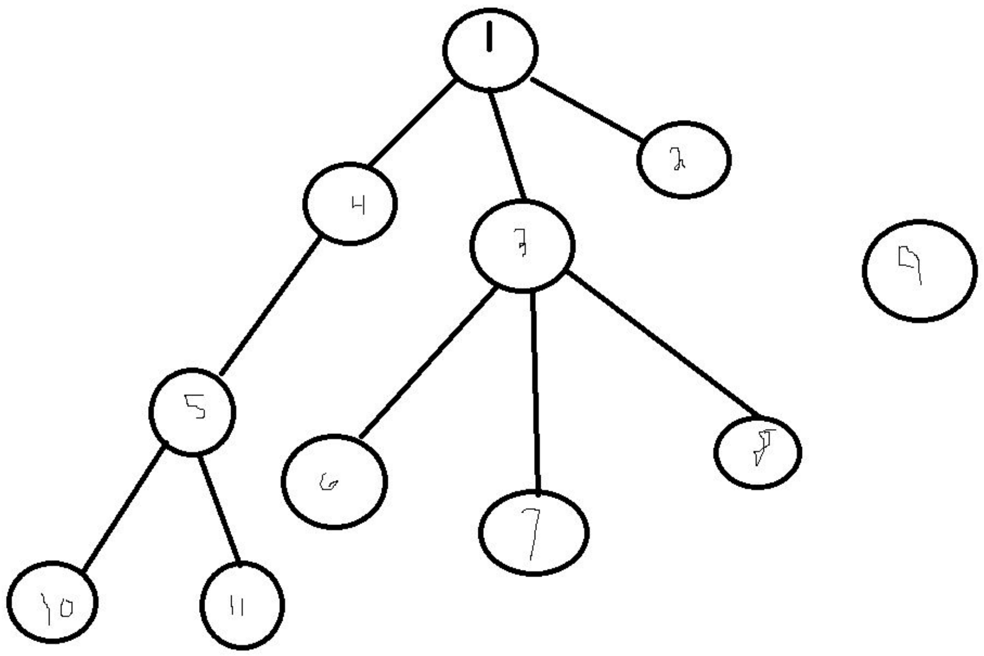
Breadth First Search/Depth First Search

Suppose we have a graph (this graph will be a tree because it's easier that way)



Suppose you want to search every element of the tree. There are two ways to do this: **B**readth **F**irst **S**earch, or **D**epth **F**irst **S**earch. (Do remember that you should be using BFS/DFS if you are not concerned about the edges being weighted, although this can change).

In both search methods, the algorithm starts at a certain vertex (today we will start at 1 in our example) and 'looks at' every single vertex that has a path to the said vertex (what 'looking at' is depends on what you want your algorithm to do).

In BFS, the siblings are processed first, with descendants done after.

The order that the BFS processes the above tree is:

1 - 4 - 7 - 2 - 5 - 6 - 7 - 8 - 10 - 11

An interesting - and important - facet of BFS is that the list of nodes it searches through is sorted by distance from the initial node.

BFS is typically done by using a queue. Here is an implementation of BFS:

```
void bfs(int s){
    queue<pair<int, int>> q;
    q.push({s,s});
    while(!q.empty()){
        int k = q.size();
        for(int i = 0; i < k; i++){
            pair<int, int> p = q.front();
            q.pop();
            for(int j : G[p.first]){
                if(j != p.second) q.push({j,p});
            }
        }
    }
}
```

On the other hand, DFS processes descendants first, with siblings done after.

The order that the DFS processes the nodes is:

1 - 4 - 5 - 10 - 11 - 3 - 6 - 7 - 8 - 2

DFS is typically done with recursion. Here is the **skeleton** of a DFS implementation:

```
vector<int> G[100];
void dfs(int v, int u){
    for(int i : G[v]){
        if(i != u) dfs(v,i);
    }
}
```

BFS processes vertices in order of ascending distance; this is great for shortest path problems executed on unweighted graphs (and variants). However, it does not keep track of the path from the initial vertex to whatever is being processed at the time - attempting to do so consumes a large (and usually unacceptable) amount of memory.

DFS, with some work, keeps track of the current path rather easily and has a very short implementation (which can change). The order in which the nodes are traversed, however, is a little less clean than BFS. **Most recursion can actually be modelled as DFS.**

The reasons for using either algorithm over the other are very diverse and vary from problem to problem; use your discretion (discretion, as always, is gained by practice).