Vertex Cover

A dynamic programming approach

Franco, Ledgard, Reátegui, Roizman & Wurttele

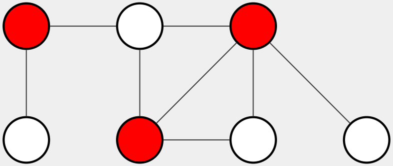
November 25, 2020

Introduction

Definition

Formal Definition

A ***vertex cover*** of an undirected graph $G = (V, E)$ is a subset $V' \subseteq V$ *s. t.* if $(u, v) \in E$, then $u \in V'$ or $v \in V'$ (or both) [CLRS09].



A more intuitive definition

A subset of vertices that includes at least one endpoint of every edge.

Vertex Cover Problem

Which is the problem?

Find a ***vertex cover*** of minimum size in a given undirected graph. Also known as ***minimum vertex cover***.

What kind of problem is this?

This problem is the ***optimization version*** of the NP-complete decision problem of determining whether a graph has a vertex cover of a given size k.

Approaches

Approaches to solve the problem

Is it solvable in polynomial time for any undirected graph?

Extremely unlikely to have to have a polynomial time algorithm for any graph [Dem11].

What about trees?

We can solve this problem in polynomial time using dynamic programming on trees!

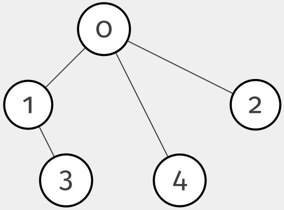
Recall: Trees

What is a tree?

A ***tree*** is an undirected graph in which any two vertices are connected by exactly one path [BW10].

Another definition

Connected acyclic undirected graph [BW10].



Dynamic Programming Approach

Elements of dynamic programming: Vertex Cover

Optimal Substructure

A solution for a given tree contains itself optimal solutions to subtrees.

Overlapping Subproblems

Subproblems to subtrees are called multiple times (you’ll see how!).

Base cases

Minimum Vertex Cover for 0 vertices

Clearly, the minimum vertex cover is 0.

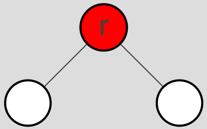
Minimum Vertex Cover for 1 vertex

Since there are no edges, the minimum vertex cover is 0 as well.

Intuition to apply DP

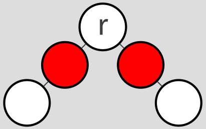
Given an undirected graph $G = (V, E)$.
We make two guesses for $v \in V$:

Root is part of a Minimum Vertex Cover



Then the root is in the cover. We are left with children subtrees as subproblems.

Root is not part of a Minimum Vertex Cover



Then the children of the root are part of the vertex cover. We are left with the grandchildren subtrees as subproblems.

Calculating the cardinality of a minimum vertex cover

A ***top-down memoized approach*** will suit better this problem, since we must traverse the tree from the root (root can be picked arbitrarily).

An easy way to memoize the subproblems is to ***save the cardinality of a minimum vertex cover*** in every node (*every node has a subtree/subproblem*).

Calculating the cardinality of a minimum vertex cover

Let $G = (V, E)$ be a tree.

$V(v)$ represents the cardinality of a minimum vertex cover for any $v \in V$.

$c \in V$ and $g \in V$ represent the children and grandchildren of a given node $v \in V$, respectively [Dem11].

Recurrence

$V(v) = \min\{1 + sum(V(c)), |c| + sum(V(g))\}$

Pseudocode

```
V(v):
    if v is null or |v.edges| == 0:
        return 0
    if v has not been visited:
        with_root = 1 + sum(V(c))
        without_root = |v.edges| + sum(V(g))
        dp[v] = min(with_root, without_root)
    return dp[v]
```

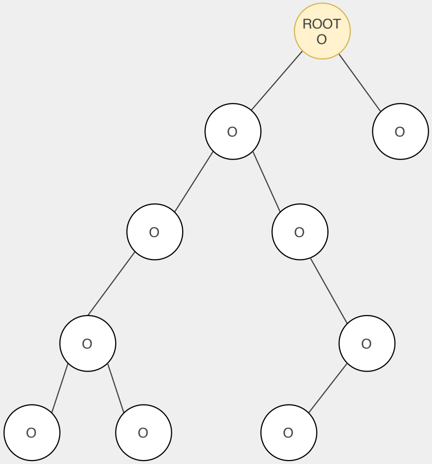
Listing 1: Dynamic programming approach

Lower level implementation (for binary trees)

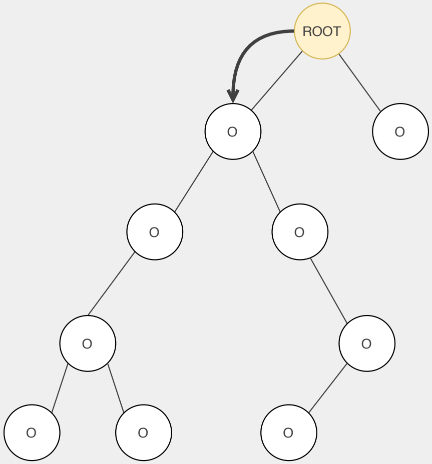
```
VC(index):
    if(notExists(index) or (notExists(right(index)
    and notExists(left(index) )):
        return 0
    if(tree[index] == 0):
        withRoot = 1 + VC(left(index)) + VC(right(index))
        withoutRoot = 0
        if(exists(left(index))):
            withoutRoot += 1 + VC(left(left(index))) +
            VC(right(left(index)))
        if(exists(right(index))):
            withoutRoot += 1 + VC(left(right(index))) +
            VC(right(right(index)))
        tree[index] = minValue(withRoot, withoutRoot)
    return tree[index]
```

Listing 2: Dynamic programming approach

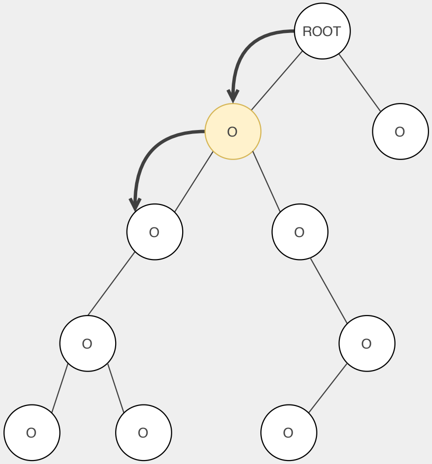
Demo of D. Programming Approach



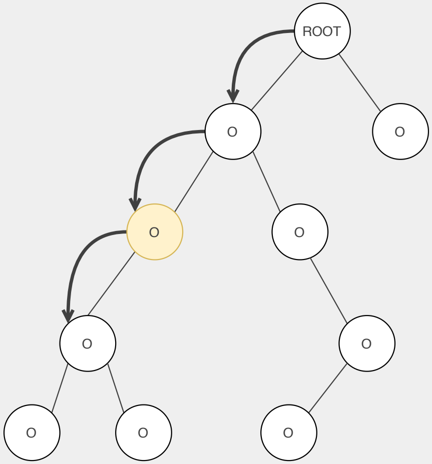
Demo of D. Programming Approach



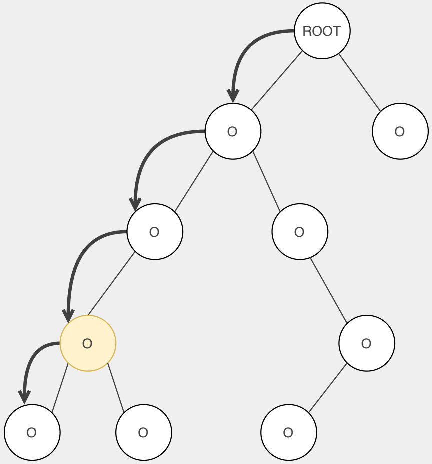
Demo of D. Programming Approach



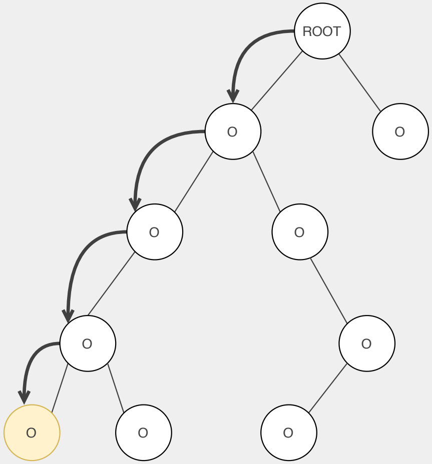
DEMO OF D. PROGRAMMING APPROACH



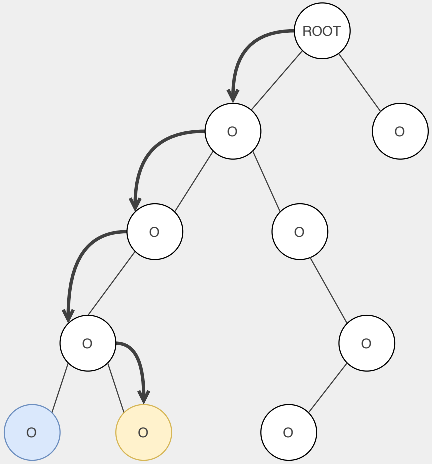
Demo of D. Programming Approach



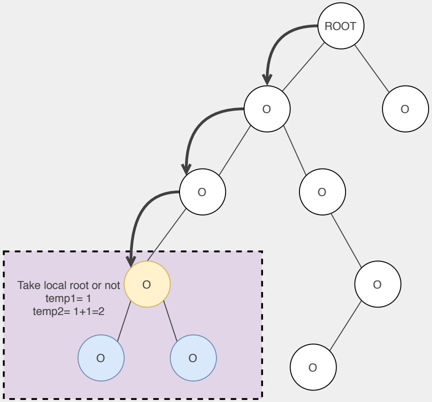
Demo of D. Programming Approach



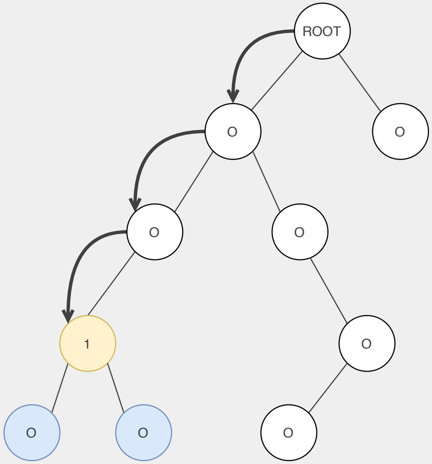
Demo of D. Programming Approach



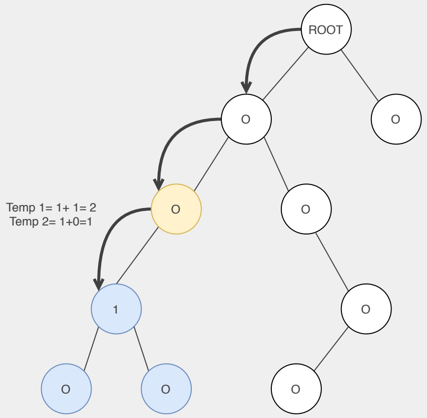
DEMO OF D. PROGRAMMING APPROACH



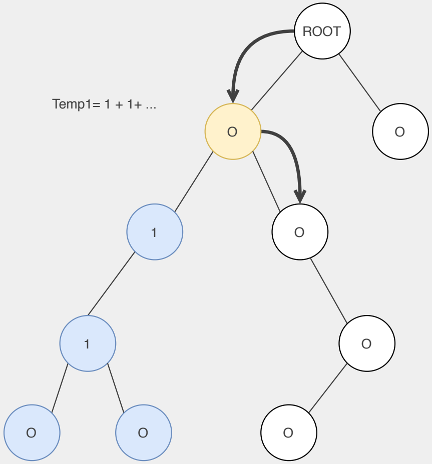
Demo of D. Programming Approach



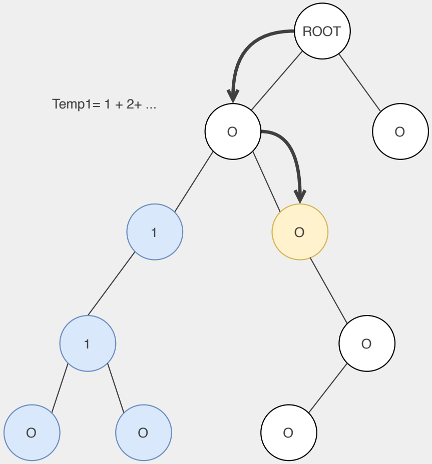
Demo of D. Programming Approach



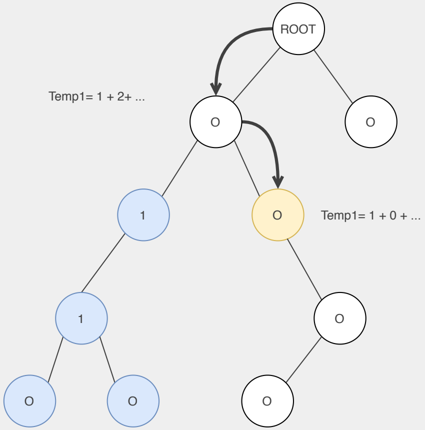
Demo of D. Programming Approach



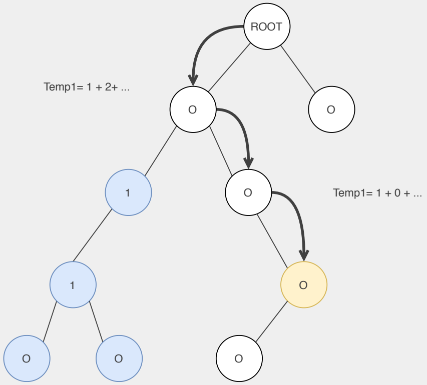
Demo of D. Programming Approach



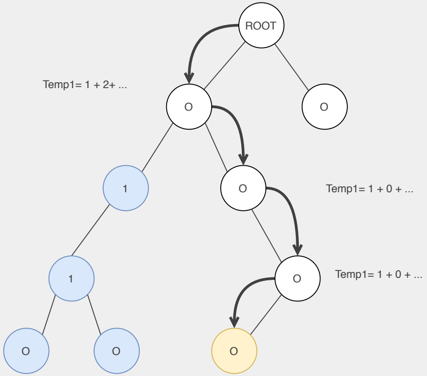
Demo of D. Programming Approach



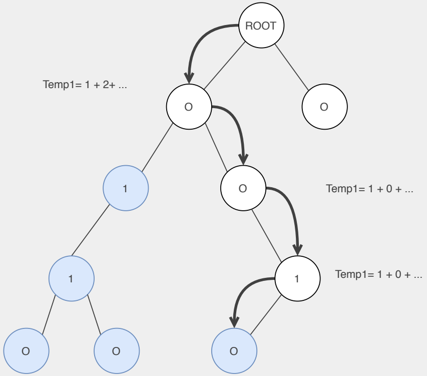
Demo of D. Programming Approach



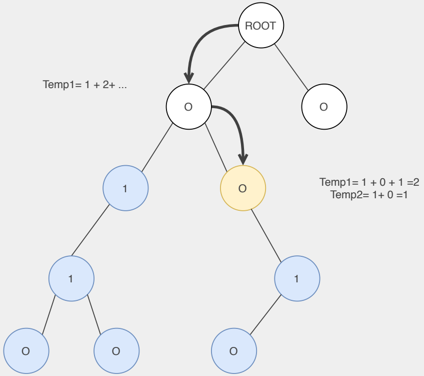
Demo of D. Programming Approach



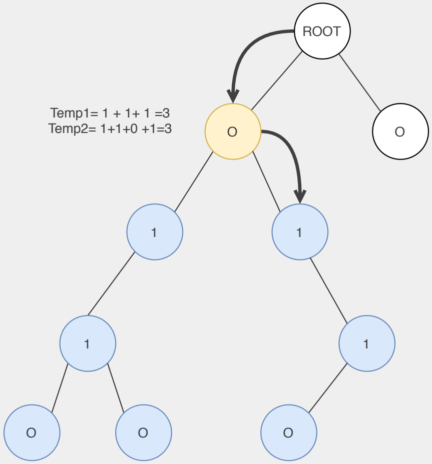
Demo of D. Programming Approach



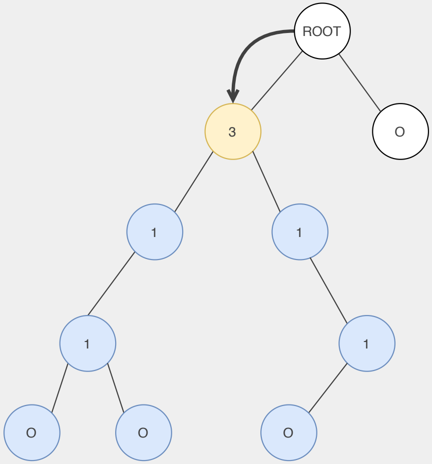
Demo of D. Programming Approach



Demo of D. Programming Approach

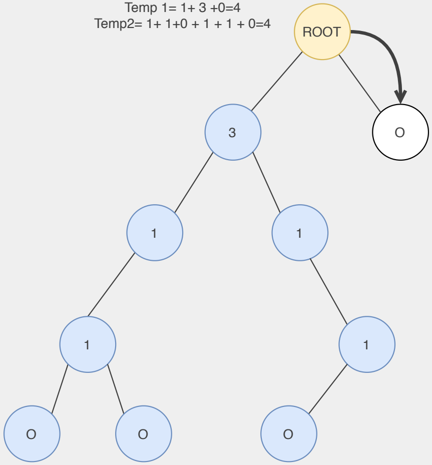


Demo of D. Programming Approach



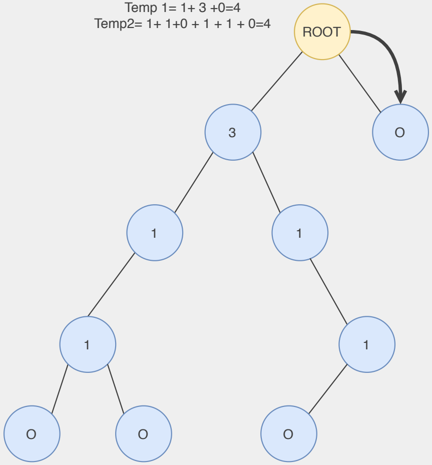
Demo of D. Programming Approach

Temp 1= 1+ 3 +0=4
Temp2= 1+ 1+0 + 1 + 1 + 0=4



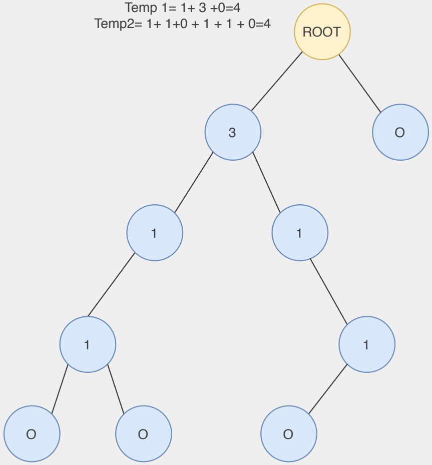
Demo of D. Programming Approach

Temp 1= 1+ 3 +0=4
Temp2= 1+ 1+0 + 1 + 1 + 0=4

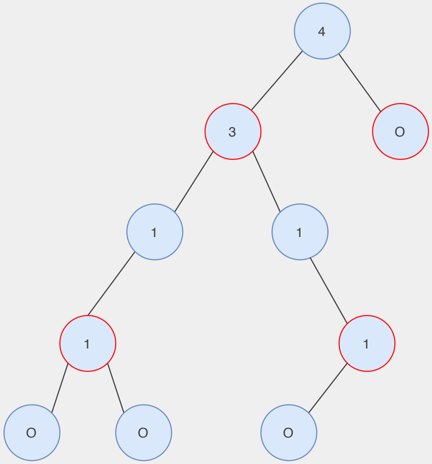


Demo of D. Programming Approach

Temp 1= 1+ 3 +0=4
Temp2= 1+ 1+0 + 1 + 1 + 0=4



Demo of D. Programming Approach



Get Vertex Cover

```


int height(Node node){
    if (node=null)
        return -1;
    return 1 + max(height(node.left), height(node.right));
}

int depth(Node root){
    return height(root);
}
```

Listing 3: Support Functions

Get Vertex Cover

```
vector<Node> vertexCover(Tree tree){
    vector<Node> result;
    bool oddDepth;
    if(depth%2==0) oddDepth = false;
    else oddDepth = true;
    for(node: tree){
        if(oddDepth){
            if (height(node)%2==0) result.push_back(node);
        }
        else{
            if (height(node)%2!=0) result.push_back(node);
        }
    }
    return result;
}
```

Listing 4: Dynamic programming approach

For any tree

```
V(v):
    if v is null or |v.edges| == 0:
        return 0
    if v has not been visited:
        with_root = 1 + sum(V(c))
        without_root = |v.edges| + sum(V(g))
        dp[v] = min(with_root, without_root)
        if root was chosen: paint root
        else: paint children
    return dp[v]
```

Listing 5: Dynamic programming approach

Time Complexity Analysis

At first glance...

We have $|V|$ subproblems. Each subproblem takes $O(V)$, since we iterate through the children and the grandchildren of every node.

But, actually...

We visit every node at most twice (as a child and as a grandchild). So the time taken by this procedure is $O(2V) = O(V)$.

Greedy Approach

Intuition

Keep finding a vertex which covers the maximum numbers of edges [Ban18].

Procedure

1. Find a vertex with maximum degree.
2. Add the vertex to the solution and remove the vertex with all the incident edges from the graph.
3. Repeat until all edges are covered.

Time Complexity

$\mathcal{O}(\log n)$

Greedy Pseudocode

```
1: function VertexCoverGreedy($G$)
2:     $C \leftarrow \emptyset$
3:     while $W \neq \emptyset$ do
4:         Pick vertex $v \in V$ of maximum degree in $G$
5:         $C \leftarrow C \cup \{v\}$
6:         $E \leftarrow E \setminus \{e \cup E : v \cup e\}$
7:     end while
8:     return C
9: end function
```

[ZoL05]

Conclusion

- The hardest step in the Dynamic programming approach is not to obtain the optimal substructure, but to count nodes that compose it (for binary trees).
- The Greedy approach is more optimal but it is just an approximation, for that reason choosing to use the Dynamic Programming approach is the best option (when it is possible) to solve the Minimum Vertex Cover Problem.

Extra resources

Vertex Cover DP implementation:
`https://github.com/StephanoWurttele/vertexCover`

References I

- Richita Bandyopadhyay, *Vertex cover problem*, 11 2018.
- Edward A. Bender and S. Gill Williamson, *Lists, decisions and graphs*, S. Gill Williamson, 2010.
- Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein, *Introduction to algorithms, third edition*, 3rd ed., The MIT Press, 2009.
- Erik Demaine, *Introduction to algorithms*, 2011.
- Michele Zito and University of Liverpool, *Vertex cover u*, 2005.