1. Consider the following graph:

(a) Construct the adjacency matrix and adjacency list for the above graph.

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></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>7</td>
</tr>
<tr>
<td>b</td>
<td></td>
<td>17</td>
<td>23</td>
<td></td>
<td></td>
<td>9</td>
<td></td>
</tr>
<tr>
<td>c</td>
<td>17</td>
<td></td>
<td>4</td>
<td></td>
<td></td>
<td>25</td>
<td></td>
</tr>
<tr>
<td>d</td>
<td></td>
<td>4</td>
<td></td>
<td>5</td>
<td>31</td>
<td></td>
<td></td>
</tr>
<tr>
<td>e</td>
<td>7</td>
<td>23</td>
<td></td>
<td></td>
<td>14</td>
<td></td>
<td></td>
</tr>
<tr>
<td>f</td>
<td></td>
<td>25</td>
<td>5</td>
<td>14</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>g</td>
<td>12</td>
<td>9</td>
<td>31</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Adjacency list:

<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>(e,7), (g,12)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>b</td>
<td>(c,17), (e,23), (g,9)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>c</td>
<td>(b,17), (d,4), (f,25)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>d</td>
<td>(c,4), (f,5), (g,31)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>e</td>
<td>(a,7), (b,23), (f,14)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>f</td>
<td>(c,25), (d,5), (e,14)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>g</td>
<td>(a,12), (b,9), (d,31)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
(b) Run dfs on the above graph and record the order vertices are visited. Assume neighbors are examined in alphabetical order.
   a e b c d f g

(c) Run bfs on the above graph and record the order vertices are visited. Assume neighbors are examined in alphabetical order.
   a e g b f d c

(d) Run Dijkstra’s algorithm on the above graph starting from vertex a and record the shortest distances to each node. Highlight or darken the edges that form the shortest paths.

   Distances:
   \[
   \begin{array}{c|c}
   \text{a} & 0 \\
   \text{b} & 21 \\
   \text{c} & 30 \\
   \text{d} & 26 \\
   \text{e} & 7 \\
   \text{f} & 21 \\
   \text{g} & 12 \\
   \end{array}
   \]

2. Suppose a depth first search is run on a rooted binary tree where the neighbor ordering is left child and then right child. The order the DFS visits the nodes of the tree is the same as something else we’ve seen previously. What is it?

   It is the same as a preorder traversal. This can be observed by looking at the code for both. The preorder traversal code has unrolled the DFS loop.

3. Dijkstra’s algorithm is guaranteed to compute the correct shortest path distances when the edge weights are non-negative. When run on graphs with negative edge weights, Dijkstra’s algorithm might return incorrect distances.

   (a) Draw a graph and indicate a start vertex such that Dijkstra’s algorithm will succeed and return correct distances.
      Starting from vertex a.

   \[
   \begin{array}{c}
   \text{a} \quad 5 \quad \text{b} \quad -3 \quad \text{c} \quad 4 \quad \text{d} \\
   \end{array}
   \]

   (b) Draw a graph and indicate a start vertex such that Dijkstra’s algorithm will fail and return incorrect distances.
      Starting from vertex a.
4. The breadth first search we described in class outputs the vertices sorted by the number of edges between each vertex and the start vertex i.e. all the vertices one edge away from the start come before the vertices two edges away from the start and so on. Alter breadth first search to also compute the number of edges between each vertex and the start vertex.

```python
def bfs(G,v):
    level = map()
    level[v] = 0
    q = queue()
    q.push(v)
    while q is not empty:
        v = q.pop()
        mark v as visited
        for w in neighbors(v):
            if w is not visited and not in q:
                q.push(w)
                level[w] = level[v] + 1
```

Extra credit: The diameter of a graph is the largest distance between any two nodes. The small world phenomenon is the phenomenon that in certain graphs the diameter is very small. For example in the graph where the nodes are people and the edges are social relationships, the distance between any two people tends to be less than six giving rise to the term “six degrees of separation”. Describe an algorithm that runs in $O(n^2 \log n + nm)$ time to compute the diameter of a graph.

Run Dijkstra’s algorithm from every single vertex. The diameter of the graph will be the largest shortest path distance found by any of the Dijkstra’s algorithm runs. This takes $n$ runs of Dijkstra’s algorithm so in total it takes $O(n^2 \log n + nm)$ time. If there are negative edge weights, we can instead use one run of Floyd-Warshall.