CS3330  Algorithms  
Midterm Exam (100 points)  
Closed books and notes (except two sheets of notes)

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.
2. (30 points) Given two strings $X$ and $Y$, the edit distance between $X$ and $Y$, $D(X, Y)$, is the minimal number of operations performed on $X$ so that $X$ becomes $Y$. The allowed operations are: delete a letter, add a letter, or change a letter. For example, $D(\text{“hurry”}, \text{“carry”}) = 2$, because “hu” can be changed to “ca” by two operations. Please design an efficient algorithm to compute $D(X, Y)$ and analyze its complexity.

**Sol.** This problem can be solved using dynamic programming. Let $X_i$, $Y_j$ be the prefix of $X$ and $Y$, respectively, where $X_i$ contains $i$ letters and $Y_j$ contains $j$ letters, and we write $d(i, j) = D(X_i,
Yj). Then by definition, \(d(0, j) = j, d(i, 0) = i\), because we need \(j\) deletions (or \(j\) additions) to make \(X_0 = Y_j\). For \(i, j > 0\), \(d(i, j) = d(i-1, j-1)\) if \(X[i] = Y[j]\), \(d(i, j) = \min(d(i-1, j-1), d(i, j-1), d(i-1, j)) + 1\) if \(X[i] \neq Y[j]\). Using this recursive definition, we may compute \(d(i, j)\) as follows:

\[
\text{editDistance}(X, Y) \{
    m = |X|; n = |Y|;
    for (i = 0, i <= m; i++) d[i, 0] = i;
    for (j = 0, j <= n; j++) d[0, j] = j;
    for (i = 1, i <= m; i++)
        for (j = 1, j <= n; j++)
            if (X[i] == Y[j]) d[i, j] = d[i-1, j-1];
            else d[i, j] = \min(d[i-1, j-1], d[i-1, j], d[i, j-1]) + 1;
    return d[m, n];
\}

The complexity of \(\text{editDistance}(X, Y)\) is \(O(|X||Y|)\).

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.

**Sol.** 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 programming, 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 \(\text{children}(n)\) 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 \(a_n = 0, b_n = 1\).
- 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:

\[
\text{treeIndSet}(\text{node } x) \{
    x.a = 0; x.b = 1;
    for (y in \text{children}(x)) {
        \text{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 r.c.

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.

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

```java
boolean invited(node x) {
    boolean good = true;
    for (y in children(x)) if (invited(y)) good = false;
    if (good) num++;
    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.

4. (10 points) A greedy algorithm for the Vertex Cover problem works as follows: Always move a vertex with the highest degree into the vertex cover, and then delete all edges incident to this vertex in the graph. Repeat the above operation until no more edges in the graph. Please provide a counter example to that this algorithm is not a 2-approximation algorithm for the Vertex Cover problem.

**Sol.** We can construct a bipartite graph G = (V, E) as a counter example. Let V = A ∪ B, where A = \{a_1, a_2, ..., a_{12}\}, B = X ∪ Y ∪ Z ∪ W.

- Let X = \{x_1, x_2, ..., x_{12}\}, and (a_i, x_i) ∈ E for 1 ≤ i ≤ 12. The degree of each x in X is 1.
- Let Y = \{y_1, y_2, ..., y_6\}, and (a_{2i-1}, y_i) ∈ E and (a_{2i}, y_i) ∈ E, for 1 ≤ i ≤ 6. The degree of each y in Y is 2.
- Let Z = \{z_1, z_2, z_3, z_4\}, and (a_{3i-j}, z_i) ∈ E, for 1 ≤ i ≤ 4 and 0 ≤ j ≤ 2. The degree of each z in Z is 3.
- Let W = \{w_1, w_2, w_3\}, and (a_{4i-j}, w_i) ∈ E, for 1 ≤ i ≤ 3 and 0 ≤ j ≤ 3. The degree of each w in W is 4.

The degree of each a in A is 4. The greedy algorithm may choose W first, followed by Z, Y and X, as the vertex cover. This greedy solution’s cost is \(C = |B| = |X|+|Y|+|Z|+|W| = 12+6+4+3 = 25\). The optimal solution is to choose A as the vertex cover and its cost is \(C^* = |A| = 12\). The ratio is \(C/C^* = 25/12 > 2\). Hence, this greedy algorithm is not a 2-approximation algorithm.