Directed Graphs: Strongly Connected Components, and Topological Sorting

Reading: KT 3.5—3.6

CS231 Fundamental Algorithms
Lyn Turbak
Department of Computer Science
Wellesley College
Tue April 12, 2022 (Revised Thu Apr 28)

Directed Graphs

A directed graph is a pair \((V, E)\) of
1. A set \(V\) of vertices (also called nodes)
2. A set \(E\) of directed edges (where each edge is a pair of two nodes†)

\[
\begin{align*}
\text{vertices} & \{ \{a,b,c,d,e,f,g,h\}, \\
& \quad \{\{a,c\}, \{b,a\}, \{b,e\}, \{b,g\}, \{c,a\}, \{c,b\}, \{c,d\}, \\
& \quad \{d,d\}, \{e,f\}, \{f,g\}, \{g,e\}, \{g,h\}\} \\
\text{directed edges} & \end{align*}
\]

† This definition does allow a self-edge from a vertex to itself.

Directed Paths

A path in a directed graph is a sequence of vertices where each vertex is connected to the next by a directed edge. A path is simple if no vertices are repeated. The length of the path is one less than the length of the sequence.

Simple path \((a,c,d,b,g,h)\); length=5  Nonsimple path \((c,a,c,d,d,b,g,e)\); length=7

Note: there’s a length 0 path from every node to itself. E.g., \((a)\)

Directed Cycles

A cycle in a directed graph is a path with length \(\geq 1\) beginning and ending at the same vertex. A simple cycle is a cycle that repeats no vertices except the first/last.

Simple cycle \((e,f,g,e)\)  Nonsimple cycle \((a,c,a,c,d,d,b,a)\)
**Mutual Reachability**

In a directed graph, vertices \( u \) and \( v \) are **mutually reachable** if there’s a path from \( u \) to \( v \) and from \( v \) to \( u \).

Examples of mutually reachable vertex pairs:
\[(a,a), (a,b), (a,c), (a,d), (b,c), (b,d), (c,d), (e,f), (e,g), (f,g)\]

Examples of vertex pairs that aren’t mutually reachable:
\[(a,e), (a,f), (a,g), (a,h), (b,e), (b,f), (b,g), (b,h),
(c,e), (c,f), (c,g), (c,h), (d,e), (d,f), (d,g), (d,h),
(e,h), (f,h), (g,h)\]

Mutual reachability is:
- **reflexive**: For all vertices \( v \), \( (v,v) \) is mutually reachable.
- **symmetric**: For all vertices \( u, v \), if \( (u,v) \) is mutually reachable, then \( (v,u) \) is mutually reachable.
- **transitive**: For all vertices \( u, v, w \), if \( (u,v) \) and \( (v,w) \) are mutually reachable, then \( (u,w) \) is mutually reachable.

So mutual reachability is an equivalence relation.

**In-degree and Out-degree**

The **in-degree** of a vertex is the number of edges entering that vertex. The **out-degree** of a vertex is the number of edges exiting that vertex.

\[
\begin{align*}
\text{in-degree}(c) &= \text{in-degree}(f) = \text{in-degree}(h) = 1 \\
\text{in-degree}(a) &= \text{in-degree}(b) = \text{in-degree}(d) = \text{in-degree}(e) = \text{in-degree}(g) = 2 \\
\text{out-degree}(h) &= 0 \\
\text{out-degree}(a) &= \text{out-degree}(e) = \text{out-degree}(f) = \text{out-degree}(g) = 1 \\
\text{out-degree}(d) &= 2 \\
\text{out-degree}(b) &= \text{out-degree}(c) = 3
\end{align*}
\]

**Strongly Connected Components**

In a directed graph \( G \), a **strongly connected component** (or just **strong component**) is a subgraph of \( G \) consisting of all vertices in the same mutually reachable equivalence class, along with all edges that connect these vertices. Edges not in any strongly connected component are **cross edges** between components.

A graph can always be partitioned into a collection of strongly connected components, where each vertex is a member of exactly one such component.

The example graph has 3 strongly connected components.

**Directed Graph Reversal**

If \( G \) is a directed graph, its reversal \( G^{\text{rev}} \) has the same vertices, but all edges in the opposite direction.
Directed Acyclic Graphs (DAGs)

A directed graph $G$ is a directed acyclic graph (DAG) iff it has no cycles. In practice, DAGs are used to represent prerequisites/dependencies, and tree-like structures with shared descendants.

This graph $H$ is a DAG

This graph $G$ is not a DAG

Topological Sort/Ordering on a DAG

A topological sort (or topological order) of a DAG $D$ is a sequence $S$ of all vertices from the DAG such that if there’s an edge in $D$ from $S_i$ to $S_j$, then $i < j$.

This graph $H$ is a DAG

Some topological sorts of $H$:  
(a, c, d, b, e, g, h)  
(a, c, d, f, b, e, g, h)  
(a, c, f, d, b, e, g, h)  
(a, f, c, d, b, e, g, h)  
(f, a, c, d, b, e, g, h)

Are there any others?

Directed Graph Representation:

We’ll assume the $n$ vertices of graph have unique IDs (UIDs) numbered $1 .. N$, and that vertex names and edges (as adjacency lists for outgoing edges) are indexed by UID. This is like undirected graphs, except that $u$ in $\text{OutAdj}[v]$ does not imply $v$ in $\text{OutAdj}[u]$. In some contexts, the directed graph might also have $\text{InAdj}$ for incoming edges.

Name array $\text{Name}$ with 1-indexed arrays

```
[ "a", "b", "c", "d", "e", "f", "g", "h" ]
```

Adjacency array $\text{OutAdj}$ with 1-indexed arrays. (Adjacent vertices are sorted in example, but need not be)

```
[ [3], # OutAdj[1]  
[2, 5, 7], # OutAdj[2]  
[1, 2, 4], # OutAdj[3]  
[2, 4], # OutAdj[4]  
[6], # OutAdj[5]  
[7], # OutAdj[6]  
[5, 8], # OutAdj[7]  
[] ] # OutAdj[8]
```

Some Typical Directed Graph Questions

What are some typical questions we want to answer about directed graphs?

- Is there a path between two vertices?
- What is the shortest path between two vertices? If edges are annotated with distances/weights, what is the shortest distance or minimal weight path between two vertices?
- Are two vertices mutually reachable? (i.e., are they in the same strongly connected component?)
- What are the strongly connected components of a graph?
- Does the graph have a cycle? Is there a cycle involving a particular vertex or edge?
- Is a graph a DAG?
- If a graph is a DAG, what is a topological sort of its vertices?
- Can we label the vertices/edges of a graph with information such that certain properties hold?
Topological Sort Algorithm

function TopSort(D) # If D is a DAG, return an array that’s a
    # sequence of the vertices of D in an order
    # respecting the directed edges in D.
    # Otherwise indicate that D has a cycle.
Remaining <- V(D) # Set of vertices to be processed.
Seq = new empty array # Initialize result sequence
while Remaining is nonempty
    choose a v in Remaining where len(InAdj[v]) = 0
        # One must exist or there’d be a cycle!
        # If one does not exist, can return an indication
        # that there’s a cycle.
    Seq@v # Extend result sequence with v
    Remove all edges in OutAdj[v] from D
        # affects InAdj for adjacent nodes
return Seq

Another Topological Sort Example

```
  a  c
  ↓  ↓
  b  d
  ↓  ↓
e  g
  ↓  ↓
f  h
```

Directed Graphs 13

Directed Graphs 14