Minimum Weighted Vertex Cover in a Tree

Given an undirected graph $G = (V, E)$, a set $S \subseteq V(G)$ is said to be a *vertex cover* of G if for every $(u, v) \in E(G)$, $u \in S$ or $v \in S$ (or both). If the graph is vertex weighted, i.e. every vertex has a weight, say the vertex v has $\text{wt}(v)$, then the weight of a vertex cover $\text{wt}(S) = \sum_{v \in S} \text{wt}(v)$.

A vertex cover S of a graph G is called a *minimum weighted vertex cover* of G if $\text{wt}(S)$ is minimum among all possible vertex covers of G.

Problem: The input to the problem is a vertex weighted rooted binary tree $T = (V, E)$ and $\text{wt} : V(T) \to \mathbb{R}^+$.

Objective: Find a vertex cover with minimum total weight.

Assumption: The *rooted binary tree* T is given as part of the input. A post order traversal of the rooted tree T ensures that for every node u that is not a leaf, both its children appear before u.

For every node $x \in V(T)$, let T_x be the subtree of T rooted at the node x.

Subproblem Definition:

For every node $x \in V(T)$, we define three subproblems.

- $\mathsf{MVC}(x, 1)$ be the weight of a minimum weight vertex cover of T_x that contains x.
- $\mathsf{MVC}(x, 0)$ be the weight of a minimum weight vertex cover of T_x that does not contain x.
- $\mathsf{VC}(x)$ be the weight of a minimum weight vertex cover of T_x.

Trivially observe that $\mathsf{VC}(x) = \min\{\mathsf{MVC}(x, 0), \mathsf{MVC}(x, 1)\}$.

Explanation for the ease of interpretation: Let S be a minimum weight vertex cover of the tree T_x subject to the constraint that $x \in S$, then S is a *representative optimal solution* for the subproblem $\mathsf{MVC}(x, 1)$. Similarly, if S is a minimum weight vertex cover of the tree T_x subject to the constraint that $x \notin S$, then S is a representative optimal solution for $\mathsf{MVC}(x, 0)$.

Recurrence of subproblem:

The recurrence of the subproblems MVC will look at the values for its two children and will solve it.

Base Case: If x is a leaf, then $\mathsf{MVC}(x, 0) = 0$ since T_x is just a single node tree and trivially $\emptyset$ is a vertex cover of T_x and that is the only possible vertex cover for the subproblem $\mathsf{MVC}(x, 0)$. Similarly, $\{x\}$ is the only possible vertex cover for the subproblem $\mathsf{MVC}(x, 1)$. Hence, $\mathsf{MVC}(x, 1) = \mathrm{wt}(x)$.

Non-Leaf nodes: Let x_1 and x_2 be two children of x. Then, we have the followings.

$$\mathsf{MVC}(x, 0) = \mathsf{MVC}(x_1, 1) + \mathsf{MVC}(x_2, 1)$$

$$\mathsf{MVC}(x, 1) = \mathrm{wt}(x) + \mathsf{VC}(x_1) + \mathsf{VC}(x_2)$$

Finally, $\mathsf{VC}(x) = \min\{\mathsf{MVC}(x, 1), \mathsf{MVC}(x, 0)\}$.

Subproblem that solves the final problem: Solving the subproblems for the root vertex r solves the final problem. Hence, $\mathsf{VC}(r)$ solves the final problem.

Algorithm Description:

1. Let ρ be a post order traversal of the vertices of tree T. Note that the root r appears at the end, and for every other node x, their children appear before x.
2. Process the nodes in the order in which they appear in ρ.
3. Repeat the following steps for every node.
4. If x is a leaf, then set $A[x, 0] = 0$, i.e. the value of $\mathsf{MVC}(x, 0)$ and $A[x, 1] = \mathrm{wt}(x)$, i.e. the value of $\mathsf{MVC}(x, 1)$. Set $A[x, 2] = \min(A[x, 0], A[x, 1])$.
5. If x is not a leaf, then let x_1 and x_2 be its two children. Set $A[x, 0] = A[x_1, 1] + A[x_2, 1]$. Set $A[x, 1] = \mathrm{wt}(x) + A[x_1, 2] + A[x_2, 2]$. Set $A[x, 2] = \min\{A[x, 0], A[x, 1]\}$.
6. Finally, return $A[r, 2]$ where the root r appears at the last.

Running Time: Computing post order traversal of a rooted tree takes $O(n)$-time where n is the number of nodes in a tree (binary) rooted at r. Hence, the ordering ρ is a post-order traversal of the nodes of the tree. The root appears at the end. For every leaf node x (including checking whether x is a leaf or not), computing $\mathsf{MVC}(x, 0), \mathsf{MVC}(x, 1)$ and $\mathsf{VC}(x)$ take $O(1)$-time. If x has some child, then the following happens. Suppose that x has t children. Then, accessing each of its children takes $O(1)$-time and then time taken to compute the value $\mathsf{MVC}(x, 1)$ takes $O(t)$-time computing and $\mathsf{MVC}(x, 0)$ takes $O(t)$-time. Since $t = 2$ here as T is a binary tree, computing $\mathsf{MVC}(x, 1)$ and $\mathsf{MVC}(x, 0)$ both take $O(1)$-time. Finally, computing $\mathsf{VC}(x)$ takes $O(1)$-time. Therefore, the total running time is $O(n)$.

Advise for assignment and exam: You must write the above mentioned details in parts when you answer a DP question in the exam. Remember that DP recurrence is separate from the algorithm description. You can write a pseudocode that can use memoization method or method of filling up table. The above algorithm uses filling up of table that is the same as tabulation method. In summary, your solution must contain the following components.

- Subproblem definition.
- Recurrence relation for the subproblem.
- The specific subproblem that solves the final problem.
- High level description of the algorithm.
- Explanation of the running time of your algorithm.

Explanations for Correctness of the Recurrence od Subproblem: You have to convince yourself for the following two statements. Think about it why that is the case. It will be proof by contradiction.

- **Statement-1**: Let S be a minimum weight vertex cover of T_x such that $x \in S$, i.e. S is an optimal solution for $\mathsf{MVC}(x, 1)$. Furthermore, let y and z be the two children for x. Then, $S = \{x\} \cup S_y \cup S_z$ such that S_y is an optimal solution for the subproblem $\mathsf{VC}(y)$ and S_z is an optimal solution for the subproblem $\mathsf{VC}(z)$.
- **Statement-2:** Let S be a minimum vertex cover of T_x such that $x \notin S$, i.e. S is an optimal solution for $\mathsf{MVC}(x, 0)$. Furthermore, let y and z be the two children for x. Then, $S = S_y \cup S_z$ such that S_y is an optimal solution for the subproblem $\mathsf{MVC}(y, 1)$ and S_z is an optimal solution for the subproblem $\mathsf{MVC}(z, 1)$.

Exercise: Even if the tree T is not a binary tree, you can design a dynamic programming algorithm with designing exactly the same subproblem definition. Think what the recurrence of the subproblem will be. A small change will work out. But it is your exercise to figure it out. Also, think a high level idea how you can implement that algorithm in $O(n)$-time.