Chapter 9
Graph algorithms
Sample Graph Problems

- Path problems.
- Connectedness problems.
- Spanning tree problems.
Path Finding

Path between 1 and 8.

Path length is 20.
Another Path Between 1 and 8

Path length is 28.
Example Of No Path

No path between 2 and 9.
Connected Graph

• Undirected graph.
• There is a path between every pair of vertices.
Example of a graph Not Connected
Connected Graph Example
Connected Components

The graph is divided into two connected components:

Component 1: 1, 2, 3, 4, 5, 6, 7

Component 2: 8, 9, 11
Connected Component

- A maximal subgraph that is connected.
  - Cannot add vertices and edges from original graph and retain connectedness.
- A connected graph has exactly 1 component.
Communication Network

Each edge is a link that can be constructed (i.e., a feasible link).
Communication Network Problems

• Is the network connected?
  ▪ Can we communicate between every pair of cities?

• Find the components.

• Want to construct smallest number of feasible links so that resulting network is connected.
Cycles And Connectedness

Removal of an edge that is on a cycle does not affect connectedness.
Cycles And Connectedness

Connected subgraph with all vertices and minimum number of edges has no cycles.
Tree

- Connected graph that has no cycles.
- An $n$ vertex connected graph with $n-1$ edges.
- A connected graph in which removal of any edge makes it unconnected.
- A cyclic graph in which addition of any edges introduces a cycle.
Spanning Tree

• Subgraph that includes all vertices of the original graph.
• Subgraph is a tree.
  ▪ If original graph has $n$ vertices, the spanning tree has $n$ vertices and $n-1$ edges.
Minimum Cost Spanning Tree

- Tree cost is sum of edge weights/costs.
A Spanning Tree

Spanning tree cost = 51.
Minimum Cost Spanning Tree

Spanning tree cost = 41.
Graph Representation

• Adjacency Matrix
• Adjacency Lists
  ▪ Linked Adjacency Lists
  ▪ Array Adjacency Lists
Adjacency Matrix

- 0/1 n x n matrix, where n = # of vertices
- A[i,j] = 1 iff (i,j) is an edge

```
1 2 3 4 5
1 0 1 0 1 0
2 1 0 0 0 1
3 0 0 0 0 1
4 1 0 0 0 1
5 0 1 1 1 0
```
Adjacency Matrix Properties

- Diagonal entries are zero.
- Adjacency matrix of an undirected graph is symmetric.

$A(i,j) = A(j,i)$ for all $i$ and $j$. 

```
     1  2  3  4  5
1  1  0  1  0  1  0
2  0  1  0  0  0  1
3  0  0  1  0  0  1
4  0  1  0  0  0  1
5  0  0  1  1  1  1
```
Adjacency Matrix (Digraph)

- Diagonal entries are zero.
- Adjacency matrix of a digraph need not be symmetric.

```
<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
```
Adjacency Matrix

- $n^2$ bits of space
- For an undirected graph, may store only lower or upper triangle (exclude diagonal).
  - $(n-1)n/2$ bits
- $O(n)$ time to find vertex degree and/or vertices adjacent to a given vertex.
- $O(1)$ time to determine if there is an edge between two given vertices.
Adjacency Lists

- Adjacency list for vertex \( i \) is a linear list of vertices adjacent from vertex \( i \).
- An array of \( n \) adjacency lists.

\[
\begin{align*}
\text{aList}[1] &= (2,4) \\
\text{aList}[2] &= (1,5) \\
\text{aList}[3] &= (5) \\
\text{aList}[4] &= (5,1) \\
\text{aList}[5] &= (2,4,3)
\end{align*}
\]
Linked Adjacency Lists

- Each adjacency list is a chain.

Array Length = \( n \)

# of chain nodes = \( 2e \) (undirected graph)

# of chain nodes = \( e \) (digraph)
Weighted Graphs

- Cost adjacency matrix.
  - $C(i,j) =$ cost of edge $(i,j)$
- Adjacency lists => each list element is a pair (adjacent vertex, edge weight)
Single-source Shortest path problem

- directed, weighted graph is the input
- specified source s.
- want to compute the shortest path from s to all the vertices.
Example:

Figure 24.2  (a) A weighted, directed graph with shortest-path weights from source $s$. (b) The shaded edges form a shortest-paths tree rooted at the source $s$. (c) Another shortest-paths tree with the same root.
Dijkstra’s algorithm:

• Works when there are no negative weight edges.

• takes time $O(e \log n)$ where $e = \text{number of edges}$, $n = \text{number of vertices}$.

• suppose $n \sim 10^5$, $e \sim 10^6$, then the number of computations $\sim 2 \times 10^6$
Data structures needed:

- Adjacency list rep. of graph
- a heap
- some additional structures (e.g. array)

\[
\text{\textbf{Initialize-Single-Source}}(G, s)
\]

1. \textbf{for} each vertex \( v \in V[G] \)
2. \hspace{1em} \textbf{do} \( d[v] \leftarrow \infty \)
3. \hspace{1em} \textbf{do} \( \pi[v] \leftarrow \text{NIL} \)
4. \hspace{1em} \textbf{do} \( d[s] \leftarrow 0 \)
key operation: relaxation on edge

For each node $v$, the algorithm assigns a value $d[v]$ which gets updated and will in the end become the length of the shortest path from $s$ to $v$.

![Diagram](image)

**Figure 24.3** Relaxation of an edge $(u, v)$ with weight $w(u, v) = 2$. The shortest-path estimate of each vertex is shown within the vertex. (a) Because $d[v] > d[u] + w(u, v)$ prior to relaxation, the value of $d[v]$ decreases. (b) Here, $d[v] \leq d[u] + w(u, v)$ before the relaxation step, and so $d[v]$ is unchanged by relaxation.
implementation of relaxation

\texttt{RELAX}(u, v, w)

1 \ \textbf{if} \ d[v] > d[u] + w(u, v)
2 \ \textbf{then} \ d[v] \leftarrow d[u] + w(u, v)
3 \ \pi[v] \leftarrow u
Dijkstra’s algorithm

\textsc{Dijkstra}(G, w, s)
1. \textsc{Initialize-Single-Source}(G, s)
2. \( S \leftarrow \emptyset \)
3. \( Q \leftarrow V[G] \)
4. \textbf{while} \( Q \neq \emptyset \)
5. \hspace{1em} \textbf{do} \( u \leftarrow \textsc{Extract-Min}(Q) \)
6. \hspace{2em} \( S \leftarrow S \cup \{u\} \)
7. \hspace{1em} \textbf{for each} vertex \( v \in \text{Adj}[u] \)
8. \hspace{2em} \textbf{do} \textsc{Relax}(u, v, w)
Dijkstra’s algorithm – Example

Figure 24.6 The execution of Dijkstra’s algorithm. The source s is the leftmost vertex. The shortest-path estimates are shown within the vertices, and shaded edges indicate predecessor values. Black vertices are in the set S, and white vertices are in the min-priority queue $Q = V - S$. (a) The situation just before the first iteration of the while loop of lines 4–8. The shaded vertex has the minimum $d$ value and is chosen as vertex $u$ in line 5. (b)–(f) The situation after each successive iteration of the while loop. The shaded vertex in each part is chosen as vertex $u$ in line 5 of the next iteration. The $d$ and $\pi$ values shown in part (f) are the final values.
Course summary