Maximum Weight Independent Set in a Tree

Given: A tree with weights for each vertex. We use \( W(v) \) to denote the weight of vertex \( v \).

To do: Find a set \( S \) of vertices in the tree, none adjacent, such that the sum of the vertex weights in \( S \), \( \sum_{v \in S} W(v) \), is the largest possible.

Step 1: Characterize optimal subproblems

For nearly every problem related to trees, we root the tree first because subproblems normally correspond to subtrees.

Let \( v \) be the root of a tree \( T \) with children \( v_1 \ldots v_c \) and grandchildren \( w_1 \ldots w_g \).

For a vertex \( w \) in the tree, let \( T_u \) denote the subtree rooted at \( u \).

If \( v \) is not in the optimal solution \( S \), then \( S = S_{v_1} \cup \ldots \cup S_{v_c} \)
where each \( S_{v_i} \) is an optimal solution for \( T_{v_i} \).

Assume the contradiction: some \( S_{v_i} \) is not optimal for \( T_{v_i} \).

Let \( S'_{v_i} \) be optimal for \( T_{v_i} \).

Then, \( \sum_{u \in S_{v_i}} W(u) < \sum_{u \in S'_{v_i}} W(u) \).

Then, \( S' = (S \setminus S_{v_i}) \cup S'_{v_i} \) is a solution for \( T \) because \( v \) is not in \( S \).

However, \( \sum_{u \in S} W(u) < \sum_{u \in S'} W(u) \), a contradiction.

If \( v \) is in the optimal solution \( S \), then \( S = \{v\} \cup S_{w_1} \cup \ldots \cup S_{w_g} \)
where each \( S_{w_i} \) is an optimal solution for \( T_{w_i} \).

Assume the contradiction: some \( S_{w_i} \) is not optimal for \( T_{w_i} \).

Let \( S'_{w_i} \) be optimal for \( T_{w_i} \).

Then, \( \sum_{u \in S_{w_i}} W(u) < \sum_{u \in S'_{w_i}} W(u) \).

Then, \( S' = (S \setminus S_{w_i}) \cup S'_{w_i} \) is a solution for \( T \) because its parent, a child of \( v \), is not in \( S \).

However, \( \sum_{u \in S} W(u) < \sum_{u \in S'} W(u) \), a contradiction.

Step 2: Recursive algorithm

\[
\text{IS-value}(T, W) \\
\text{let } v \text{ be the root of } T \\
\text{let } v_1 \ldots v_c \text{ be the children of } v
\]
let $w_1 \ldots w_g$ be the grandchildren of $v$
return $\max\{\text{IS-value}(T_{v_1}, W) + \ldots + \text{IS-value}(T_{v_c}, W)\}$
$\quad\text{IS-value}(T_{w_1}, W) + \ldots + \text{IS-value}(T_{w_g}, W) + W(v)\}$

Or, equivalently ...

\begin{verbatim}
IS-value(T,W)
  return max{ IS-value-with-root(T,W), IS-value-no-root(T,W) }
IS-value-with-root(T,W)
  let $v_1 \ldots v_c$ be the children of root $v$ of $T$
  return IS-value-no-root(T_{v_1}, W) + \ldots + IS-value-no-root(T_{v_c}, W) + W(v)
IS-value-no-root(T,W)
  let $v_1 \ldots v_c$ be the children of root $v$ of $T$
  return IS-value(T_{v_1}, W) + \ldots + IS-value(T_{v_c}, W)
\end{verbatim}

Though this algorithm does not have exponential running time, it does solve overlapping subproblems. For a full binary tree $T$ on $n$ vertices, the algorithm makes at least $n^{3/2}/9$ recursive calls.

Let $h$ be the height of $T$. Then, the running time is: $T(h) \geq 2T(h-1) + 4T(h-2)$, $T(1) \geq 1$.

We prove by induction that $T(h) \geq 3^{h-1}$:

**Base:** $T(1) \geq 1 = 3^0$

**Induction assumption:** $T(h) \geq 3^{h-1}$ for $h \leq K$ for some constant $K \geq 0$.

**Induction step:** $T(K+1) \geq 2T(K) + 4T(K-1)$
\begin{align*}
&\geq 2 \cdot 3^{K-1} + 4 \cdot 4^{K-2} \\
&\geq (2\cdot 3 + 4)4^{K-2} \\
&\geq 9 \cdot 3^{K-2} \\
&\geq 3^K.
\end{align*}

Since $n=2^{h+1}-1$, we have
\[n^{3/2} \leq (2^{h+1})^{3/2} = 2^{(h+1)3/2} \leq 3^{h+1}.
\]
In other words, $n^{3/2}/9 \leq 3^{h-1} \leq T(h)$.

**Step 3: Dynamic programming algorithm**
We can reduce the running time to $O(n)$ by observing that there are $2n$ different subproblems to solve, two for each vertex $u$ in the tree:

1. $\text{IS-value}(T_u, W)$
2. $\text{IS-value-no-root}(T_u, W)$

Once again, there are two ways to avoid solving the same subproblem twice. Here is the bottom-up technique:

```
IS-value(T, W)
  is = an (vertices of T)x(optimal,noroot) table
  for $h = \text{height}(T)$ .. 0 do
    for each vertex $u$ at height $h$ do
      let $u_1 ... u_c$ be the children of $u$
      $is[u,\text{noroot}] = is[u_1,\text{optimal}] + ... + is[u_c,\text{optimal}]$
      $is[u,\text{optimal}] = \max \{ is[u,\text{noroot}],$
                          $is[u_1,\text{noroot}] + ... + is[u_c,\text{noroot}] + W(u) \}$
  return $is[\text{root}(T),\text{optimal}]$
```

In this algorithm, we compute $2n$ values in the table, and reference each element once, so the algorithm runs in $O(n)$ time.

**Step 4: Reconstructing an optimal solution**

An exercise for the reader.

[back to examples](#)