Lecture 39: Dijkstra’s Algorithm

CSCI 62
Fall, 2016
Kim Bruce & Peter Mawhorter

Graph Representations

- Adjacency Matrix
- Adjacency List

Adjacency Matrix

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>B</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>C</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>D</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

- Good for dense graphs
- Constant time lookup for edges.
- Symmetric if undirected.

Adjacency Lists

- Good for sparse graphs, saves space
  - lists on right can be vectors, array on left could be map
  - Linear time lookup for edges.
Complexity of BFS

- Enqueue the start node
- while the queue is not empty
  - Dequeue a node
  - if the the node has not been visited previously,
    - visit it
    - enqueue all of the node's children

- Adjacency List:
  - Each edge contributes both end points to queue
  - $O(\max(v,e))$ if each visit takes constant time

- More expensive with adjacency matrix

Single-Source Shortest Paths

- Like Breadth-first search
  - If all paths have length 1
  - Otherwise use priority queue and mark with predecessor so can find shortest path.
  - Assume all edges have non-negative weights.
  - Due to Dijkstra

```plaintext
<table>
<thead>
<tr>
<th></th>
<th>Denver</th>
<th>Chicago</th>
<th>Williams</th>
</tr>
</thead>
<tbody>
<tr>
<td>SF</td>
<td>20</td>
<td>13</td>
<td></td>
</tr>
<tr>
<td>LA</td>
<td>15</td>
<td>42</td>
<td></td>
</tr>
<tr>
<td>Boston</td>
<td></td>
<td>7</td>
<td>11</td>
</tr>
</tbody>
</table>
```

```plaintext
<table>
<thead>
<tr>
<th></th>
<th>Denver</th>
<th>Chicago</th>
<th>Williams</th>
</tr>
</thead>
<tbody>
<tr>
<td>SF</td>
<td>$\infty$</td>
<td>13</td>
<td>$\infty$</td>
</tr>
<tr>
<td>LA</td>
<td>$\infty$</td>
<td>42</td>
<td>$\infty$</td>
</tr>
<tr>
<td>Boston</td>
<td></td>
<td>$\infty$</td>
<td>11</td>
</tr>
</tbody>
</table>
PQ = [SF], adding LA and Denver

PQ = [LA, D], adding B, D

PQ = [D, B], adding C, LA, SF

PQ = [C, B], adding B
Single Source Shortest Path Problem

- From a starting node s, find the shortest path (and its length) to all other (reachable) nodes
- The collection of all shortest paths form a tree, called... the shortest path tree!
- If all edges have the same weight, we can use BFS.
- Otherwise ...

Single Source Shortest Path Problem

- If all edges have weights $\geq 0$, then we use Dijkstra’s algorithm
  - Essentially just BFS with priority queue
  - Priorities are best known distance to a node from the source
  - Keep track of parents as in BFS so can get path
- Example of a “greedy” algorithm!
Dijkstra

- Variables
  - graph G
  - vertex_t s // start node
  - double length[n][n] // Edge lengths (adj. list)
  - double dist[n] // Current best distance
  - vertex_t parent[n] // Current parent
  - pqueue Q // priority queue

Dijkstra Algorithm

- Set dist[v] to \( \infty \) for all \( v \), except \( \text{dist}[s] = 0 \)
- Add \( s \) to \( Q \) with priority \( 0.0 \)
- loop while \( Q \) is not empty:
  - get node cur with min priority \( d \) (distance from \( s \))
  - if \( d \leq \text{dist}[cur] \)
    - for each outgoing edge \( cur \rightarrow v \):
      - if \( \text{dist}[cur] + \text{length}[cur][v] < \text{dist}[v] \):
        - \( \text{dist}[v] = \text{dist}[cur] + \text{length}[cur][v] \)
        - parent[v] = cur
        - Add \( v \) to \( Q \) with new priority \( \text{dist}[v] \)

Run Dijkstra on Sample Graph

Run-Time of Dijkstra

- Let \( v = \#\text{vertices}, e = \#\text{edges} \)
- Adding and removing from priority queue, \( O(\log v) \)
  - Each goes on and off once, so \( O(v \log v) \)
- reduce_priority \( O(\log v) \)
  - Worst case, once for each edge, so \( O(e \log v) \)
- Total time \( O((e+v) \log v) \)