Recapture Dynamic programming

Chapter 6 in Kleinberg & Tardos

- Independent Set on Trees

Dynamic Programming Summary

Recipe.

1. Characterize structure of problem.
2. Recursively define *value* of optimal solution.
3. Compute and store *values* of optimal solution iteratively.
4. Construct optimal solution itself from computed information.

Dynamic programming techniques.

- Binary choice: weighted interval scheduling
- Multi-way choice: segmented least squares.
- Adding a new variable: knapsack.
- Dynamic programming over intervals (more subproblems): RNA secondary structure.

TU Delft

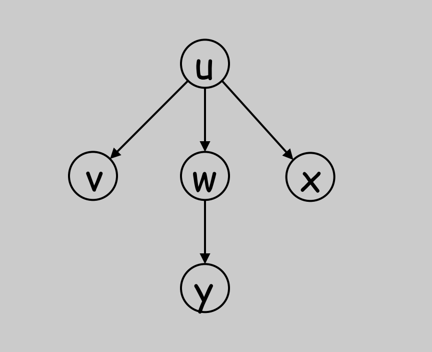
Weighted Independent Set on Trees

neighbors are not allowed

Weighted independent set on trees. Given a tree and node weights $w_v > 0$, find an independent set S that maximizes $\Sigma_{v \in S} w_v$.

Brute Force. $O(2^n)$

With dynamic programming… efficiently solvable!

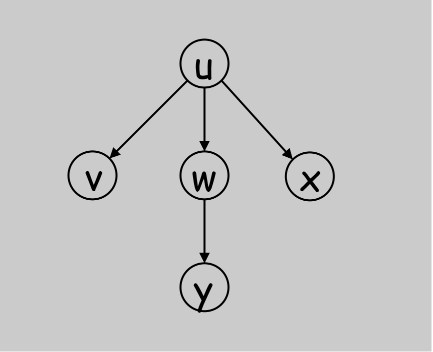


Weighted Independent Set on Trees

neighbors are not allowed

Weighted independent set on trees. Given a tree and node weights $w_v > 0$, find an independent set S that maximizes $\Sigma_{v \in S} w_v$.

Idea. Use dynamic programming to optimize sum of weights OPT(u) for a tree with root u.



Start with defining a search tree:

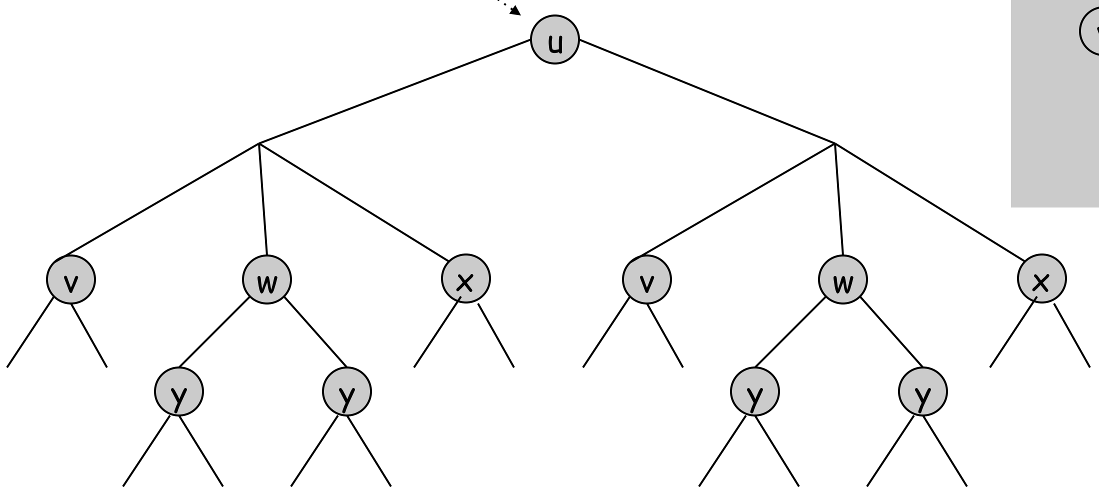
Q. Starting at root u, what are the options?

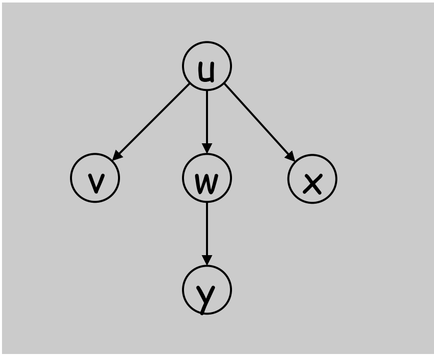
A.

1. include u, or
2. don't include u

Independent Set: Brute Force Search Tree

include u? yes (right) or no (left)?





Observation. Number of nodes grows exponentially with problem size.

Observation. Search tree may contain redundant sub-problems (e.g. y).

Two types of subproblems y: where y may be chosen, or not.

Idea. Dynamic programming:

1. Store and reuse solutions to subproblems.
2. Compute these bottom-up.

Weighted Independent Set on Trees

neighbors are not allowed

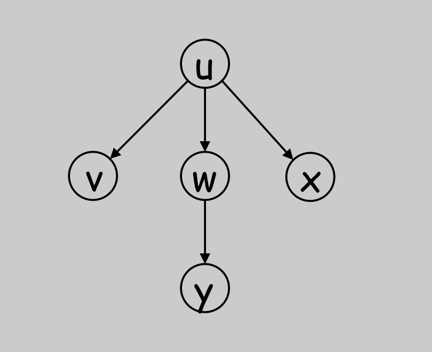
Weighted independent set on trees. Given a tree and node weights $w_v > 0$, find an independent set S that maximizes $\Sigma_{v \in S} w_v$.

Idea. Use dynamic programming to optimize sum of weights for a tree with root u.

Q. Starting at root u, what are the options?

A.

1. include u (and thus don't include children of u), or
2. don't include u (and possibly include all children of u).



children(u) = { v, w, x }

Q. How to express the value of an optimal solution in these cases?

A.

1. $= w_u$ + sum over optimal solution of children, excluding children
2. = sum over optimal solution of children

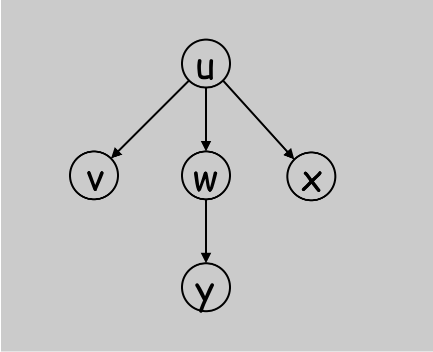
Idea. Use different notation for optimal solution with and without u.

Weighted Independent Set on Trees

Idea. Use different notation for OPT with and without u.

- OPT_{in} (u) = max weight independent set rooted at u, containing u.
- OPT_{out}(u) = max weight independent set rooted at u, not containing u.

$$OPT(u) = \max\left\{OPT_{in}(u), OPT_{out}(u)\right\}$$



children(u) = { v, w, x }

The two subcases are:

1. include u and don't include children of u, or
2. don't include u and possibly include all children of u.

Give recursive formulas for OPT_{in} and OPT_{out}.

$$OPT_{in}(u) = w_u + \sum_{v \in \text{children}(u)} OPT_{out}(v)$$

$$OPT_{out}(u) = \sum_{v \in \text{children}(u)} \max\left\{OPT_{in}(v),\ OPT_{out}(v)\right\}$$

TU Delft

Independent Set on Trees: DP Algorithm

Claim. The following dynamic programming algorithm efficiently finds a maximum weighted independent set in trees.

```
Weighted-Independent-Set-In-A-Tree(T) {
   Root the tree at a node r
   foreach (node u of T in postorder) {
      if (u is a leaf) {
         M_in [u] = w_u
         M_out[u] = 0
      }
      else {
         M_in [u] = Σ_{v∈children(u)} M_out[v]  +  w_u
         M_out[u] = Σ_{v∈children(u)} max(M_out[v], M_in[v])
      }
   }
   return max(M_in[r], M_out[r])
}
```

start from bottom:
ensures a node is visited after all its children

Q. What is the run time of this algorithm?

A. Takes O(n) time since we visit nodes in postorder and examine each edge exactly once. ▪