IE170: Algorithms in Systems Engineering: Lecture 15

Jeff Linderoth

Department of Industrial and Systems Engineering
Lehigh University

February 26, 2007

Taking Stock

Last Time

- DP for Lot Sizing
- Greedy Algorithm for activity scheduling

This Time

- The Wonderful World of Graph Theory
- You should read Chap 22

Graphs

- A graph is an abstract object used to model such connectivity relations.
- A graph consists of a list of items, along with a set of connections between the items.
- The study of such graphs and their properties, called graph theory, is hundreds of years old.
- Graphs can be visualized easily by creating a physical manifestation showing the connection relationships

Graph Types

- The connections in the graph may or may not have an orientation or a direction.
- We may (or may not) allow more than one connection between a pair of items
- We may (or may not) not allow an item to be connected to itself.

- For now, we consider graphs that are
  - undirected, i.e., the connections do not have an orientation, and
  - simple, i.e., we allow only one connection between each pair of items and no connections from an item to itself.

(A few) Applications of Graphs

- Maps
- Internet/World Wide Web
- Social Networks
- Circuits
- Scheduling
- Communication Networks
- Matching and Assignment
- Chemistry and Physics

Graph Terminology and Notation

- In an undirected graph, the "items" are usually called vertices (sometimes also called nodes).
- We denote the set of vertices as V and index them (in our code) from 0 to $n-1$, where $n = |V|$.
- The connections between the vertices are (for now) unordered pairs called edges.
- Often, when the pair is ordered, people call them arcs
- The set of edges is denoted E and $m = |E| \leq n(n-1)/2$.

Graph Terminology and Notation

- An undirected graph $G = (V, E)$ is then composed of a set of vertices V and a set of edges $E \subseteq V \times V$.
- If $e = (i, j) \in E$, then
  - i and j are called the endpoints of e,
  - e is said to be incident to i and j, and
  - i and j are said to be adjacent vertices.
- The number of vertices adjacent to v in G is known as the degree of v

More Terminology

- Let $G = (V, E)$ be an undirected graph.
- A subgraph of G is a graph composed of an edge set $E' \subseteq E$ along with all incident vertices.
- A subset V' of V, along with all incident edges is called an induced subgraph.
- A path in G is a sequence of vertices such that each vertex is adjacent to the vertex preceding it in the sequence.
- A path is simple if no vertex occurs more than once in the sequence.
- A cycle is a path that is simple except that the first and last vertices are the same.
- A tour is a cycle that includes all the vertices.

Connectivity in Graphs

- An undirected graph is said to be connected if there is a path between any two vertices in the graph.
- A graph that is not connected consists of a set of connected components that are the maximal connected subgraphs.
- Given a graph, one of the most basic questions one can ask is whether vertices i and j are in the same component.
- In other words, is there a path from i to j?
- If so, what is the shortest path (number of edges) from i to j.
- (We'll ask that today in lab)

Representing Graphs on a Computer

Two Graph Representations

1. Adjacency Lists
2. Adjacency Matrix

Adjacency List

- Array of $|V|$ sets, one for each vertex
- Vertex $u's$ list has all vertices such that $(u, v) \in E$
- e.g. Java: `private ArrayList<TreeSet<Integer>> AdjList_;`

Adjacency Matrix

- Matrix $A \in \{0, 1\}^{|V| \times |V|}$
- $a_{ij} = 1$ if and only if $(i, j) \in E$

Comparing Graph Representations

Adjacency List

- Space: $O(|V| + |E|) = O(n + m)$
- Time to list vertices adjacent to v: $O(\text{degree}(v))$
- Time to tell if $(u, v) \in E$: $O(\text{degree}(u))$

Adjacency Matrix

- Space: $O(|V|^2) = O(n^2)$
- Time to list vertices adjacent to v: $O(|V|)$
- Time to tell if $(u, v) \in E$: $O(1)$

Useful Java Code

- We'll use adjacency list in our graph implementations. (Most graphs are fairly sparse)
- The following code creates a "random" graph on n verices, with each edge occurring indepently with probability p

Random Graph Constructor

```
public Graph(int n, double p)
{
  numV_ = n;
  AdjList_ = new ArrayList<TreeSet<Integer>>(numV_);

  for (int i=0; i < numV_; i++) {
    AdjList_.add(new TreeSet<Integer>());
  }

  for (int i=0; i < numV_; i++) {
    for (int j=i+1; j < numV_; j++) {
      if (Math.random() < p) {
        insert(i,j);
      }
    }
  }
}
```


Adding an Edge

```
public void insert(int u, int v)
{

  assert(u >= 0 && v >= 0 && u < numV_ && v < numV_);

  AdjList_.get(u).add(v);

  //XXX Here we assume undirected graph
  AdjList_.get(v).add(u);
  numE_++;
}
```


Graph Search Algorithms

- There are two "workhorse" algorithms for searching graphs that form the basis for many more complicated algorithms.
  - Breadth-First Search (BFS): Search "broadly"
  - Depth-First Search (DFS): Search "deeply"
- We'll do BFS first.
  - Don't worry – you'll get to do DFS too!
- BFS: Discovers all nodes at distance k from starting node s before discovering any node at distance $k+1$
- Send a "wave" out from a starting vertex s

Aside: Queue 'em up

- BFS is conveneniently implemented with a data structure known as a FIFO queue.
- FIFO: First-In-First-Out. Just like a regular line, like you have to stand in at Disneyworld.
- Java has a Queue interface for you. Two methods are of specific interest are `poll()` and `add()`. You can check the docs for more.

Queue Interface

Queue<E>

- `boolean add(E e)`: Inserts the specified element into this queue
- `E poll()`: Retrieves and removes the head of this queue, or returns null if this queue is empty.

- Remember queue is an interface, so you can't really create one. You need to specify an actual implementation, e.g.
- `Queue<Integer> myqueue = new ArrayList<Integer>();`

BFS

BFS

- Input: Graph $G = (V, E)$, source node $s \in V$
- Output: $d(v)$, distance (smallest # of edges) from s to v $\forall v \in V$
- Output: $\pi(v)$, predecessor of v on the shortest path from s to v

---

Oh no! DP again

- $\delta(s, v)$: shortest path from s to v
- Lemma: If $(u, v) \in E$, then $\delta(s, v) \leq \delta(s, u) + 1$

BFS

```
BFS(V, E, s)
  for each u in V \ {s}
  do d(u) ← ∞
     π(u) ← NIL
  d[s] ← 0
  Q ← ∅
  ADD(Q, s)
  while Q ≠ ∅
  do u ← POLL(Q)
     for each v in Adj[u]
     do if d[v] = ∞
          then d[v] ← d[u] + 1
               π[v] = u
               ADD(Q, v)
```


Analysis

- How many times is each vertex added?
  - Answer: Once. So $|V|$ for add operation
- How many times is adjaceny list of vertex v scanned?
  - Answer: Once. Since $\sum_{v \in V} \text{size}(Adj[v]) = 2|E|$ (for undirected), we have $|E|$ here.
- Running time: $O(|V| + |E|)$: Linear in the input size of the graph (for adjacency list implementation)

Next Time

- Graphs, Graphs, and more Graphs.