1. What is the running time of DFS if the graph is given as an adjacency list and adjacency matrix? Justify your running time.

Solution:

If the number of nodes in the graph is V and the number of edges in the graph is E, the running time are:

Adjacency list: $O(V+E)$

Adjacency matrix: $O(V^2)$

In DFS, we will visit every node once and visit those nodes are adjacent to them. In both case, we first need $O(V)$ time to visit all nodes.

If we use adjacency list, we only need to visit every edge in each list, which is $O(E)$. So, the total running time is $O(V)+O(E)=O(V+E)$.

If we use adjacency matrix, although there are only E edges, once we visit a node (i.e. the i^{th} node), we will visit the element in the i^{th} row of the adjacency matrix. In total, we will visit every element in the adjacency matrix (no matter it is 0 or 1), which is V^2. So the total running time will be $O(V^2)+O(V)=O(V^2)$.

2. Write a method that takes any two nodes u and v in a tree T, and quickly determines if the node u in the tree is a *descendant* or *ancestor* of node v

Solution:

We can simply run DFS on a tree T and set the start node as the root and record the discover time and finish time of every node. Because T is a tree, the number of edges is $n-1$. So, the total running time is $O(V+E)=O(n+n-1)=O(n)$. As we have proved in the textbook, given any 2 nodes u, v in the DFS tree, if $u.d<v.d<v.f<u.f$, u is the ancestor of v. Else if $v.d<u.d<u.f<v.f$, u is the descendant of v. Because we only need $O(n)$ time to complete DFS and store the discover time and finish time. Other operations are in constant time ($O(1)$), so the total running time is $O(n)$. More specifically, we need $O(n)$ time to initialize and $O(1)$ time to determine the relationship between 2 nodes.

3. (u (v (y (x x) y) v) u) (w (z z) w)

4.

Alg: find_subsets:

```
procedure subsets(nums: array):
    var combinations : array
    dfs(nums, 0, [], combinations)
    return combinations
```

```
precedure dfs(nums: array, index: integer, combination: array, combinations, array):
    combinations.append(combination.copy)// insert the copy of combination to combinations
    begin
        for i := index to nums.length:
            combination.append(nums[i])
            dfs(nums, i + 1, combination, combinations)
            combination.pop()
    end;
```