Lecture 5: Dynamic Programming on Trees

Yury Makarychev
TTIC and the University of Chicago

Dynamic Programming on Trees

DP Tables

1 dimensional: Puzzle, Job Scheduling, Typesetting, …

$T[1]$	$T[2]$								$T[n]$

2 dimensional: Knapsack, …

$T[0,W]$									$T[n,W]$
$T[0,0]$									$T[n,0]$

k dimensional

DP Tables

1 dimensional: Puzzle, Job Scheduling, Typesetting, …

$T[1]$	$T[2]$								$T[n]$

2 dimensional: Knapsack, …

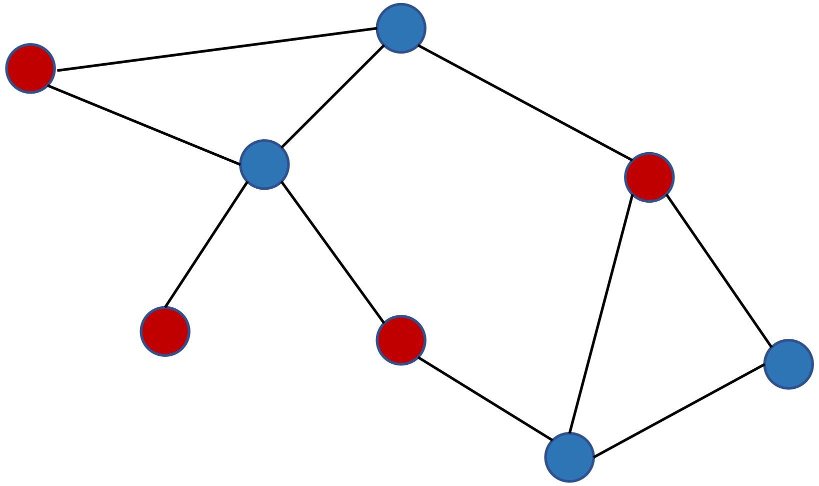
$T[0,W]$									$T[n,W]$
$T[0,0]$									$T[n,0]$

k dimensional

Maximum Weight Independent Set

- We are given a graph $G = (V, E)$ with vertex weights w_u

A subset $I \subseteq V$ is an independent set if for every edge $(u, v) \in E$, at most one of the vertices u and v is I.

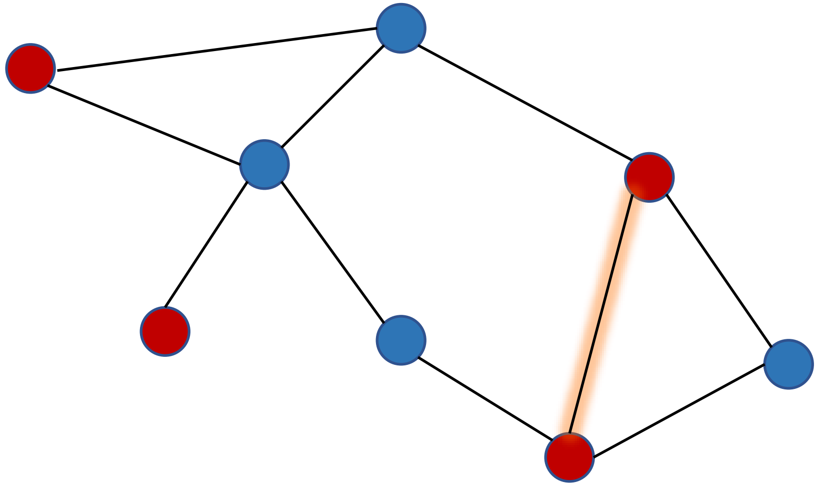


There are no edges between points in the independent set.

Maximum Weight Independent Set

- We are given a graph $G = (V, E)$ with vertex weights w_u

A subset $I \subseteq V$ is an independent set if for every edge $(u, v) \in E$, at most one of the vertices u and v is I.



This is not an independent set!

Maximum Weight Independent Set

Given a graph $G = (V, E)$ find an independent set I of maximum weight

$$w(I) = \sum_{u \in I} w_u$$

Maximum Weight Independent Set

Independent set is a very important combinatorial problem.

E.g. has applications in compiler design

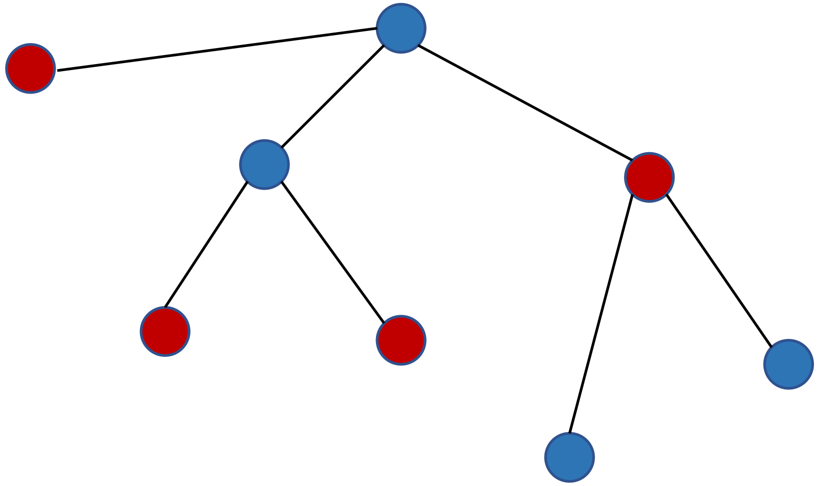
It is NP-hard. It’s even very hard to get any reasonable approximate solution in the worst case.

We can solve it on trees!

Maximum Weight Independent Set

- We are given a tree $T = (V, E)$ with vertex weights w_u

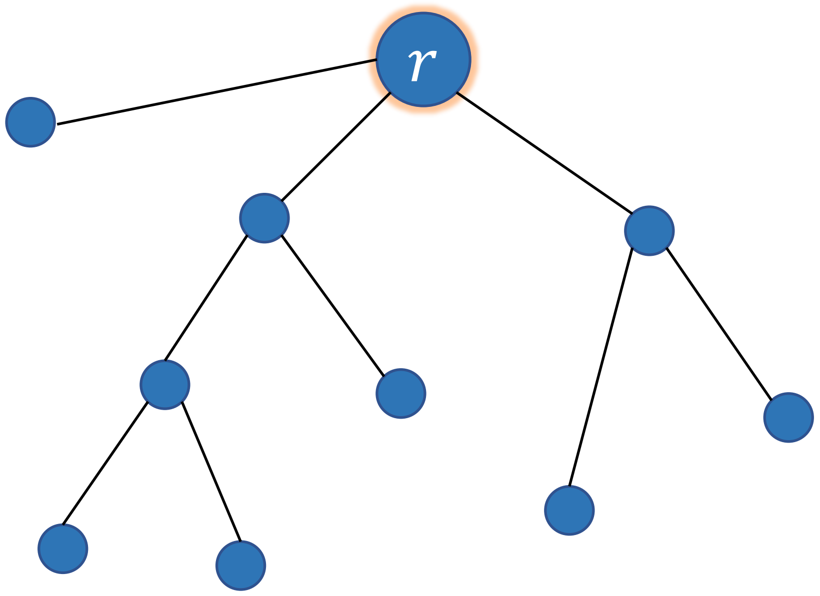
A subset $I \subseteq V$ is an independent set if for every edge $(u, v) \in E$, at most one of the vertices u and v is I.



There are no edges between points in the independent set.

Dynamic Program

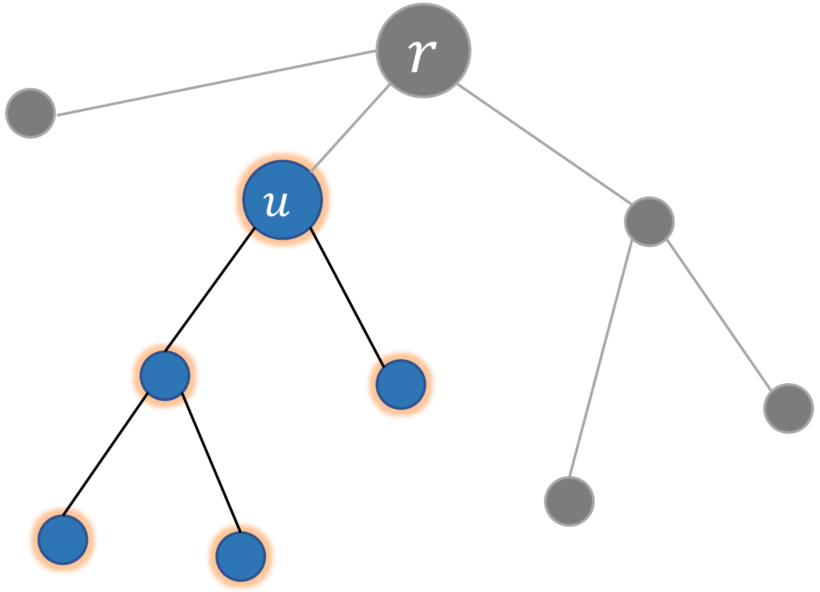
Choose a root r (arbitrarily)



Dynamic Program

Choose a root r (arbitrarily)

For every vertex u,
let T_u be the subtree rooted at u.



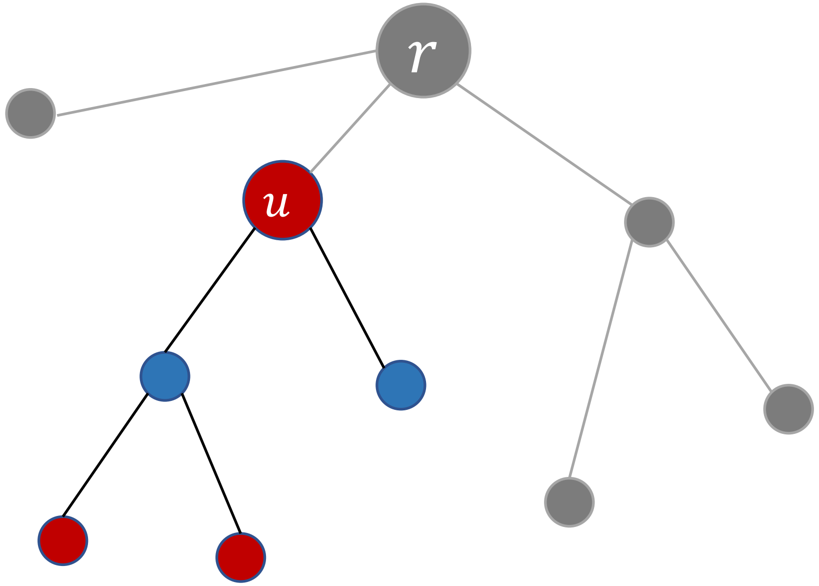
Dynamic Program

Choose a root r (arbitrarily)

For every vertex u,
let T_u be the subtree rooted at u.

Define subproblems:

- let $A[u]$ be the weight of a maximum weight independent set in T_u
- …



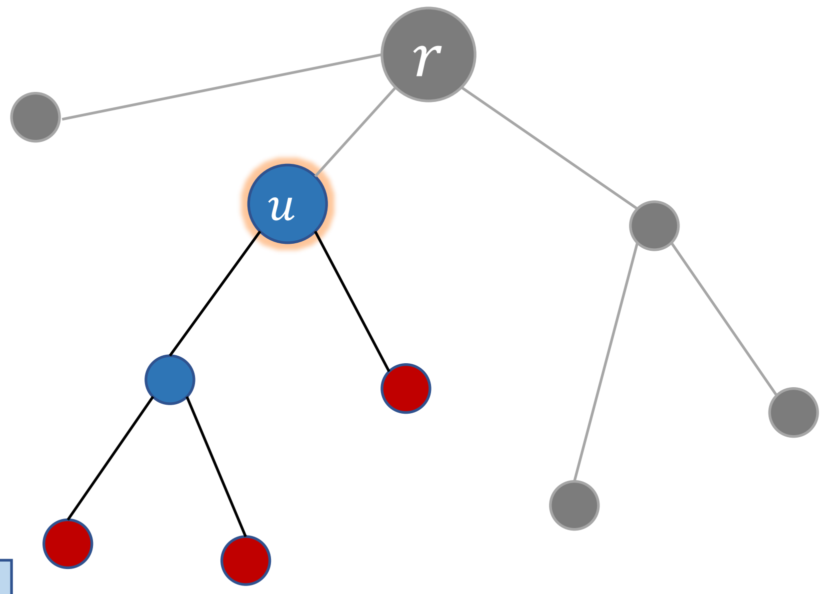
Dynamic Program

Choose a root r (arbitrarily)

For every vertex u,
let T_u be the subtree rooted at u.

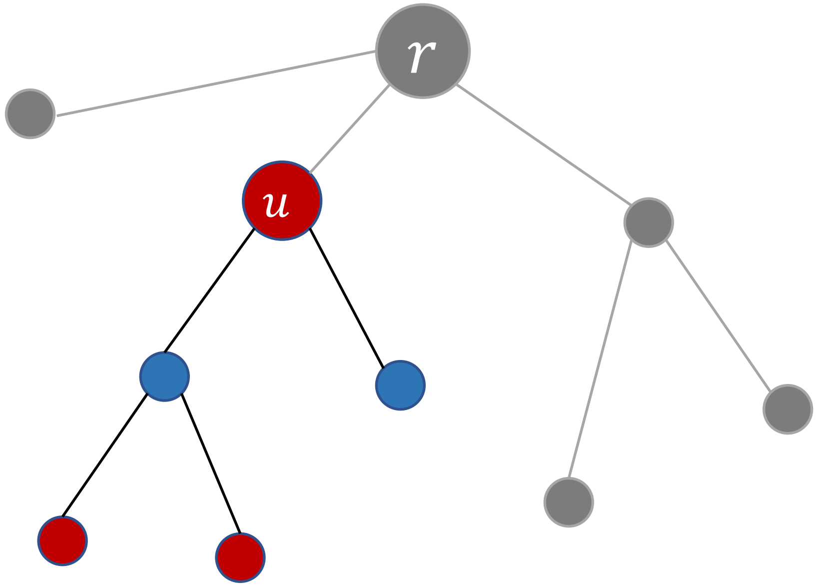
Define subproblems:

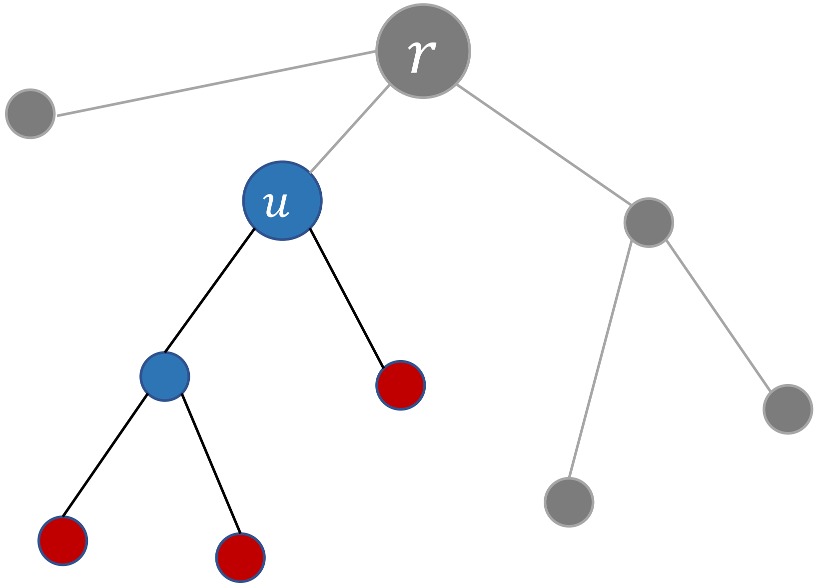
- let $A[u]$ be the weight of a maximum weight independent set in T_u
- $B[u]$ be the weight of a maximum weight independent set I in T_u s.t. $u \notin I$



DP: Quiz!

$A[u] \leq B[u]$ or $A[u] \geq B[u]$ or *it depends* ...

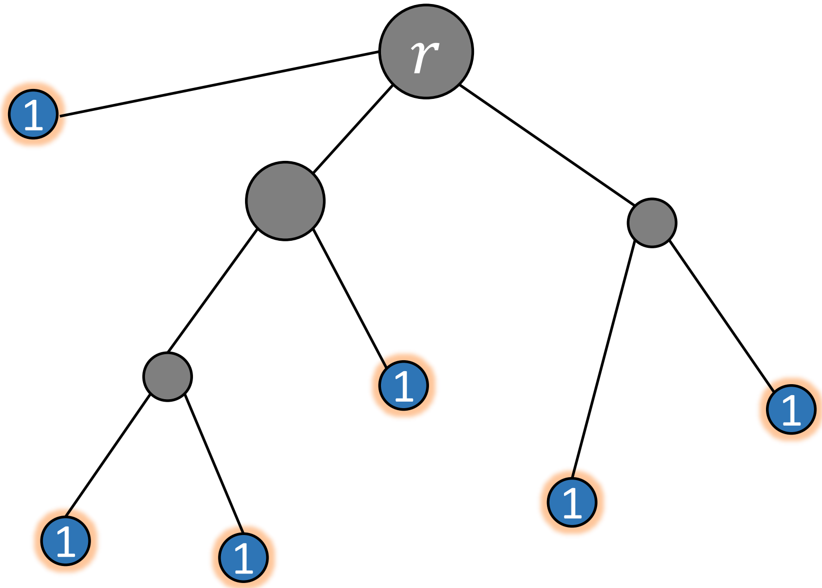




$A[u]$ is the weight of a maximum weight independent set in T_u

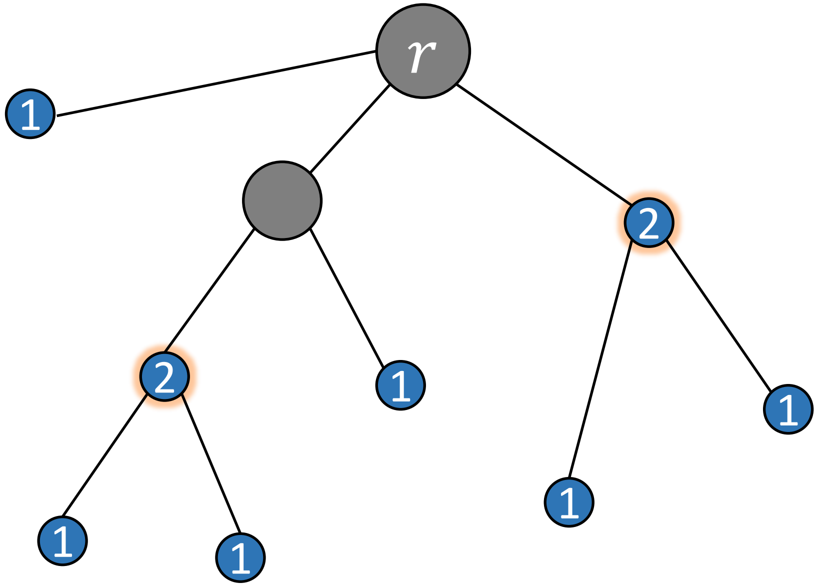
$B[u]$ is the weight of a maximum weight independent set I in T_u s.t. $u \notin I$

DP: Bottom-up Approach



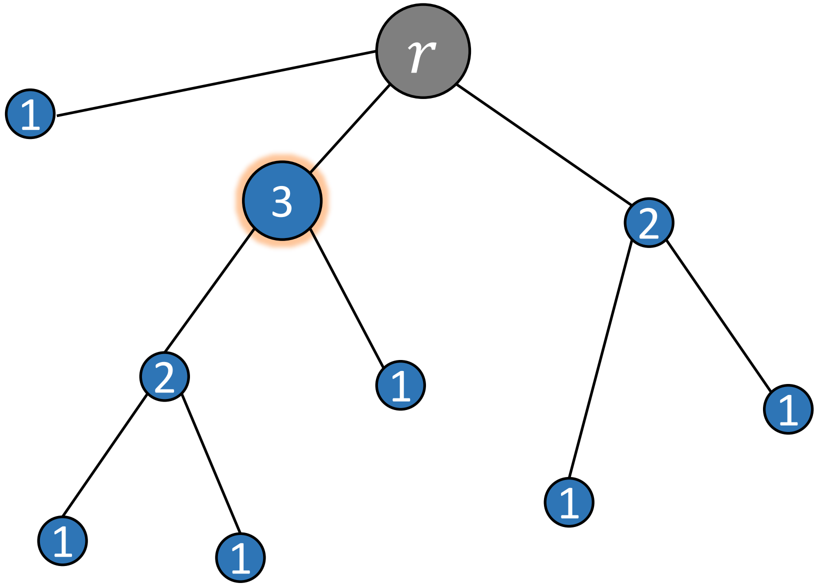
- Compute $A[u]$ and $B[u]$ for all leaves u

DP: Bottom-up Approach



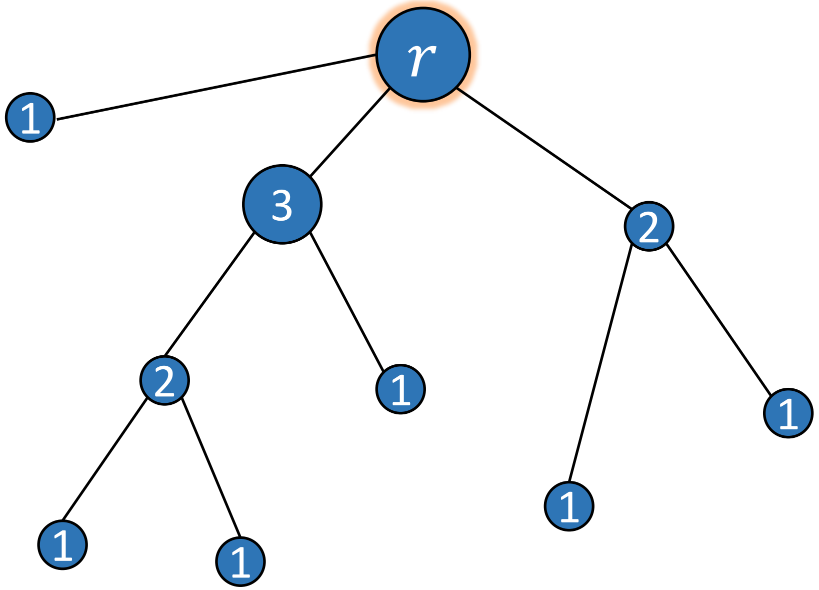
- Compute $A[u]$ and $B[u]$ for all leaves u
- Compute $A[u]$ and $B[u]$ for parents of the leaves.

DP: Bottom-up Approach



- Compute $A[u]$ and $B[u]$ for all leaves u
- Compute $A[u]$ and $B[u]$ for parents of the leaves.
- … their parents

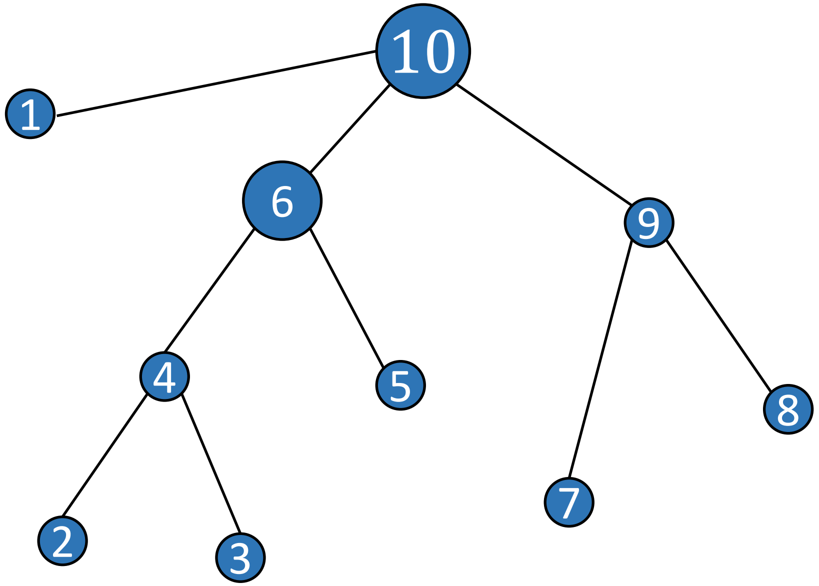
DP: Bottom-up Approach



- Compute $A[u]$ and $B[u]$ for all leaves u
- Compute $A[u]$ and $B[u]$ for parents of the leaves.
- … their parents
- until we reach the root

output $\boldsymbol{A[r]}$

DP: Bottom-up Approach



The specific order in which we fill out the DP entries is not important, as long as we

compute the entries for u after we computed the entries for all children of u.

Options

- Based on the depth of T_u (as we saw)
- Use depth-first traversal. Compute vertices in the post-order (see the figure).
- Based on the distance to the root (i.e. depth)

Recursion with Memoization

function FillOutDP (vertex u)

if $A[u]$ and $B[u]$ are assigned, return $A[u]$, $B[u]$

if u is a leaf,

use initialization formulas to compute $A[u]$, $B[u]$

if u is not a leaf,

recursively call FillOutDP(v) for all children v of u

compute $A[u]$ and $B[u]$ using recurrence formulas

return

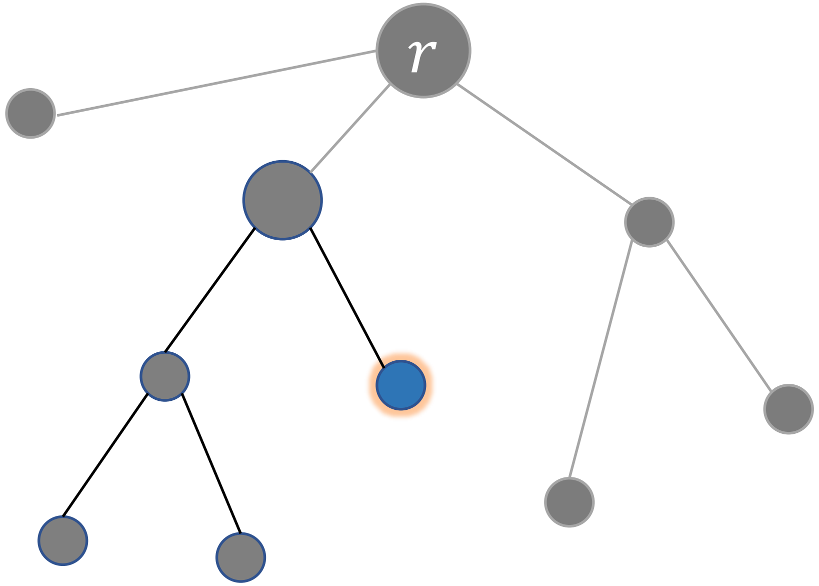
Algorithm: FillOutDP(r)

DP: Initialization

Consider a leaf u.

$$A[u] = ?$$
$$B[u] = ?$$

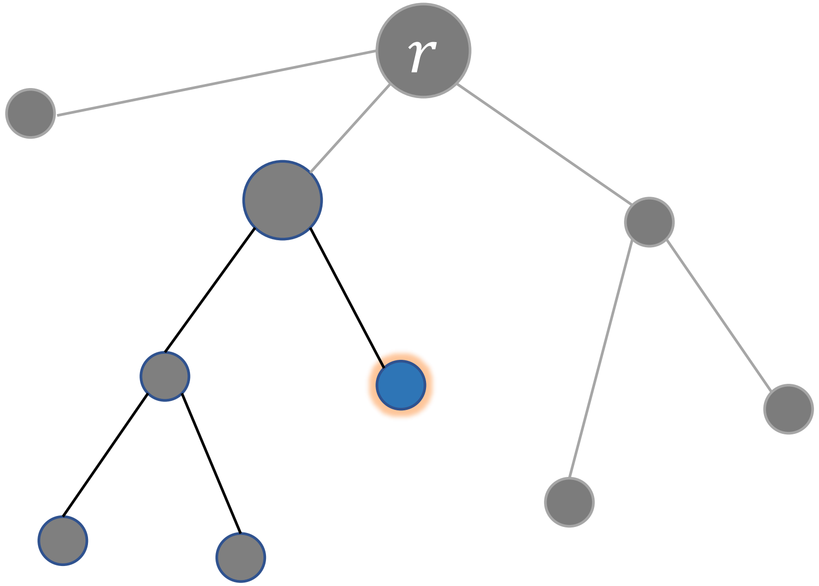
Any suggestions?



DP: Initialization

Consider a leaf u.

$$A[u] = w_u$$
$$B[u] = 0$$



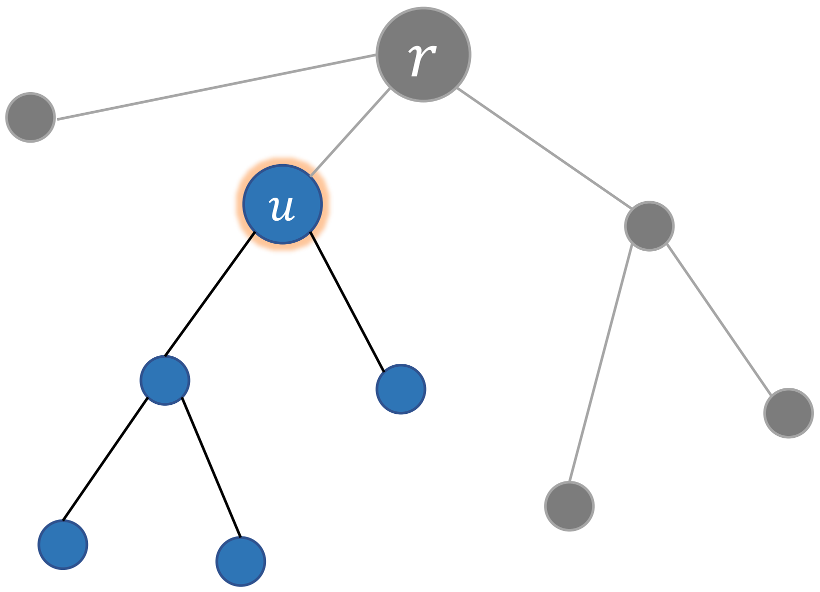
DP: Recurrence

Consider non-leaf vertex u. Assume that all vertices in T_u other than u have been already processed.

Let I be an optimal solution for T_u.

There are two options:

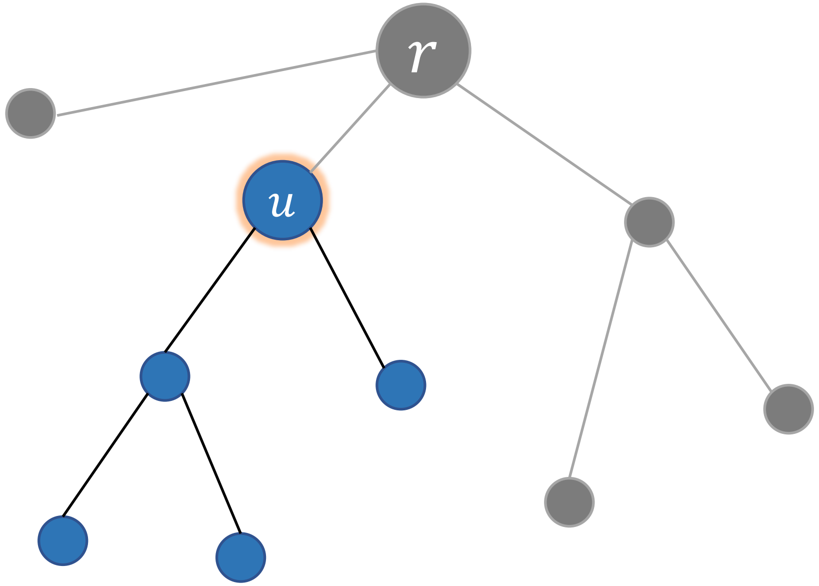
- $u \notin I$. Each child of u may be in I or not in I
- $u \in I$. Then children of u are not in I.



DP: Recurrence

- $u \notin I$. Each child of u may be in I or not in I

Q: What is the best way to construct I?



DP: Recurrence

- $u \notin I$. Each child of u may be in I or not in I

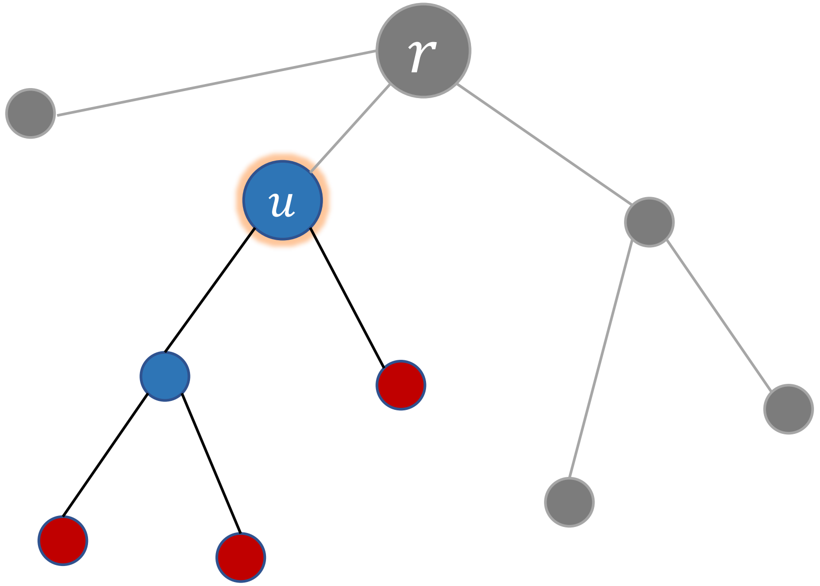
Q: What is the best way to construct I?

A: Simply choose a maximum independent I_v in tree T_v for each child v of u.

$$I = \bigcup_{v \text{ is a child of } u} I_v$$

I is an independent set since

- there are no edges between subtrees T_v and T_w for distinct children v and w
- No problems with edges incident on u, since $u \notin I$



DP: Recurrence

- $u \notin I$. Each child of u may be in I or not in I

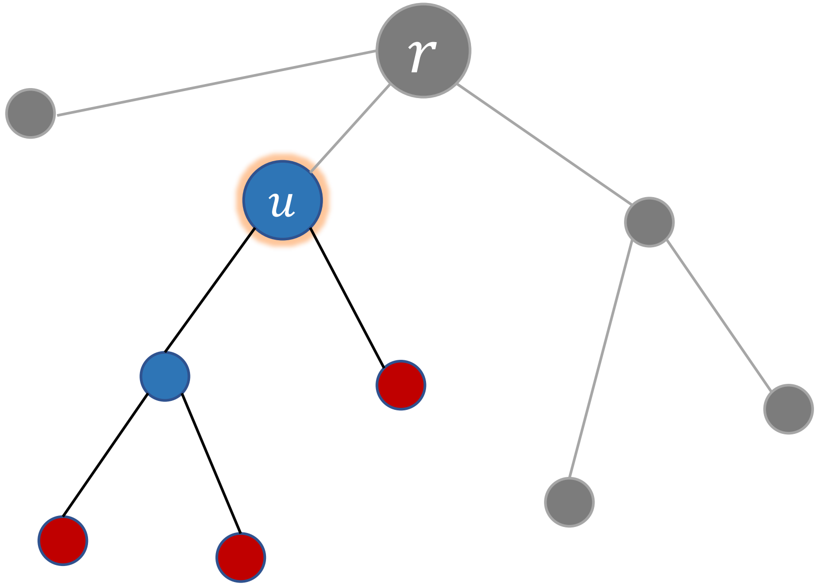
Q: What is the best way to construct I?

A: Simply choose a maximum independent I_v in tree T_v for each child v of u.

$$I = \bigcup_{v \in C} I_v$$

$$B[u] = w(I) = \sum_{v \in C} A[v]$$

where C is the set of children of u



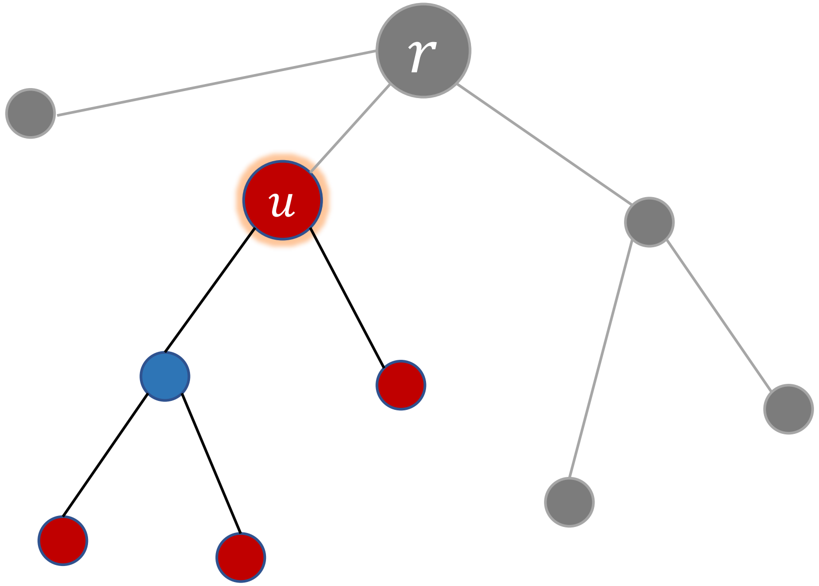
DP: Recurrence

- $u \in I$. Children of u are not in I.

Q: What is the best way to construct I?

Can we proceed the same way as before?

A: No! If we do, both endpoints of an edge (u, v) may get into I.



DP: Recurrence

- $u \in I$. Children of u are not in I.

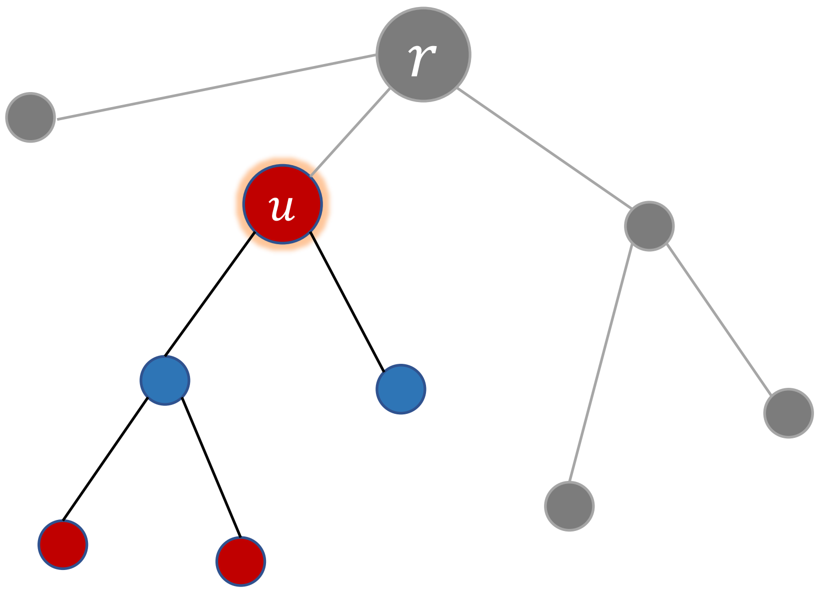
Q: What is the best way to construct I?

Can we proceed the same way as before?

A: Find maximum independent set I_v in each tree T_v for $v \in C$ s.t. $v \notin I_v$. Let

$$I = \bigcup I_v$$

$$w(I) = w_u + \sum B[v]$$



DP: Recurrence

- $u \in I$. Children of u are not in I.

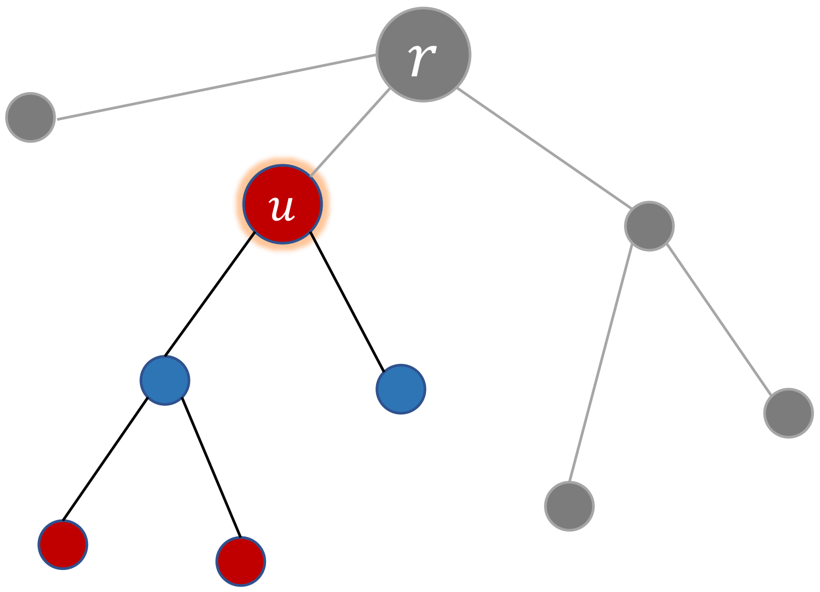
Q: What is the best way to construct I?

Can we proceed the same way as before?

A: Find maximum independent set I_v in each tree T_v for $v \in C$ s.t. $v \notin I_v$. Let

$$I = \bigcup I_v$$

$$w(I) = w_u + \sum B[v]$$



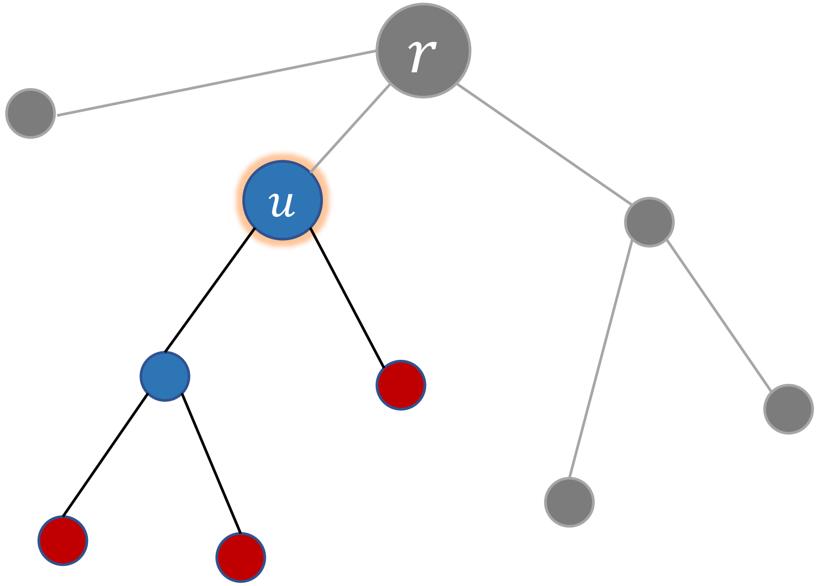
We are done!

There is only one option for $B[u]$ $(u \notin I)$

$$B[u] = \sum_{v \in C} A[v]$$

There are two options for $A[u]$ $(u \notin I \text{ or } u \in I)$

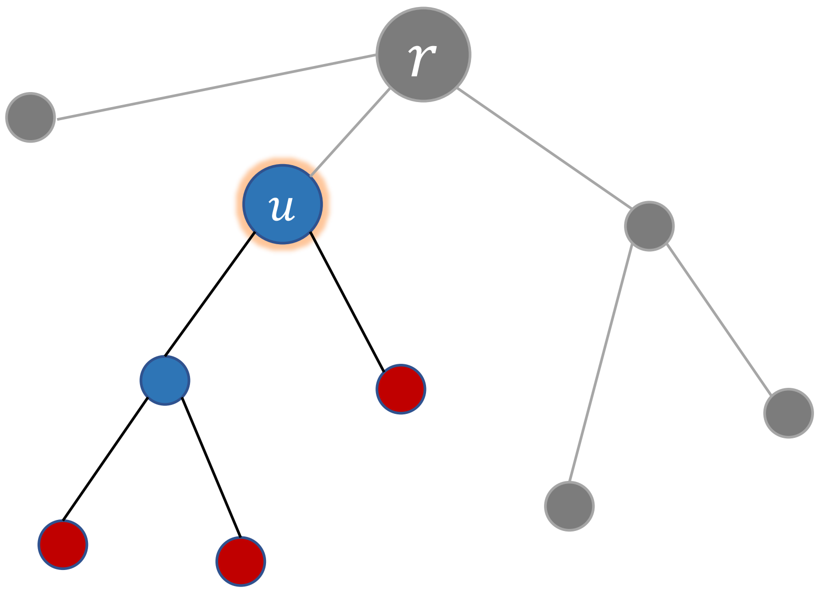
$$\mathrm{A}[u] = \max\left(\sum_{v \in C} A[v], w_u + \sum_{v \in C} B[v]\right)$$



Independent Set on Trees

Questions?

Running time?



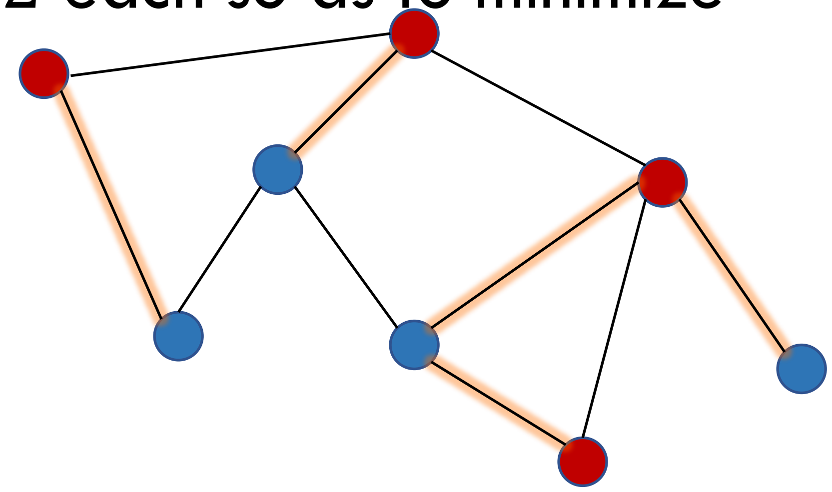
Minimum Bisection Problem

- We are given a graph $G = (V, E)$ with non-negative edge costs/weights w_e

$$n = |V| \text{ is even} \quad \text{and} \quad m = |E|$$

- Partition V into two sets L and R of size $n/2$ each so as to minimize the size / cost of the cut (L, R)

$$cost(L, R) = \sum_{\substack{(u,v)\in E \\ u\in L, v\in R}} w_{uv}$$



Minimum Bisection Problem

Minimum Bisection is an example of a large class of graph partitioning problem. They have numerous applications in practice.

The problem is NP-hard…

We can solve it on trees using DP (this class).

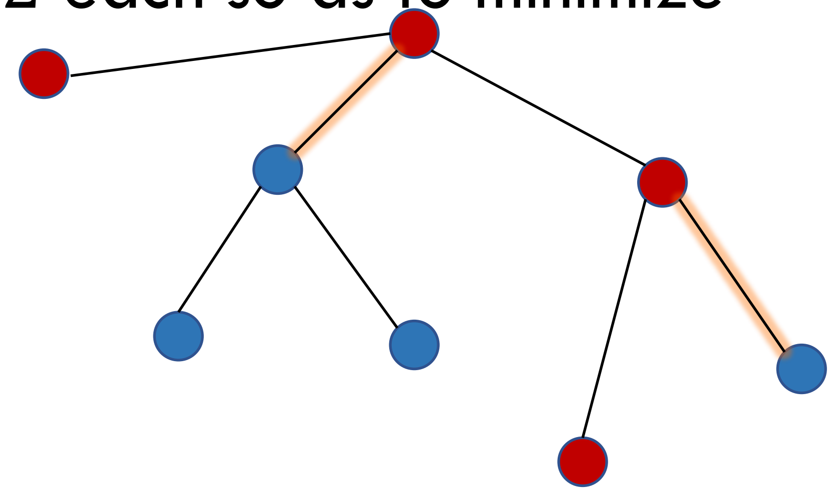
This DP yields an approximation algorithm for arbitrary graphs (TTIC 31100 and CMSC 39010-1).

Minimum Tree Bisection Problem

- We are given a tree $T = (V, E)$ with non-negative edge costs/weights w_e

- Partition V into two sets L and R of size $n/2$ each so as to minimize the size / cost of the cut (L, R)

$$cost(L, R) = \sum_{\substack{(u,v)\in E \\ u\in L, v\in R}} w_{uv}$$



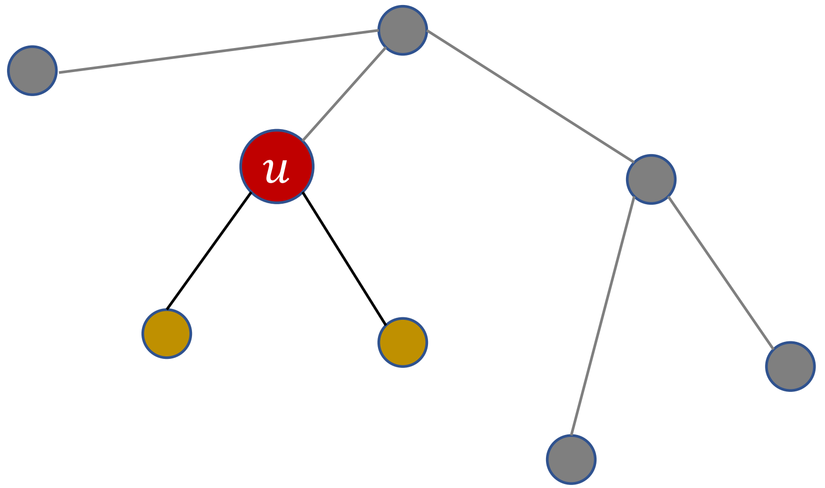
DP: Subproblems

Again: choose a root r and define subtrees T_u

Subproblem (u, Δ)

Partition T_u into L and R s.t.

- $u \in L$
- $|L| - |R| = \Delta$
- the goal is to minimize the cost of cut edges in T_u



$M_L[u, \Delta]$ is the minimum cost for (u, Δ)

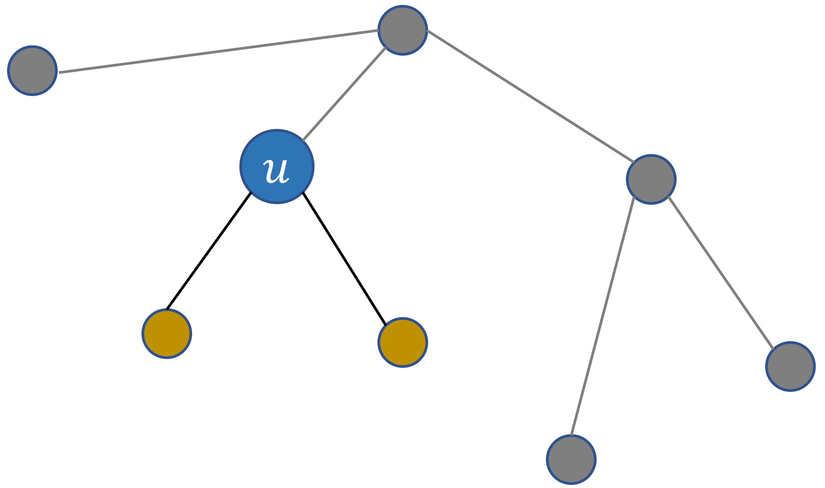
DP: Subproblems

What if we require that $u \in R$, rather than $u \in L$?

Another Subproblem (u, Δ)

Partition T_u into L and R s.t.

- $\boxed{u \in R}$
- $|L| - |R| = \Delta$
- the goal is to minimize the cost of cut edges in T_u



$M_R[u, \Delta]$ is the minimum cost for (u, Δ)

DP: Subproblems

solution for subproblem (u, Δ)

$\Downarrow$

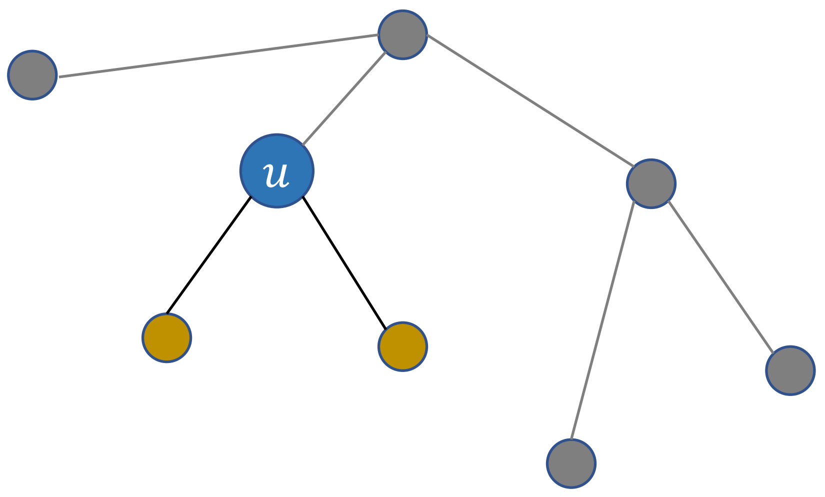
swap L and R

$\Downarrow$

Another Subproblem $(u, -\Delta)$

$$M_L[u, \Delta] = M_R[u, -\Delta]$$

It's sufficient to compute and store only $M_L[u, \Delta]$.

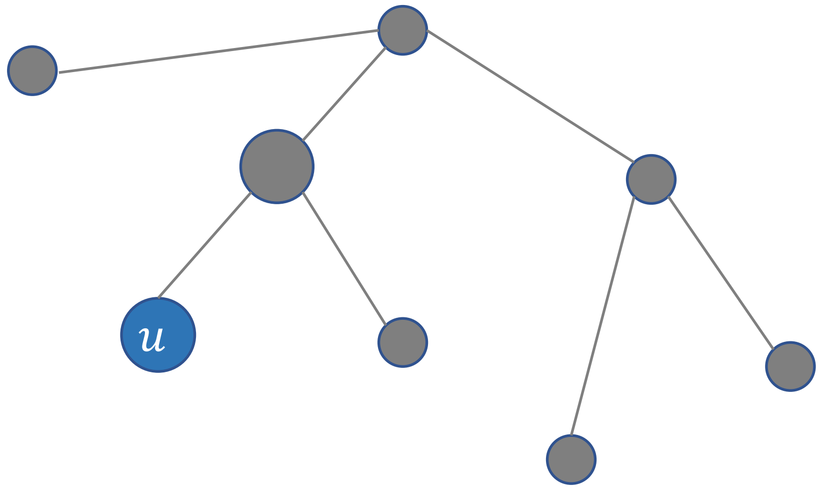


DP: Initialization

$M_L[u, \Delta] = ?$

We don't have any choice:

- $L = \{u\}$
- $R = \emptyset$
- $|L| - |R| = 1$



$M_L[u, 1] = 0$

$M_L[u, \Delta] = +\infty$ for $\Delta \neq 1$

DP: Recurrence

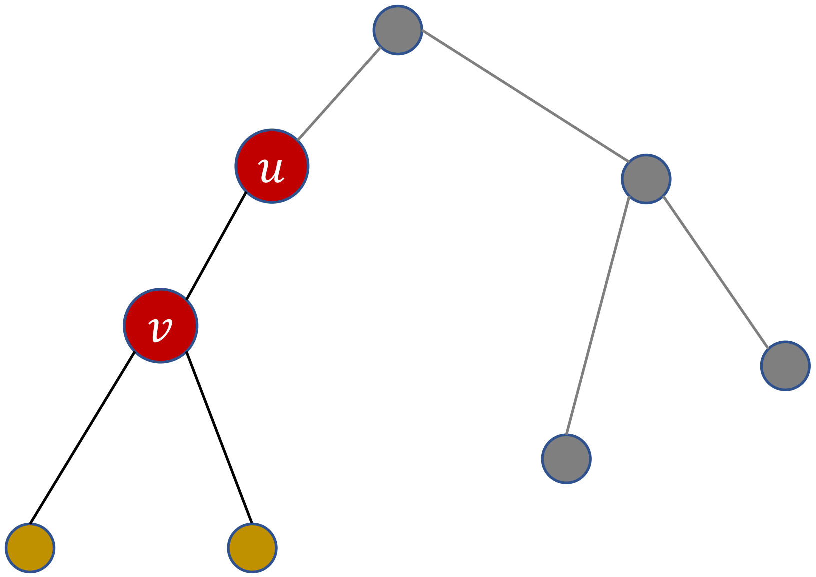
Assume that T is a binary tree

Warm Up: u has a single child v. Two options:

$$v \in L \text{ and } v \in R$$

If $v \in L$:

- edge (u, v) is not cut
- $\Delta' = |L \cap T_v| - |R \cap T_v| = |L| - 1 - |R|$
  $= \Delta - 1$
- optimal solution has cost $M_L[v, \Delta - 1]$

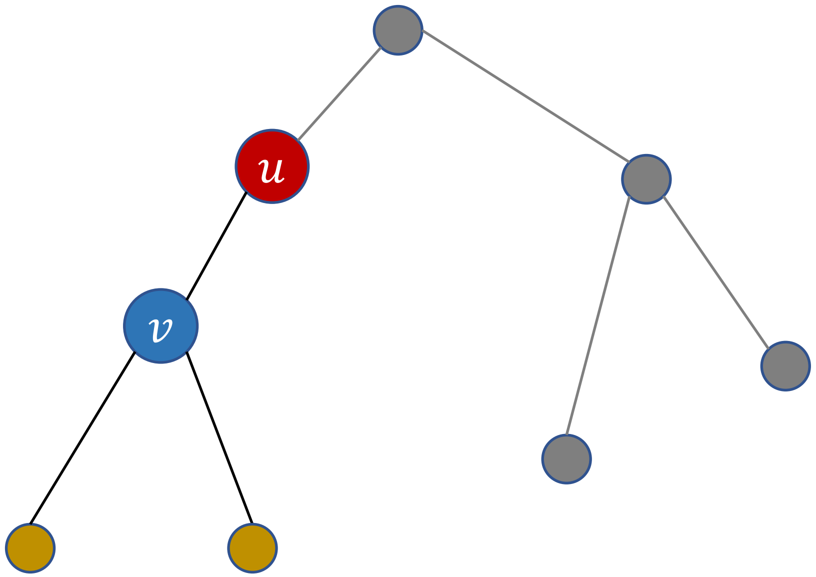


DP: Recurrence

If $v \in R$:

- edge (u, v) is cut
- $\Delta' = |L \cap T_v| - |R \cap T_v| = |L| - 1 - |R|$
  $= \Delta - 1$
- optimal solution has cost

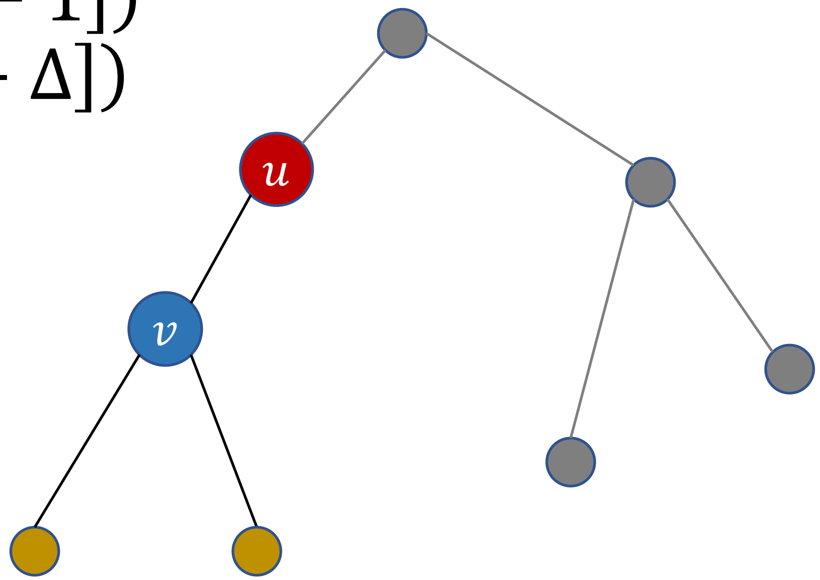
$$w_{(u,v)} + M_R[v, \Delta - 1]$$



DP: Recurrence

If u has a single child v then

$$\begin{aligned} M_L[u, \Delta] &= \min(M_L[v, \Delta - 1], w_{uv} + M_R[v, \Delta - 1]) \\ &\equiv \min(M_L[v, \Delta - 1], w_{uv} + M_L[v, 1 - \Delta]) \end{aligned}$$

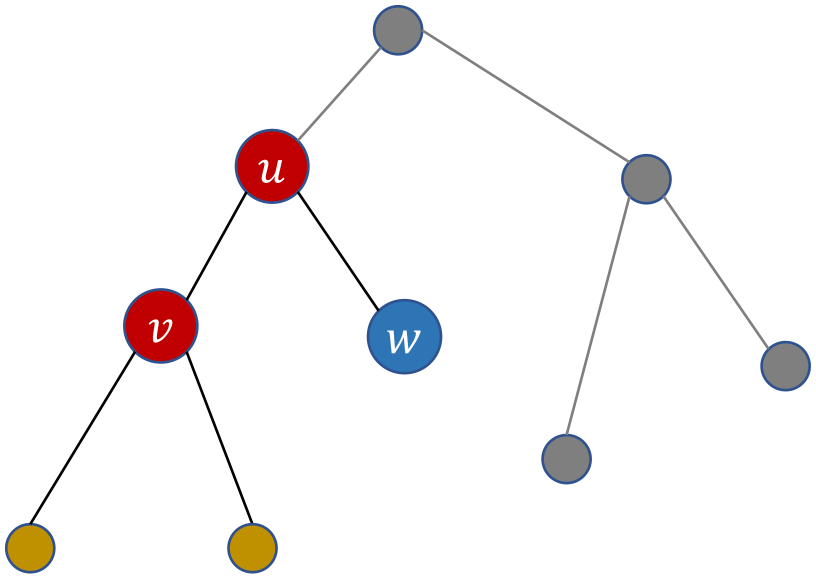


DP: Recurrence

Assume that u has two children v and w.

There are 4 cases:

	$\boldsymbol{w} \in \boldsymbol{L}$	$\boldsymbol{w} \in \boldsymbol{R}$
$\boldsymbol{v} \in \boldsymbol{L}$	I	II
$\boldsymbol{v} \in \boldsymbol{R}$	III	IV



DP: Recurrence

Assume that u has two children $v \in L$ and $w \in R$.

Assume that

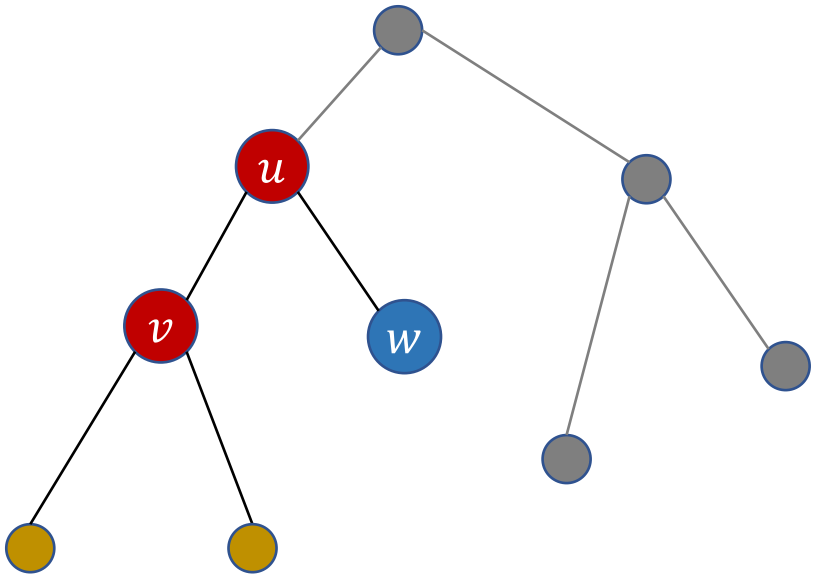
$$\Delta_v = |L \cap T_v| - |R \cap T_v|$$
$$\Delta_w = |L \cap T_w| - |R \cap T_w|$$

Subject to these assumptions:

The best partition for T_v has cost $M_L[v, \Delta_v]$

The best partition for T_w has cost $M_R[w, \Delta_w]$

Edge (u, v) is not cut, but (u, w) is cut.

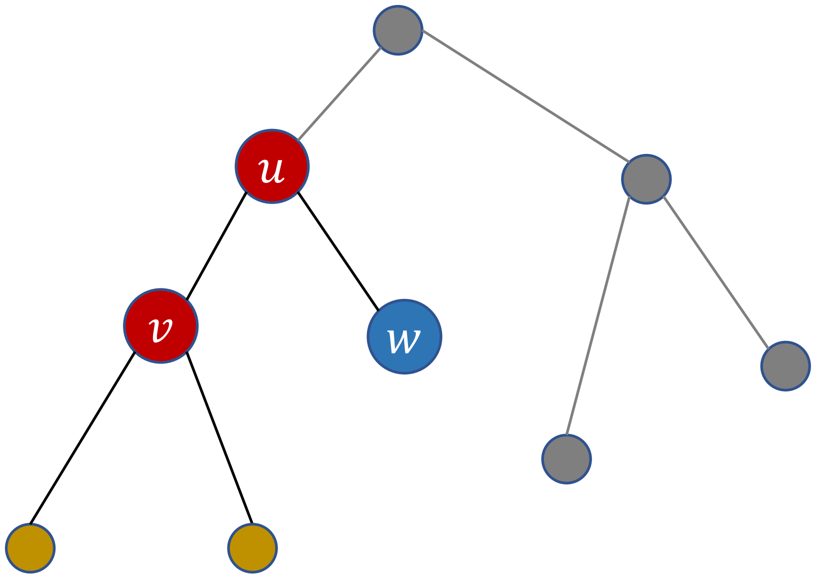


$$M_L[v, \Delta_v] + M_R[w, \Delta_w] + w_{uw}$$

DP: Recurrence

Analyze all 4 cases in the same way:

$$
\min\Big(\\
M_L[v,\Delta_v] + M_L[w,\Delta_w], \\
M_L[v,\Delta_v] + M_R[w,\Delta_w] + w_{uw}, \\
M_R[v,\Delta_v] + M_L[w,\Delta_w] + w_{uv}, \\
M_R[v,\Delta_v] + M_R[w,\Delta_w] + w_{uv} + w_{uw} \\
\Big)
$$

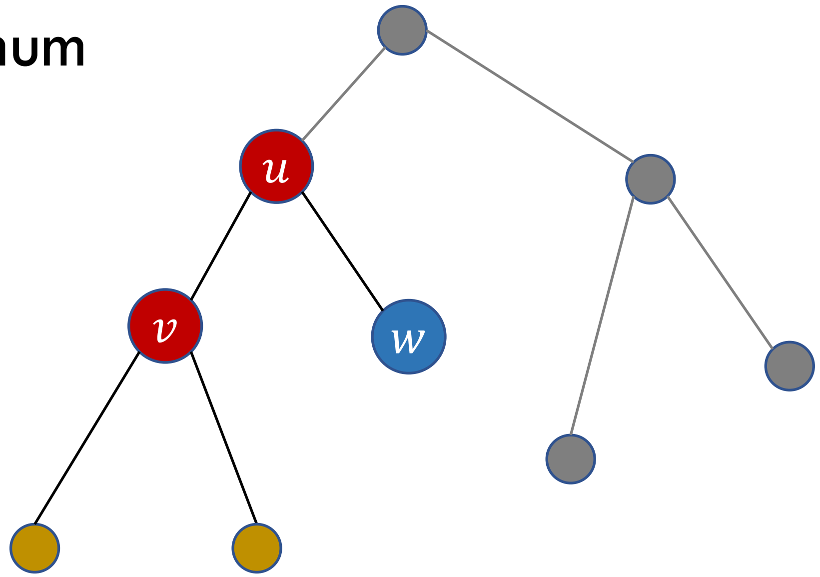


…but we don't know Δ_v and Δ_w

DP: Recurrence

$$\Delta_v = |L \cap T_v| - |R \cap T_v|$$
$$\Delta_w = |L \cap T_w| - |R \cap T_w|$$

To compute $M_L[u, \Delta]$, we need to compute the minimum of the formula we got over all possible Δ_v and Δ_w.

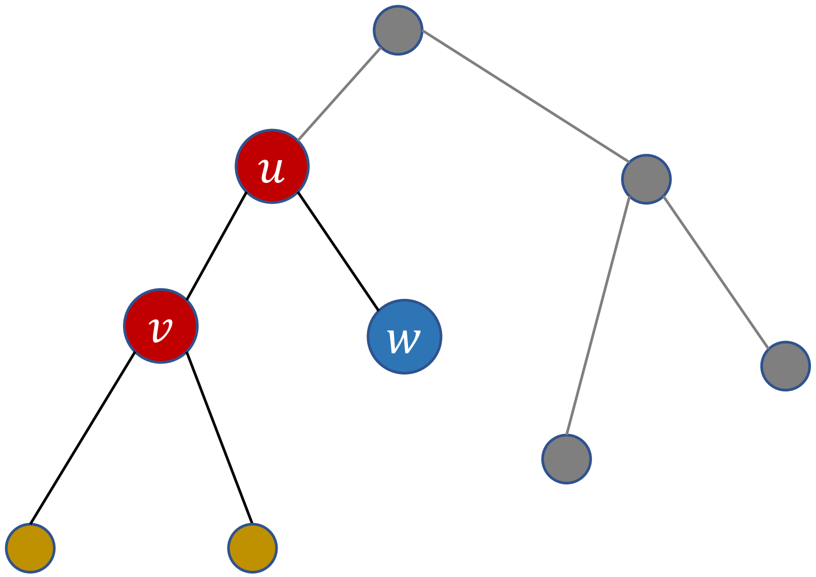


$$\begin{array}{rl} & |L| = |L \cap T_v| + |L \cap T_w| + 1 \\ - & |R| = |R \cap T_v| + |R \cap T_w| \\ \hline & \Delta = \Delta_v + \Delta_w + 1 \end{array}$$

DP: Recurrence

Compute the minimum over Δ_v and $\Delta_w \equiv \Delta - \Delta_v - 1$ of

$$\min(\\ M_L[v, \Delta_v] + M_L[w, \Delta_w],\\ M_L[v, \Delta_v] + M_R[w, \Delta_w] + w_{uw},\\ M_R[v, \Delta_v] + M_L[w, \Delta_w] + w_{uv},\\ M_R[v, \Delta_v] + M_R[w, \Delta_w] + w_{uv} + w_{uw}\\ )$$



Running time

We obtain an algorithm for binary trees.

Find running time.

- Table $M_L[u, \Delta]$ has $O(n \times n)$ entries, since $\Delta \in \{-n, \dots, n\}$
- To compute $M_L[u, \Delta]$ we go over all values of Δ_v.
  Perform $O(n)$ iterations.

Total running time: $O(n^3)$

Total memory: $O(n^2)$

Algorithm for Binary Trees

Questions?

Arbitrary Trees

Assume that u has k children: $v_1, \ldots, v_k$.

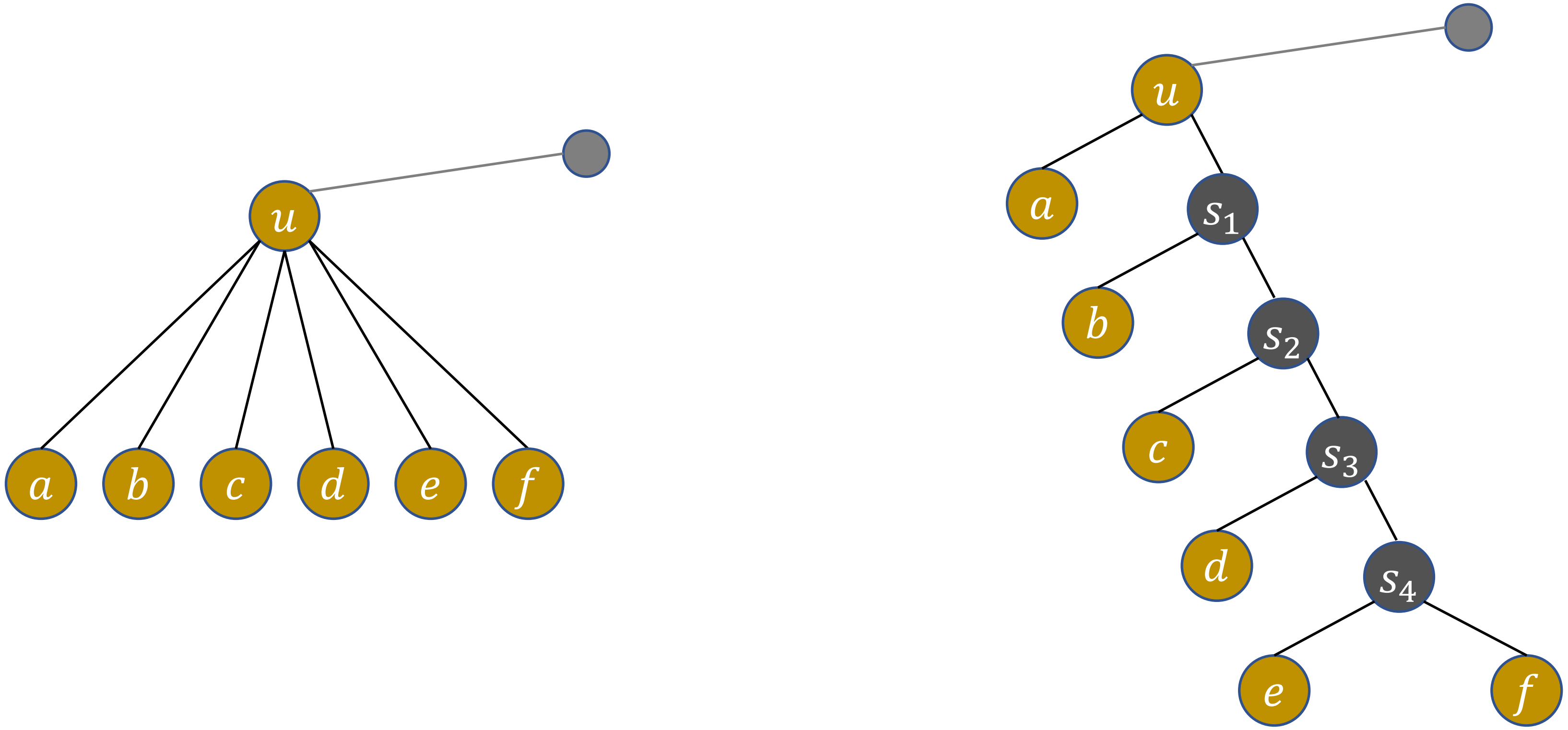
Consider all possible cases.

$$v_1 \in L, \ldots, v_k \in L$$
$$\ldots$$
$$v_1 \in L, \ldots, v_k \in R$$
$$\ldots$$
$$v_1 \in R, \ldots, v_k \in R$$

Wait! Are we in trouble? We have 2^k cases instead of 4.

Reduction to a Binary Tree

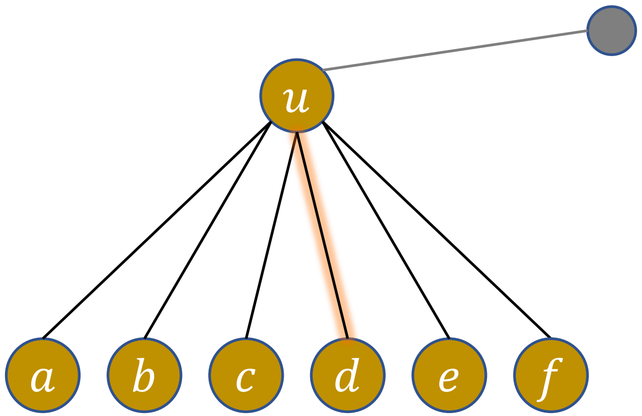
Transform our tree to a binary tree with Steiner vertices as follows:

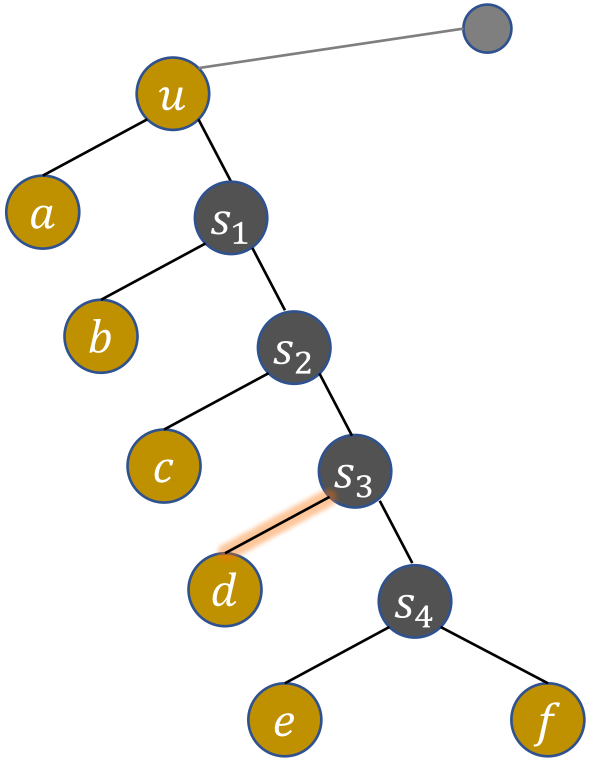


Process each vertex of degree $k > 2$. Add $k - 2$ Steiner vertices: $s_1, \dots, s_{k-2}$

Reduction to a Binary Tree

Transform our tree to a binary tree with Steiner vertices as follows:

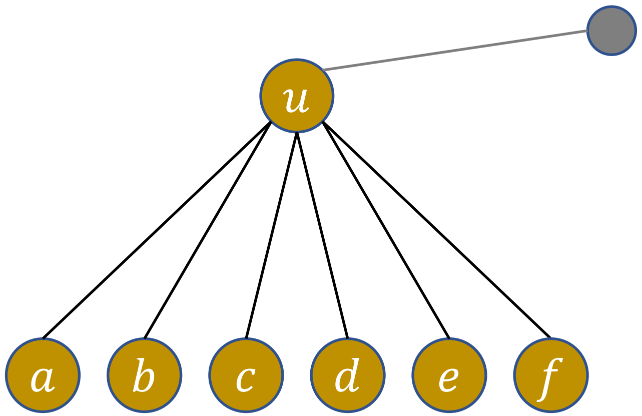


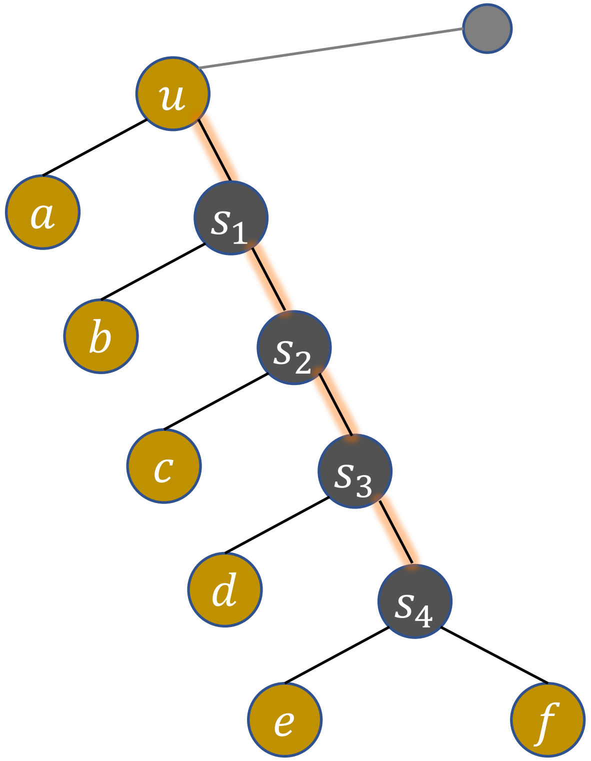


Keep the original edges.

Reduction to a Binary Tree

Transform our tree to a binary tree with Steiner vertices as follows:

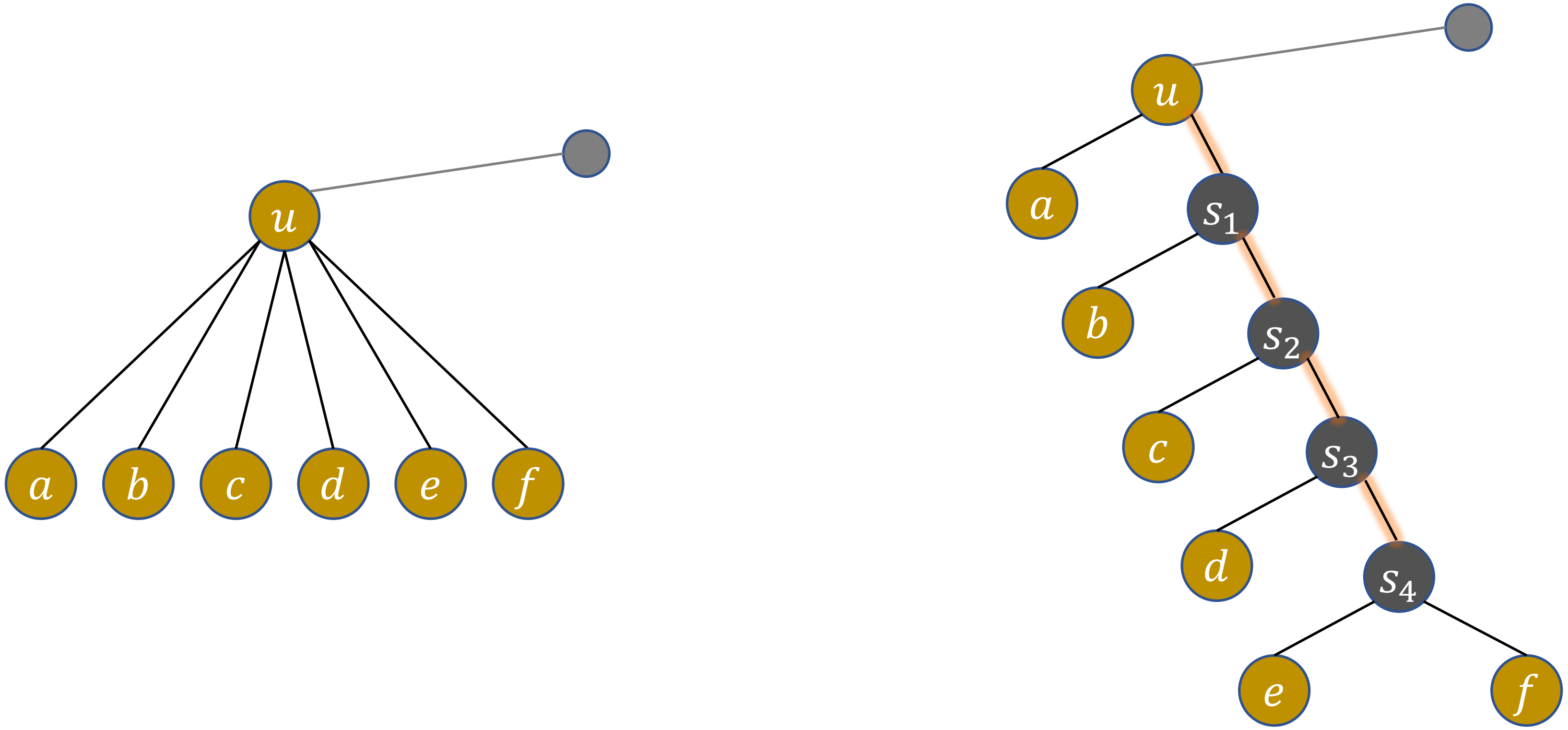




Assign ∞ weight to edges (u, s_1) and between Steiner vertices.

Reduction to a Binary Tree

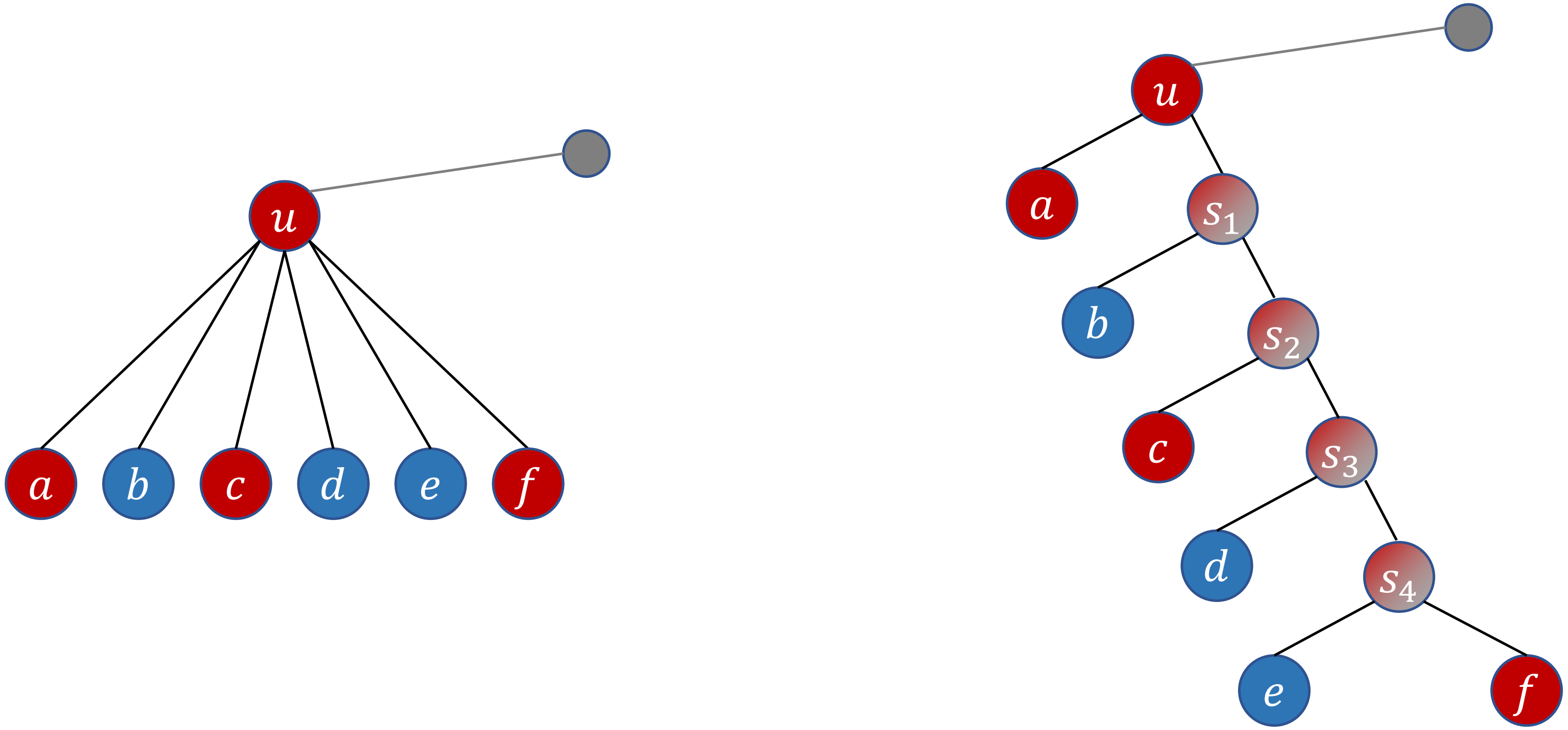
Transform our tree to a binary tree with Steiner vertices as follows:



$s_1, \ldots, s_{k-2}$ must be in the same set L or R as u: otherwise, the cost is infinite

Reduction to a Binary Tree

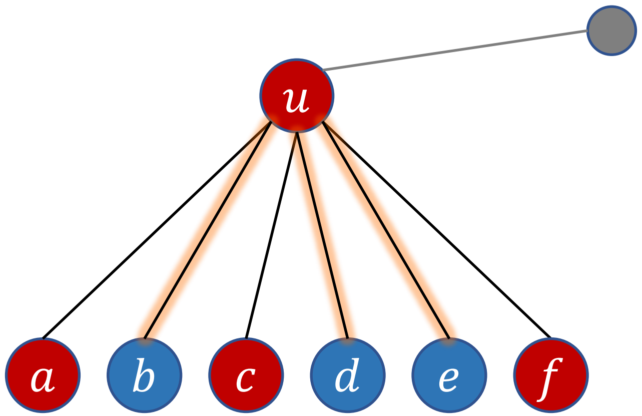
Transform T to a binary tree T' with Steiner vertices as follows:

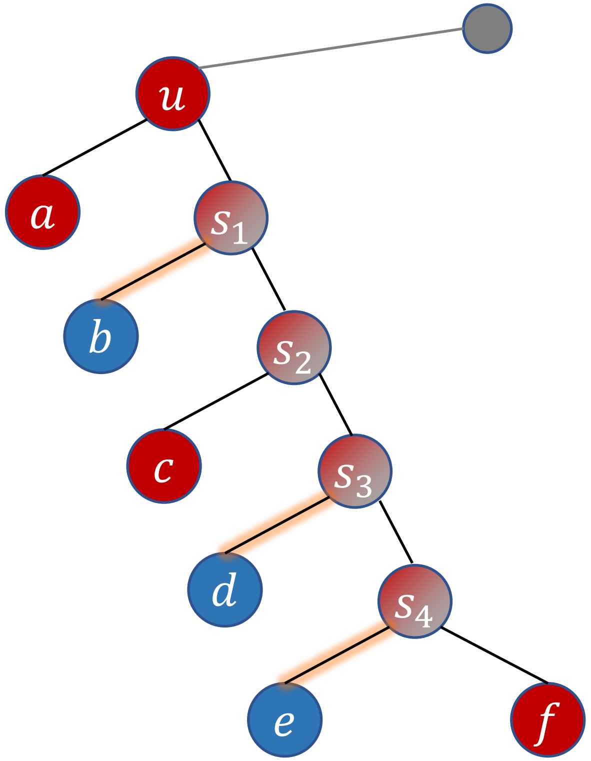


There is a one-to-one correspondence between partitions of T and T'

Reduction to a Binary Tree

Transform T to a binary tree T' with Steiner vertices as follows:

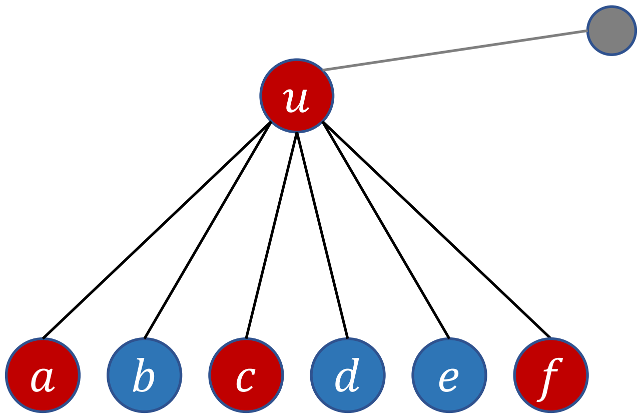


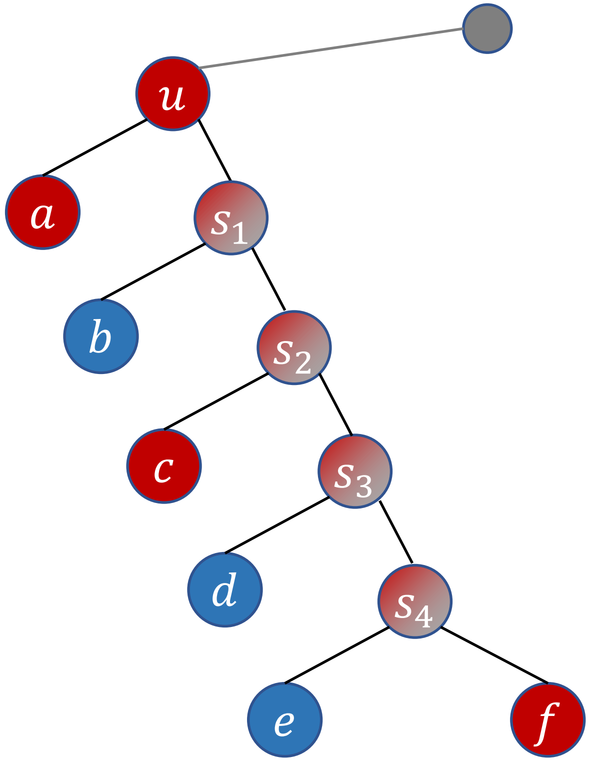


The costs of cut edges are the same.

Reduction to a Binary Tree

Transform T to a binary tree T' with Steiner vertices as follows:





Don’t count Steiner vertices when we compute $|L| - |R|$.

Reduction to a Binary Tree

We reduced the problem in arbitrary trees to a problem in Steiner trees where the balanceness requirement is:

$$|L \setminus S| - |R \setminus S| = \Delta$$

where S is the set of Steiner vertices.

DP: New Recurrence

Compute the minimum over Δ_v and

$$\Delta_w = \begin{cases} \Delta - \Delta_v - 1 & \text{if } u \notin S \\ \Delta - \Delta_v & \text{if } u \in S \end{cases}$$

of

$$\min(\\ M_L[v, \Delta_v] + M_L[w, \Delta_w], \\ M_L[v, \Delta_v] + M_R[w, \Delta_w] + w_{uw}, \\ M_R[v, \Delta_v] + M_L[w, \Delta_w] + w_{uv}, \\ M_R[v, \Delta_v] + M_R[w, \Delta_w] + w_{uv} + w_{uw} \\ )$$