CMSC 132: Object-Oriented Programming II

Graph Implementations & Single Source Shortest Path Algorithm

Department of Computer Science
University of Maryland, College Park

Graph Implementation

How do we represent edges?
- Adjacency matrix
  - 2D array of neighbors
- Adjacency list
  - List of neighbors
- Adjacency set / map
  - Set / map of neighbors

Important for very large graphs
- Affects efficiency / storage
Adjacency Matrix

Representation
- 2D array
  - Position j, k → edge between nodes n_j, n_k

Example

Adjacency Matrix

Representation (cont.)
- Single array for entire graph
- Undirected graph
  - Only upper / lower triangle matrix needed
  - Since n_j, n_k implies n_k, n_j
- Unweighted graph
  - Matrix elements ⇒ boolean
- Weighted graph
  - Matrix elements ⇒ weight
Adjacency List

Representation

- For each node, store
  - List of neighbors / successors
    - Linked list
    - Array list
- For weighted graph
  - Also store weight for each edge
- Undirected graph
  - For every edge (a, b)
    - Nodes a & b need to store each other as neighbor
- Directed graph
  - May also need to store list of predecessors

Adjacency List

Example

- Unweighted graph

<table>
<thead>
<tr>
<th>Node</th>
<th>Neighbor List</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2 → 3</td>
</tr>
<tr>
<td>2</td>
<td>1 → 3 → 4</td>
</tr>
<tr>
<td>3</td>
<td>1 → 2 → 4 → 5</td>
</tr>
<tr>
<td>4</td>
<td>2 → 3 → 5</td>
</tr>
<tr>
<td>5</td>
<td>3 → 4 → 6</td>
</tr>
</tbody>
</table>

- Weighted graph

<table>
<thead>
<tr>
<th>Node</th>
<th>Neighbor List</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>(2, 3.7) → (3, 5.0)</td>
</tr>
<tr>
<td>2</td>
<td>(1, 3.7) → (3, 1.0) → (4, 10.2)</td>
</tr>
<tr>
<td>3</td>
<td>(1, 5.0) → (2, 1.0) → (4, 8.0) → (5, 3.0)</td>
</tr>
<tr>
<td>4</td>
<td>(2, 10.2) → (3, 8.0) → (5, 1.5)</td>
</tr>
<tr>
<td>5</td>
<td>(3, 3.0) → (4, 1.5) → (5, 6.0)</td>
</tr>
</tbody>
</table>
Adjacency Set / Map

**Representation**
- For each node, store
  - Set or map of neighbors / successors
- For unweighted graph
  - Use set of neighbors
- For weighted graph
  - Use map of neighbors, w/ value = weight of edge
- Undirected graph
  - For every edge (a, b)
    - Nodes a & b need to store each other as neighbor
- For directed graph
  - May also need to store map of predecessors

Graph Space Requirements

**Adjacency matrix**
- $\frac{1}{2} N^2$ entries (for graph with N nodes, E edges)
- Many empty entries for large, sparse graphs

**Adjacency list**
- $2E$ entries

**Adjacency set / map**
- $2E$ entries
- Space overhead per entry
  - Higher than for adjacency list
Graph Time Requirements

- **Adjacency matrix**
  - Can find individual edge (a,b) quickly
  - Examine entry in array Edge[a,b]
  - Constant time operation

- **Adjacency list / set / map**
  - Can find all edges for node (a) quickly
  - Iterate through collection of edges for a
  - On average E / N edges per node

Graph Time Requirements

- **Average Complexity of operations**
  - For graph with N nodes, E edges

<table>
<thead>
<tr>
<th>Operation</th>
<th>Adj Matrix</th>
<th>Adj List</th>
<th>Adj Set/Map</th>
</tr>
</thead>
<tbody>
<tr>
<td>Find edge</td>
<td>O(1)</td>
<td>O(E/N)</td>
<td>O(1)</td>
</tr>
<tr>
<td>Insert edge</td>
<td>O(1)</td>
<td>O(E/N)</td>
<td>O(1)</td>
</tr>
<tr>
<td>Delete edge</td>
<td>O(1)</td>
<td>O(E/N)</td>
<td>O(1)</td>
</tr>
<tr>
<td>Enumerate edges for node</td>
<td>O(N)</td>
<td>O(E/N)</td>
<td>O(E/N)</td>
</tr>
</tbody>
</table>
Choosing Graph Implementations

- **Graph density**
  - Ratio edges to nodes (dense vs. sparse)

- **Graph algorithm**
  - **Neighbor based**
    - For each node X in graph
      - For each neighbor Y of X  // adj list faster if sparse
        doWork( )
  - **Connection based**
    - For each node X in ...
      - For each node Y in ...
        - if (X,Y) is an edge  // adj matrix faster if dense
          doWork( )

---

Single Source Shortest Path

- **Common graph problem**
  1. Find path from X to Y with lowest edge weight
  2. Find path from X to any Y with lowest edge weight

- **Useful for many applications**
  - Shortest route in map
  - Lowest cost trip
  - Most efficient internet route

- **Dijkstra’s algorithm solves problem 2**
  - Can also be used to solve problem 1
  - Would use different algorithm if only interested in a single destination
Shortest Path – Dijkstra’s Algorithm

- **Maintain**
  - Nodes with known shortest path from start ⇒ S
  - Cost of shortest path to node K from start ⇒ C[K]
    - Only for paths through nodes in S
  - Predecessor to K on shortest path ⇒ P[K]
    - Updated whenever new (lower) C[K] discovered
    - Remembers actual path with lowest cost

---

Shortest Path – Intuition for Dijkstra’s

- At each step in the algorithm
  - Shortest paths are known for nodes in S
  - Store in C[K] length of shortest path to node K (for all paths through nodes in { S })
  - Add to { S } next closest node
Shortest Path – Intuition for Dijkstra’s

- Update distance to J after adding node K
  - Previous shortest path to K already in C[K]
  - Possibly shorter path to J by going through node K
  - Compare C[J] with C[K] + weight of (K,J), update C[J] if needed

Shortest Path – Dijkstra’s Algorithm

S = ∅
P[ ] = none for all nodes
C[start] = 0, C[ ] = ∞ for all other nodes

while ( not all nodes in S )
  find node K not in S with smallest C[K]
  add K to S
  for each node J not in S adjacent to K
    if ( C[K] + cost of (K,J) < C[J] )
      C[J] = C[K] + cost of (K,J)
      P[J] = K

*Optimal solution computed with greedy algorithm*
Dijkstra’s Shortest Path Example

- Initial state
  - $S = \emptyset$

<table>
<thead>
<tr>
<th>C</th>
<th>P</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0 none</td>
</tr>
<tr>
<td>2</td>
<td>$\infty$ none</td>
</tr>
<tr>
<td>3</td>
<td>$\infty$ none</td>
</tr>
<tr>
<td>4</td>
<td>$\infty$ none</td>
</tr>
<tr>
<td>5</td>
<td>$\infty$ none</td>
</tr>
</tbody>
</table>

Dijkstra’s Shortest Path Example

- Find shortest paths starting from node 1
  - $S = 1$

<table>
<thead>
<tr>
<th>C</th>
<th>P</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0 none</td>
</tr>
<tr>
<td>2</td>
<td>$\infty$ none</td>
</tr>
<tr>
<td>3</td>
<td>$\infty$ none</td>
</tr>
<tr>
<td>4</td>
<td>$\infty$ none</td>
</tr>
<tr>
<td>5</td>
<td>$\infty$ none</td>
</tr>
</tbody>
</table>
Dijkstra’s Shortest Path Example

- Update C[K] for all neighbors of 1 not in \{ S \}
- S = \{ 1 \}

<table>
<thead>
<tr>
<th></th>
<th>C</th>
<th>P</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>none</td>
</tr>
<tr>
<td>2</td>
<td>5</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>8</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>∞</td>
<td>none</td>
</tr>
<tr>
<td>5</td>
<td>∞</td>
<td>none</td>
</tr>
</tbody>
</table>

C[2] = \min (\infty, C[1] + (1,2)) = \min (\infty, 0 + 5) = 5
C[3] = \min (\infty, C[1] + (1,3)) = \min (\infty, 0 + 8) = 8

Find node K with smallest C[K] and add to S

- S = \{ 1, 2 \}
Dijkstra’s Shortest Path Example

- Update C[K] for all neighbors of 2 not in S
- S = \{ 1, 2 \}

<table>
<thead>
<tr>
<th></th>
<th>C</th>
<th>P</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>none</td>
</tr>
<tr>
<td>2</td>
<td>5</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>6</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>15</td>
<td>2</td>
</tr>
<tr>
<td>5</td>
<td>∞</td>
<td>none</td>
</tr>
</tbody>
</table>

C[3] = \min(8, C[2] + (2,3)) = \min(8, 5 + 1) = 6
C[4] = \min(∞, C[2] + (2,4)) = \min(∞, 5 + 10) = 15

Dijkstra’s Shortest Path Example

- Find node K with smallest C[K] and add to S
- S = \{ 1, 2, 3 \}

<table>
<thead>
<tr>
<th></th>
<th>C</th>
<th>P</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>none</td>
</tr>
<tr>
<td>2</td>
<td>5</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>6</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>15</td>
<td>2</td>
</tr>
<tr>
<td>5</td>
<td>∞</td>
<td>none</td>
</tr>
</tbody>
</table>
Dijkstra's Shortest Path Example

- Update $C[K]$ for all neighbors of 3 not in $S$
- \{ $S$ \} = 1, 2, 3

\[
\begin{array}{c|c|c}
   C & P & \text{none} \\
   1 & 0 & \text{none} \\
   2 & 5 & 1 \\
   3 & 6 & 2 \\
   4 & 9 & 3 \\
   5 & \infty & \text{none} \\
\end{array}
\]

$C[4] = \min(15 , C[3] + (3,4)) = \min(15 , 6 + 3) = 9$

Dijkstra's Shortest Path Example

- Find node $K$ with smallest $C[K]$ and add to $S$
- \{ $S$ \} = 1, 2, 3, 4

\[
\begin{array}{c|c|c}
   C & P & \text{none} \\
   1 & 0 & \text{none} \\
   2 & 5 & 1 \\
   3 & 6 & 2 \\
   4 & 9 & 3 \\
   5 & \infty & \text{none} \\
\end{array}
\]
**Dijkstra’s Shortest Path Example**

- Update $C[K]$ for all neighbors of 4 not in $S$
- $S = \{ 1, 2, 3, 4 \}$

<table>
<thead>
<tr>
<th></th>
<th>C</th>
<th>P</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>none</td>
</tr>
<tr>
<td>2</td>
<td>5</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>6</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>9</td>
<td>3</td>
</tr>
<tr>
<td>5</td>
<td>18</td>
<td>4</td>
</tr>
</tbody>
</table>

$C[5] = \min (\infty, C[4] + (4,5)) = \min (\infty, 9 + 9) = 18$

**Dijkstra’s Shortest Path Example**

- Find node $K$ with smallest $C[K]$ and add to $S$
- $S = \{ 1, 2, 3, 4, 5 \}$

<table>
<thead>
<tr>
<th></th>
<th>C</th>
<th>P</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>none</td>
</tr>
<tr>
<td>2</td>
<td>5</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>6</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>9</td>
<td>3</td>
</tr>
<tr>
<td>5</td>
<td>18</td>
<td>4</td>
</tr>
</tbody>
</table>
Dijkstra’s Shortest Path Example

- All nodes in S, algorithm is finished
- \( S = \{ 1, 2, 3, 4, 5 \} \)

<table>
<thead>
<tr>
<th></th>
<th>C</th>
<th>P</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>none</td>
</tr>
<tr>
<td>2</td>
<td>5</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>6</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>9</td>
<td>3</td>
</tr>
<tr>
<td>5</td>
<td>18</td>
<td>4</td>
</tr>
</tbody>
</table>

Dijkstra’s Shortest Path Example

- Find shortest path from start to K
  - Start at K
  - Trace back predecessors in P[ ]
- Example paths (in reverse)
  - 2 → 1
  - 3 → 2 → 1
  - 4 → 3 → 2 → 1
  - 5 → 4 → 3 → 2 → 1