Graphs & Graph Algorithms 2

Fawzi Emad
Chau-Wen Tseng

Department of Computer Science
University of Maryland, College Park

Overview

- Spanning trees
- Minimum spanning tree
  - Kruskal’s algorithm
- Shortest path
  - Djikstra’s algorithm
- Graph implementation
  - Adjacency list / matrix
Spanning Tree

- Tree connecting all nodes in graph
- \(N-1\) edges for \(N\) nodes
- Can build tree during traversal

**Spanning Tree Construction**

for all nodes \(X\)

- set \(X.tag = False\)
- set \(X.parent = Null\)

\{ Discovered \} = \{ 1st node \}

while ( \{ Discovered \} \(\neq \emptyset\) )

- take node \(X\) out of \{ Discovered \}
- if (\(X.tag = False\))
  - set \(X.tag = True\)
  - for each successor \(Y\) of \(X\)
    - if (\(Y.tag = False\))
      - set \(Y.parent = X\)  // add \((X,Y)\) to tree
      - add \(Y\) to \{ Discovered \}
Breadth & Depth First Spanning Trees

Breadth-first

Depth-first

Depth-First Spanning Tree Example
Breadth-First Spanning Tree Example

Minimum Spanning Tree (MST)

- Spanning tree with minimum total edge weight
- Multiple MSTs possible (with same weight)
MST – Kruskal’s Algorithm

sort edges by weight (from least to most)

tree = ∅

for each edge (X,Y) in order
  if it does not create a cycle
    add (X,Y) to tree
  stop when tree has N–1 edges

*Optimal solution computed with greedy algorithm*

MST – Kruskal’s Algorithm Example
MST – Kruskal’s Algorithm

When does adding \((X,Y)\) to tree create cycle?

1. **Traversal approach**
   1. Traverse tree starting at \(X\)
   2. Cycle if reach \(Y\)

2. **Connected subgraph approach**
   1. Maintain set of nodes for each connected subgraph
   2. Initialize one connected subgraph for each node
   3. When edge \((X,Y)\) is added to tree
      - Merge sets containing \(X, Y\)
   4. Cycle if \(X, Y\) in same set

MST – Connected Subgraph Example

<table>
<thead>
<tr>
<th>MST</th>
<th>Sets</th>
<th>Edge being considered for addition</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>([A]), ([B]), ([C]), ([D])</td>
<td>(&lt;A, B&gt;) Include, since it connects two nodes in distinct sets</td>
</tr>
<tr>
<td>2.</td>
<td>([A, B]), ([C]), ([D])</td>
<td>(&lt;A, C&gt;) Include, since it connects two nodes in distinct sets</td>
</tr>
</tbody>
</table>

Original graph:

- \(A\) \(B\) \(C\) \(D\)
- Edges: \(A-B\) 5, \(A-C\) 9, \(B-C\) 13, \(D-C\) 15, \(B-D\) 17
MST – Connected Subgraph Example

Single Source Shortest Path

- Common graph problem
  - Find path from X to Y with lowest edge weight
  - 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

- Can solve both problems with same algorithm
Shortest Path – Djikstra’s Algorithm

Maintain

- Nodes with known shortest path ⇒ { S }
- Cost of shortest path to node ⇒ C[...]
- For paths through nodes in { S }

Algorithm

- Repeat until all nodes in { S }
  - Find node K not in { S } with smallest C[K]
  - Add K to { S }
  - Update C[M] for all neighbors M of K not in { S }
    - By checking whether C[M] can be reduced by first going
to K, then adding weight for edge (K,M)

\[
\begin{align*}
\{ S \} &= \emptyset \\
C[X] &= 0 \\
C[Y] &= \infty \text{ for all other nodes}
\end{align*}
\]

while ( not all nodes in { S } )

- find node K not in { S } with smallest C[K]
- add K to { S }
- for each node M not in { S } adjacent to K
  - C[M] = \min ( C[M] , C[K] + \text{cost of (K,M)} )

*Optimal solution computed with greedy algorithm*
Shortest Path – Intuition for Djikstra’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

- Update distance to \( J \) after adding node \( K \)
  - Previous shortest paths already in \( C[K] \)
  - Possibly shorter path by going through node \( K \)
  - Compare \( C[J] \) to \( C[K] \) + weight of \((K,J)\)
Dijkstra’s Shortest Path Example

Initial state
- \{ S \} = \emptyset
- C[1] = 0
- C[2] = \infty
- C[3] = \infty
- C[4] = \infty
- C[5] = \infty

Find node K with smallest C[K] and add to \{ S \}
- \{ S \} = 1
- C[1] = 0
- C[2] = \infty
- C[3] = \infty
- C[4] = \infty
- C[5] = \infty
Dijkstra’s Shortest Path Example

- Update \( C[K] \) for all neighbors of 1 not in \( \{ S \} \)
- \( \{ S \} = 1 \)
- \( C[1] = 0 \)
- \( C[2] = 5 \)
- \( C[3] = 8 \)
- \( C[4] = \infty \)
- \( C[5] = \infty \)

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

Dijkstra’s Shortest Path Example

- Find node \( K \) with smallest \( C[K] \) and add to \( \{ S \} \)
- \( \{ S \} = 1, 2 \)
- \( C[1] = 0 \)
- \( C[2] = 5 \)
- \( C[3] = 8 \)
- \( C[4] = \infty \)
- \( C[5] = \infty \)
Dijkstra’s Shortest Path Example

- Update $C[K]$ for all neighbors of 2 not in $\{ S \}$
- $\{ S \} = 1, 2$
- $C[1] = 0$
- $C[2] = 5$
- $C[3] = 6$
- $C[4] = 15$
- $C[5] = \infty$

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

Dijkstra’s Shortest Path Example

- Find node $K$ with smallest $C[K]$ and add to $\{ S \}$
- $\{ S \} = 1, 2, 3$
- $C[1] = 0$
- $C[2] = 5$
- $C[3] = 6$
- $C[4] = 15$
- $C[5] = \infty$
Dijkstra's Shortest Path Example

- Update $C[K]$ for all neighbors of 3 not in $\{ S \}$
- $\{ S \} = 1, 2, 3$
- $C[1] = 0$
- $C[2] = 5$
- $C[3] = 6$
- $C[4] = 9$
- $C[5] = \infty$

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

Dijkstra's Shortest Path Example

- Find node $K$ with smallest $C[K]$ and add to $\{ S \}$
- $\{ S \} = 1, 2, 3, 4$
- $C[1] = 0$
- $C[2] = 5$
- $C[3] = 6$
- $C[4] = 9$
- $C[5] = \infty$
Dijkstra’s Shortest Path Example

- Update $C[K]$ for all neighbors of 4 not in $\{ S \}$
- $\{ S \} = 1, 2, 3, 4$
- $C[1] = 0$
- $C[2] = 5$
- $C[3] = 6$
- $C[4] = 9$
- $C[5] = 18$

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

Dijkstra’s Shortest Path Example

- Find node $K$ with smallest $C[K]$ and add to $\{ S \}$
- $\{ S \} = 1, 2, 3, 4, 5$
- $C[1] = 0$
- $C[2] = 5$
- $C[3] = 6$
- $C[4] = 9$
- $C[5] = 18$
Dijkstra’s Shortest Path Example

- No more nodes in \{ S \}
- \{ S \} = 1, 2, 3, 4, 5
- \( C[1] = 0 \)
- \( C[2] = 5 \)
- \( C[3] = 6 \)
- \( C[4] = 9 \)
- \( C[5] = 18 \)

Graph Implementation

- Representations
  - Explicit edges (a,b)
    - Maintain set of edges for every node
  - Adjacency matrix
    - 2D array of neighbors
  - Adjacency list
    - Linked list of neighbors

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

- **Representation**
  - 2D array
  - Position \( j, k \) \( \Rightarrow \) edge between nodes \( n_j, n_k \)
  - Unweighted graph
    - Matrix elements \( \Rightarrow \) boolean
  - Weighted graph
    - Matrix elements \( \Rightarrow \) weight

---

**Example**

![Adjacency Matrix Example](image)
Adjacency Matrix

Properties
- Single array for entire graph
- Only upper / lower triangle matrix needed for undirected graph
  - Since \( n_j, n_k \) implies \( n_k, n_j \)

Adjacency List

Representation
- Linked list for each node
- Unweighted graph
  - store neighbor
- Weighted graph
  - store neighbor, weight
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 → 5</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>

Graph Space Requirements

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

Adjacency list
- E edges
- Each edge stores reference to node & next edge

Explicit edges
- E edges
- Each edge stores reference to 2 nodes
Graph Time Requirements

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

<table>
<thead>
<tr>
<th>Operation</th>
<th>Adj Matrix</th>
<th>Adj List</th>
</tr>
</thead>
<tbody>
<tr>
<td>Find edge</td>
<td>$O(1)$</td>
<td>$O(E/N)$</td>
</tr>
<tr>
<td>Insert node</td>
<td>$O(1)$</td>
<td>$O(E/N)$</td>
</tr>
<tr>
<td>Insert edge</td>
<td>$O(1)$</td>
<td>$O(E/N)$</td>
</tr>
<tr>
<td>Delete node</td>
<td>$O(N)$</td>
<td>$O(E)$</td>
</tr>
<tr>
<td>Delete edge</td>
<td>$O(1)$</td>
<td>$O(E/N)$</td>
</tr>
</tbody>
</table>