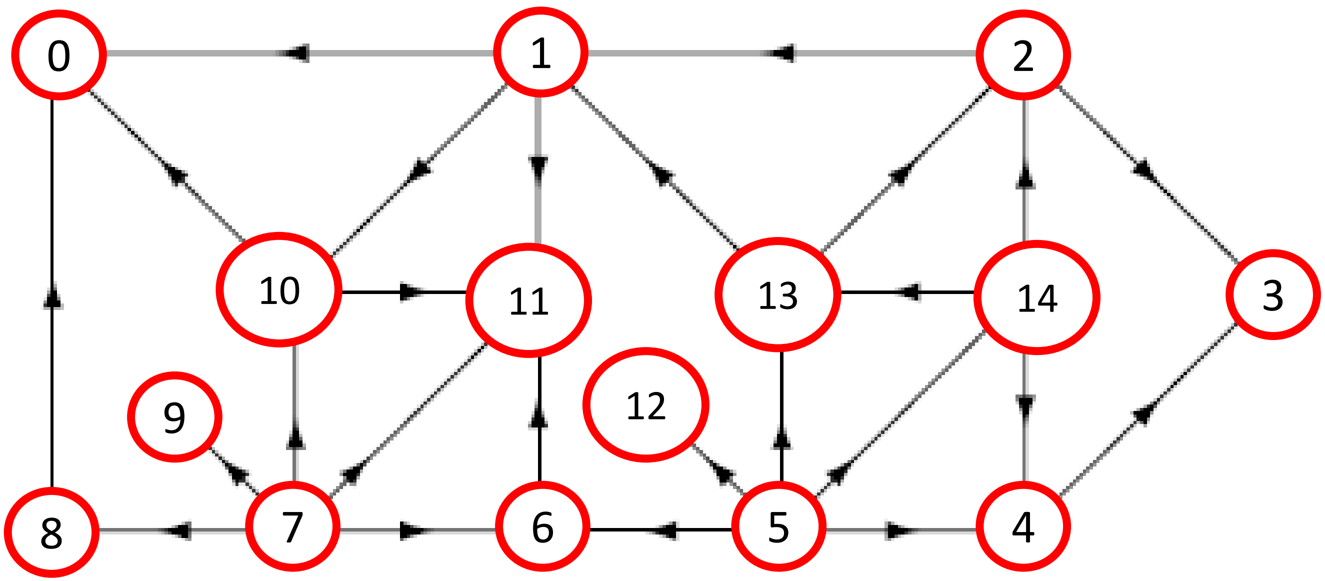
Repeat until all vertices of DAG have been visited

1. Find a vertex *v* all of whose unvisited vertices lie in its out-neighborhood
2. Insert *v* at the beginning of the list
3. Mark *v* as visited

Use straightforward algorithm to find topological sort for the following DAG:



Solution



Topologically sorted list:

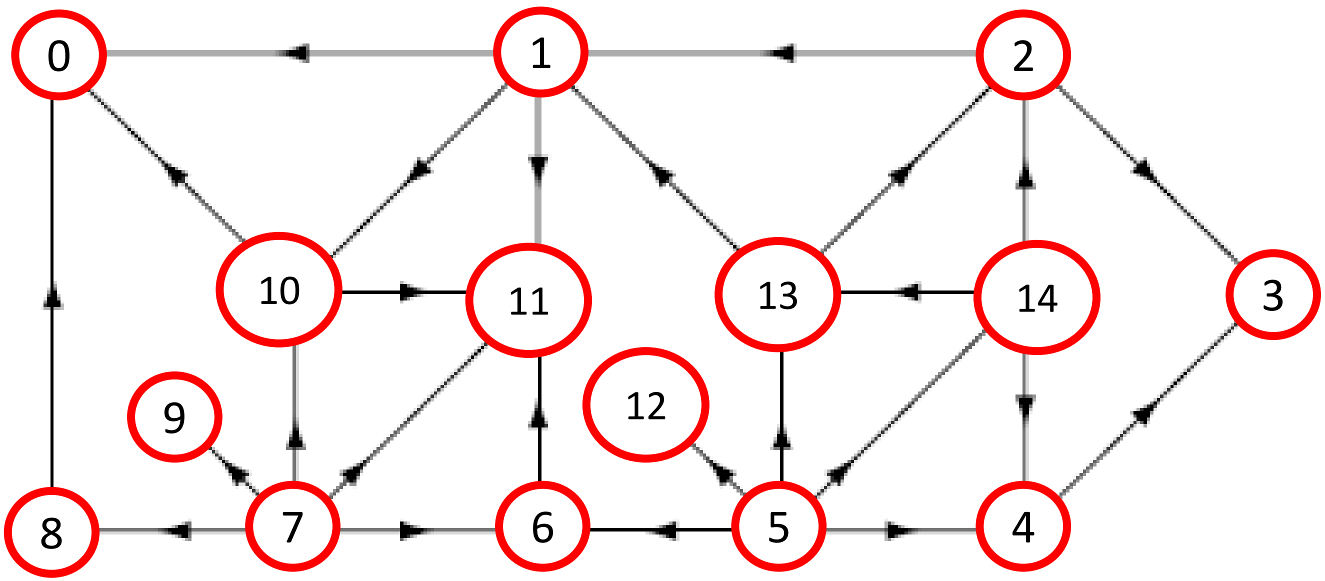
5 7 8 9 12 14 13 2 1 4 3 10 0 6 11

More efficient topological sort using DFT

Perform a DFT traversal keeping track of the **order in which the vertices are explored**. A vertex becomes **explored** when all the vertices in its out-neighborhood have been visited, i.e., when we backtrack from the vertex.

The topological sort order is the **reverse** of the explored order.

Action for a sample list



Topologically sorted list:

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

Pseudocode for Topological Sort using DFT

```
procedure TopologicalSort(D,TopList[0:n– 1])
Input: D (dag with vertex set V = {0, . . . , n – 1} and edge set E)
Output: TopList[0:n– 1] (array containing a topologically sorted list of the
                                                        vertices in D)
      Mark[0:n– 1]     a 0/1 array initialized to 0s
      Counter ← n – 1
      for v ← 0 to n – 1 do
         if Mark[v] ← 0 then
           DFSOutTopLabel(D,v)
         endif
      endfor
       procedure DFSOutTopLabel(D,v) recursive
           Mark[v] ← 1
           for each w ∈ V such that vw ∈ E do
               if Mark[w] = 0 then
                  DFSOutTopLabel(D,w)
               endif
           endfor
           TopList[Counter] ← v
           Counter ← Counter – 1
        end DFSOutTopLabel
end TopologicalSort
```

Analysis of computing time

TopologicalSort has the same complexity as DFT, i.e.,

$$O(m+n)$$

How does earth and mars schedule a vacation?

They planet.