MORE ALGORITHMS

ANNOUNCEMENTS:

• course almost done: L28 is summary, evals
• A6, A7 due soon
• end of regular consulting: Fri, 5/1
• special hours to finish regrades, pick up work will be announced on the website
• consulting: forms in 303 Upson, starting 1PM today
• final exam info: Final Exam (on website) (review info, prior exam posted soon)
• final exam conflicts? see website; due Friday 5/2!

OVERVIEW:

• shortest path algorithm for weighted graph (Dijkstra’s algorithm)
• all pairs source shortest path (Floyd’s algorithm)
• minimum cost spanning trees (Prim’s algorithm, Kruskal’s algorithm)
1. Shortest Path for Weighted Graphs

1.1 Assumptions

• could be directed or undirected
• non-negative weights

1.2 Dijkstra’s Algorithm

• very famous
• example of greedy algorithm
• on-line demo:
  http://www-b2.is.tokushima-u.ac.jp/~ikeda/suuri/dijkstra/Dijkstra.shtml
1.3  Wordy Gist: Based ON BFS

- BFS: visit the nodes by “levels” or “layers”
  - put new (unvisited) nodes in Q
  - look at each node at each layer
  - process each node and repeat
  - don’t re-process already-visited nodes
- New twist!
  - don’t treat all unvisited nodes as equals
  - want smallest accumulation of weights
  - so, need to sum weights along the way and maybe pick a different node than what’s in front of the Q
1.4 Physical Gist

- Want shortest path from A to I
- Imagine graph is weights and strings, in which strings are cut to scaled lengths
- Pick up weights one at a time

- Pick up A
- String becomes tight first at B
  record: A→B

- Pick up B
- String becomes tight first at E
  record: A→B→E
- String now becomes tighter at D
- Why? After E comes F or H, each of which is longer than D
- record: A→D
- We could have gotten to E via A

- Pick up D, followed by F
- But, G will be in “next round”
- so, record: A→D→G

- Eventually:
- record: A→D→G→H→I
- Forms a tree
1.5 Pseudocode Gist: Version 1

- Longish algorithm that uses cost in organizing priority queue to choose nodes
- a bit expanded on “wordy gist” from before:
  - pick the highest priority node (the smallest dist)
  - tag the node, record previous, update cost:
    PQ element: <node, accumulating cost>
  - repeat until no more PQ or no more unvisited nodes
    (note: tagging happens *after* extract from PQ)
- Visualization:

<table>
<thead>
<tr>
<th>Current PQ Entry</th>
<th>Current Node</th>
<th>Adjacent Nodes</th>
<th>PQ</th>
<th>Previous Node</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;A, 0&gt;</td>
<td>A</td>
<td></td>
<td>[ ]</td>
<td></td>
</tr>
<tr>
<td>&lt;B, 1&gt;</td>
<td>B</td>
<td>C</td>
<td>[&lt;B, 1&gt;] A</td>
<td></td>
</tr>
<tr>
<td>&lt;C, 2&gt;</td>
<td>C</td>
<td>D</td>
<td>[&lt;C, 2&gt;] A</td>
<td></td>
</tr>
<tr>
<td>&lt;D, 4&gt;</td>
<td>D</td>
<td>E</td>
<td>[&lt;D, 4], &lt;E, 5&gt; C</td>
<td></td>
</tr>
<tr>
<td>&lt;E, 5&gt;</td>
<td>E</td>
<td>F</td>
<td>[&lt;E, 5], &lt;F, 8&gt; D</td>
<td></td>
</tr>
<tr>
<td>&lt;F, 6&gt;</td>
<td>F</td>
<td></td>
<td>[&lt;F, 8&gt;] E</td>
<td></td>
</tr>
</tbody>
</table>
1.6 Code Gist: Version 1

```
resetVertices();
boolean done = false;
SeqStructure toDo = new Heap(edgeCount); // use min heap!
SeqStructure path = new QueueAsList(); // should use stack
Vertex originVertex = (Vertex)verticies.get(origin);
Vertex endVertex = (Vertex)verticies.get(end);
originVertex.setPrev(null);
toDo.put(new MinPQElement(originVertex,0));
while(!done & !toDo.isEmpty()) {
    MinPQElement entry = (MinPQElement) toDo.get();
    Vertex currentVertex = (Vertex) entry.getItem();
    if (!currentVertex.isVisited()) {
        currentVertex.visit();
        currentVertex.setCost(entry.getPriority());
        currentVertex.setPrev(
            ((Vertex)entry.getItem()).getPrev());
        if (currentVertex.equals(endVertex))
            done = true;
    } else {
        for (Iterator edges = currentVertex.
            getEdgeIterator(); edges.hasNext(); ) {
            Edge currentEdge = (Edge) edges.next();
            Vertex nextVertex = currentEdge.getDest();
            if (!nextVertex.isVisited()) {
                int nextCost = currentEdge.getWeight() +
                    currentVertex.getCost();
                nextVertex.setCost(nextCost);
                nextVertex.setPrev(currentVertex);
                toDo.put(new
                    MinPQElement(nextVertex,nextCost));
            }
        } // end for
    } // end if
} // end while
path.put(endVertex);
while(endVertex.hasPrev()) {
    endVertex = endVertex.getPrev();
    path.put(endVertex);
}
return path;
```
1.7 Pseudocode Gist: Version 2

- Data:
  - s: start vertex
  - c(i,j): cost from i to j
  - dist(n): distance from s to n (initially $\infty$)
  - PQ to store neighboring nodes and choose the one with min cost at each “layer” (note: PQ size is edgeCount -> max # of adj nodes)

- Algorithm:
  
  $\text{dist}(s) \leftarrow 0$
  
  while (some vertices are unvisited)
    
    $v \leftarrow$ unmarked vertex with smallest dist (get from the PQ)
    
    tag $v$
    
    for each node $w$ adjacent to $v$
      
      $\text{dist}(w) = \min(\text{dist}(w), \text{dist}(v) + c(v,w))$
      
    end for
  
  end while
1.8 Code Gist: Version 2

public SeqStructure dijkstra3(Object origin, Object end) {

    resetVerticies(Integer.MAX_VALUE);
    SeqStructure toDo = new Heap(edgeCount);
    SeqStructure path = new QueueAsList();

    Vertex originVertex = (Vertex) verticies.get(origin);
    Vertex endVertex = (Vertex) verticies.get(end);

    originVertex.setPrev(null);
    originVertex.setCost(0);
    toDo.put(new MinPQElement(originVertex, 0));

    while (!toDo.isEmpty()) {
        MinPQElement entry = (MinPQElement) toDo.get();
        Vertex currentVertex = (Vertex) entry.getItem();
        currentVertex.visit();

        for (Iterator edges = currentVertex.getEdgeIterator();
             edges.hasNext(); ) {

            Edge currentEdge = (Edge) edges.next();
            Vertex nextVertex = currentEdge.getDest();
            int nextCost =
                currentEdge.getWeight() + currentVertex.getCost();

            if (nextVertex.getCost() > nextCost) {
                nextVertex.setCost(nextCost);
                nextVertex.setPrev(currentVertex);
                toDo.put(new MinPQElement(nextVertex, nextCost));
            }
        }
    }

    path.put(endVertex);
    while (endVertex.hasPrev()) {
        endVertex = endVertex.getPrev();
    }

    path.put(endVertex);
    return path;
}
1.9 Proof Gist

- Induction on iterations of while loop
  - each iteration moves one new node into lifted set
  - do induction on set of nodes ordered in the sequence
    in which they get put into the lifted set
- Induction:
  - base case: path from origin to self is 0
  - inductive hypothesis: assume that the shortest paths to
    all nodes currently in the lifted set have been
    computed correctly
  - inductive hypothesis: the next node that gets lifted is correct
1.10 Run-time Analysis for Adjacency List

- dominant operation of method is while loop (processing unvisited nodes)
- time for processing each vertex:
  - each vertex processed once
  - all edges from a vertex might be processed
  - so, for each node, add up time for each edge
  - so, $O(|V| + |E|)$ (see BFS time)
- PQ ops?
  - worst case: each edge has a node to queue and dequeue (see `for` loop and inner `if`)
  - so, PQ has max length of $|E|$
  - from heap: put is $O(\log n)$, get is $O(\log n)$
  - so, adding each edge’s contribution gives $O(|E| \log |E|)$
- total: $O(|V| + |E| \log |E|)$

1.11 Adjacency Matrix

- see DS&A pg 577
- $O(|V|^2 + |E| \log |E|)$
2. All Pairs Shortest Path

2.1 Problem

- given edge weighted graph
- for each pair of vertices find length of shortest path

2.2 One Solution

- run Dijkstra’s algorithm $|V| +$ times
- use each vertex as the origin

2.3 Floyd’s Algorithm

- use adjacency matrix
- see 16.4.2 in DS&A
3. Spanning Trees

3.1 Interesting Thing About Traversals

- BFS, DFS don’t repeat -> no cycles
- can backtrack to find a new unvisited node, but won’t repeat it
- what does that look like?
- a rooted tree!
- ex) BFS = \{A, B, D, E, G, H, F, I, C\}

```
A → D → G
    ↓     ↓
   B → E → H
       ↓     ↓
      C ← F ← I
```
3.2 Spanning Tree

- effectively a subset of a graph:
  - all nodes sames as in G
  - tree edges must be graph edges (but nec all!)
  - connected
  - acyclic

- constructing?
  - pick a starting edge
  - add edges with unvisited dest nodes
3.3 Minimal Spanning Tree

- given: undirected, weighted graph
- weight of spanning tree = sum of tree edge weights
- **minimum spanning tree**:  
  - any spanning tree with smallest weight  
  - could have many such trees

3.4 Application

- see DS&SD pg 899  
- find a cheap way to connect a bunch of nodes  
  - as in something travelling an entire graph  
  - plane needs to travel to a set of cities  
  - wants cheapest path to take that still hits all cities
3.5 Compare to SSSP

- SSSP: shortest path to a node
  what’s cheapest way to get from A to Z using nodes \{A,…,Z\}
- MST: smallest sum of weights connecting each node
  what’s cheapest way to connect all nodes \{A,…,Z\}?

\[ \text{weighted, undirected graph} \]

\[ \text{SSSP for A→C: } \{A,C\} \]
Tree: \{ \{A,B\}, \{A,C\} \}

\[ \text{MST for graph} \]
Tree: \{ \{A,B\}, \{B,C\} \}
3.6 Prim’s Algorithm

- modify Dijsktra’s Algorithm:
  - put edges in PQ
  - associate edges with length of edge (don’t add costs)
  - otherwise, algorithm is the same

3.7 Kruskal’s Algorithm

- add edges by increasing order of weights
- not allowed to add edges that form cycles
- see DS&A 16.5.2
4. Exercises

- Modify the heap code to use a minimum heap.
- Modify the heap code to provide a sorted string for describing a priority queue.
- Prove by induction that Dijkstra’s algorithm is correct.
- Implement Prim’s algorithm.