CSE 326: Data Structures
Graphs – Topological Sort

Hal Perkins
Spring 2007
Lectures 22-23

Agenda

• Basic graph terminology
• Graph representations
• Topological sort

• Reference: Weiss, Ch. 9

Graph… ADT?

• Not quite an ADT…
operations not clear

• A formalism for representing relationships between objects

Graph \( G = (V, E) \)

– Set of vertices:
  \( V = \{v_1, v_2, \ldots, v_n\} \)

– Set of edges:
  \( E = \{e_1, e_2, \ldots, e_m\} \)
where each \( e_i \) connects two vertices \( \{v_{i1}, v_{i2}\} \)

Graph

\( V = \{\text{Han}, \text{Leia}, \text{Luke}\} \)

\( E = \{\text{Luke, Leia}, \text{(Han, Leia)}, \text{(Leia, Han)}\} \)

Some Applications:
Moving Around Washington

What’s the shortest way to get from Seattle to Pullman?
Edge labels:
Some Applications: Moving Around Washington

What’s the fastest way to get from Seattle to Pullman?
Edge labels:

Some Applications: Reliability of Communication

If Wenatchee’s phone exchange goes down, can Seattle still talk to Pullman?

Some Applications: Bus Routes in Downtown Seattle

If we’re at 3rd and Pine, how can we get to 1st and University using Metro?

Graph Definitions

In directed graphs, edges have a specific direction:

In undirected graphs, they don’t (edges are two-way):

\( v \) is adjacent to \( u \) if \( (u, v) \in E \)
More Definitions:
Simple Paths and Cycles

A simple path repeats no vertices (except that the first can be the last):

\[ p = \{\text{Seattle, Salt Lake City, San Francisco, Dallas}\} \]
\[ p = \{\text{Seattle, Salt Lake City, Dallas, San Francisco, Seattle}\} \]

A cycle is a path that starts and ends at the same node:

\[ p = \{\text{Seattle, Salt Lake City, Dallas, San Francisco, Seattle}\} \]
\[ p = \{\text{Seattle, Salt Lake City, Seattle, San Francisco, Seattle}\} \]

A simple cycle is a cycle that repeats no vertices except that the first vertex is also the last (in undirected graphs, no edge can be repeated)

Trees as Graphs

• Every tree is a graph!
• Not all graphs are trees!

A graph is a tree if
– There are no cycles (directed or undirected)
– There is a path from the root to every node

Directed Acyclic Graphs (DAGs)

DAGs are directed graphs with no (directed) cycles.

Aside: If program call-graph is a DAG, then all procedure calls can be inlined

Graph Representations

0. List of vertices + list of edges
1. 2-D matrix of vertices (marking edges in the cells) “adjacency matrix”
2. List of vertices each with a list of adjacent vertices “adjacency list”

Things we might want to do:
• iterate over vertices
• iterate over edges
• iterate over vertices adj. to a vertex
• check whether an edge exists

Vertices and edges may be labeled
**Representation 1: Adjacency Matrix**

A $|V| \times |V|$ array in which an element $(u, v)$ is true if and only if there is an edge from $u$ to $v$.

**Weighted Edges**

- **adjacency matrix**:
  $A[u][v] = \begin{cases}  \text{weight} & \text{if } (u, v) \in E \\ 0 & \text{if } (u, v) \notin E \end{cases}$

**Representation 2: Adjacency List**

A $|V|$-ary list (array) in which each entry stores a list (linked list) of all adjacent vertices.

**Representation**

- **adjacency list**: 

---

**space requirements**: runtime:
**Application: Topological Sort**

Given a directed graph, \( G = (V, E) \), output all the vertices in \( V \) such that no vertex is output before any other vertex with an edge to it.

Is the output unique?

---

**Topological Sort: Take One**

1. Label each vertex with its *in-degree* (\# of inbound edges)
2. While there are vertices remaining:
   a. Choose a vertex \( v \) of *in-degree zero*; output \( v \)
   b. Reduce the in-degree of all vertices adjacent to \( v \)
   c. Remove \( v \) from the list of vertices

**Runtime:**

---

**Topological Sort: Take Two**

1. Label each vertex with its in-degree
2. Initialize a queue \( Q \) to contain all in-degree zero vertices
3. While \( Q \) not empty
   a. \( v = Q\text{.dequeue} \); output \( v \)
   b. Reduce the in-degree of all vertices adjacent to \( v \)
   c. If new in-degree of any such vertex \( u \) is zero
      \( Q\text{.enqueue}(u) \)

**Runtime:**

Note: could use a stack, list, set, box, … instead of a queue

---

`void Graph::topsort(){
    Vertex v, w;
    labelEachVertexWithItsIn-degree();
    for (int counter=0; counter < NUM_VERTICES; counter++){
        v = findNewVertexOfDegreeZero();
        v.topologicalNum = counter;
        for each w adjacent to v
            w.indegree--;
    }
} `
void Graph::topsort(){
    Queue q(NUM_VERTICES); int counter = 0; Vertex v, w;
    labelEachVertexWithItsIn-degree();
    q.makeEmpty();
    for each vertex v
        if (v.indegree == 0)
            q.enqueue(v);
    while (!q.isEmpty()){
        v = q.dequeue();
        v.topologicalNum = ++counter;
        for each w adjacent to v
            if (--w.indegree == 0)
                q.enqueue(w);
    }
}

Runtime: