Graphs

Neva Cherniavsky
Summer 2006

Graph... ADT?

• Not quite an ADT...
operations not clear

• A formalism for representing relationships between objects
Graph G = (V, E)

> Set of vertices:
V = {v₁, v₂, ..., vₙ}

> Set of edges:
E = {e₁, e₂, ..., eₘ}

where each eᵢ connects two vertices (vᵢ₁, vᵢ₂)

Graphs In Practice

• Web graph
  > Vertices are web pages
  > Edge from u to v is a link to v appears on u

• Call graph of a computer program
  > Vertices are functions
  > Edge from u to v if u calls v

• Task graph for a work flow
  > Vertices are tasks
  > Edge from u to v if u must be completed before v begins

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) ∈ E

Weighted Graphs

Each edge has an associated weight or cost.

- Clinton to Mukilteo: 20
- Kingston to Edmonds: 30
- Bainbridge to Seattle: 35
- Bremerton to Seattle: 60

Paths and Cycles

• A **path** is a list of vertices (v₀, v₁, ..., vₙ) such that (vᵢ, vᵢ₊₁) ∈ E for all 0 ≤ i < n.

• A **cycle** is a path that begins and ends at the same node.

- p = (Seattle, Salt Lake City, Chicago, Dallas, San Francisco, Seattle)
Path Length and Cost

- **Path length**: the number of edges in the path
- **Path cost**: the sum of the costs of each edge

Seattle ➔ San Francisco ➔ Dallas ➔ Salt Lake City ➔ Chicago

- Length: 5
- Cost: 11.5

Trees as Graphs

- Every tree is a graph with some restrictions:
  - the tree is directed
  - there are no cycles (directed or undirected)
  - there is a directed path from the root to every node

Directed Acyclic Graphs (DAGs)

- DAGs are directed graphs with no cycles.

Graph 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 $v$.

Graph Representation 2: Adjacency List

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

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: Bus Routes in Downtown Seattle

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

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?

Another example

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:
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--;
    }
}

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.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.enqueue(u)

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

Example
Queue = 1 2 5
Output =

Example
Queue = 1 2
Output = 5

Example
Queue = 1 2 5 2
Output = 5 2
**Graph Connectivity**

Undirected graphs are **connected** if there is a path between any two vertices.

Directed graphs are **strongly connected** if there is a path from any one vertex to any other.

Directed graphs are **weakly connected** if there is a path between any two vertices, ignoring direction.

A **complete** graph has an edge between every pair of vertices.

**Graph Traversals**

- Breadth-first search (and depth-first search) work for arbitrary (directed or undirected) graphs - not just mazes!
  - Must mark visited vertices so you do not go into an infinite loop!
- Either can be used to determine connectivity:
  - Is there a path between two given vertices?
  - Is the graph (weakly) connected?
- Which one:
  - Uses a queue?
  - Uses a stack?
  - Always finds the **shortest path** (for unweighted graphs)?

**Example**

Queue = 3
Output = 5 2 1 6

Queue = 4
Output = 5 2 1 6 7 3

**Exercise**

- Design the algorithm to initialize the in-degree array. Assume the adjacency list representation.