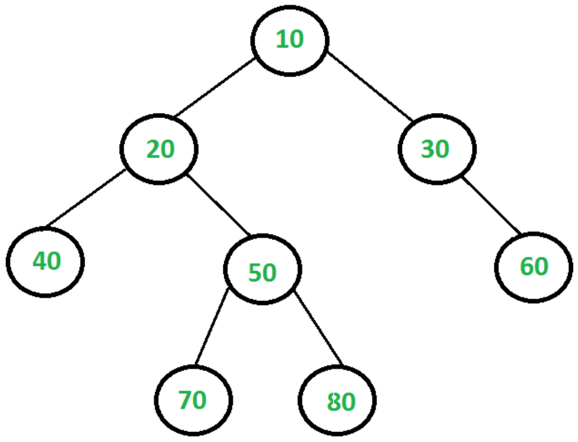
Dynamic Programming | Set 26 (Largest Independent Set Problem)

Given a Binary Tree, find size of the **L**argest **I**ndependent **S**et(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.

```
     LISS(X) = MAX { (1 + sum of LISS for all grandchildren of X),
                     (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 suproblems are called again, this problem has Overlapping Subprolems 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.

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.

Related Topics:

- Handshaking Lemma and Interesting Tree Properties
- Advantages of BST over Hash Table
- Given a binary tree, how do you remove all the half nodes?
- K'th Largest Element in BST when modification to BST is not allowed

- Vertex Cover Problem | Set 2 (Dynamic Programming Solution for Tree)
- Check whether a binary tree is a complete tree or not | Set 2 (Recursive Solution)
- Check whether a binary tree is a full binary tree or not
- Find sum of all left leaves in a given Binary Tree

Tags: Dynamic Programming

Tweet 3 g+1 2

Writing code in comment? Please use **ideone.com** and share the link here.

48 Comments GeeksforGeeks 1 Login

Recommend Share Sort by Newest

Join the discussion…

Neha • 6 days ago

Please tell me the importance of these lines
if (root->left == NULL && root->right == NULL)
return (root->liss = 1);

Are they actually needed? if Yes then on which case we need this check?

And -> if (root->liss)
return root->liss;
it means if root's liss is already set than return the same .

Is my understanding is correct . Please Explain?

• Reply • Share ›

sayyid tabish • 3 months ago

Implementation of LISS without modifying the tree structure.

Used hashing.

T.C. : O(n)

S.C. : O(n)

http://ideone.com/8Ukpv6

• Reply • Share ›

oO • 4 months ago

My Solution O(n) using recursion:

int LIS(struct node *root, int &I, int &N)

```
{
------ if ( root==NULL )
------ {
----------- I = N = 0;
----------- return 0;
------ }
-
------ int il=0, nl=0;
------ LIS(root->left, il, nl); //Get for the left
------ int ir=0, nr=0;
------ LIS(root->right, ir, nr); //Get for the right
------ I = 1 + nl + nr;
------ N = max(max(il+ir, il+nr), max(nl+ir, nl+nr));
------ return max(I,N);
}
```

1 • Reply • Share ›

oO oO • 4 months ago

The idea is the following:
1) For every node, we calculate the LIS if the node is included (I) and the LIS if the node is not included (N) - we store these values in I and L respectively.
2) After we have the I and L values for the left and right subtrees,
a) if we are including the node, we cannot use I (included) values for the left and right subtrees (IR, IL).
b) If we are not including the root, we may or may not choose to include one of its children.

• Reply • Share ›

disqus_5wZW2V8OEI • 5 months ago

Hey,

I am using this approach in which we don't need to store values for liss in every node, rather we just return values of liss_included and liss_excluded
for every child to it's parent and then the parent would do the same by computing bottom-up:
liss_in=le+re+1; liss_ex=li+ri;
and then for the root of the tree, just return the max of two values.
It works for the given case, can someone please provide any test case where it'll not work.

http://ideone.com/UhTc2V

• Reply • Share ›

StrictMath disqus_5wZW2V8OEI • 4 months ago

Too bad that you couldn't figure that out:
a) First your code has the following in liss_util:

--------liss_util(root->left, li, le);
--------liss_util(root->left, ri, re);

Both as LEFT!!

b) Second if you change the bottom one to root->right, it fails for the test case given in the problem above and prints a '4' instead of '5'.

• Reply • Share ›

disqus_5wZW2V8OEI StrictMath • 4 months ago
Hey, thanks for pointing that out. I corrected the code and the logic.

• Reply • Share ›

arenped • 5 months ago
public void largestIndependentSet(Node root, ref int size, ref Dictionary<node, node=""> ds)

{

if(root==null) return;

largestIndependentSet(root.left,ref size,ref ds);

largestIndependentSet(root.right,ref size,ref ds);

bool l=root.left==null?true:!ds.ContainsKey(root.left);

bool r=root.right==null?true:!ds.ContainsKey(root.right);

if (l && r)

{

ds.Add(root, root);

size++;

see more

• Reply • Share ›

code_craze • 5 months ago
http://ideone.com/eZvOdK#stdin

For printing the list :)

1 • Reply • Share ›

Anil Kumar K K • 6 months ago
This problem can be solved without DP with O(n). Please suggest any bug/flaw in my idea.

int first = 1;

int first = 1;

int

largest_ind_set(node_t *root)

{

int count = 0;

if(first)

{

count++;

first = 0;

}

see more

• Reply • Share ›

Sivakumar K R • 6 months ago

Here's my O(n) solution. i have made use of variable level to store the level info. And used the map to store the result of the current level and return the result to parent calls. Once you have consumed the LIS value of grandchildren(level+2), we should clear it.
It takes O(n) space. Please test it. It will not alter the tree structure.

http://ideone.com/eyK4Rm

• Reply • Share ›

gopayya • 6 months ago

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#include<malloc.h>
#include<math.h>
#define Buffersize 100

#define max 50

struct node
{
int vid;
int wt;
struct node *lchild;
struct node *rchild;
```

```
int maxwt;

struct node *next;
char string[max];
```

see more

^ | ⌄ • Reply • Share ›

vipinkaushal • 10 months ago

a O(n^2) solution to find the independent set (not only cardinality) using post order traversal

```
void findIndependentSet(node *root,node *set[],int *i)
{
if(root==NULL)return;
if(root->lchild==NULL&&root->rchild==NULL){ set[(*i)++]=root; return;}
findIndependentSet(root->rchild,set,i);
findIndependentSet(root->lchild,set,i);
if(isIndependent(root,set,i))
set[(*i)++]=root;

return;
}
bool isIndependent(node* root,node *set[],int *n)
{
for(int i=0;i<*n;i++)
if(root->lchild==set[i]||root->rchild==set[i])return false;
return true;
}
```

^ | ⌄ • Reply • Share ›

AlienOnEarth • a year ago

GeeksforGeeks:

The below 2 lines are redundant. The DP algorithm will work without these 2 lines.

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

6 ^ | ⌄ • Reply • Share ›

Monkey D. Luffy ➦ AlienOnEarth • 4 months ago

right you are AlienOnEarth

^ | ⌄ • Reply • Share ›

avik_d ➦ AlienOnEarth • 9 months ago

how to print the LIS ??

^ | ⌄ • Reply • Share ›

Reply • Share ›

hj • a year ago

Can this problem be seen as a variant of Graph Coloring Problem? The largest number of nodes having the same color will form largest independent set.
Of course the graph coloring depends on the order in which vertices are seen.. On the other hand, we can find independent subsets and use for graph coloring?

2 • Reply • Share ›

aman hj • a year ago

Yes it can be done taking in consideration the notion of bipartite graph.

• Reply • Share ›

Cristian Florica • a year ago

What is the approach for the "Maximum Weighted Independent Set (MWIS) problem"?

• Reply • Share ›

Anonymous Cristian Florica • 3 months ago

MWIS[X]=max(X->data+sum of MWIS[grandchildren], sum of MWIS[children])

• Reply • Share ›

prashant jha • a year ago

```
#include<iostream>
using namespace std;
struct node
{
node* lchild;
int data;
node* rchild;
node(int d)
{
data=d;
lchild=NULL;
rchild=NULL;
}
}*root=NULL;
void create(node* &root,int d)
{
int n;
if(d==-1)
```

see more

• Reply • Share ›

Nitesh • a year ago

Nitesh • a year ago

LIS will always contain leaves? is there any case in which it will not??

# if it always contains leafs then a better solution is

```
# include<stdio.h>
# include<iostream>
# include<stdlib.h>
int arr[100],t;
struct node
{ int data,flag;
struct node *left, *right;
};
int lic(node *root)
{
if(root==NULL)
return 0;
else
{
```

see more

1 • Reply • Share ›

nitesh78 → Nitesh • 10 months ago

no consider a tree with a root and 2 nodes at level 1 and 2 nodes at level 2 and 2 nodes at level 3 and 1 node at level 4 , if leaves are include we will have we can have only 4 nodes in lis but if leaves are not include optinally we can have 5 nodes.

• Reply • Share ›

Guru Gorantla → Nitesh • a year ago

It need not contain leaves all the time. For example if a tree has a depth say "d" and dth row represent only leaves. Let this dth row contain only one leaf then your LISS need not contain Leafed row. why ? In general when we consider an example we tend to take a almost balanced binary tree, as we increase depth d, the number of nodes in that depth increases with d(if almost balanced) . So instead consider a tree with only one leaf.

• Reply • Share ›

nitesh78 → Guru Gorantla • 10 months ago

sorry i was wrong, in unweighted tree their exists a optimal solution which includes leaves.

• Reply • Share ›

Behind D Walls • a year ago

recursion code goes into infinite loop for keys 1-50 or more elements why?

^ | ⌄ • Reply • Share ›

Guest • 2 years ago

is the solution nothing but the root and all the leaves possible for a tree...i know there would be an exception when there r two levels but else for other cases ,my conclusion is right,isnt it?? pls help

1 ^ | ⌄ • Reply • Share ›

hj ➦ Guest • a year ago

I was thinking the same too.. but on second thought, this in fact forms the minimum independent set, apart from the case where there are two levels.
Consider the case where there are 5 levels.. (assume complete tree) root + leaves + some elements from 3rd level can also be included in the largest independent set.

^ | ⌄ • Reply • Share ›

Sreenivas Doosa • 2 years ago

Awesome logic duuude :) Appreciate it..!

^ | ⌄ • Reply • Share ›

Gaurav Gulzar • 2 years ago

/*Try this and please reply if u find anything wrong*/.

```
//Utility Function
int _LISS(Node *root, int *count) {.
if(! root)
return 0;

if(!(_LISS(root->left, count) + _LISS(root->right, count))) {.
(*count)++;
return 1;
}

return 0;
}

int LISS(Node *root) {.
int count = 0;.
_LISS(root,&count);
return count;
}
```

^ | ⌄ • Reply • Share ›

gaurav • 2 years ago

[sourcecode language="C++"]

```
/* Paste your code here (You may delete these lines if not writing code) */

//
// LongestIndSet.cpp
//
//
// Created by Gaurav Gulzar on 08/08/13.
//
//

#include
using namespace std;

typedef struct tNode
{
int data;
struct tNode* left;
```

see more

• Reply • Share ›

RAHUL23 • 2 years ago

CAN ANYONE TELL HOW TO PRINT MAXIMUM LIST SET??
IT WOULD BE OF GR8 HELP

THANKS IN ADVANCE

1 • Reply • Share ›

sudhanshu • 2 years ago

Another method O(n) :-

1) Perform level order traversal and in an array(declared globally), increment the count corresponding to that level.

2) So you get a count of the number of nodes corresponding to each level in an array in O(n).

3) Now, get the max subset sum ensuring adjacent levels(cells) aren't picked up. O(n) [simple dp].

This way we can also print them easily

Takes O(n) time overall.

• Reply • Share ›

saurabh sudhanshu • a year ago

And by your algorithm you will end up with answer 4 for the same case.

• Reply • Share ›

Born Actor • 2 years ago

```
 #include <iostream>
#include<string>
#include<sstream>
#include<iomanip>
#include <stdio.h>
#include <math.h>
#include <vector>
#include <stdlib.h>
using namespace std;
int count=0; //count variable counts the number of times the recursive call was made.
class node
{
        public:
        node*l;
        node *r;
        node *p;
        int value;
        int lss;
```

see more

• Reply • Share ›

illuminati • 2 years ago

one of the few problems solved with memoisation on GFG.

1 • Reply • Share ›

Karan Verma • 2 years ago

If the root&#039s children are the leaves themselves,then it&#039ll be wrong.

• Reply • Share ›

abhishek08aug • 2 years ago

Intelligent :D

• Reply • Share ›

Abhijeet • 2 years ago

Isn't the size of largest independent set equal MAX(#of nodes at even depths, #of nodes at odd depths)? If yes, BFS based solution will do with the O(n) complexity.

1 • Reply • Share ›

Niks • 2 years ago

```
 // 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;
};

static int index = 0;
```

see more

Reply • Share ›

apsc • 2 years ago

Can someone please tell how to extend it to print nodes of LIS, also without changing the structure of node how can we use memoization. Thanks.

Reply • Share ›

Santhosh • 2 years ago

Here's my take on the problem. Simple traversal based solution. Let me know if there's any logical error or cases I missed.

This works from bottom up. I assume that all leaf nodes are part of the solution and work from there.

Algo: all nodes contain a bool print.

1) Postorder traversal. If the node is root, set print=true
2) If node->left is null check if right node print value.
3) set current->print = !node->right->print.
4) If node->right is null set current->print=!node->left->print
5) If there are both left and right nodes, check if they are printed.
6) If any one of them is printed then skip current node, else print it.

```
 void postorder(struct node* node)
{
```

```
{
    if (node == NULL)
        return;
```

see more

• Reply • Share ›

Srinath Santhosh • 2 years ago

I am sure pretty sure this approach is correct,a lot more intuitive and easier to implement.

• Reply • Share ›

Santhosh Santhosh • 2 years ago

If this is found to be logically correct, can be easily extended to get count. If we are not allowed to change structure of node then a simple solution would be to push the bool values on to a stack and pop for the number of children of the node and compare them.

• Reply • Share ›

sreeram • 2 years ago

Without modifying original tree

[#include
#include

int max(int x, int y) { return (x > y)? x: y; }

struct node
{
int data;
struct node *left, *right;
};
int LISSUtil(struct node *root,int *lcref,int *rcref)
{
if (root == NULL){
*lcref=*rcref=0;
return 0;}

see more

1 • Reply • Share ›

op • 2 years ago

What if the max arguments happens to be same. Then in that case we will have two sets of same size. Like if we add 90 to inorder last node i.e. 60. then [40, 70, 80, 10, 60] and [40, 70, 80, 30, 90].

Your programs works fine.
But at the time of printing we need to take care of this case, if it is required to display all the maximum size sets.

• Reply • Share ›

viki • 2 years ago
There is no need to have these lines in second solution. I mean, removing these lines will suffice the solution.
Cheers

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

• Reply • Share ›

bond • 2 years ago
its a recent tc div2 hard problem :D

• Reply • Share ›

Subscribe Add Disqus to your site Privacy

-
-
-
-
    - Interview Experiences
    - Advanced Data Structures
    - Dynamic Programming
    - Greedy Algorithms
    - Backtracking
    - Pattern Searching
    - Divide & Conquer
    - Mathematical Algorithms
    - Recursion
    - Geometric Algorithms
-

- ## Popular Posts

    - All permutations of a given string
    - Memory Layout of C Programs
    - Understanding “extern” keyword in C
    - Median of two sorted arrays
    - Tree traversal without recursion and without stack!
    - Structure Member Alignment, Padding and Data Packing
    - Intersection point of two Linked Lists
    - Lowest Common Ancestor in a BST.
    - Check if a binary tree is BST or not
    - Sorted Linked List to Balanced BST
- Follow @GeeksforGeeks

- ## Recent Comments

    - lt_k

      i need help for coding this function in java...

      Java Programming Language · 1 hour ago

    - Piyush

      What is the purpose of else if (recStack[*i])...

      Detect Cycle in a Directed Graph · 1 hour ago

    - Andy Toh

      My compile-time solution, which agrees with the...

      Dynamic Programming | Set 16 (Floyd Warshall Algorithm) · 1 hour ago

- lucy

  because we first fill zero in first col and...

  Dynamic Programming | Set 29 (Longest Common Substring) · 2 hours ago

- lucy

  @GeeksforGeeks i don't n know what is this long...

  Dynamic Programming | Set 28 (Minimum insertions to form a palindrome) · 3 hours ago

- manish

  Because TAN is not a subsequence of RANT. ANT...

  Given two strings, find if first string is a subsequence of second · 3 hours ago