1. (40 points) Below is a weighted undirected graph. (a) Show the DFS (depth-first search) tree found by the recursive DFS algorithm and list the vertices in the order of adding to the DFS tree. (b) Show the DFS tree found by the non-recursive DFS algorithm and list the vertices in the order of adding to the DFS tree. (c) Show the BFS (breadth-first search) tree found by the BFS algorithm and list the vertices in the order of adding to the BFS tree. (d) Show the MST (minimum spanning tree) found by Prim’s algorithm and list the vertices in the order of adding to the MST. (e) Compute the shortest paths from vertex A to all other vertices using Dijkstra’s algorithm and list the vertices in the order of adding to the cloud. For all the above questions, we start with vertex A and ties are broken by alphabet order of vertices.

**Sol.** (a) [8] The order of vertices is A, B, C, D, E, F, I, G, H.

(b) [8] The order of vertices is A, B, F, G, H, I, E, D, C.
(c) [8] The order of vertices is A, B, F, G, C, E, I, H, D.

(d) [8] The order of vertices is A, B, G, I, E, D, C, H, F.

(e) [8] The paths are given in the following tree and the distances from A are given inside the circle of each vertex. The order of vertices is A, B, G, I, C, E, H, D, F.

2. (30 points) Let \( S = \{ L_i \mid 1 \leq i \leq n \}, \) \( L_i \) is a sorted list of numbers \}, and for each list \( L_i \), let \( a_i = |L_i| \) be the number of elements in \( L_i \). We are given a function \( \text{merge}(L, L') \) which takes time \( O(|L| + |L'|) \) to merge \( L \) and \( L' \) into one sorted list, and returns the result. To merge all lists in \( S \) into one sorted list, we may use the following algorithm:

\[
\text{mergeAll}(S) \{
    n = |S|;
\}
\]
while (n > 1) {
    S' = { }; // assume S = { Li | 1 ≤ i ≤ n }
    for (i = 1; i<n; i=i+2) S' = S' U { merge(Li, Li+1) }
    S = S'; n = |S|;
}}

(a) Please analyze the complexity of mergeAll(S) in terms of n and a_i = |L_i| for 1 ≤ i ≤ n.
(b) Show that mergeAll is not the fast algorithm by providing a counter example.
(c) Design an efficient algorithm to solve this problem.

**Sol:**

(a) [8] Let A = \( \sum_{1 \leq i \leq n} a_i \), where a_i is the size of original L_i. The size of S is halved after by the for loop. So the body of the while loop will execute log n times. The total cost of each for loop is O(A). So the total complexity is O(A log n).

(b) [8] Suppose S contains 4 lists of sizes 1k, 1k, 1k, 100k, respectively, where k is an integer. The algorithm mergeAll will take 2 iterations and the total cost is 2*(103k) = 206k. If we merge the first two lists, the cost is 2k, we then merge the result of the first merge with the third list of size 1k, the cost is 3k. Finally, we merge it with the list of 100k, the cost is 103k. The total cost is 2k + 3k + 103k = 106k. So mergeAll takes almost twice the cost comparing to the cumulative merge.

(c) [14] We may use the greedy technique like Huffman’s coding always to merge two shortest lists first.

```
greedyMergeAll(S) {
    Q = createQueue(S);
    while (Q.size > 1) {
        L1 = removeMin(Q);
        L2 = removeMin(Q);
        insert(Q, merge(L1, L2));
    }
}
```

Let d_i be the number of times that the original L_i is involved in the merge operation. Since the queue operations removeMin and insert take O(log n) time, the total complexity of greedyMergeAll is O(n log n + \( \sum_{1 \leq i \leq n} d_i a_i \)). Using the proof for Huffman’s coding, this is optimal for all the algorithms using merge. When all a_i are equal, d_i = log n, and the total cost is the same as O(A log n).

3. (30 points) In a company, the supervisor-supervisee relation can be represented by a single tree T, with the president being the root of the tree. Given the tree T, you are asked to compute the maximal number of employees that can be invited to a party such that an employee and his/her immediate supervisor cannot be invited at the same time. Please design an efficient algorithm for this problem and analyze its time complexity.
This problem can be solved using either dynamic programming or greedy techniques, both using the depth-first search framework (or equivalently, the post-order traversal). For dynamic programing, let us define the following: For any set S of nodes in T, we say S is independent if a parent and its child cannot both in S. For any node n in T, let children(n) denote the children in T; \( a_n \) be the maximal size of an independent set of nodes in the subtree rooted by n without the node n; \( b_n \) be the maximal size of an independent set of nodes in the subtree rooted by n with the node n; and \( c_n = \max(a_n, b_n) \).

If \( r \) is the root of T, then \( c_r \) is the answer we are looking for.

If n is a leaf node, then \( a_n = 0 \), and \( b_n = 1 \).

If n has children, then \( a_n = \sum_{x \in \text{children}(n)} c_x \) and \( b_n = 1 + \sum_{x \in \text{children}(n)} a_x \).

The above relation allows us to design a linear time, bottom-up algorithm to compute \( c_r \).

Assuming the class node has values a, b, and c:

```
    treeIndSet(node x) {
        x.a = 0; x.b = 1;
        for (y in children(x)) {
            treeIndSet(y);
            x.a += y.c;
            x.b += y.a;
        }
        x.c = max(x.a, x.b);
    }
```

Let \( r \) be the root of T, then the first call is `treeIndSet(r)` and the final solution is \( c_r \).

The greedy technique always takes leaf nodes as invited and then removed from the tree along with their supervisors. In actual implementation, we don’t do removal. Just invite a node if none of its children is invited. We use a global variable num to record the number of invited people.

```
    int num = 0;
    invited(T);
    print “the max number of invited people is ” + num;

    boolean invited(node x) {
        boolean good = true;
        for (y in children(x)) if (invited(y)) good = false;
        if (good) num++; // x is invited
        return good;
    }
```

Note: Both algorithms visit each node once, so it’s a linear time algorithm. Using the breadth-first search’s level numbers cannot produce the right answer.