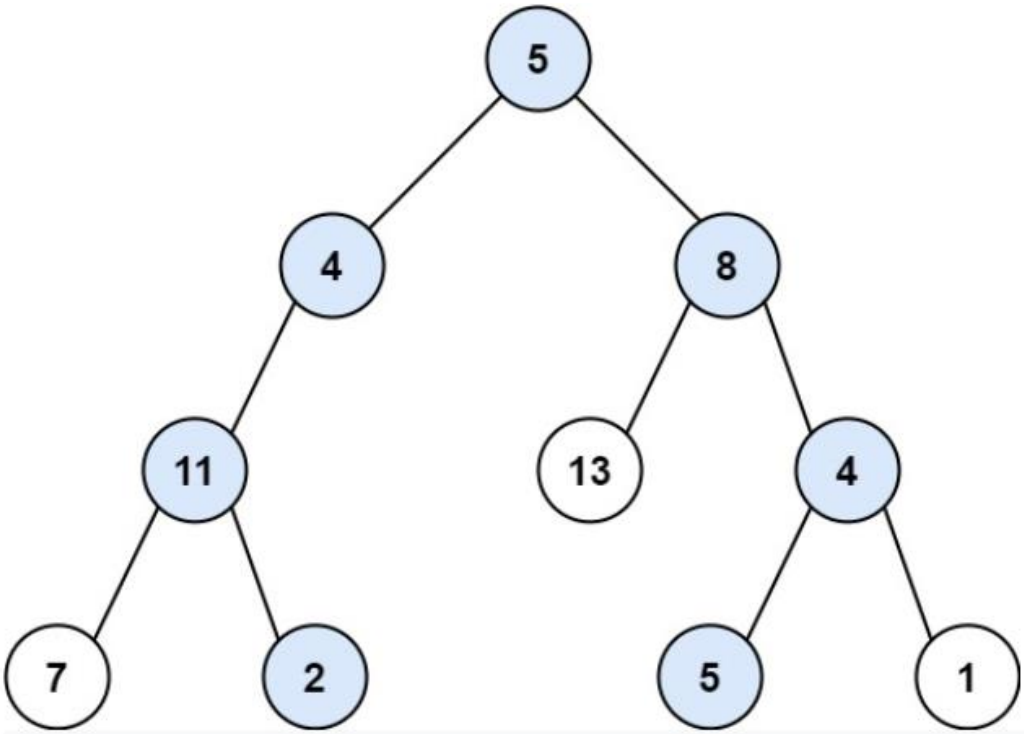
Bài 1: Path Sum II (medium)

https://leetcode.com/problems/path-sum-ii/

Cho cây nhị phân cố gốc là root, mỗi node của cây có 1 giá trị và 1 số nguyên target. Yêu cầu tìm các đường đi từ gốc đến lá (lá là node không có node con nào dưới nó) và có tổng các node trên đường đi đó bằng target




```
Input: root = [5,4,8,11,null,13,4,7,2,null,null,5,1], targetSum = 22
Output: [[5,4,11,2],[5,8,4,5]]
```

Số lượng node giới hạn trong khoảng 0 <= N <= 5000

Using DFS algorithm:

- Theoretical basis to apply DFS algorithm: Duyệt cây theo chiều sâu, lưu lại tổng hiện tại của tất cả các node đã đi qua. Lưu lại đường đi vào một mảng, khi gặp node là lá, so sánh tổng hiện tại với target, nếu bằng thì thêm vào kết quả trả về.
- Memory space, execution time and data structure.
    - Memory space: O(N)
    - Execution time: O(N)
    - Data structure: Binary Tree
- Code

```
class Solution {

public:

    vector<vector<int>> ans;
```

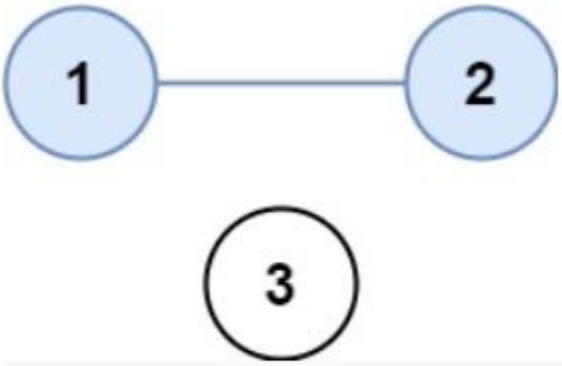
```
    vector<int> temp;
    void dfs(TreeNode* root, int curSum, int tar) {
        temp.push_back(root->val);
        curSum += root->val;
        if (!root->left && !root->right) {
            if (curSum == tar) ans.push_back(temp);
            temp.pop_back();
            return;
        }
        if (root->left) {
            dfs(root->left, curSum, tar);
        }
        if (root->right) {
            dfs(root->right, curSum, tar);
        }
        temp.pop_back();
        return;
    }

    vector<vector<int>> pathSum(TreeNode* root, int targetSum) {
        if (!root) return {};
        dfs(root, 0, targetSum);
        return ans;
    }
};
```

Bài 2: Number of Provinces (medium)

https://leetcode.com/problems/number-of-provinces/

Cho N thành phố, một thành phố có thể có kết nối với 1 hoặc nhiều thành phố khác. Yêu cầu trả về số lượng nhóm các thành phố sau khi được kết nối với nhau. Cho ma trận isConnected kích thước N*N, trong đó nếu A[i][j] = 1 nghĩa là thành phố i và j có kết nối với nhau, bằng 0 thì ngược lại.




```
Input: isConnected = [[1,1,0],[1,1,0],[0,0,1]]
Output: 2
```

Giới hạn: 0 <= N <= 200

Using DFS algorithm:

- Theoretical basis to apply DFS:

Duyệt từng phần tử từ 0 đến N. Sử dụng DFS để đánh dấu những vị trí có kết nối tới phần tử hiện tại. Tạo một mảng đánh dấu các phần tử đã được duyệt qua để tránh việc phải lại duyệt một phần tử nhiều lần làm chậm thời gian chương trình.

- Memory space, execution time and data structure.
  - Memory space: O(N^2)
  - Execution time: O(N^2)
  - Data structure: Array
- Code

```
class Solution {
public:
    void dfs(vector<vector<int>> &a, int s, vector<bool> &v){
        v[s] = true;
        for (int i=0; i<a[s].size(); i++){
            if (a[s][i] == 1 && v[i] == false){
                dfs(a, i, v);
            }
        }
```

```
        return;
    }
    int findCircleNum(vector<vector<int>>& isConnected) {
        int count = 0;
        int n = isConnected.size();
        vector<bool> v(n,false);
        for (int i=0; i<n; i++){
            if (v[i] == false) {
                count++;
                dfs(isConnected, i, v);
            }
        }
        return count;
    }
};
```

Bài 3: Maximum Depth of Binary Tree (easy)

https://leetcode.com/problems/maximum-depth-of-binary-tree/

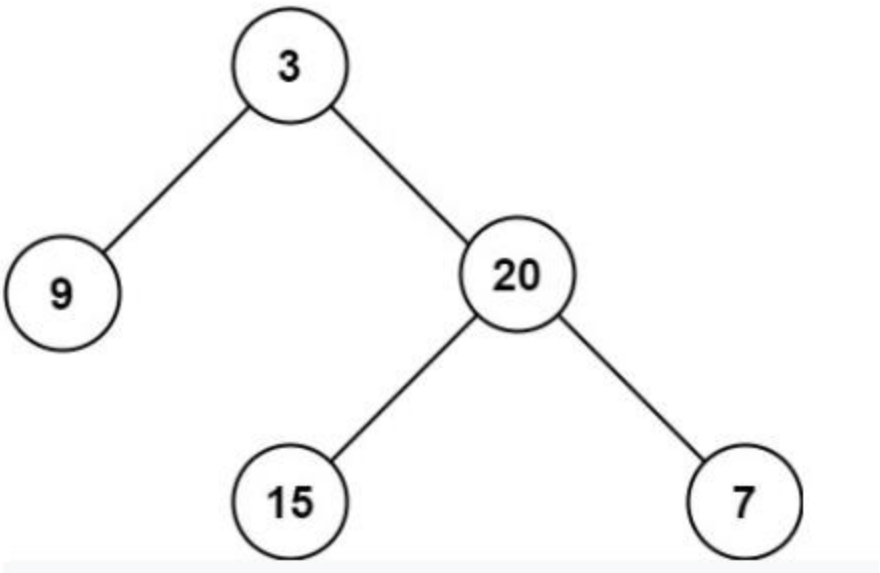
Cho 1 cây nhị phân, trả về độ sâu lớn nhất của cây.

Độ sâu lớn nhất của cây là tổng số nốt trên đường đi từ nốt gốc tới nốt lá xa nhất của cây.

Điều kiện:

Tổng số nốt của cây từ 0 đến 10^4.

Ví dụ:



root = [3,9,20,null,null,15,7]

Độ sâu lớn nhất của cây là 3.

Depth-first search:

Dùng depth-first search duyệt các đường đi từ gốc đến các nốt của cây.

Mỗi lần duyệt đến nốt con, tăng biến đếm độ sâu thêm 1. Trả về giá trị độ sâu lớn nhất tìm được.

Code:

```
class Solution {
public:
   int ans;
   void dfs(TreeNode *root, int height) {
      if (root == NULL)
         return;
      ans = max(ans, height);
      if (root->right != NULL)
         dfs(root->right, height + 1);
      if (root->left, height + 1)
         dfs(root->left, height + 1);
   }
   int maxDepth(TreeNode *root) {
```

```
        ans = 0;
        dfs(root, 1);
        return ans;
    }
};
```

Bài 4: Count Good Nodes in Binary Tree (easy)

https://leetcode.com/problems/count-good-nodes-in-binary-tree/

Cho 1 cây nhị phân có gốc là nốt *root*, 1 nốt X trên cây được gọi là tốt nếu 1 đường đi từ gốc *root* tới nốt X sẽ không có 1 nốt nào có giá trị trên nốt đó lớn hơn giá trị trên nốt X. Xác định tổng số nốt như trên.

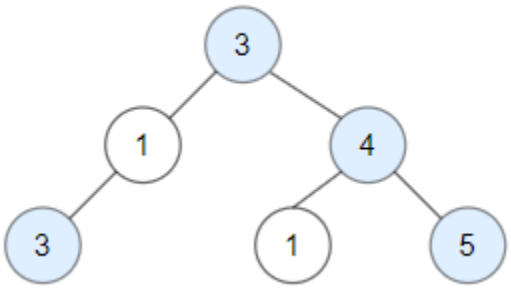
Điều kiện:

Tổng số nốt của cây từ 1 đến 10^5

Giá trị mỗi nốt nằm trong khoảng từ -10^4 đến 10^4.

Ví dụ:

root = [3,1,4,3,null,1,5]



Những nốt màu xanh là những nốt tốt, tổng số nốt tốt là 4.

Depth-first search:

Dùng depth-first search để xác định đường đi từ gốc đến từ nốt trên cây.

Trong lúc duyệt, lưu lại giá trị lớn nhất của từng nốt đã đi qua, so sánh giá trị này với giá trị của nốt hiện tại. Nếu thỏa mãn đề bài, tăng biến đếm thêm 1.

Code:

```
class Solution {
public:
    int ans;
    void dfs(TreeNode *root, int maxVal) {
        if (root->val >= maxVal)
            ++ans;
        if (root->left != NULL)
            dfs(root->left, max(root->left->val, maxVal));
        if (root->right != NULL)
            dfs(root->right, max(root->right->val, maxVal));
    }
    int goodNodes(TreeNode *root) {
        ans = 0;
        if (root == NULL)
            return 0;
        dfs(root, root->val);
        return ans;
    }
};
```

Bài 5: Course Schedule (medium)

https://leetcode.com/problems/course-schedule/

mAn đang đi học đại học. Có tổng số N khóa học An có thể chọn để học, đánh số từ 0 đến N-1. Mỗi khóa học a có thể có điều kiện phải hoàn thành một khóa học b khác trước khi có thể chọn a để học. Điều kiện cho các khóa học được lưu trong 1 mảng 2 chiều prerequisites với cấu trúc prerequisites[i] = [ai, bi]
Viết 1 thuật toán kiểm tra xem liệu An có thể chọn học tất cả các khóa học hay không. Trả về true nếu có hoặc false nếu không.

mGiới hạn:
1 <= numCourses <= 100000
0 <= prerequisites.length <= 5000

prerequisites[i].length == 2
0 <= ai, bi < numCourses
All the pairs prerequisites[i] are unique.

Approach:

Coi mỗi khóa học là một đỉnh, tìm xem có chu trình trong đồ thị không.

Nếu không có thì in ra YES, còn ngược lại in NO.

Hướng dẫn tìm chu trình:

https://www.geeksforgeeks.org/detect-cycle-in-a-graph/