1. [Quick Review]

(a) Draw the graph represented by the following adjacency matrix.

<table>
<thead>
<tr>
<th></th>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
<th>e</th>
<th>f</th>
<th>g</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>7</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>b</td>
<td>5</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>c</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>d</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>5</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>e</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>f</td>
<td>4</td>
<td>1</td>
<td></td>
<td>2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>g</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**ANSWER** See Figure 1.

(b) Draw the adjacency list representation for this graph.

**ANSWER** See Figure 1.

(c) What is the sequence of vertices visited when we perform depth-first search from a?

**ANSWER** a c d b e g f (Note: not unique)

(d) What is the sequence of vertices visited when we perform breadth-first search from a?

**ANSWER** a b c d e g f (Note: not unique)

(e) Find the shortest path to all the other nodes from a using Dijkstra’s algorithm.

**ANSWER** See Figure 2

---

Figure 1: Answer for question 1(a) and (b)

Figure 2: Shortest path for question 1(e)
2. [Representations of Graph]

(a) An undirected graph is a graph where the edges are unordered, i.e., edge 
\((u,v)\) is the same as edge \((v,u)\). How can adjacency list and adjacency 
matrix compactly represent an undirected, weighted graph? Show how to 
query if an edge \((i,j)\) exists in the graph.

**ANSWER** To check if \((i,j)\) exists, we query for edge \((i,j)\) if \(i < j\) and 
query for \((j,i)\) if \(j < i\). We only need to store one copy of edge \((i,j)\). For 
adjacency matrix, we can use half the matrix by using a ragged 2D array.

(b) Let \(n_i\) be the number of outgoing edges of a vertex \(i\) and \(m_i\) be the number 
of incoming edges of a vertex \(i\). Show how to modify the adjacency list 
representation so that we can list all incoming edges of \(i\) in \(O(m_i)\) time 
and all outgoing edges of \(i\) in \(O(n_i)\) time.

**ANSWER** See Figure 3 for an illustration of the data structure.

![Figure 3: Answer for question 2(b)](image)

(c) Let \(n_i\) be the number of vertices adjacent to a vertex \(i\). Suppose we want 
to support the following four operations on a graph: insert\((i,j)\), which adds 
an edge \((i,j)\) into the graph; delete\((i,j)\), which removes the edge \((i,j)\) from 
the graph; exists\((i,j)\), which checks if edge \((i,j)\) exists in the graph; and 
neighbours\((i)\), which returns the list of vertices adjacent to \(i\). Give a data 
structure that supports insert\((i,j)\), delete\((i,j)\) and exists\((i,j)\) in \(O(1)\) time 
on average, and neighbours\((i)\) in \(O(n_i)\) time.

**ANSWER** Use an adjacency list, where the lists are doubly linked, and 
a hash table where \((i,j)\) is the key. Hash entries for key \((i,j)\) contains 
references to a node representing \((i,j)\) in the adjacency list.

3. [Breadth-first Search] A 1-2 graph is a directed weighted graph whose edges 
have weights either 1 or 2. Show how to modify breadth-first search so that it 
can calculate the shortest paths from a given vertex in \(O(V + E)\) time.

**ANSWER** Transform the input \(G\) into an unweighted graph \(G' = (V', E')\) by 
inserting additional vertices into \(G\): For every edge \((u,v)\) with weight 2, insert 
a new vertex \(x\) and replace edge \((u,v)\) with edges \((u, x)\) and \((x, v)\). BFS on \(G'\) 
is still \(O(V + E)\) because \(|V'| = O(V + E)\) and \(|E'| = O(E)\).

4. [Longest Path] Bob thinks that computing the single-source longest path in a 
positive weighted graph can be done in the same running time as single-source 
shortest path by modifying Dijkstra’s algorithm. Is he correct? Either show 
what modifications are needed, or gives a different algorithm.

**ANSWER** Finding longest paths cannot be solved by modifying Dijkstra’s algorithm. 
To find longest path, use exhaustive search, which will give exponential 
running time. (NOTE to TAs: This is a good place to tell stories about 
NP-complete problems.)