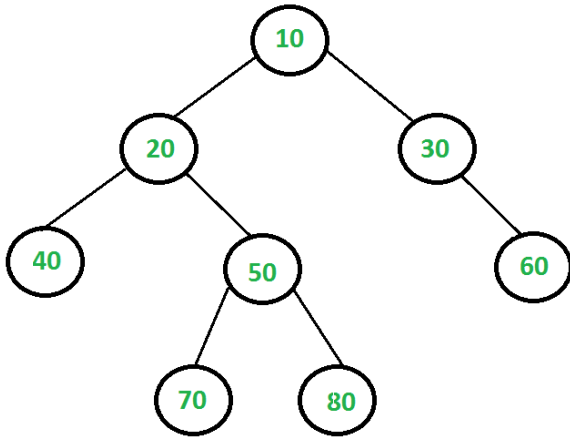


Given a Binary Tree, find size of the **Largest Independent Set(LIS)** in it. A subset of all tree nodes is an independent set if there is no edge between any two nodes of the subset.

For example, consider the following binary tree. The largest independent set(LIS) is {10, 40, 60, 70, 80} and size of the LIS is 5.



A Dynamic Programming solution solves a given problem using solutions of subproblems in bottom up manner. Can the given problem be solved using solutions to subproblems? If yes, then what are the subproblems? Can we find largest independent set size (LISS) for a node X if we know LISS for all descendants of X? If a node is considered as part of LIS, then its children cannot be part of LIS, but its grandchildren can be. Following is optimal substructure property.

1) Optimal Substructure:

Let LISS(X) indicates size of largest independent set of a tree with root X

$$\text{LISS}(X) = \text{MAX} \{ (1 + \text{sum of LISS for all grandchildren of } X), (\text{sum of LISS for all children of } X) \}$$

The idea is simple, there are two possibilities for every node X, either X is a member of the set or not a member. If X is a member, then the value of LISS(X) is 1 plus LISS of all grandchildren. If X is not a member, then the value is sum of LISS of all children.

2) Overlapping Subproblems

Following is recursive implementation that simply follows the recursive structure mentioned above.

```
// A naive recursive implementation of Largest Independent Set problem
#include <stdio.h>
#include <stdlib.h>

// A utility function to find max of two integers
int max(int x, int y) { return (x > y)? x: y; }

/* A binary tree node has data, pointer to left child and a pointer to
   right child */
struct node
{
    int data;
    struct node *left, *right;
};

// The function returns size of the largest independent set in a given
// binary tree
int LISS(struct node *root)
{
    if (root == NULL)
        return 0;

```

```

// Caculate size excluding the current node
int size_excl = LISS(root->left) + LISS(root->right);

// Calculate size including the current node
int size_incl = 1;
if (root->left)
    size_incl += LISS(root->left->left) + LISS(root->left->right);
if (root->right)
    size_incl += LISS(root->right->left) + LISS(root->right->right);

// Return the maximum of two sizes
return max(size_incl, size_excl);
}

// A utility function to create a node
struct node* newNode( int data )
{
    struct node* temp = (struct node *) malloc( sizeof(struct node) );
    temp->data = data;
    temp->left = temp->right = NULL;
    return temp;
}

// Driver program to test above functions
int main()
{
    // Let us construct the tree given in the above diagram
    struct node *root = newNode(20);
    root->left = newNode(8);
    root->left->left = newNode(4);
    root->left->right = newNode(12);
    root->left->right->left = newNode(10);
    root->left->right->right = newNode(14);
    root->right = newNode(22);
    root->right->right = newNode(25);

    printf ("Size of the Largest Independent Set is %d ", LISS(root));

    return 0;
}

```

Output:

Size of the Largest Independent Set is 5

Time complexity of the above naive recursive approach is exponential. It should be noted that the above function computes the same subproblems again and again. For example, LISS of node with value 50 is evaluated for node with values 10 and 20 as 50 is grandchild of 10 and child of 20.

Since same subproblems are called again, this problem has Overlapping Subproblems property. So LISS problem has both properties (see [this](#) and [this](#)) of a dynamic programming problem. Like other typical **Dynamic Programming(DP) problems**, recomputations of same subproblems can be avoided by storing the solutions to subproblems and solving problems in bottom up manner.

Following is C implementation of Dynamic Programming based solution. In the following solution, an additional field 'liss' is added to tree nodes. The initial value of 'liss' is set as 0 for all nodes. The recursive function LISS() calculates 'liss' for a node only if it is not already set.

```

/* Dynamic programming based program for Largest Independent Set problem */
#include <stdio.h>
#include <stdlib.h>

// A utility function to find max of two integers
int max(int x, int y) { return (x > y)? x: y; }

```

```

/* A binary tree node has data, pointer to left child and a pointer to
   right child */
struct node
{
    int data;
    int liss;
    struct node *left, *right;
};

// A memoization function returns size of the largest independent set in
// a given binary tree
int LISS(struct node *root)
{
    if (root == NULL)
        return 0;

    if (root->liss)
        return root->liss;

    if (root->left == NULL && root->right == NULL)
        return (root->liss = 1);

    // Calculate size excluding the current node
    int liss_excl = LISS(root->left) + LISS(root->right);

    // Calculate size including the current node
    int liss_incl = 1;
    if (root->left)
        liss_incl += LISS(root->left->left) + LISS(root->left->right);
    if (root->right)
        liss_incl += LISS(root->right->left) + LISS(root->right->right);

    // Maximum of two sizes is LISS, store it for future uses.
    root->liss = max(liss_incl, liss_excl);

    return root->liss;
}

```

```

// A utility function to create a node
struct node* newNode(int data)
{
    struct node* temp = (struct node *) malloc( sizeof(struct node) );
    temp->data = data;
    temp->left = temp->right = NULL;
    temp->liss = 0;
    return temp;
}

// Driver program to test above functions
int main()
{
    // Let us construct the tree given in the above diagram
    struct node *root = newNode(20);
    root->left = newNode(8);
    root->left->left = newNode(4);
    root->left->right = newNode(12);
    root->left->right->left = newNode(10);
    root->left->right->right = newNode(14);
    root->right = newNode(22);
    root->right->right = newNode(25);

    printf ("Size of the Largest Independent Set is %d ", LISS(root));

    return 0;
}

```

Output

Size of the Largest Independent Set is 5

Time Complexity: $O(n)$ where n is the number of nodes in given Binary tree.

Following extensions to above solution can be tried as an exercise.

- 1)** Extend the above solution for n -ary tree.
- 2)** The above solution modifies the given tree structure by adding an additional field 'liss' to tree nodes. Extend the solution so that it doesn't modify the tree structure.
- 3)** The above solution only returns size of LIS, it doesn't print elements of LIS. Extend the solution to print all nodes that are part of LIS.