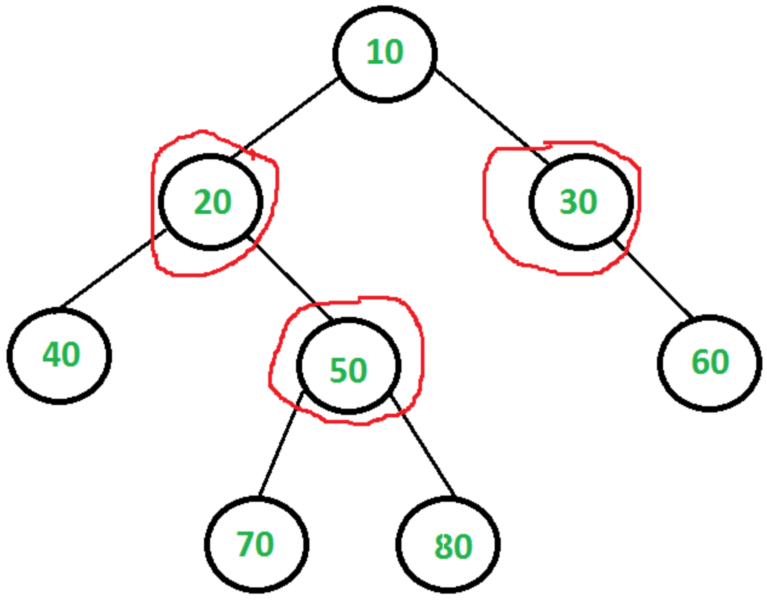
Vertex Cover Problem | Set 2 (Dynamic Programming Solution for Tree)

A vertex cover of an undirected graph is a subset of its vertices such that for every edge (u, v) of the graph, either 'u' or 'v' is in vertex cover. Although the name is Vertex Cover, the set covers all edges of the given graph.
The problem to find minimum size vertex cover of a graph is NP complete. But it can be solved in polynomial time for trees. In this post a solution for Binary Tree is discussed. The same solution can be extended for n-ary trees.

For example, consider the following binary tree. The smallest vertex cover is {20, 50, 30} and size of the vertex cover is 3.



The idea is to consider following two possibilities for root and recursively for all nodes down the root.
1) Root is part of vertex cover: In this case root covers all children edges. We recursively calculate size of vertex covers for left and right subtrees and add 1 to the result (for root).

2) Root is not part of vertex cover: In this case, both children of root must be included in vertex cover to cover all root to children edges. We recursively calculate size of vertex covers of all grandchildren and number of children to the result (for two children of root).

Below is C implementation of above idea.

```
// A naive recursive C implementation for vertex cover problem for a tree
#include <stdio.h>
#include <stdlib.h>

// A utility function to find min of two integers
int min(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 minimum vertex cover
int vCover(struct node *root)
{
    // The size of minimum vertex cover is zero if tree is empty or there
    // is only one node
    if (root == NULL)
        return 0;
    if (root->left == NULL && root->right == NULL)
        return 0;
    // Calculate size of vertex cover when root is part of it
    int size_incl = 1 + vCover(root->left) + vCover(root->right);

    // Calculate size of vertex cover when root is not part of it
    int size_excl = 0;
    if (root->left)
      size_excl += 1 + vCover(root->left->left) + vCover(root->left->right);
    if (root->right)
      size_excl += 1 + vCover(root->right->left) + vCover(root->right->right)

    // Return the minimum of two sizes
    return min(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 smallest vertex cover is %d ", vCover(root));

    return 0;
}
```

Output:

```
Size of the smallest vertex cover is 3
```

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, vCover of node with value 50 is evaluated twice as 50 is grandchild of 10 and child of 20.
Since same suproblems are called again, this problem has Overlapping Subprolems property. So Vertex Cover problem has both properties (see this and this) of a dynamic programming problem. Like other typical Dynamic Programming(DP) problems, re-computations 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 'vc' is added to tree nodes. The initial value of 'vc' is set as 0 for all nodes. The recursive function vCover() calculates 'vc' for a node only if it is not already set.

```
/* Dynamic programming based program for Vertex Cover problem for
   a Binary Tree */
#include <stdio.h>
#include <stdlib.h>

// A utility function to find min of two integers
int min(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 vc;
    struct node *left, *right;
};

// A memoization based function that returns size of the minimum vertex cover
int vCover(struct node *root)
{
    // The size of minimum vertex cover is zero if tree is empty or there
    // is only one node
    if (root == NULL)
        return 0;
    if (root->left == NULL && root->right == NULL)
        return 0;

    // If vertex cover for this node is already evaluated, then return it
    // to save recomputation of same subproblem again.
    if (root->vc != 0)
        return root->vc;

    // Calculate size of vertex cover when root is part of it
    int size_incl = 1 + vCover(root->left) + vCover(root->right);

    // Calculate size of vertex cover when root is not part of it
    int size_excl = 0;
    if (root->left)
      size_excl += 1 + vCover(root->left->left) + vCover(root->left->right);
    if (root->right)
      size_excl += 1 + vCover(root->right->left) + vCover(root->right->right)

    // Minimum of two values is vertex cover, store it before returning
    root->vc =  min(size_incl, size_excl);

    return root->vc;
}
```

```
// 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->vc = 0; // Set the vertex cover as 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 smallest vertex cover is %d ", vCover(root));

    return 0;
}
```

Output:

```
Size of the smallest vertex cover is 3
```

References:
http://courses.csail.mit.edu/6.006/spring11/lectures/lec21.pdf

Exercise:
Extend the above solution for n-ary trees.

This article is contributed by **Udit Gupta**. 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
- 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
- Remove nodes on root to leaf paths of length < K

Tags: Dynamic Programming

Tweet 2 g+1 1

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

8 Comments GeeksforGeeks 1 Login

Recommend Share Sort by Newest

Join the discussion…

satya • a month ago

if a node has leaf as any of its children , it should be in the vertex cover . from then alternate nodes should go into vertex cover

Here is the code:

```
public class VertexCover {

public static class Node{
int data;
Node left;
Node right;
int vcover;
Node(int d){
this.data = d;
this.left=null;
this.right=null;
vcover = -1;
}
```

see more

Reply • Share ›

guest • 2 months ago

how is this problem different from LISS question

Reply • Share ›

Aditya Joshi • 2 months ago

I don't think this will work. You should instead have a dynamic programming state (current_node, should_take) where should_take is a boolean. If should_take is 1, it means that in the current state, we MUST take the current node. If it is 0, we COULD or COULD NOT take it. We should try both possibilities.

So,
dp(curr_state, 1) = 1 + sum of dp(children, 0)

dp(curr_state, 0) = min(1 + dp(children, 0), dp(children, 1))

• Reply • Share ›

jackqiu Aditya Joshi • 2 months ago
your thinking is almost right,but in dp(curr_state,1) , it does not mean that all the children is 0 , to you need to check you thinking again

• Reply • Share ›

Jan • 2 months ago
For the case "2) Root is not part of vertex cover", if root only has one child, shouldn't we just add 1 instead of 2 (for size_excl)?

• Reply • Share ›

GeeksforGeeks Mod Jan • 2 months ago
Thanks for pointing this out. We have updated the article.

• Reply • Share ›

Jogi • 2 months ago
Thanks Udit

In case of n-ary tee, size_excl would be initialized as n instead of 2, everything else would be similar. Please confirm if this is fine.

• Reply • Share ›

jackqiu Jogi • 2 months ago
I think your thinking is right! you can have a try

• 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 · 2 hours ago

    - Piyush

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

      Detect Cycle in a Directed Graph · 2 hours ago

    - Andy Toh

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

      Dynamic Programming | Set 16 (Floyd Warshall Algorithm) · 2 hours 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

•