CSE 326 Data Structures

Dave Bacon

Graphs

Logisitics

- Turn in Homework 6…

- Project 3 code due on Monday!
- Project 3 writeup due on Thursday!

- Homework 7 will be due….

- Read Chapter 9 of Weiss

- Complain about the class on the survey on the webpage!

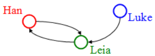
Graph… ADT?

- Not quite an ADT… operations not clear
- A formalism for representing relationships between objects

  Graph `G = (V,E)`

  – *Set of vertices*:
  $\mathbf{V} = \{\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_n\}$

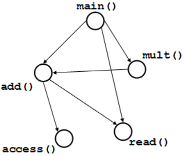
  – *Set of edges*:
  $\mathbf{E} = \{\mathbf{e}_1, \mathbf{e}_2, \ldots, \mathbf{e}_m\}$
  where each $\mathbf{e}_i$ connects two vertices $(\mathbf{v}_{i1}, \mathbf{v}_{i2})$




```
V = {Han, Leia, Luke}
E = {(Luke, Leia),
     (Han, Leia),
     (Leia, Han)}
```

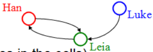
Directed Acyclic Graphs (DAGs)

DAGs are directed graphs with no (directed) cycles.



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

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"

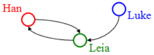
Things we might want to do:

- 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| x |V|` array in which an element `(u,v)` is true if and only if there is an edge from `u` to `v`



	Han	Luke	Leia
Han			
Luke			
Leia			

space requirements:

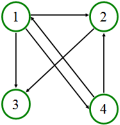
runtime:

Representation

- adjacency **matrix**:

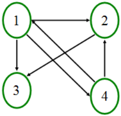
$$A[u][v] = \begin{cases} \text{weight} & , \text{ if } (u, v) \in E \\ 0 & , \text{ if } (u, v) \notin E \end{cases}$$

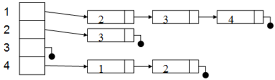
	1	2	3	4
1				
2				
3				
4				



Representation

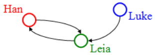
- adjacency **list:**

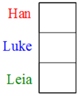




Representation 2: Adjacency List

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

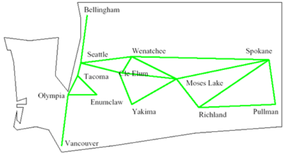




space requirements:

runtime:

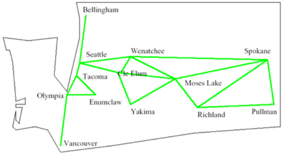
Some Applications: Moving Around Washington



What's the *shortest way* to get from Seattle to Pullman?

Edge labels:

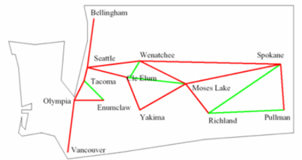
Some Applications: Moving Around Washington



What's the *fastest way* to get from Seattle to Pullman?

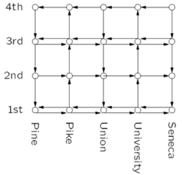
Edge labels:

Some Applications: Reliability of Communication



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

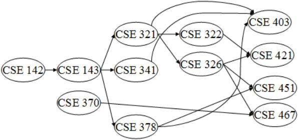
Some Applications:
Bus Routes in Downtown Seattle



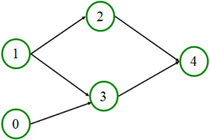
If we're at 3[rd] and Pine, how can we get to 1[st] and University using Metro?

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?



Valid Topological Sorts:

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*
3. Reduce the in-degree of all vertices adjacent to *v*
   a. Remove *v* from the list of vertices

Runtime:

```
void Graph::topsort(){
  Vertex v, w;

  labelEachVertexWithItsInDegree();

  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)

<u>Note</u>: could use a stack, list, set, box, … instead of a queue

Runtime:

```
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);
  }
}
```

intialize the queue

get a vertex with indegree 0

insert new eligible vertices

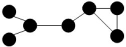
Runtime:

Graph Traversals

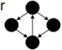
- Breadth-first search (and depth-first search) work for arbitrary (directed or undirected) graphs - not just mazes!
  - Must mark visited vertices so you do not go into an infinite loop!
- Either can be used to determine connectivity:
  - Is there a path between two given vertices?
  - Is the graph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

Graph Connectivity

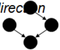
Undirected graphs are *connected* if there is a path between any two vertices



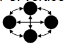
Directed graphs are *strongly connected* if there is a path from any one vertex to any other



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



The Shortest Path Problem

Given a graph G, edge costs $c_{i,j}$, and vertices s and t in G, find the shortest path from s to t.

For a path $p = v_0\ v_1\ v_2 \ldots v_k$

- *unweighted length* of path $p = k$ (a.k.a. *length*)
- *weighted length* of path $p = \sum_{i=0..k-1} c_{i,i+1}$ (a.k.a *cost*)

Path length equals path cost when ?

Single Source Shortest Paths (SSSP)

Given a graph *G*, edge costs $c_{i,j}$, and vertex *s*, find the shortest paths from *s* to all vertices in G.

- Is this harder or easier than the previous problem?

All Pairs Shortest Paths (APSP)

Given a graph *G* and edge costs $c_{i,j}$, find the shortest paths between all pairs of vertices in G.

- Is this harder or easier than SSSP?

- Could we use SSSP as a subroutine to solve this?

Variations of SSSP

- Weighted vs. unweighted
- Directed vs undirected
- Cyclic vs. acyclic
- Positive weights only vs. negative weights allowed
- Shortest path vs. longest path
- …

Applications

- Network routing
- Driving directions
- Cheap flight tickets
- Critical paths in project management (see textbook)
- …

SSSP: Unweighted Version

Ideas?

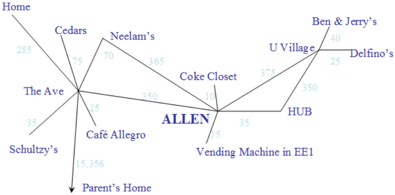
```
void Graph::unweighted (Vertex s){
  Queue q(NUM_VERTICES);
  Vertex v, w;
  q.enqueue(s);
  s.dist = 0;

  while (!q.isEmpty()){
    v = q.dequeue();
    for each w adjacent to v
      if (w.dist == INFINITY){
        w.dist = v.dist + 1;
        w.path = v;
        q.enqueue(w);
      }
  }
}
```

each edge examined at most once – if adjacency lists are used

each vertex enqueued at most once

total running time: O(   )

Weighted SSSP: The Quest For Food



Can we calculate shortest distance to all nodes from Allen Center?

Dijkstra, Edsger Wybe

Legendary figure in computer science; was a professor at University of Texas.

Supported teaching introductory computer courses without computers (pencil and paper programming)

Supposedly wouldn't (until very late in life) read his e-mail; so, his staff had to print out messages and put them in his box.

E.W. Dijkstra (1930-2002)

1972 Turning Award Winner,
Programming Languages, semaphores, and …