Graph Algorithm Selection (10/50 points)
You are planning a trip to Los Angeles, California and will be flying into Los Angeles’ LAX airport. You have a directed graph of all the possible flights in the US with the price of the ticket as the weight of the edges. For each of the following situations, choose the appropriate graph algorithm and briefly (1-2 sentences) explain your choice.

1. You want to fly out of JFK Airport in New York and want to find out if there is a direct flight from JFK to LAX.
   a) breadth-first search  
   b) depth-first search  
   c) Prim’s algorithm  
   d) Kruskal’s algorithm  
   e) Dijkstra’s algorithm  
   f) Floyd’s algorithm  

   Explanation:
   **Implementing breadth-first search one depth can find the direct flight efficiently.**

2. You don’t have very much money and want to find the cheapest possible flight or series of flights from JFK Airport in New York to LAX.
   a) breadth-first search  
   b) depth-first search  
   c) Prim’s algorithm  
   d) Kruskal’s algorithm  
   e) Dijkstra’s algorithm  
   f) Floyd’s algorithm  

   Explanation:
   **Dijkstra’s algorithm is efficient to find the shortest paths from one source vertex.**

3. You are planning the trip for a large group of people that will be flying out of airports all over the country. You want to find out the cheapest possible flight or series of flights from any airport in the country to LAX.
   a) breadth-first search  
   b) depth-first search  
   c) Prim’s algorithm  
   d) Kruskal’s algorithm  
   e) Dijkstra’s algorithm  
   f) Floyd’s algorithm  

   Explanation:
   **Dijkstra’s algorithm:** Dijkstra’s algorithm can be used to find the shortest paths to one destination vertex by reversing all edges (flights).  
   **Floyd’s algorithm:** Floyd’s algorithm is to find all source shortest paths, which can of course be used to find the shortest paths to one destination vertex.
Graph Algorithm Complexity (10/50 points)
Below is pseudo-code for Dijkstra’s algorithm. Assume that implementation of the algorithm uses a heap for the priority queue and an adjacency list for representing the graph. It is unknown whether the graph is sparse or dense. Assume that the size of the priority queue is |E|. Derive the complexity. Show line-by-line complexity analysis using appropriate indentation. Show overall (un-simplified) complexity in the first box at the bottom of the page. Then, simplify the complexity and show it in the second box at the bottom of the page.

Dijkstra(G,s0)

//Initialize
n = |V|; O(1)
createtable(n); //stores k,d,p O(V)
createpq(|E|); //empty pq O(1)
table[s0].d = 0; O(1)
insertpq(0,s0); O(1)

while (!pq.isempty) do O(E)
v0 = pq.getMin(); O(logE)
if (!table[v0].k) then //not known O(V/E)
table[v0].k = true; O(1)
for each vi ∈ Adj[v0] do O(1 + E/V)
   newd=table[v0].d + weight(vi,v0); O(1)
   if (table[vi].d) > newd) then O(1)
      table[vi].d = newd; O(1)
      table[vi].p = v0; O(1)
      insertpq(newd,vi); O(logE)

4. Un-simplified Complexity
O(1 + V + 1 + 1 + 1 + E(logE + V/E(1 + (1 + E/V)(1 + 1(1 + 1 + logE)))))

5. Simplified Complexity
O(V*logE + E*logE)
Shortest Path Algorithms (10/50 points)
Use the following graph for this problem.

6. Show each iteration of Dijkstra’s algorithm on the graph with a starting point of node F. The first iteration is provided for you.

```
Iteration 1      Iteration 2      Iteration 3      Iteration 4      Iteration 5      Iteration 6

<table>
<thead>
<tr>
<th></th>
<th>k</th>
<th>D</th>
<th>p</th>
<th></th>
<th>k</th>
<th>d</th>
<th>p</th>
<th></th>
<th>k</th>
<th>d</th>
<th>p</th>
<th></th>
<th>k</th>
<th>d</th>
<th>p</th>
<th></th>
<th>k</th>
<th>d</th>
<th>p</th>
<th></th>
<th>k</th>
<th>d</th>
<th>p</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>F</td>
<td>-</td>
<td>-</td>
<td>A</td>
<td>F</td>
<td>-</td>
<td>-</td>
<td>A</td>
<td>F</td>
<td>7</td>
<td>D</td>
<td>A</td>
<td>F</td>
<td>7</td>
<td>D</td>
<td>A</td>
<td>T</td>
<td>7</td>
<td>D</td>
<td>A</td>
<td>T</td>
<td>7</td>
<td>D</td>
<td></td>
</tr>
<tr>
<td>B</td>
<td>F</td>
<td>-</td>
<td>-</td>
<td>B</td>
<td>F</td>
<td>9</td>
<td>E</td>
<td>B</td>
<td>F</td>
<td>9</td>
<td>E</td>
<td>B</td>
<td>F</td>
<td>9</td>
<td>E</td>
<td>B</td>
<td>T</td>
<td>9</td>
<td>E</td>
<td>B</td>
<td>T</td>
<td>9</td>
<td>E</td>
<td></td>
</tr>
<tr>
<td>C</td>
<td>F</td>
<td>4</td>
<td>F</td>
<td>C</td>
<td>F</td>
<td>4</td>
<td>F</td>
<td>C</td>
<td>T</td>
<td>4</td>
<td>F</td>
<td>C</td>
<td>T</td>
<td>4</td>
<td>F</td>
<td>C</td>
<td>T</td>
<td>4</td>
<td>F</td>
<td>C</td>
<td>T</td>
<td>4</td>
<td>F</td>
<td></td>
</tr>
<tr>
<td>D</td>
<td>F</td>
<td>2</td>
<td>F</td>
<td>D</td>
<td>F</td>
<td>2</td>
<td>F</td>
<td>D</td>
<td>T</td>
<td>2</td>
<td>F</td>
<td>D</td>
<td>T</td>
<td>2</td>
<td>F</td>
<td>D</td>
<td>T</td>
<td>2</td>
<td>F</td>
<td>D</td>
<td>T</td>
<td>2</td>
<td>F</td>
<td></td>
</tr>
<tr>
<td>E</td>
<td>F</td>
<td>1</td>
<td>F</td>
<td>E</td>
<td>T</td>
<td>1</td>
<td>F</td>
<td>E</td>
<td>T</td>
<td>1</td>
<td>F</td>
<td>E</td>
<td>T</td>
<td>1</td>
<td>F</td>
<td>E</td>
<td>T</td>
<td>1</td>
<td>F</td>
<td>E</td>
<td>T</td>
<td>1</td>
<td>F</td>
<td></td>
</tr>
<tr>
<td>F</td>
<td>T</td>
<td>0</td>
<td>-</td>
<td>F</td>
<td>T</td>
<td>0</td>
<td>-</td>
<td>F</td>
<td>T</td>
<td>0</td>
<td>-</td>
<td>F</td>
<td>T</td>
<td>0</td>
<td>-</td>
<td>F</td>
<td>T</td>
<td>0</td>
<td>-</td>
<td>F</td>
<td>T</td>
<td>0</td>
<td>-</td>
<td></td>
</tr>
</tbody>
</table>
```

7. Draw the resulting directed graph representing the single source shortest paths from node F. Include edge weights in your drawing.
Text Compression Algorithms (10/50 points)
The following question is related to text compression using the Huffman Encoding Algorithm.

8. Given the following text string, populate the frequency table:

‘CREATIVE PROJECTS WITH LEGOS MINDSTORMS’

Frequency Table

<table>
<thead>
<tr>
<th>A</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>G</th>
<th>H</th>
<th>I</th>
<th>J</th>
<th>L</th>
<th>M</th>
<th>N</th>
<th>O</th>
<th>P</th>
<th>R</th>
<th>S</th>
<th>T</th>
<th>V</th>
<th>W</th>
<th>Space</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>1</td>
<td>4</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>3</td>
<td>1</td>
<td>3</td>
<td>4</td>
<td>4</td>
<td>1</td>
<td>1</td>
<td>4</td>
</tr>
</tbody>
</table>

9. Suppose that the Frequency Table is as follows (by the way, it is not). Draw the Huffman Tree. When creating the Huffman Tree, you must apply the following rules for tie-breaking during the combination of two subtrees:

- If choosing between two subtrees with the same weight but different number of nodes, always pick the subtree with the smallest number of nodes first.
- If choosing between two subtrees with the same weight and the same number of nodes, always pick the subtree whose member has the lowest alphabetical order (i.e., pick a subtree containing ‘a’ before picking a subtree containing ‘b’).

Frequency Table

<table>
<thead>
<tr>
<th>X</th>
<th>Q</th>
<th>K</th>
<th>Z</th>
<th>V</th>
<th>W</th>
<th>H</th>
<th>G</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>5</td>
<td>7</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>3</td>
<td>4</td>
<td>2</td>
</tr>
</tbody>
</table>

10. Suppose that the string is ‘cabfeddefb’ and the Huffman tree is as follows:
What is the Huffman encoding of the string?

100101101011111110111001011100101

11. What is the total number of bits for Huffman encoding of the string? 32
Dynamic Programming (10/50 points)
12. The binomial coefficient is defined as...

\[
\binom{n}{k} = \begin{cases} 
\frac{n-1}{k-1} + \frac{n-1}{k}, & 0 < k < n \\
1, & k = 0 \text{ or } k = n 
\end{cases}
\]

A recursive algorithm that solves the binomial coefficient follows:

```c
int BiCoeff(int n, int k) {
    if (k==0 || k==n) return 1;
    else return BiCoeff(n-1,k-1) + BiCoeff(n-1,k);
}
```

Rewrite the recursive algorithm as an efficient, iterative, bottom-up dynamic program. Assume the presence of a global, two-dimensional array: int mem[maxN+1][maxK+1]. Your algorithm must run in O(nk) time.

```c
int BiCoeff(int n, int k) {
    int mem [n+1][k+1];

    for (int i = 0; i <= n; i++)
        mem [i][0] = 1;

    for (int j = 1; j <= k; j++)
        for (int i = j; i <= n; i++)
            if (i == j) mem [i][j] = 1;
            else mem [i][j] = mem [i-1][j-1] + mem [i-1][j];

    return mem [n][k];
}
```
Alternative Answer (slightly better)

```c
int BiCoeff(int n, int k)
{
    int mem[n+1][k+1];

    for (int j = 0; j <= k; j++)
    {
        for (int i = j; i <= n; i++)
        {
            if ((i == j) || (j == 0)) mem[i][j] = 1;
            else mem[i][j] = mem[i-1][j-1] + mem[i-1][j];
        }
    }
    return mem[n][k];
}
```