More on Dynamic Programming

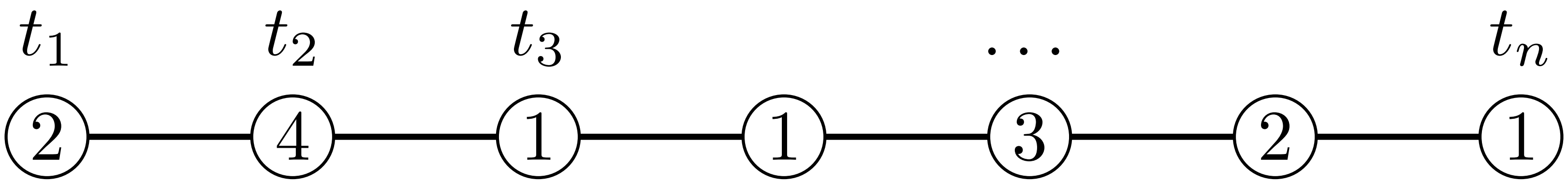
Drink as Much as Possible: A Variant

Robert still wants to drink as much a possible.

- Robert walks through the streets of King's Landing and encounters n taverns $t_1, t_2, \ldots, t_n$, in order
- When Robert encounters a tavern t_i, he can either stop for a drink or continue walking.
- Tavern t_i has $w_i \in \mathbb{N}$ liters of wine.
- If Robert drinks in tavern t_i then he will be too drunk to drink in tavern t_{i+1}. He will be able to drink again by the time he reaches t_{i+2}
- **Goal:** Compute the maximum amount of wine (in liters) Roberts can drink

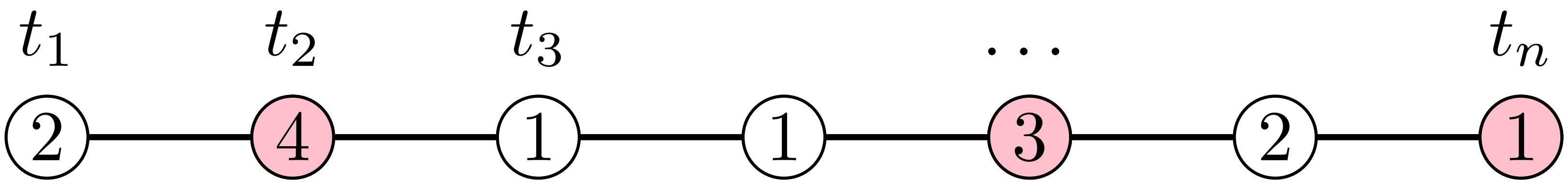
Max-Weight Independent Set on Paths

Definition: An *independent set* (IS) of a graph $G = (V, E)$ is a set $\mathcal{I} \subseteq V$ such that $\forall (u, v) \in E,\ u \notin \mathcal{I}$ or $v \notin \mathcal{I}$.



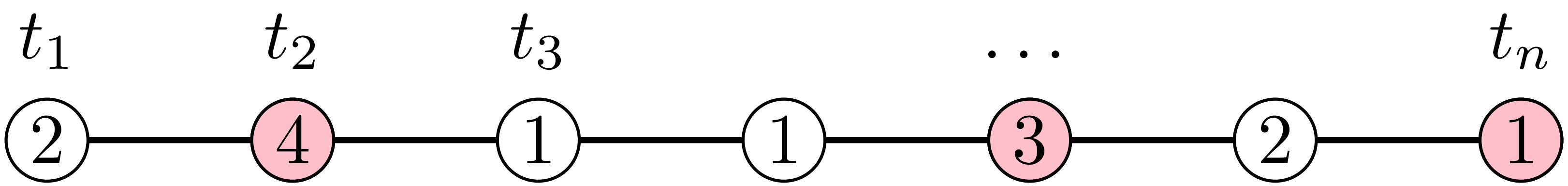
Max-Weight Independent Set on Paths

Definition: An *independent set* (IS) of a graph $G = (V, E)$ is a set $\mathcal{I} \subseteq V$ such that $\forall (u, v) \in E,\ u \notin \mathcal{I}$ or $v \notin \mathcal{I}$.



Max-Weight Independent Set on Paths

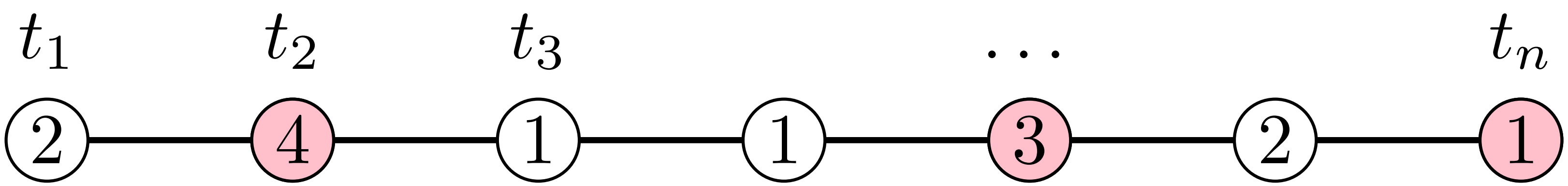
Definition: An *independent set* (IS) of a graph $G = (V, E)$ is a set $\mathcal{I} \subseteq V$ such that $\forall (u, v) \in E$, $u \notin \mathcal{I}$ or $v \notin \mathcal{I}$.



Linear-Time Dynamic Programming Algorithm

Max-Weight Independent Set on Paths

Definition: An *independent set* (IS) of a graph $G = (V, E)$ is a set $\mathcal{I} \subseteq V$ such that $\forall (u, v) \in E,\ u \notin \mathcal{I}$ or $v \notin \mathcal{I}$.



Linear-Time Dynamic Programming Algorithm

Sketch of the algorithm:

$OPT[i] =$ Maximum-weight IS w.r.t. the subpath $t_1, \dots, t_i$

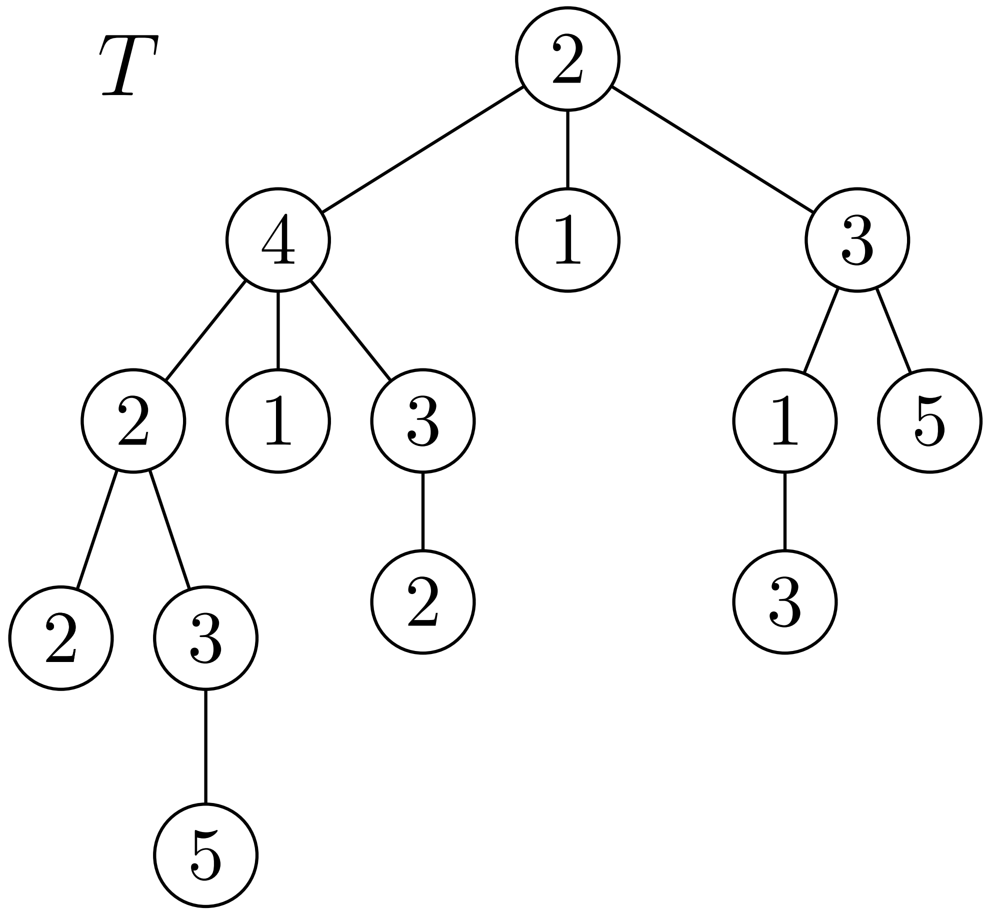
$$OPT[0] = 0 \qquad OPT[1] = w_1$$

$$OPT[i] = \max\{w_i + OPT[i-2], OPT[i-1]\}$$

Max-Weight Independent Set on Trees

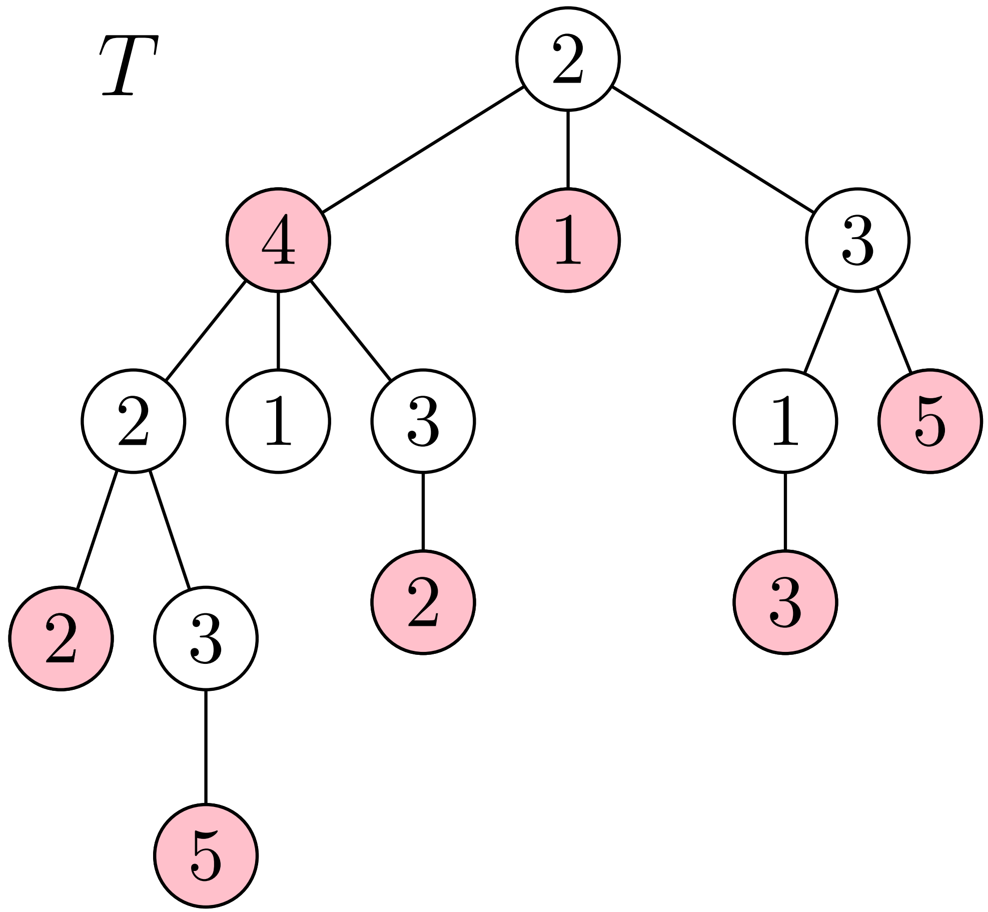
Max-Weight Independent Set on Trees

Problem: Given a tree T with integer weights on its vertices, compute the weight of a Max-Weight IS of T.



Max-Weight Independent Set on Trees

Problem: Given a tree T with integer weights on its vertices, compute the weight of a Max-Weight IS of T.



Max-Weight Independent Set on Trees

Given $v \in V(T)$, let T_v be the subtree of T rooted at v, and let $w(v)$ be the *weight* of v.

Subproblems:

• $OPT^+[v] =$ Weight of a maximum-weight IS of T_v with the constraint that v must belong to the IS.

• $OPT^-[v] =$ Weight of a maximum-weight IS of T_v with the constraint that v must *not* belong to the IS.

• $OPT[v] = \max\{OPT^+[v], OPT^-[v]\}$.

Max-Weight Independent Set on Trees

Given $v \in V(T)$, let T_v be the subtree of T rooted at v, and let $w(v)$ be the *weight* of v.

Subproblems:

• $OPT^+[v] =$ Weight of a maximum-weight IS of T_v with the constraint that v must belong to the IS.

• $OPT^-[v] =$ Weight of a maximum-weight IS of T_v with the constraint that v must *not* belong to the IS.

• $OPT[v] = \max\{OPT^+[v], OPT^-[v]\}$.

Base case: If v is a leaf in T, then:

• $OPT^+[v] = w(v)$, and
• $OPT^-[v] = 0$

Max-Weight Independent Set on Trees

Recursive formula(s):

Let $C(v)$ be set of the children of v in T.

- $OPT^+[v] = w(v) + \sum_{u \in C(v)} OPT^-[u]$.
- $OPT^-[v] = \sum_{u \in C(v)} OPT[u] = \sum_{u \in C(v)} \max\{OPT^+[u], OPT^-[u]\}$.

Max-Weight Independent Set on Trees

Recursive formula(s):

Let $C(v)$ be set of the children of v in T.

- $OPT^+[v] = w(v) + \sum_{u \in C(v)} OPT^-[u]$.
- $OPT^-[v] = \sum_{u \in C(v)} OPT[u] = \sum_{u \in C(v)} \max\{OPT^+[u], OPT^-[u]\}$.

Optimal solution:

- $OPT[r] = \max\{OPT^+[r], OPT^-[r]\}$, where r is the root of T

Max-Weight Independent Set on Trees

Recursive formula(s):

Let $C(v)$ be set of the children of v in T.

- $OPT^+[v] = w(v) + \sum_{u \in C(v)} OPT^-[u]$.

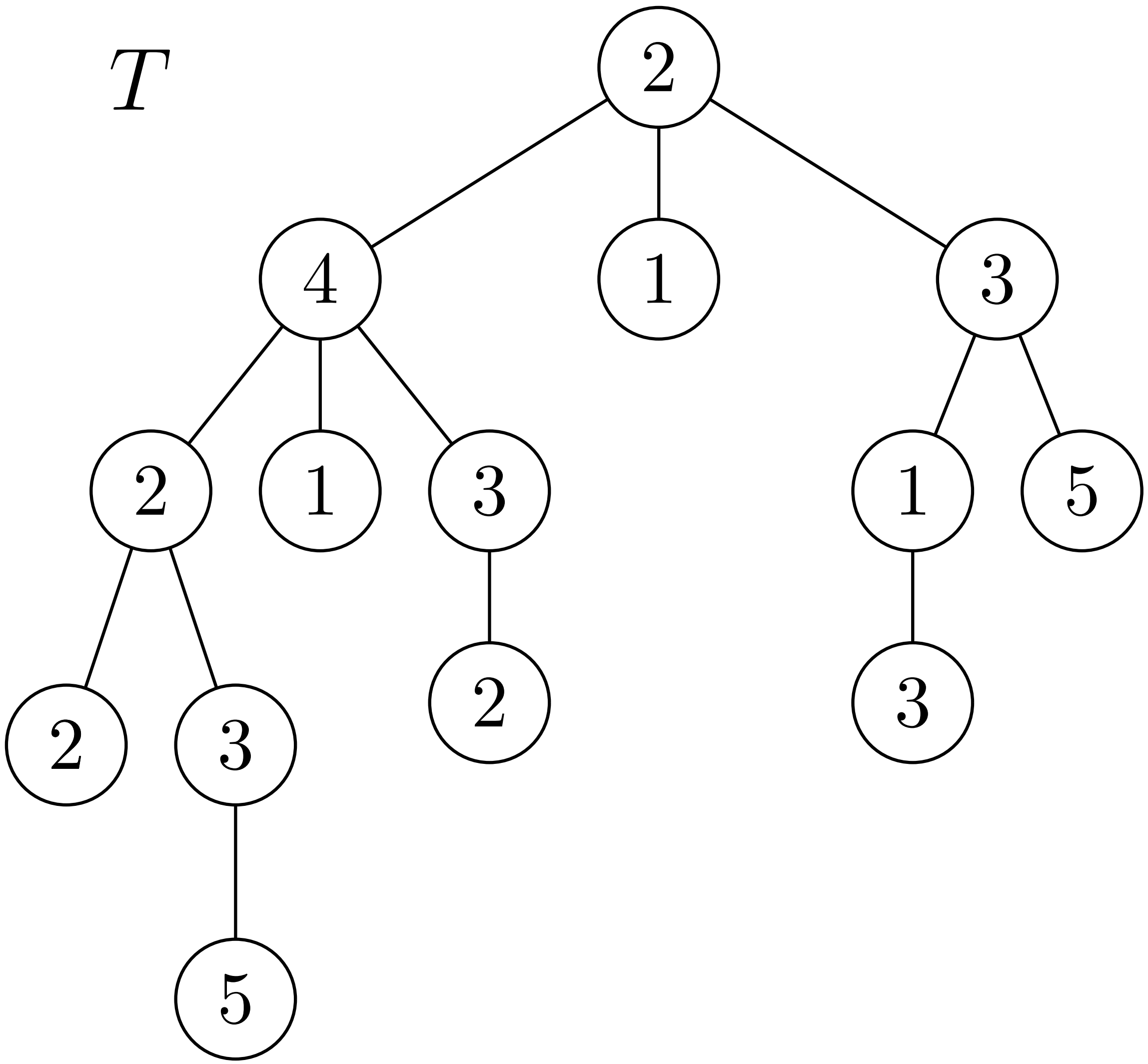
- $OPT^-[v] = \sum_{u \in C(v)} OPT[u] = \sum_{u \in C(v)} \max\{OPT^+[u], OPT^-[u]\}$.

Optimal solution:

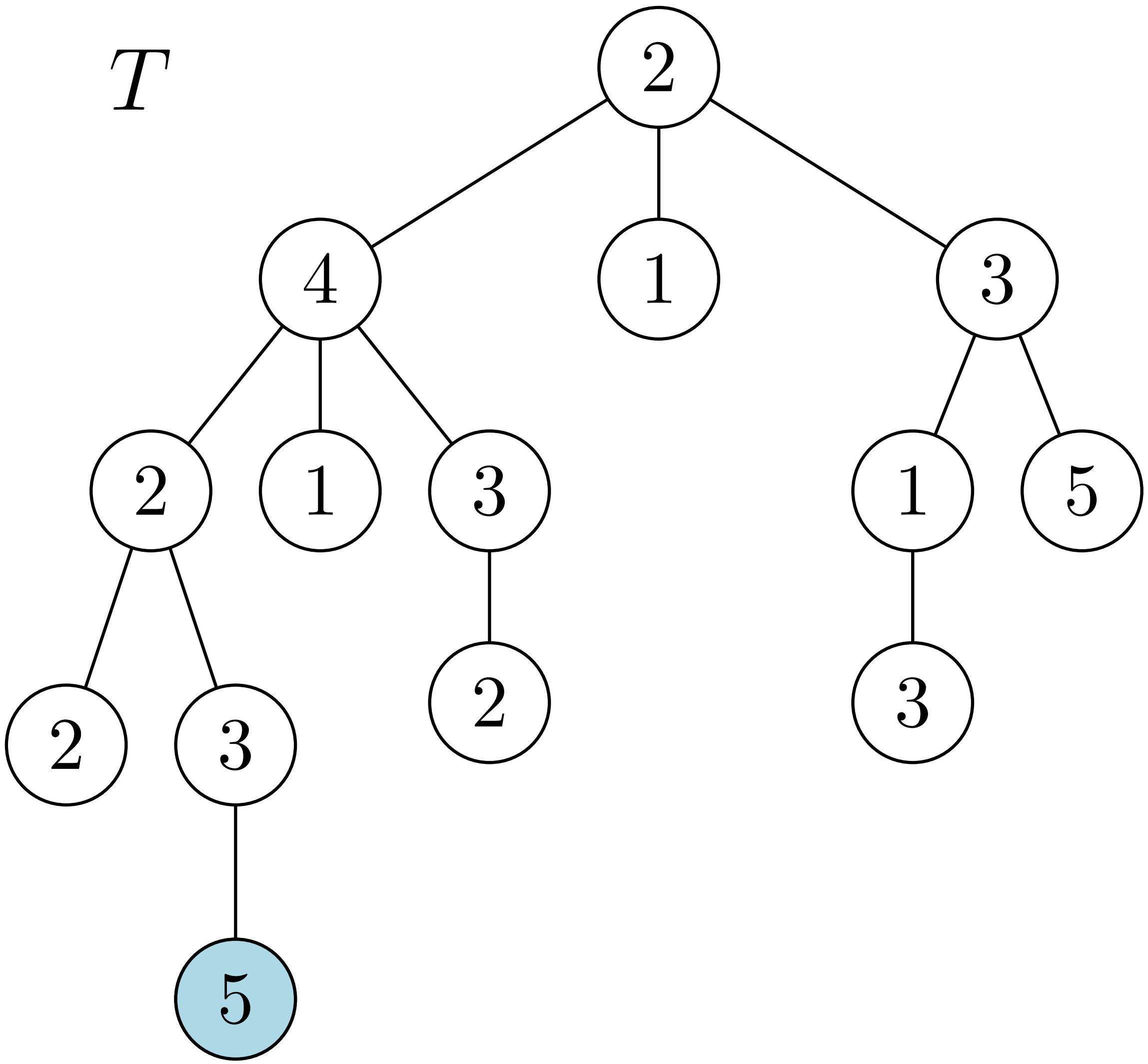
- $OPT[r] = \max\{OPT^+[r], OPT^-[r]\}$, where r is the root of T

Order of subproblems: ?

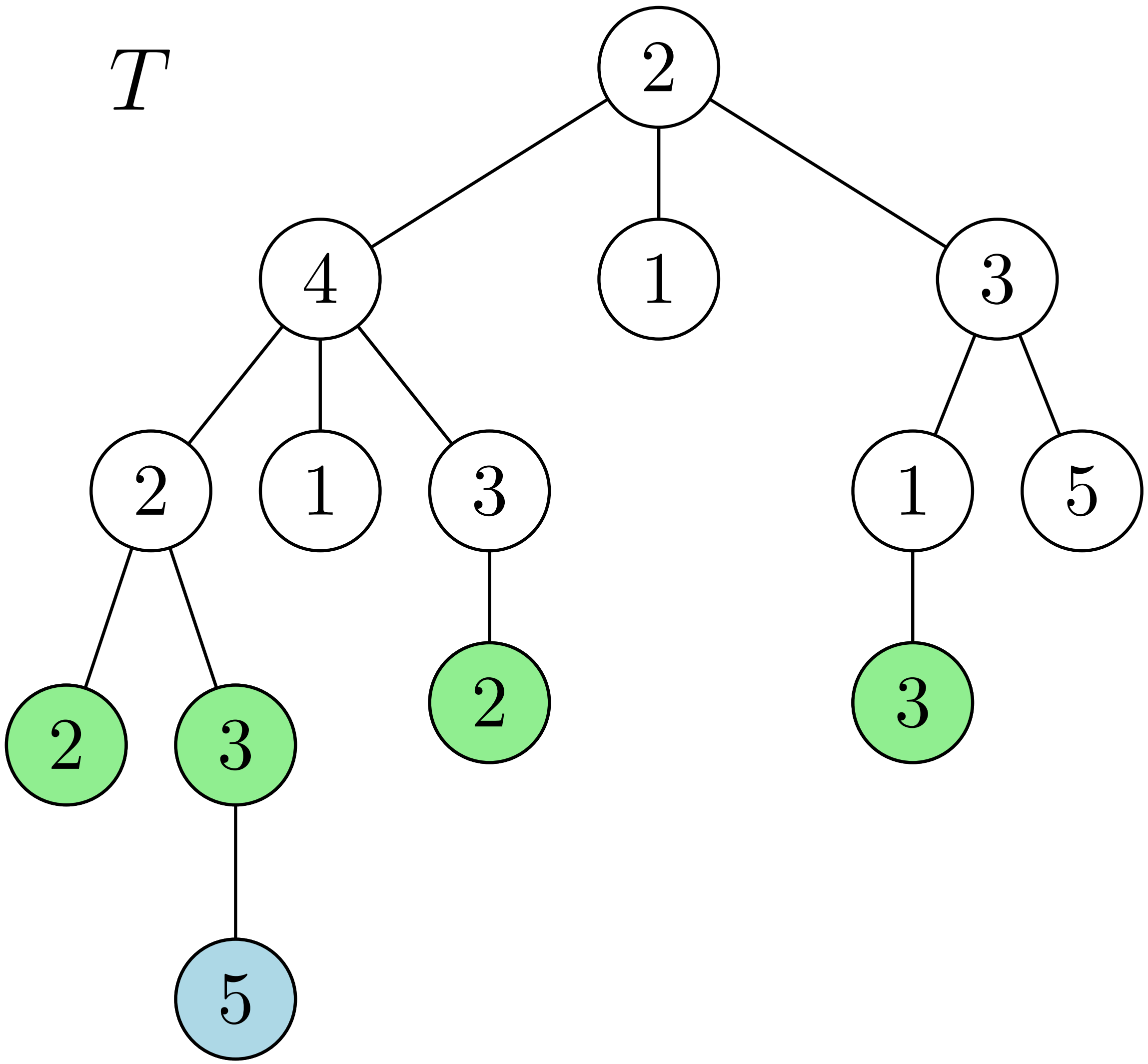
Order of Subproblems



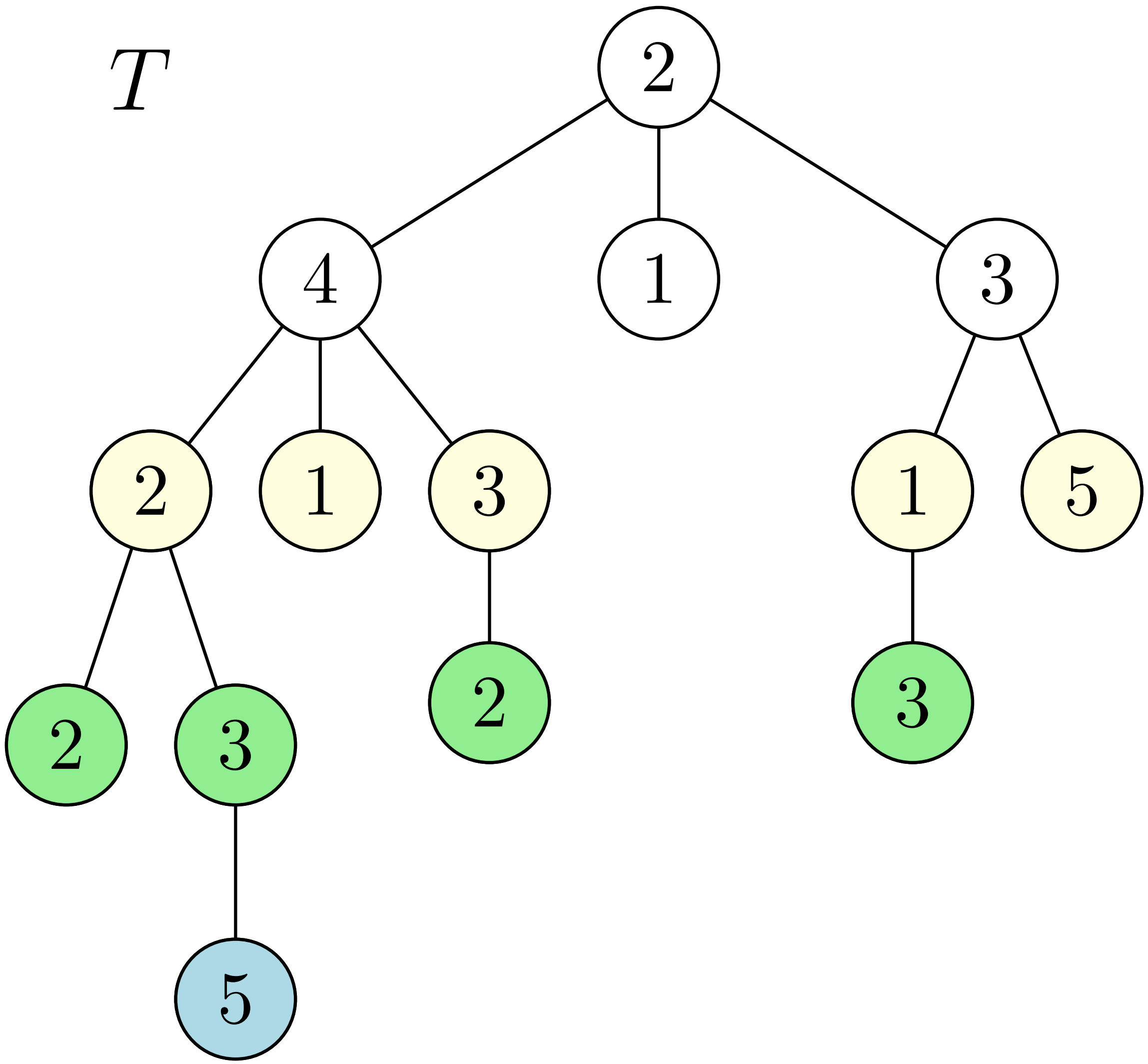
Order of Subproblems



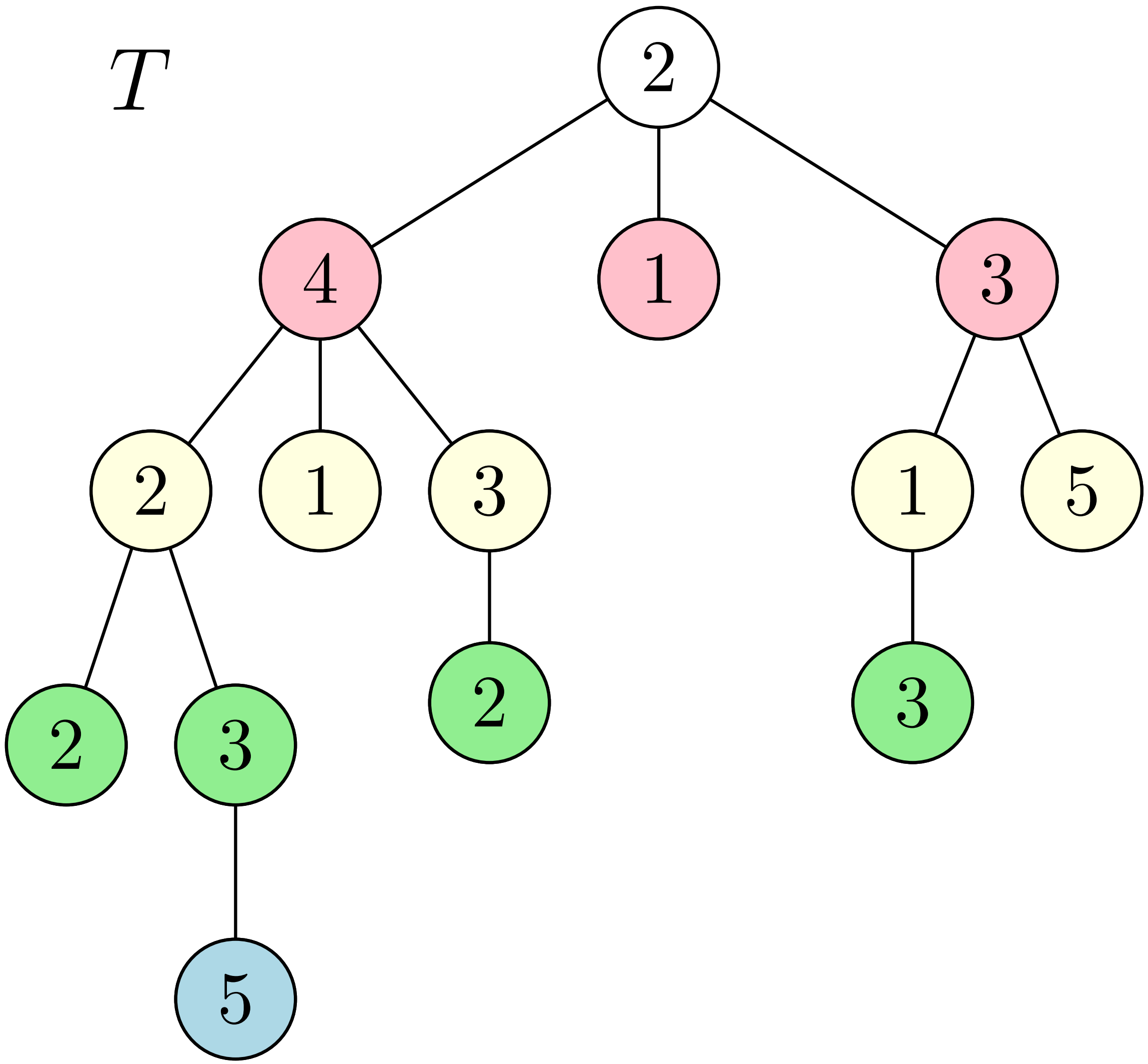
Order of Subproblems



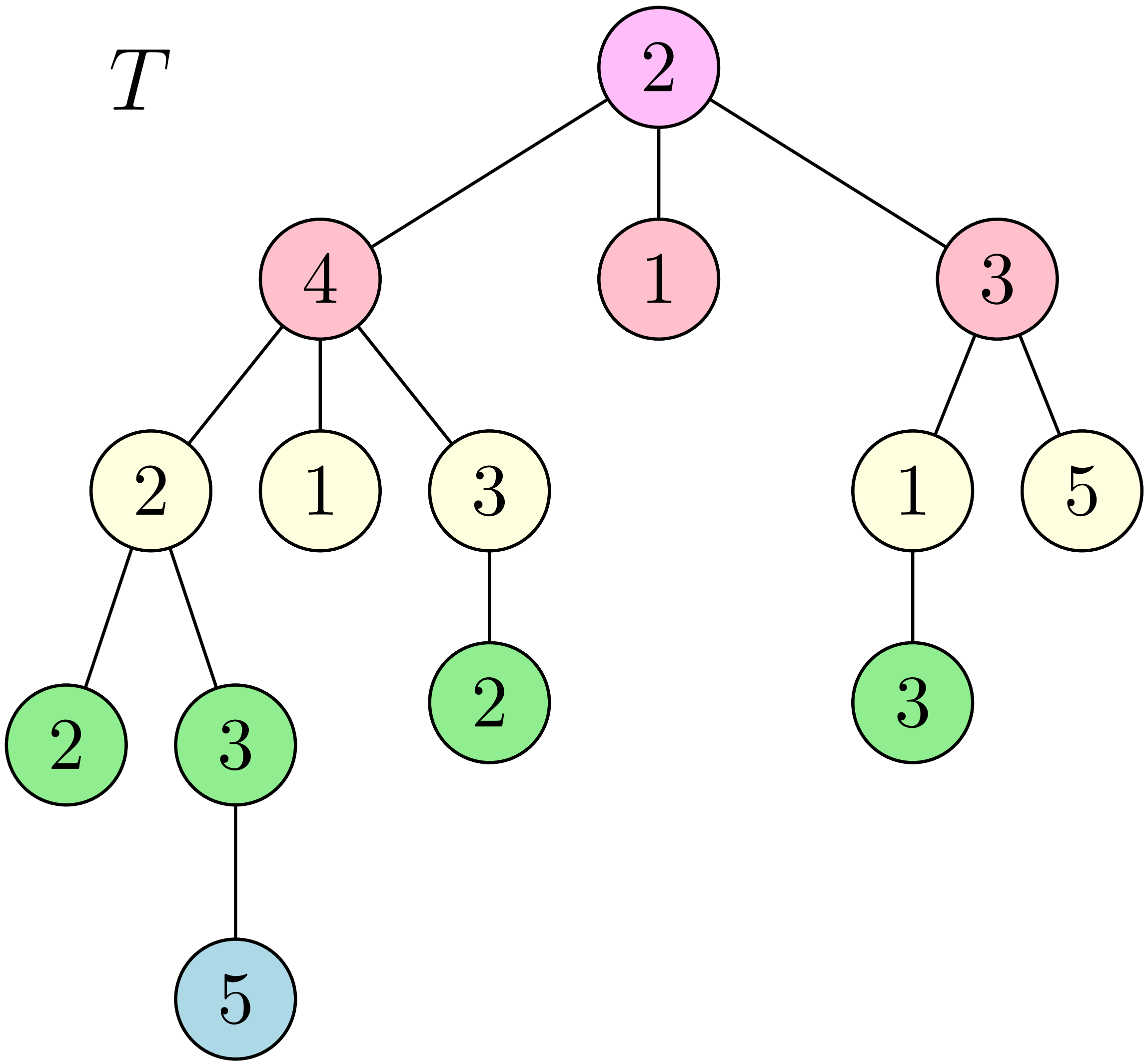
Order of Subproblems



Order of Subproblems

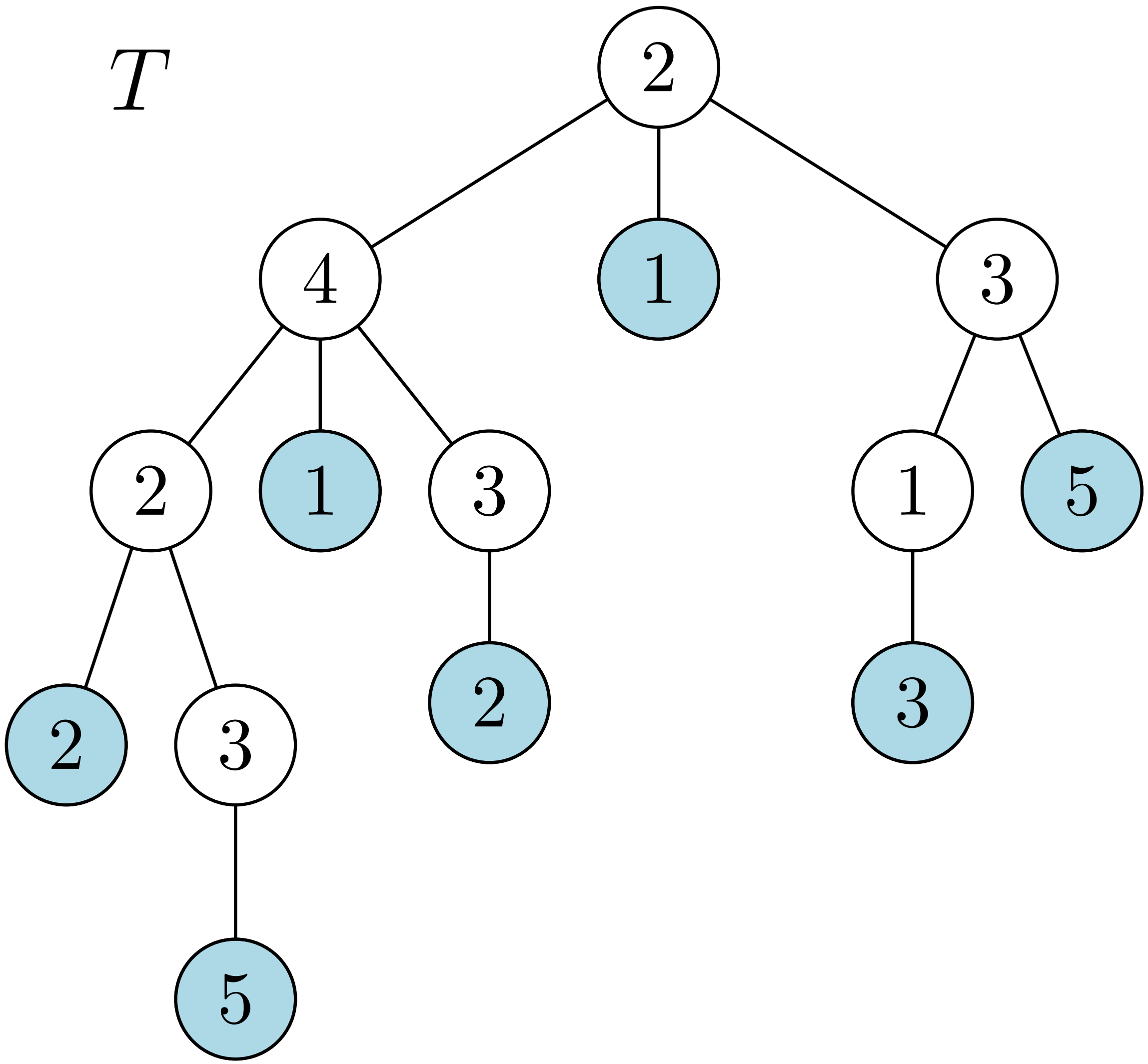


Order of Subproblems

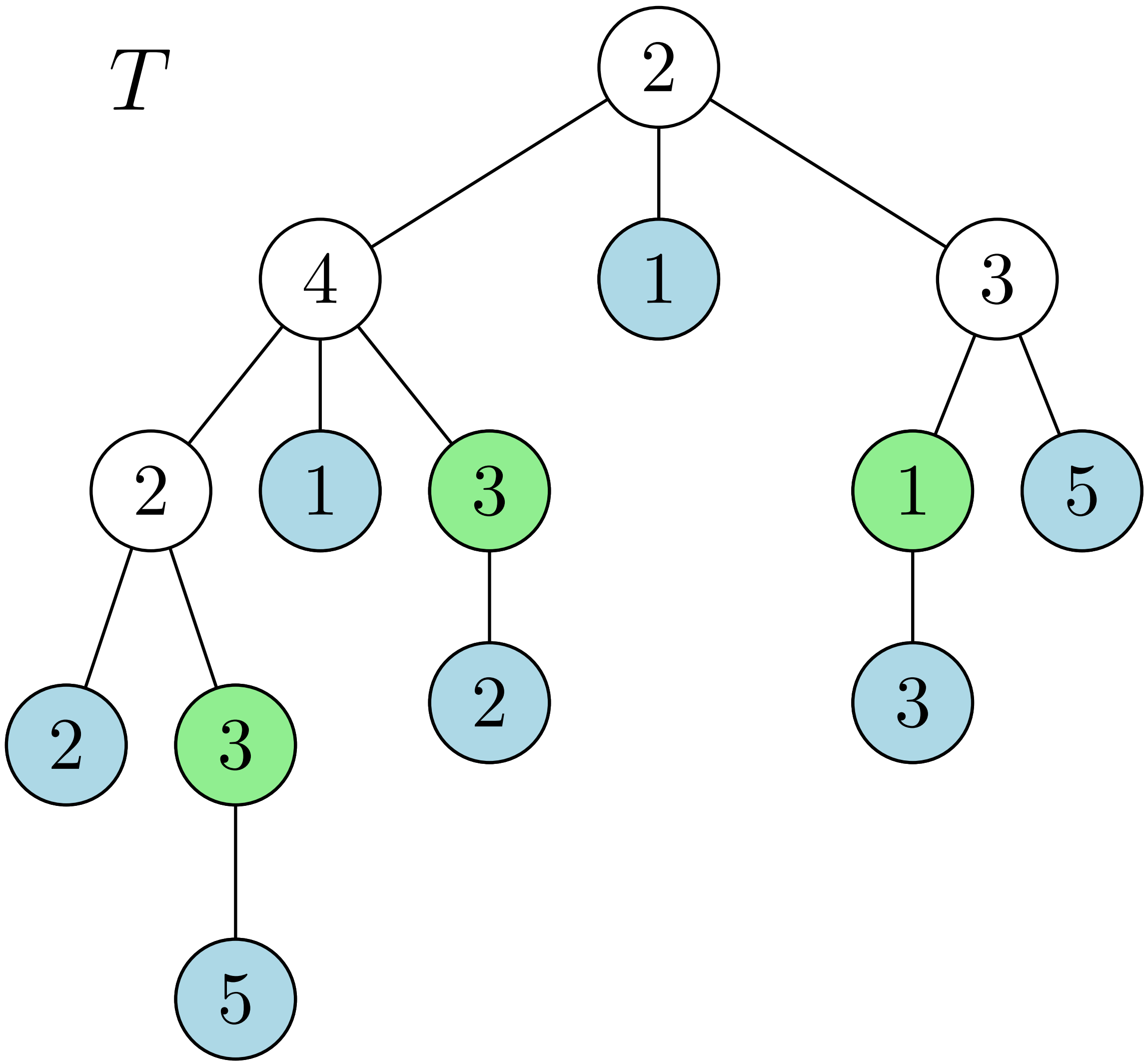


In order of decreasing depth in T

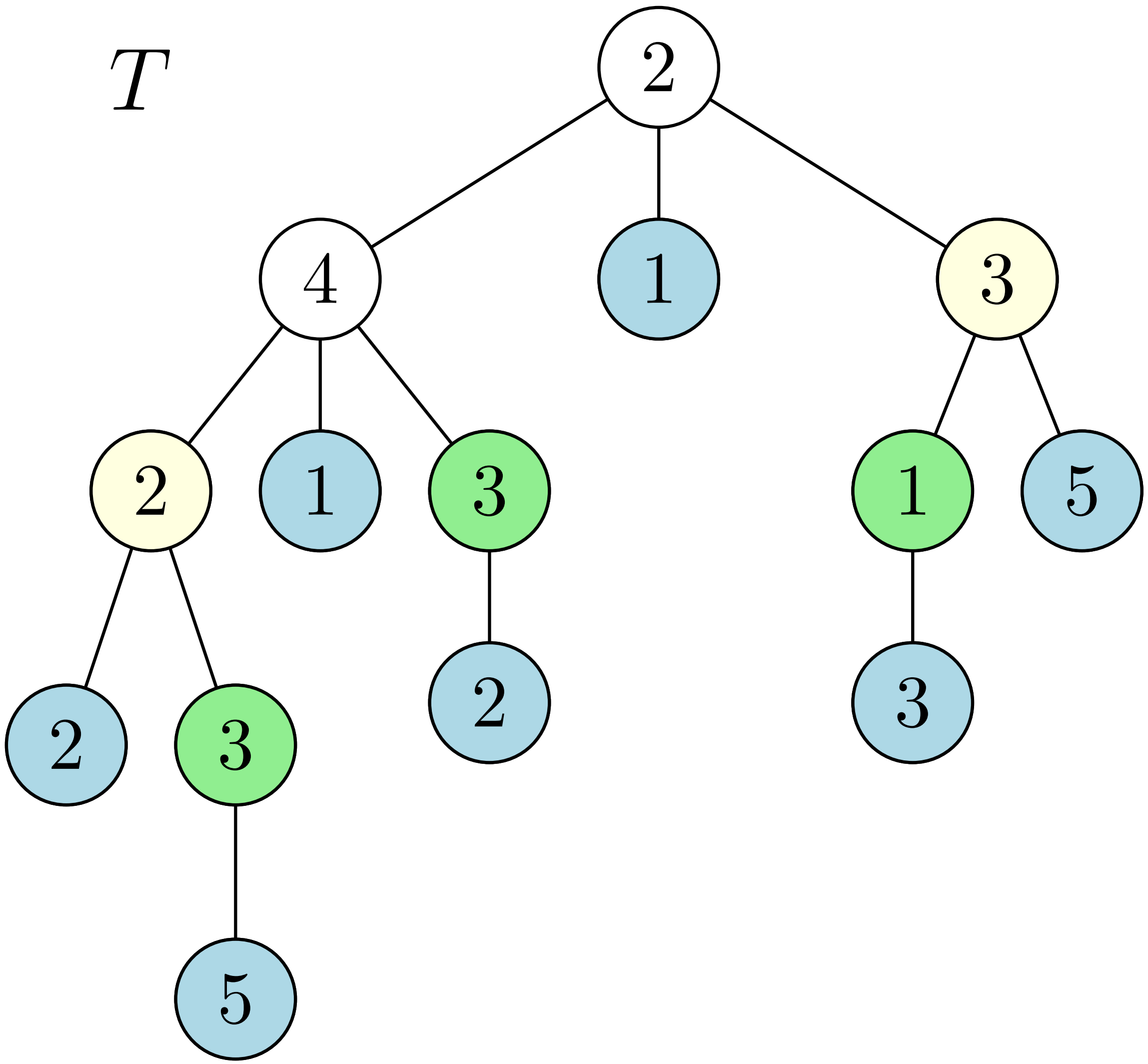
Order of Subproblems



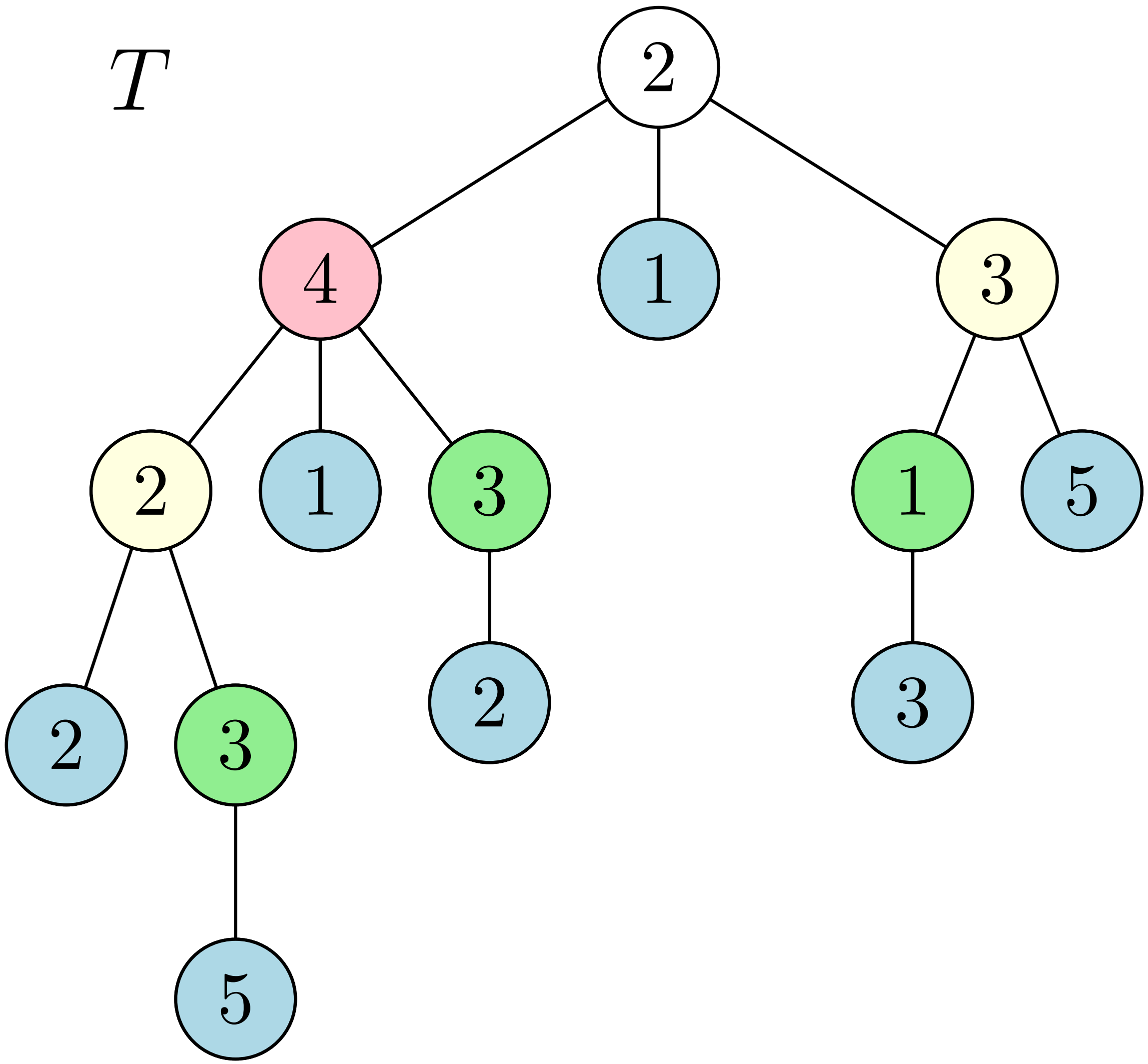
Order of Subproblems



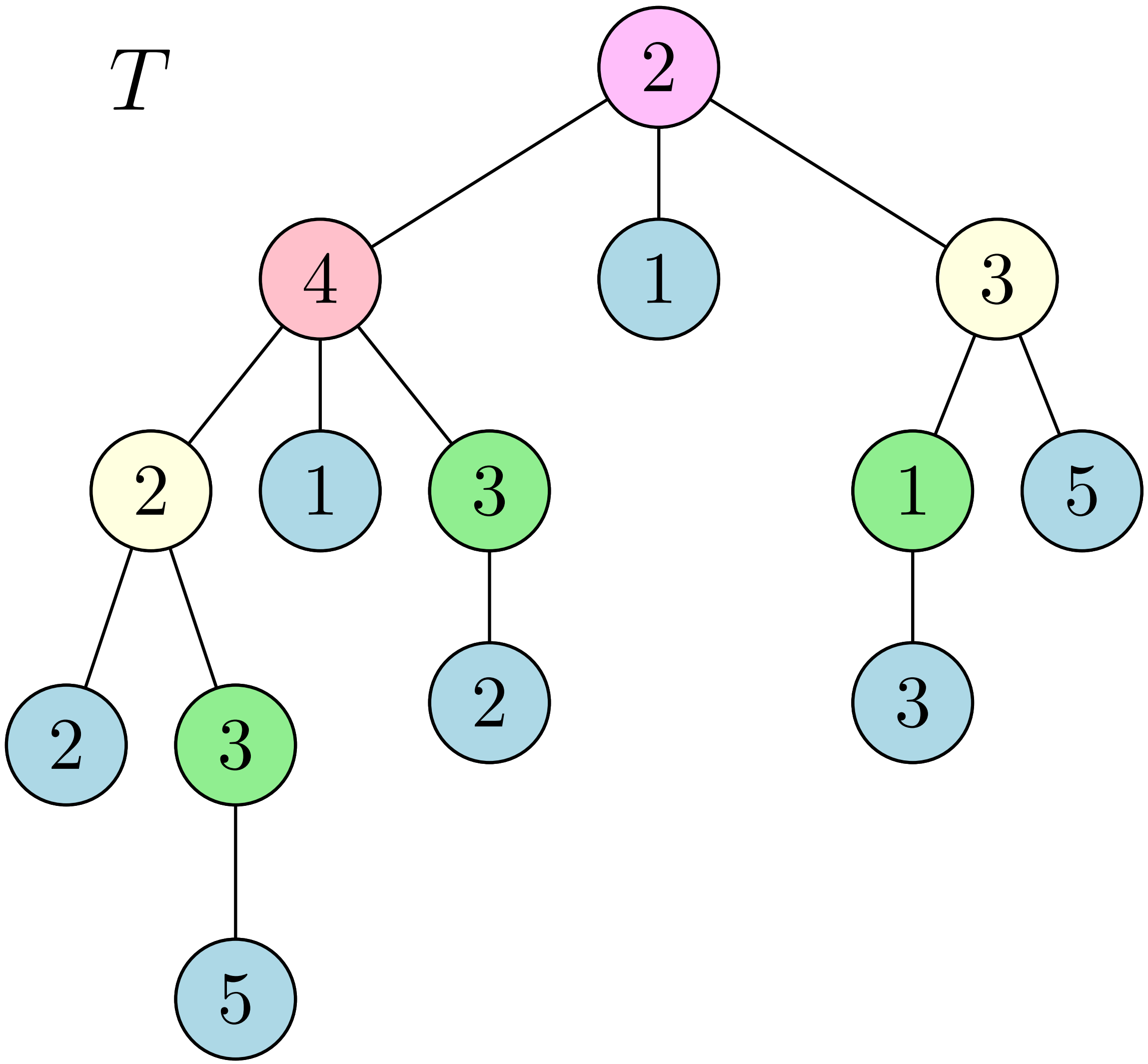
Order of Subproblems



Order of Subproblems

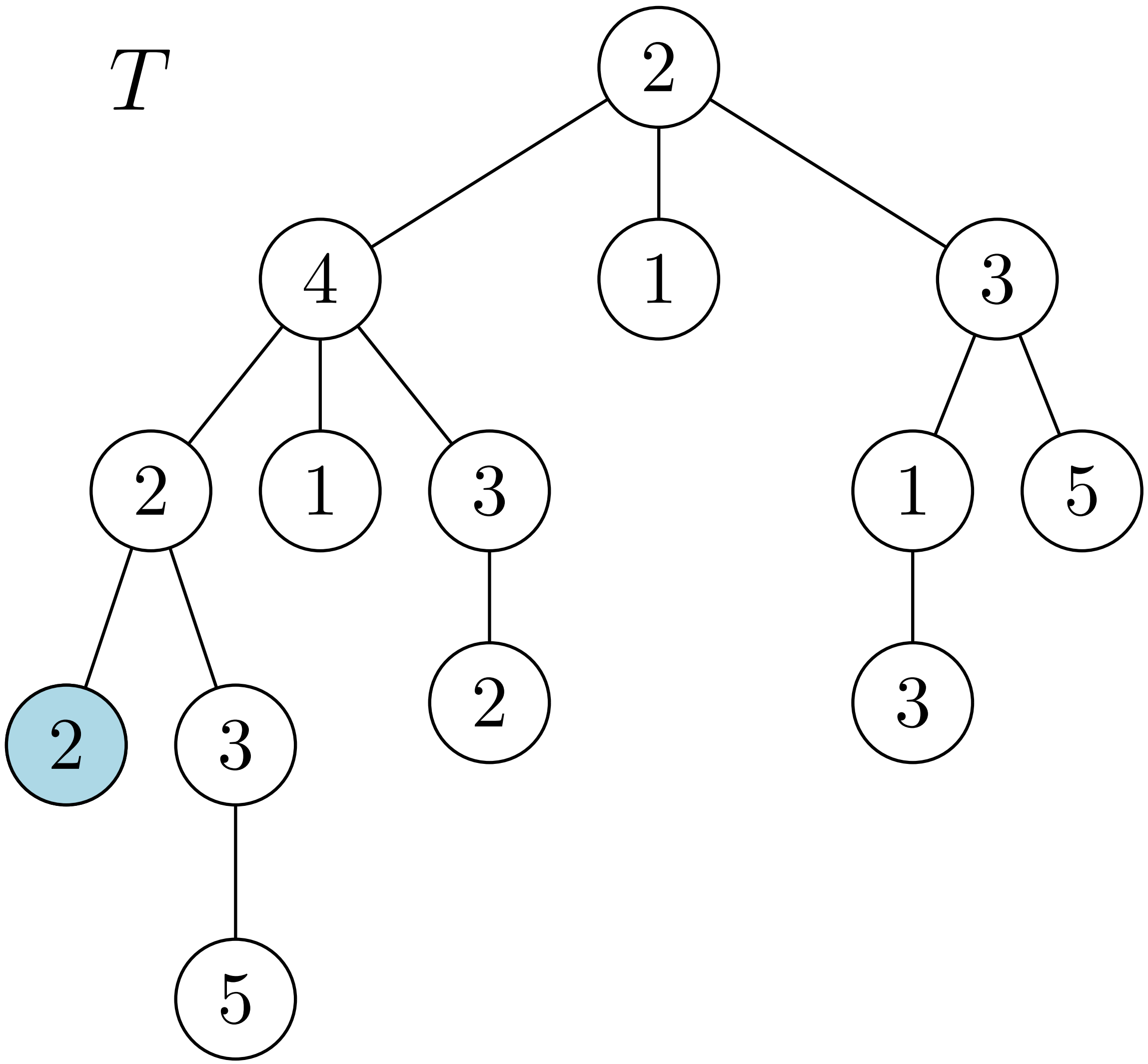


Order of Subproblems

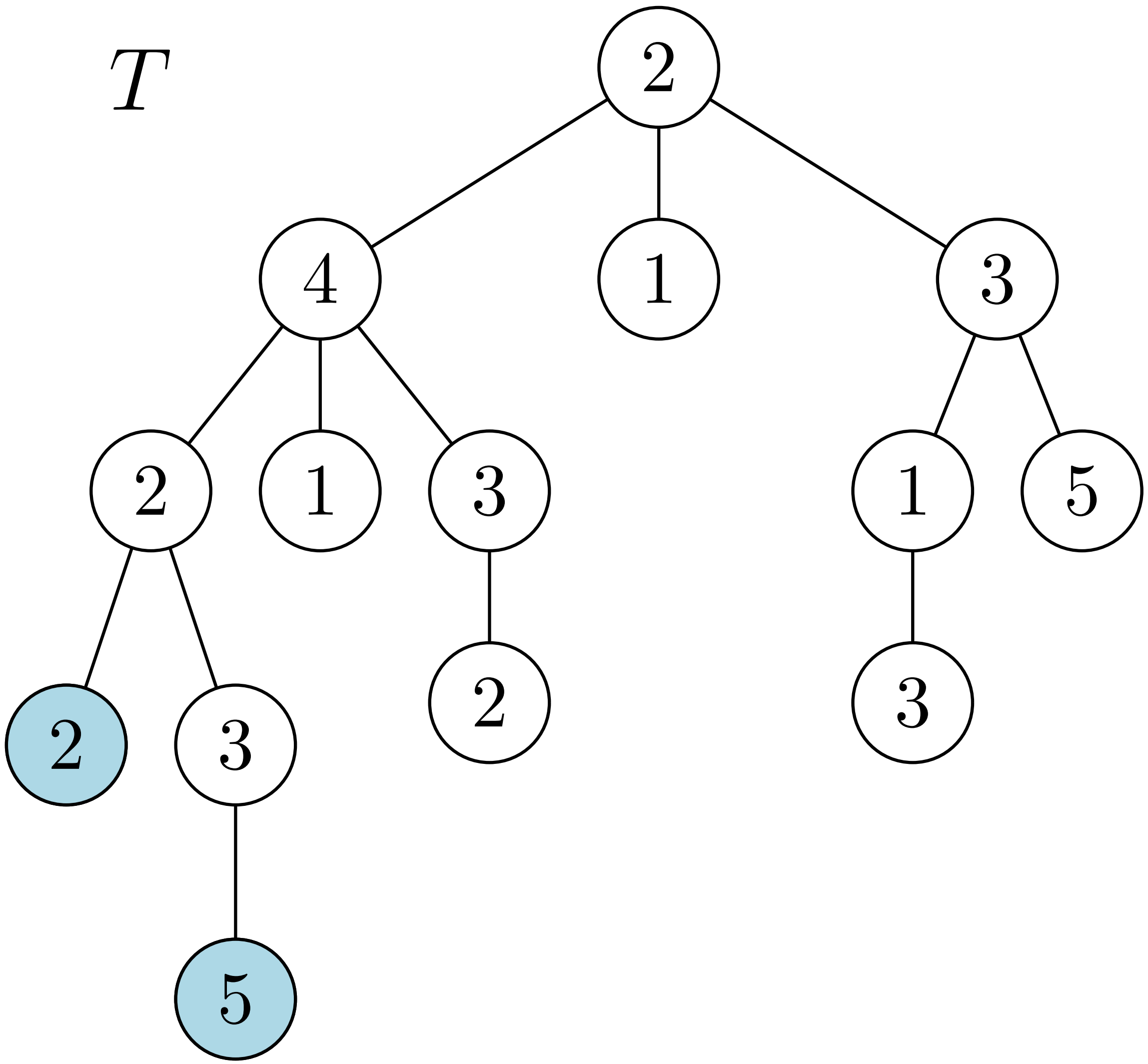


In order of increasing subtree heights

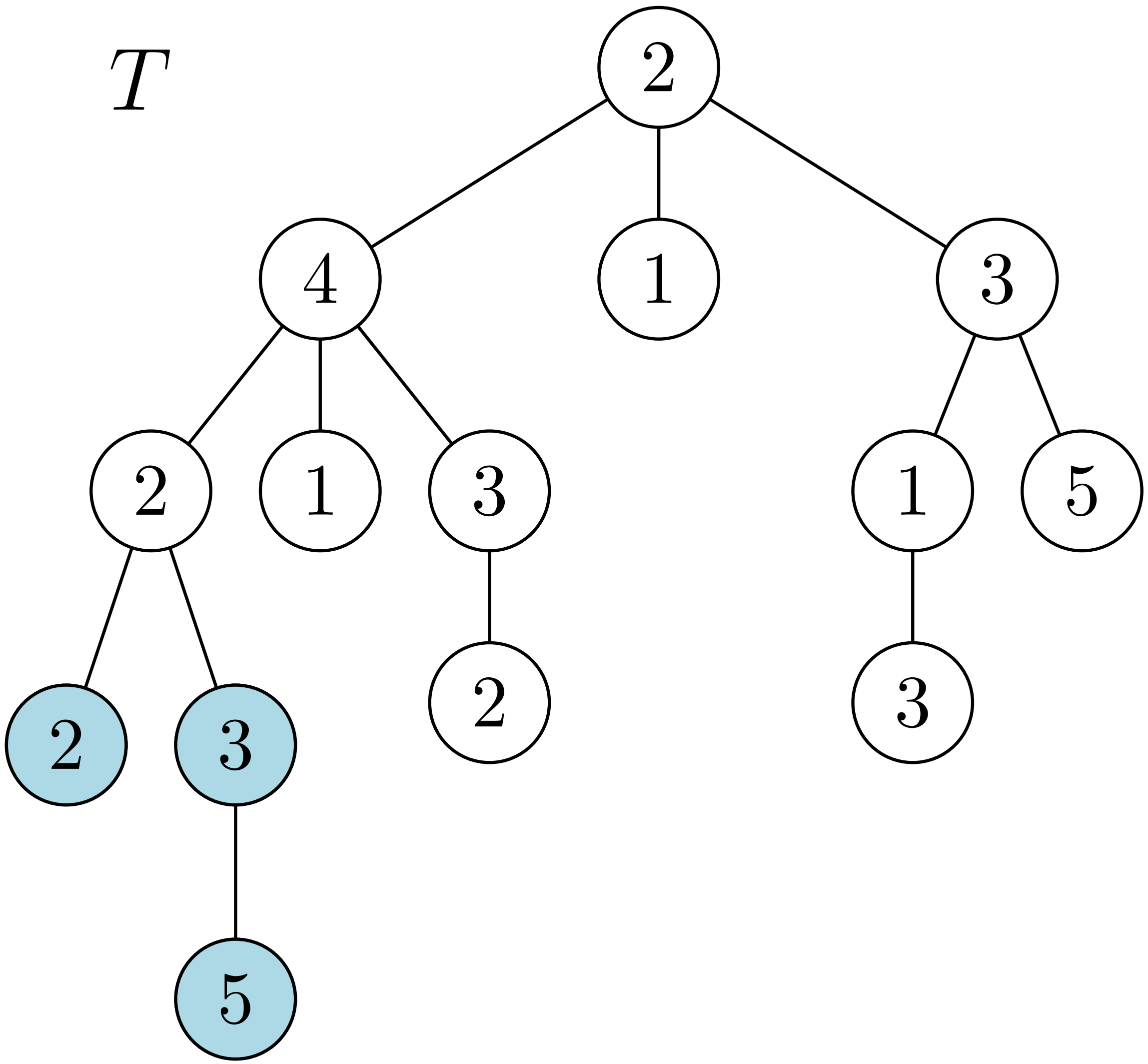
Order of Subproblems



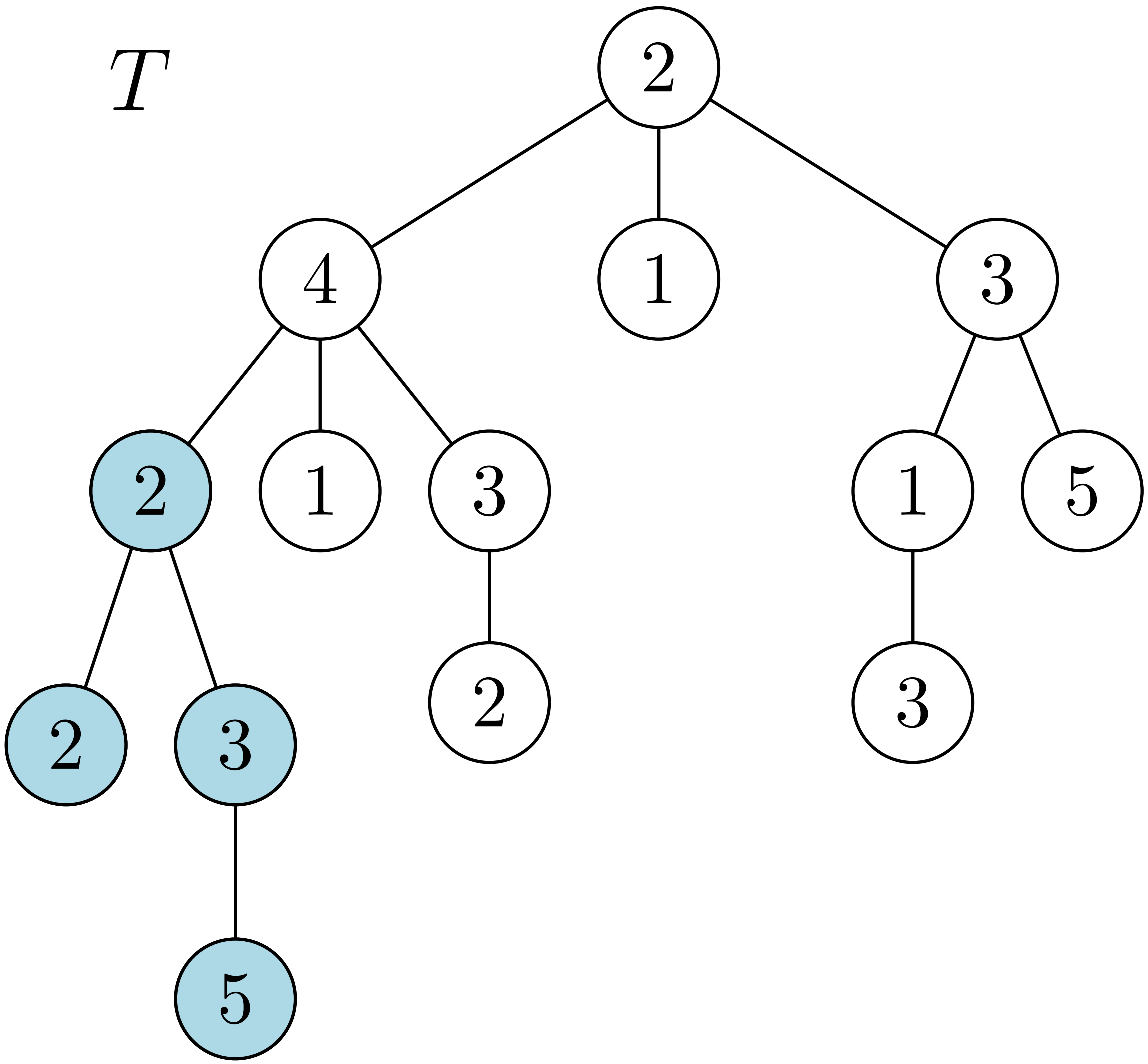
Order of Subproblems



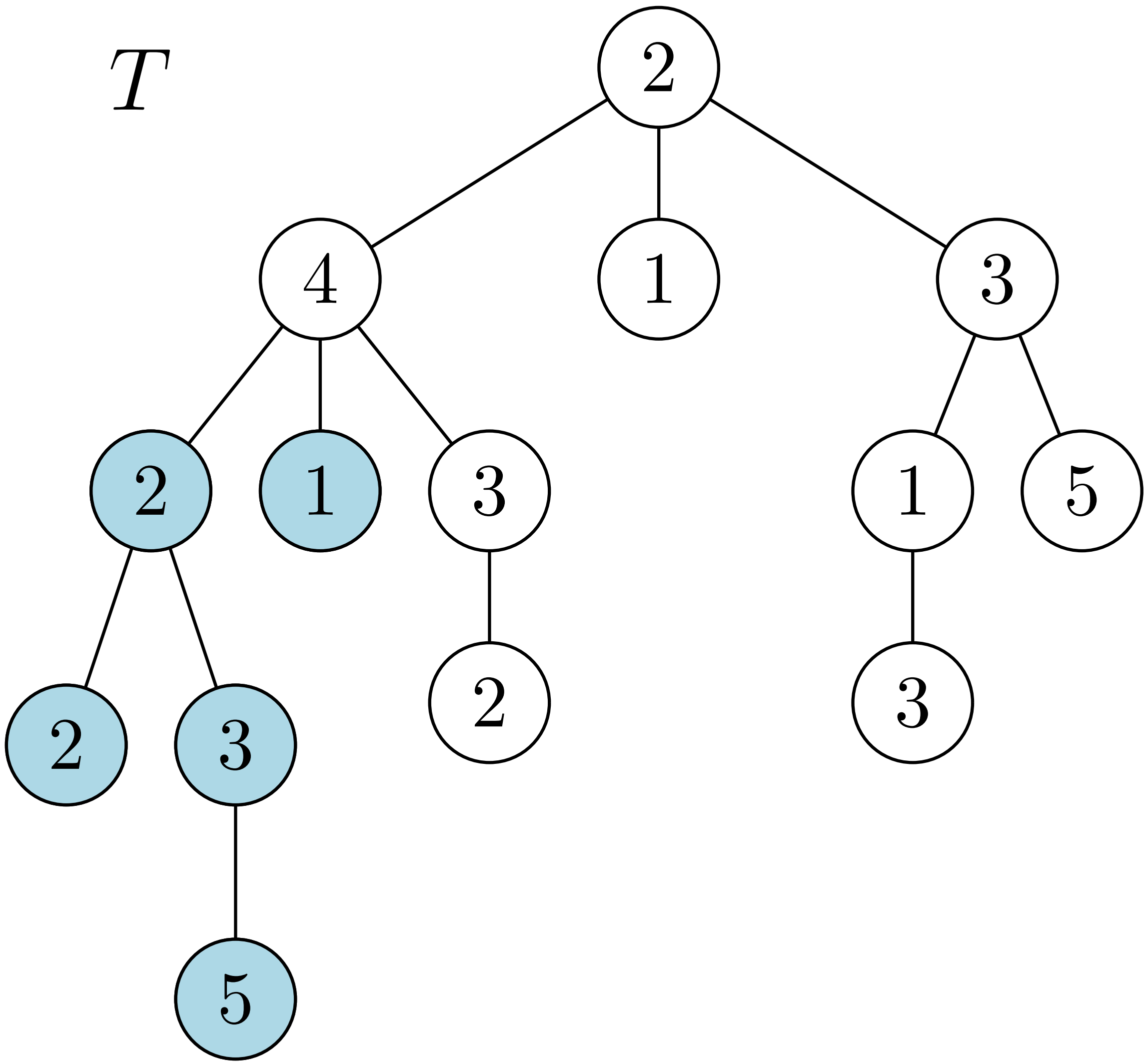
Order of Subproblems



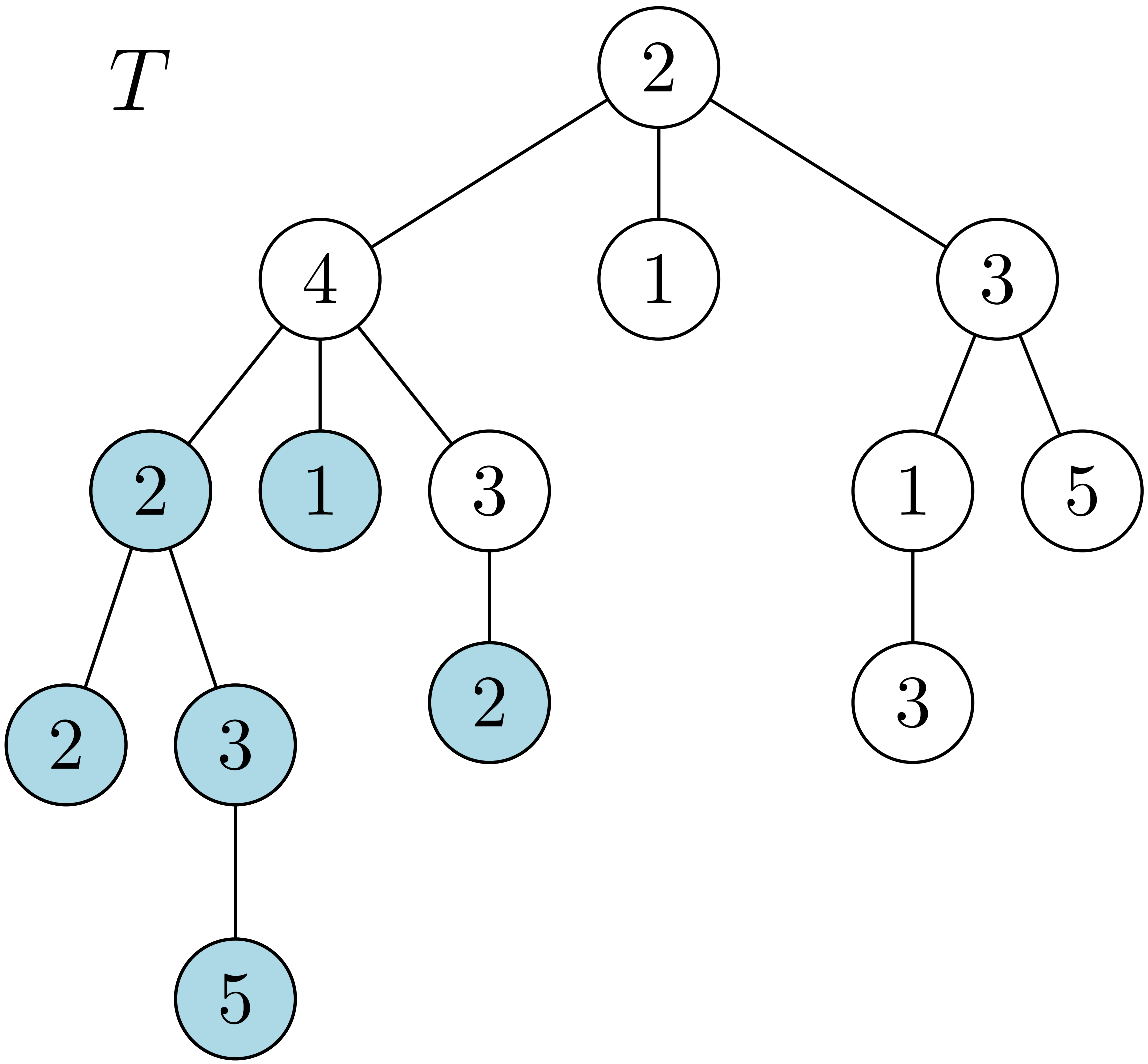
Order of Subproblems



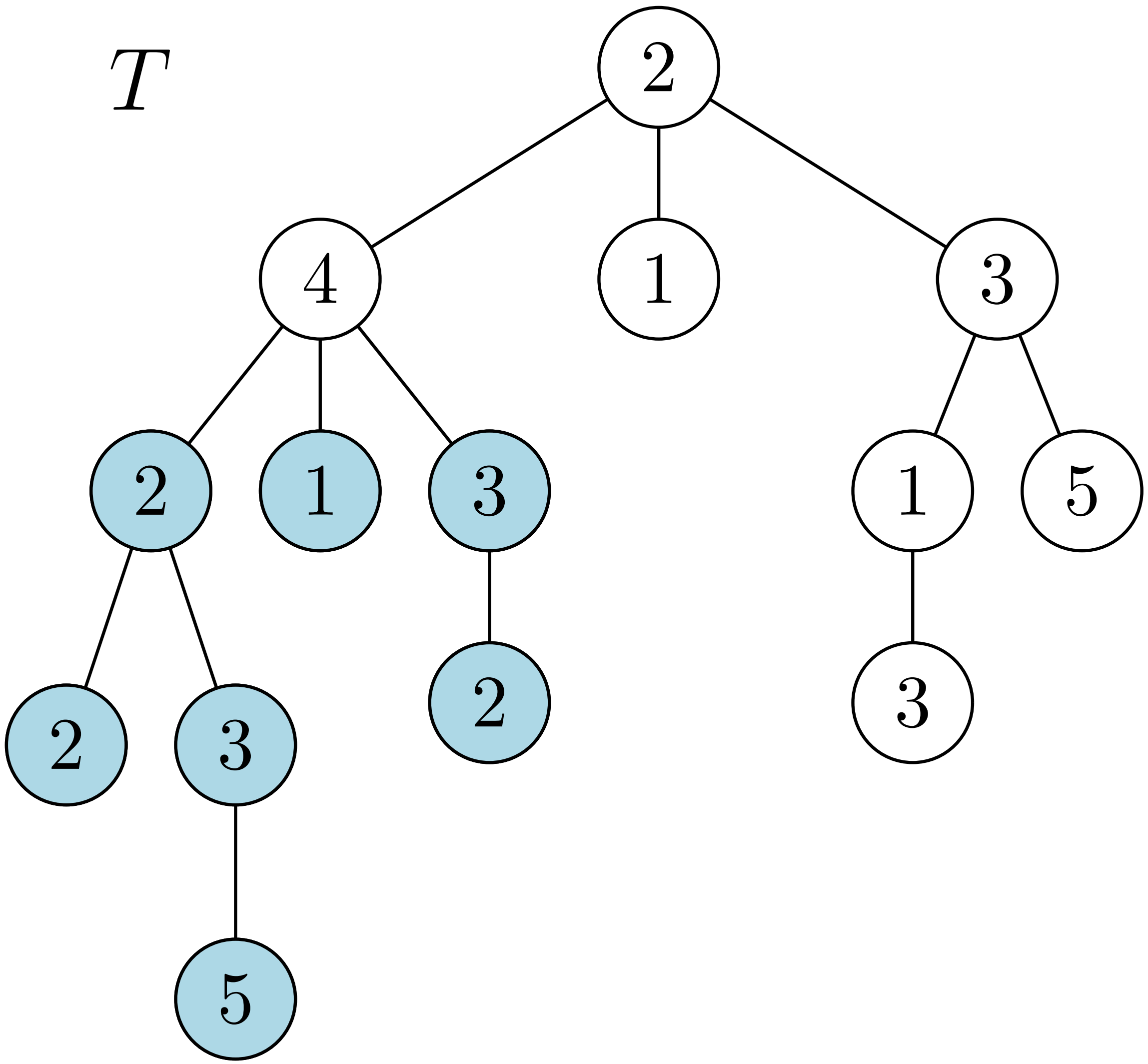
Order of Subproblems



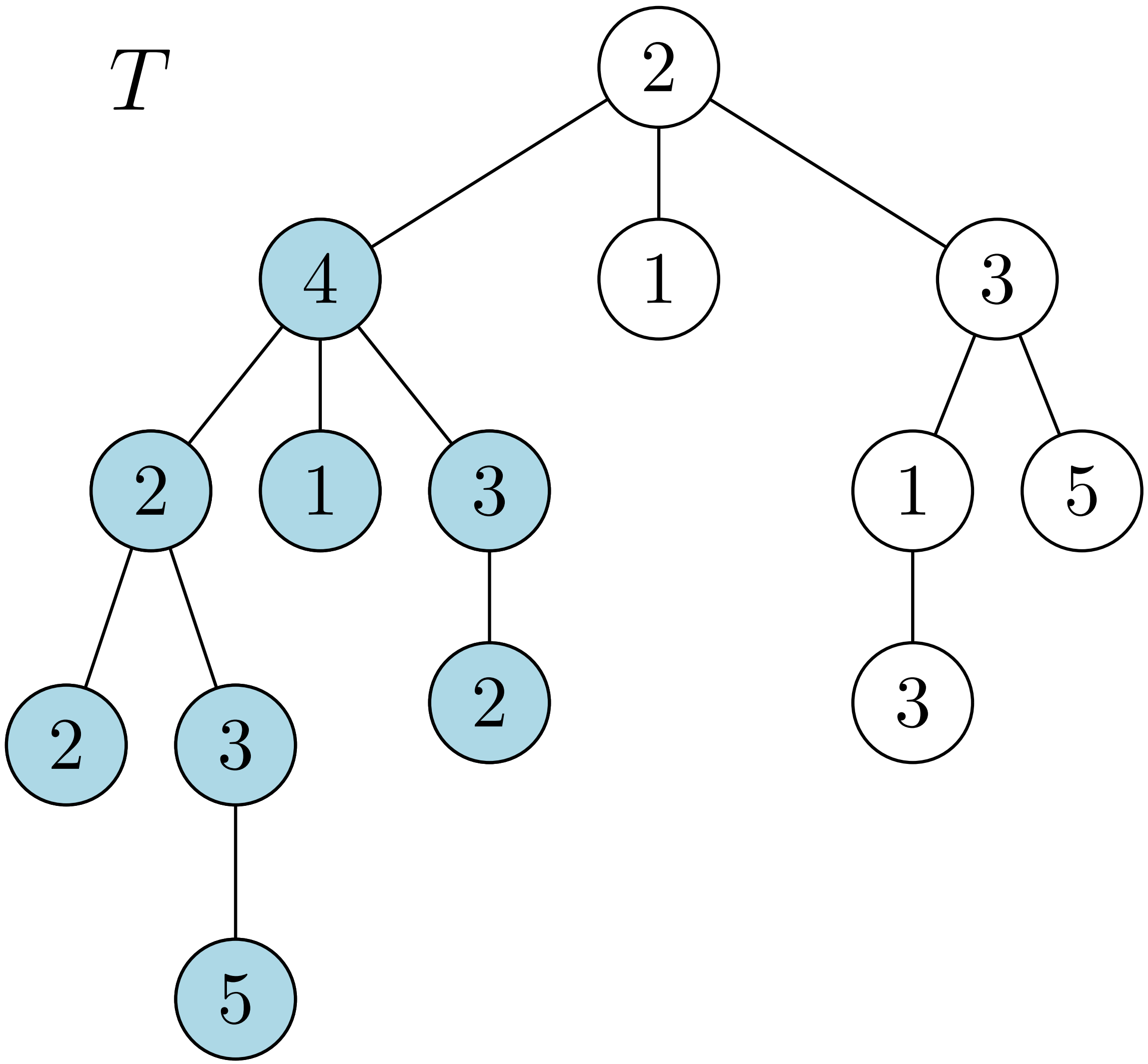
Order of Subproblems



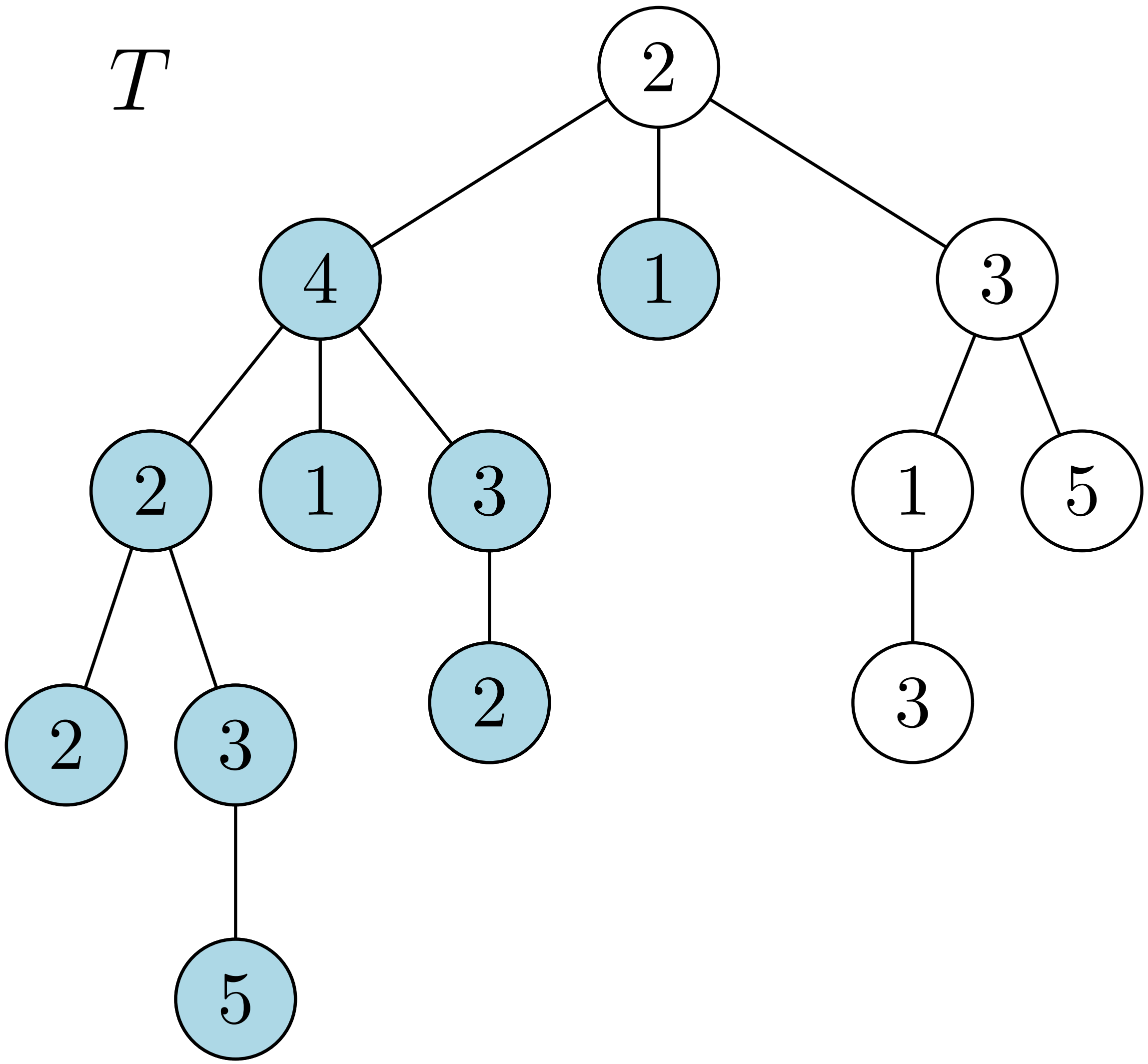
Order of Subproblems



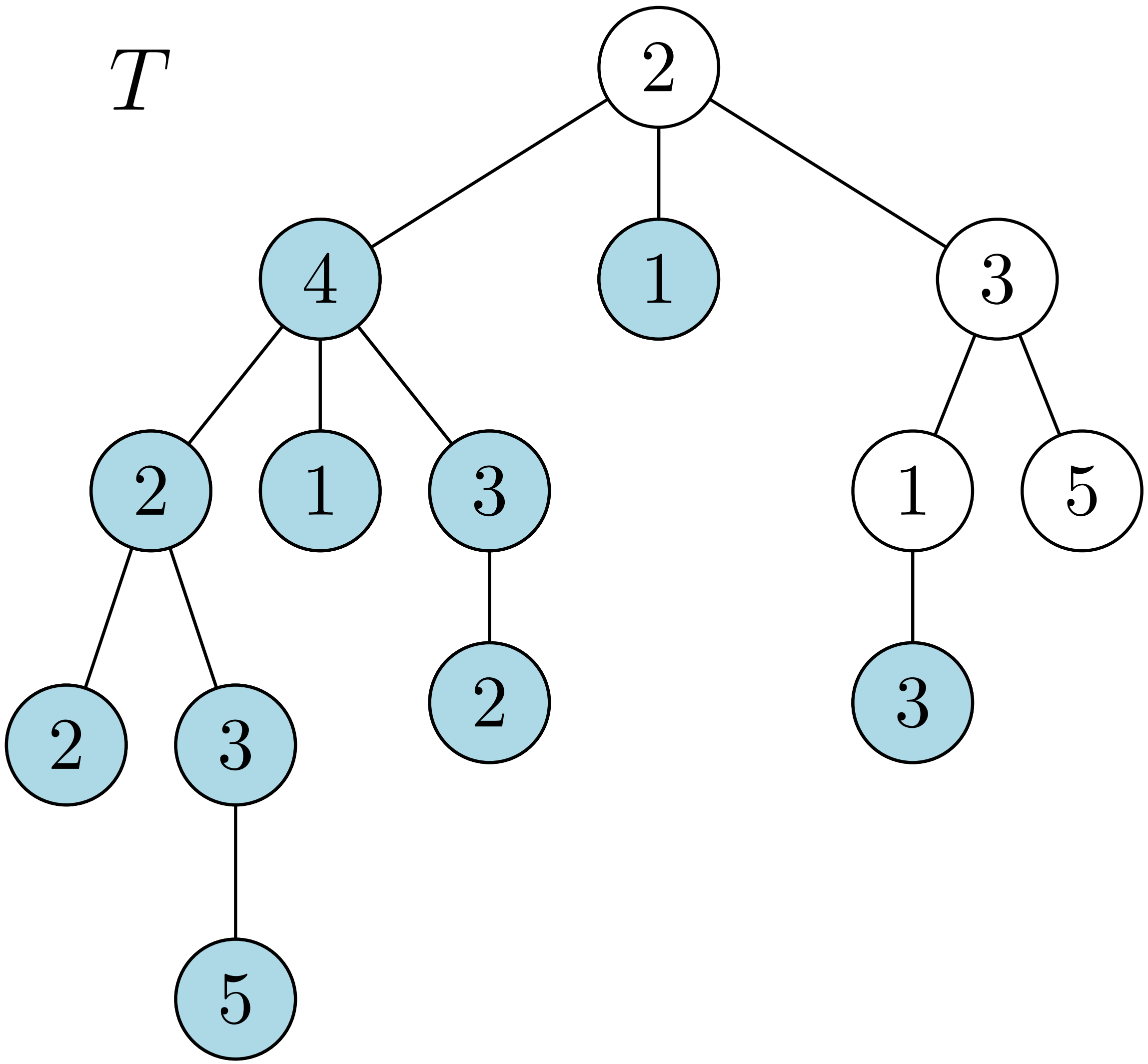
Order of Subproblems



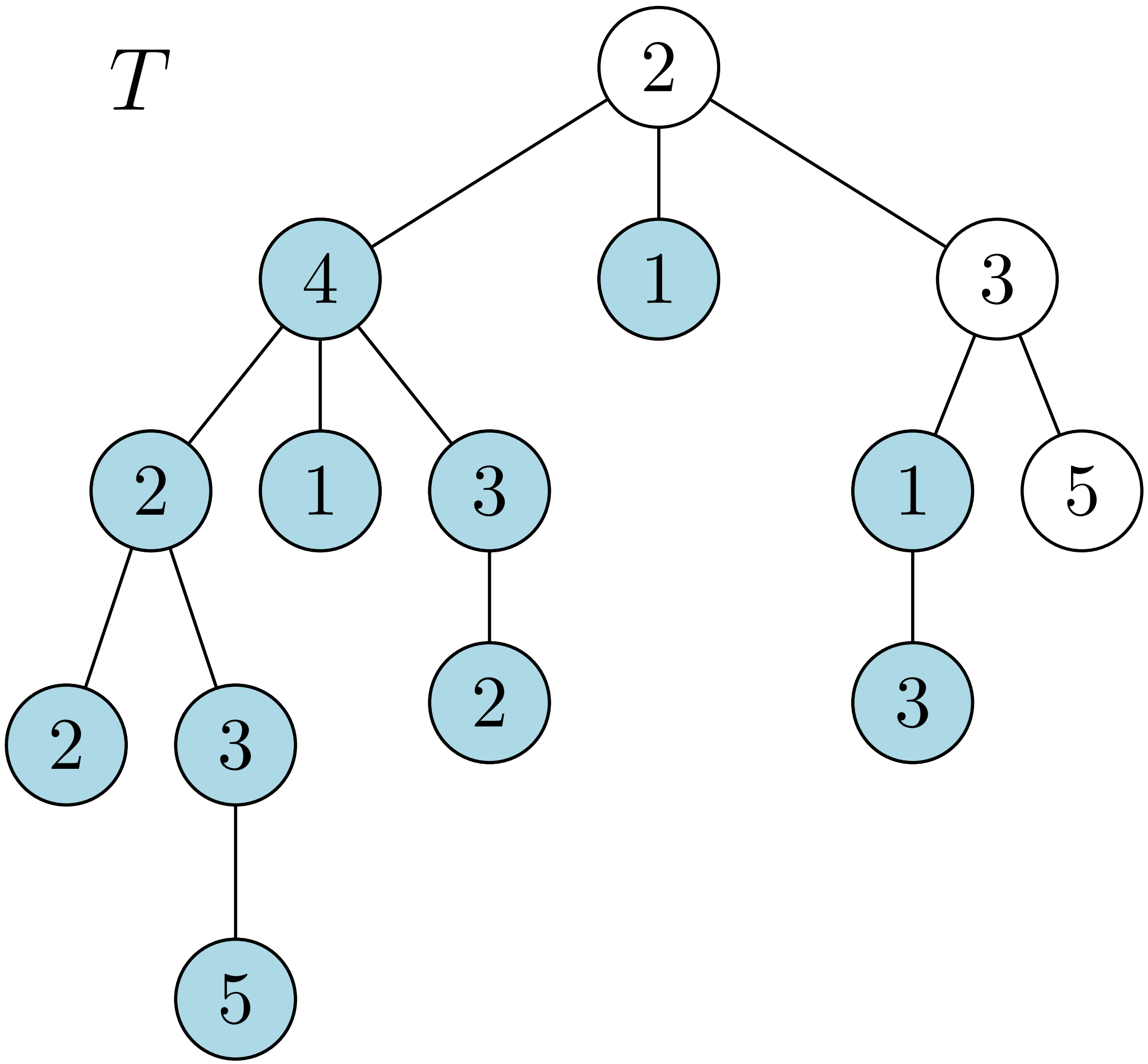
Order of Subproblems



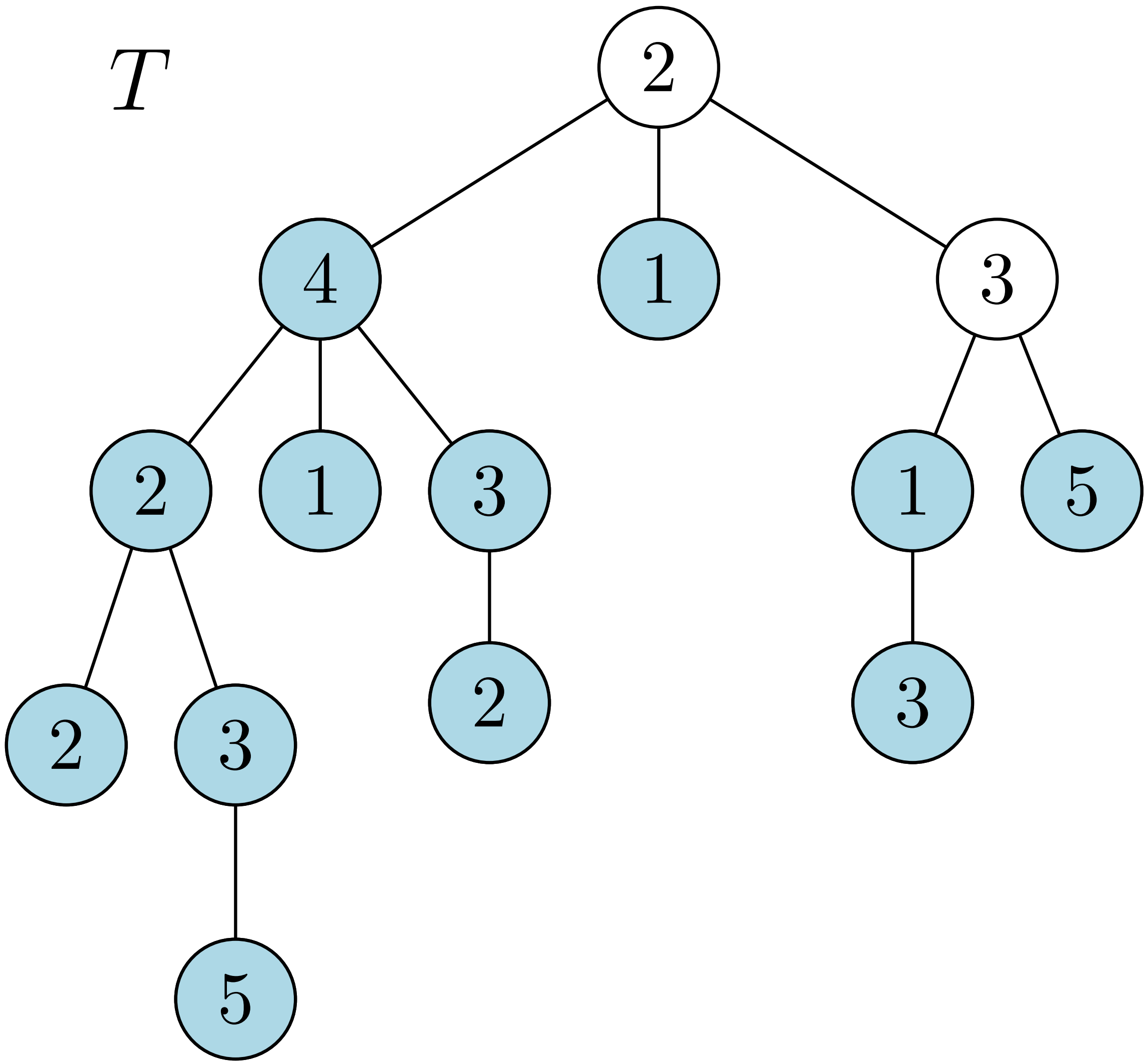
Order of Subproblems



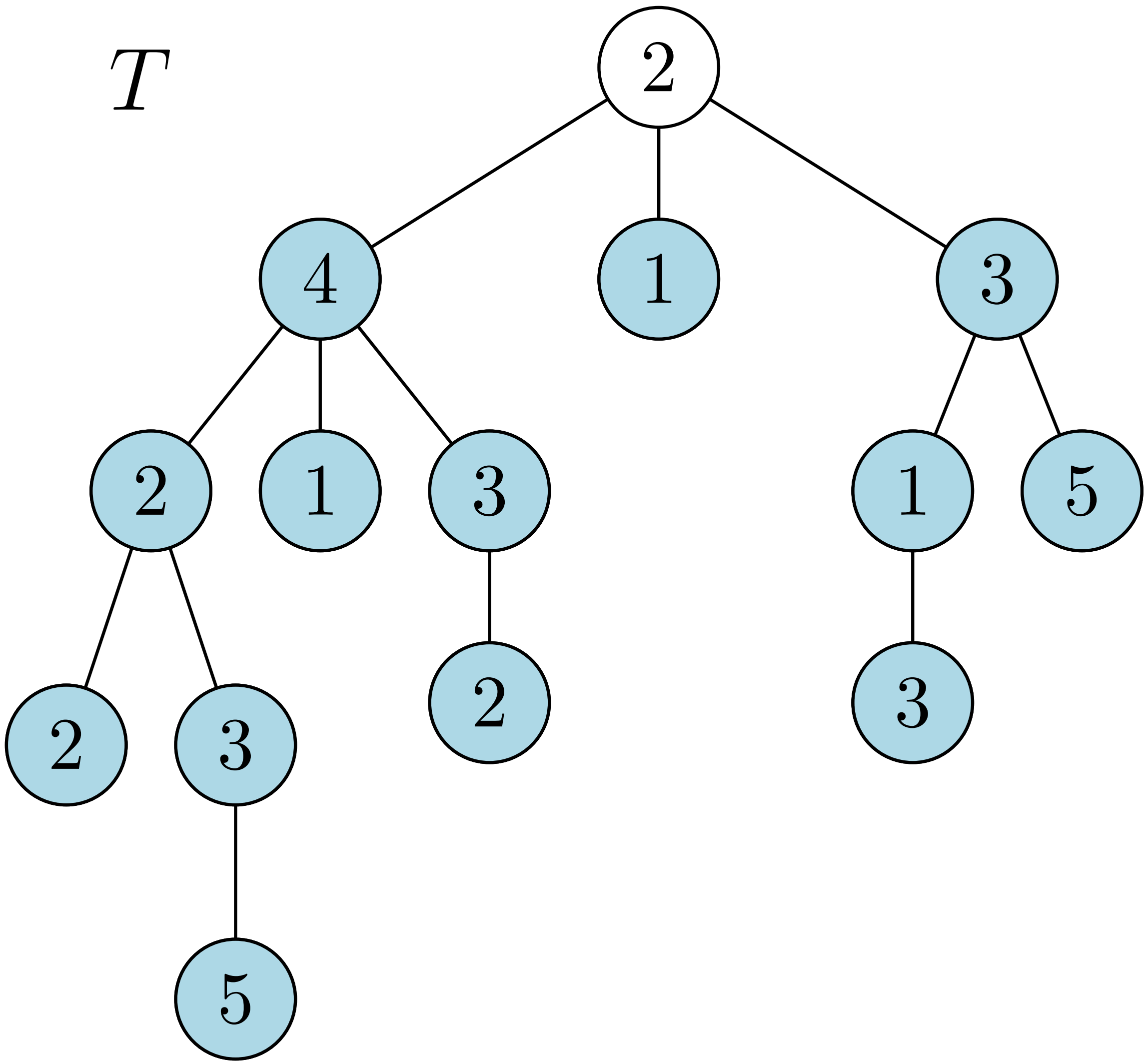
Order of Subproblems



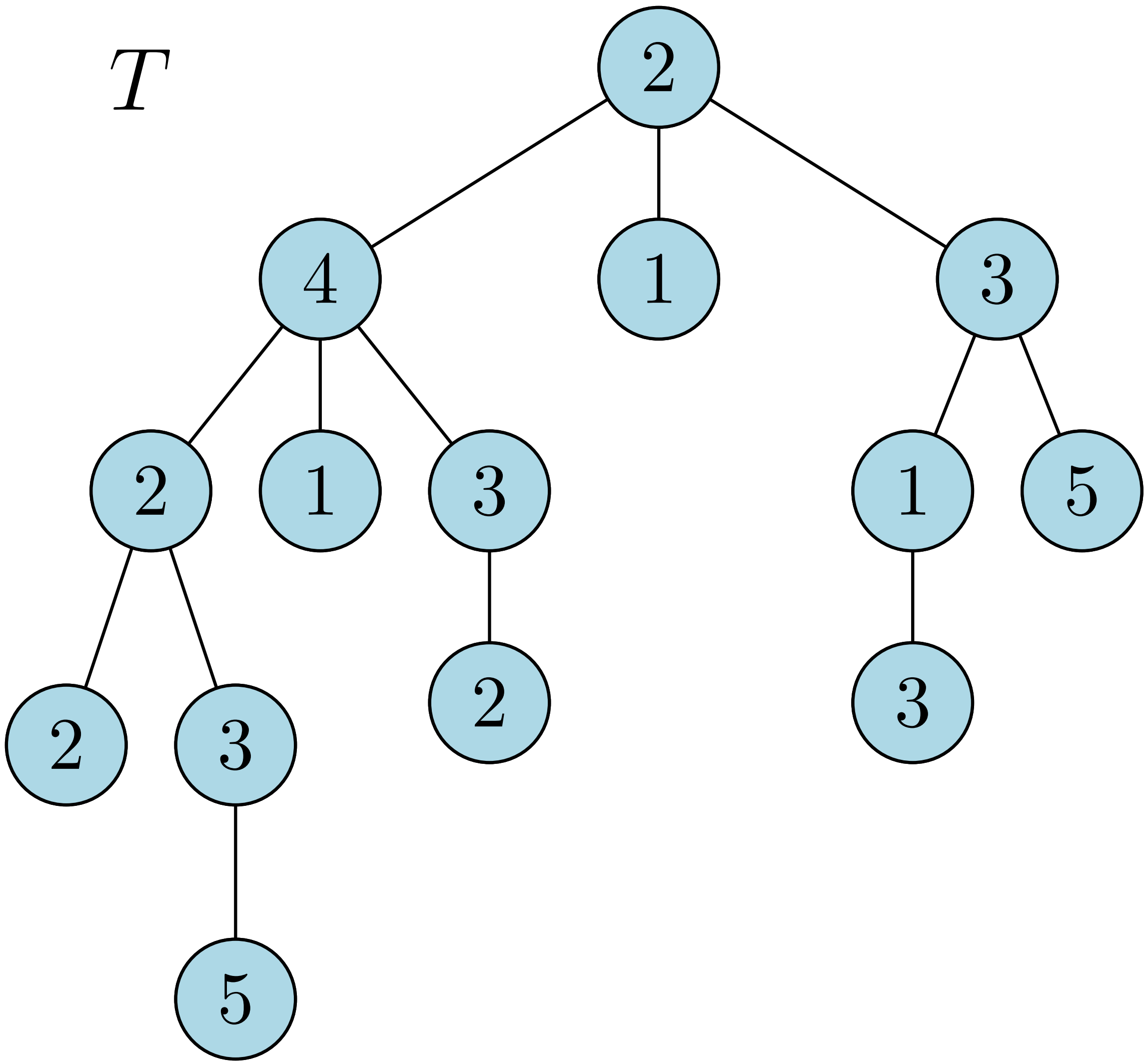
Order of Subproblems



Order of Subproblems



Order of Subproblems



In DFS postoder

Time Complexity

- Suppose we can find a suitable order in $O(n)$ time.
- The time spent on vertex v is: $O(1 + |C(v)|)$
- Overall time complexity, up to multiplicative constants:

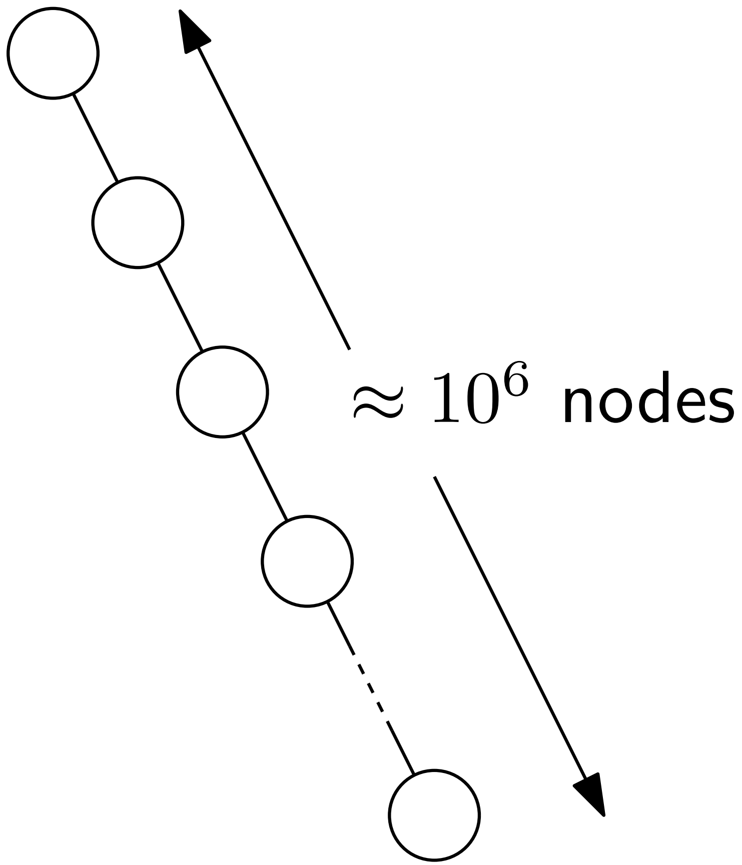
$$\sum_{v \in V(T)} (1 + |C(v)|) = n + \sum_{v \in V(T)} |C(v)| = n + (n-1) = O(n)$$

A possible implementation with DFS

```cpp
struct Node
{
    int weight;
    std::vector<Node*> children;
};

std::pair<int,int> dfs(Node* v)
{
    int opt_plus = v->weight;
    int opt_minus = 0;
    for(Node *u : v->children)
    {
        std::pair<int,int> opt_u = dfs(u);
        opt_plus += opt_u.second;
        opt_minus += std::max(opt_u.first, opt_u.second);
    }

    return std::make_pair(opt_plus, opt_minus);
}

Node* root = load_tree(); //Read T. Return a pointer to its root.
std::pair<int,int> opt = dfs(root);
std::cout << std::max(opt.first, opt.second) << "\n";
```

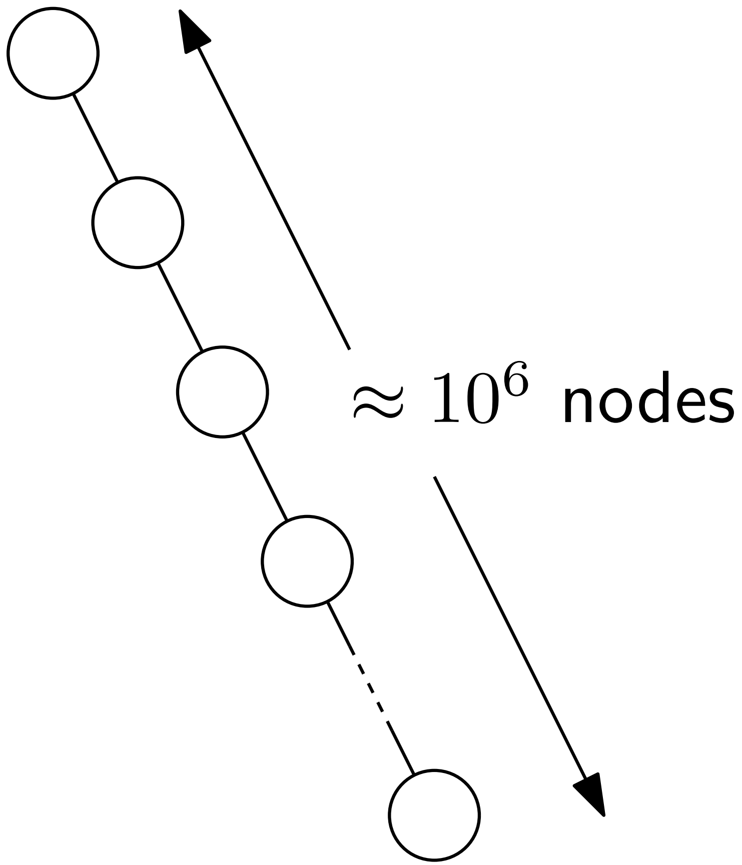
A Nasty Instance

What happens if the previous code is run on this tree?



A Nasty Instance

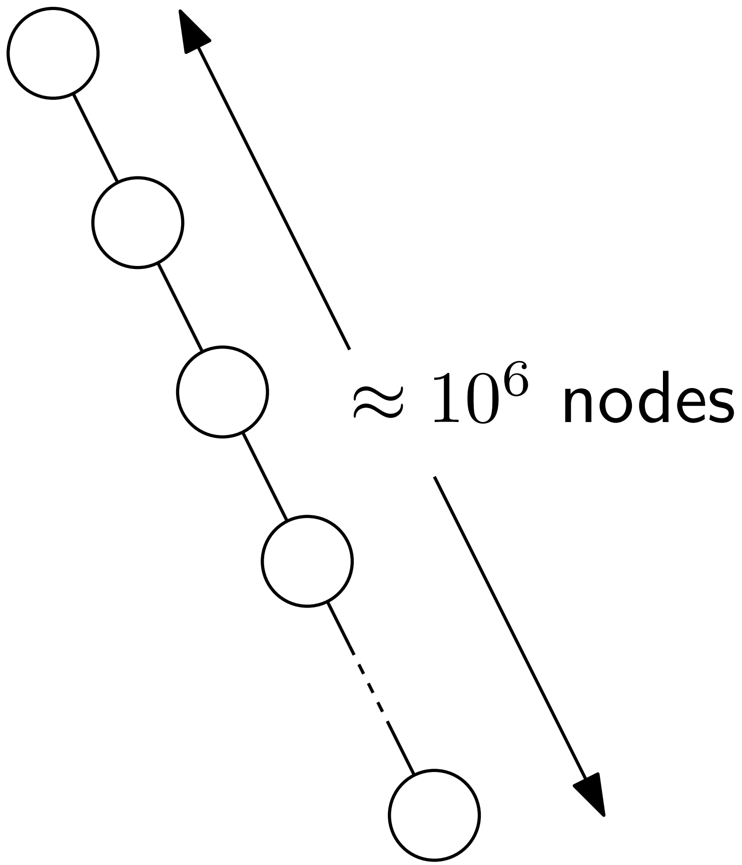
What happens if the previous code is run on this tree?




```
$ ./max_weight_is < nasty_instance.in
$ Segmentation fault
```

A Nasty Instance

What happens if the previous code is run on this tree?

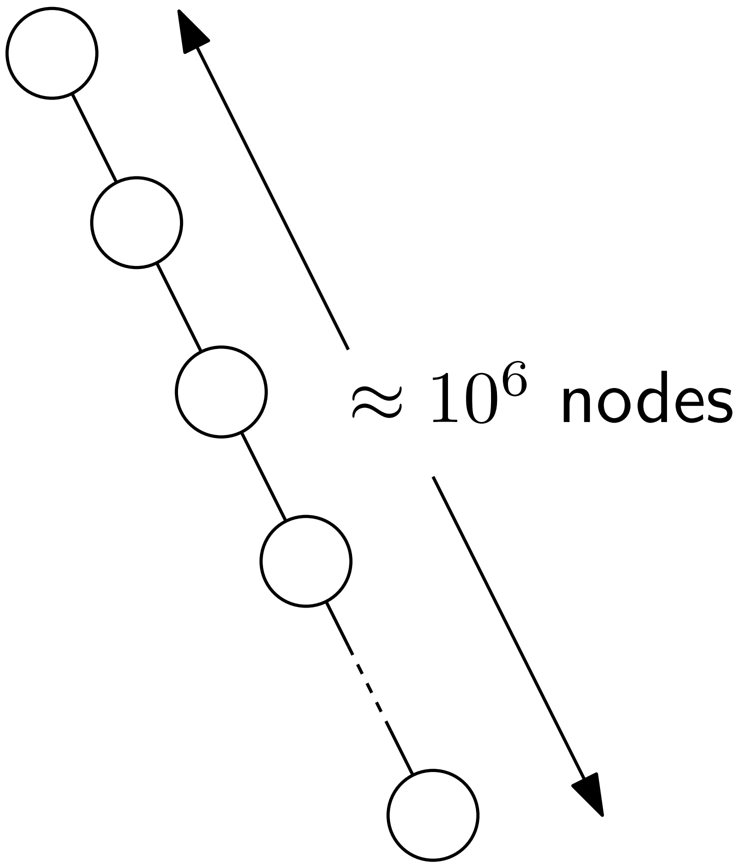



```
$ ./max_weight_is < nasty_instance.in
$ Segmentation fault
```

Why?

A Nasty Instance

What happens if the previous code is run on this tree?



Solutions

- Non recursive DFS
- Different order
  (use BFS to construct levels)
- Explicitly manage DFS stack

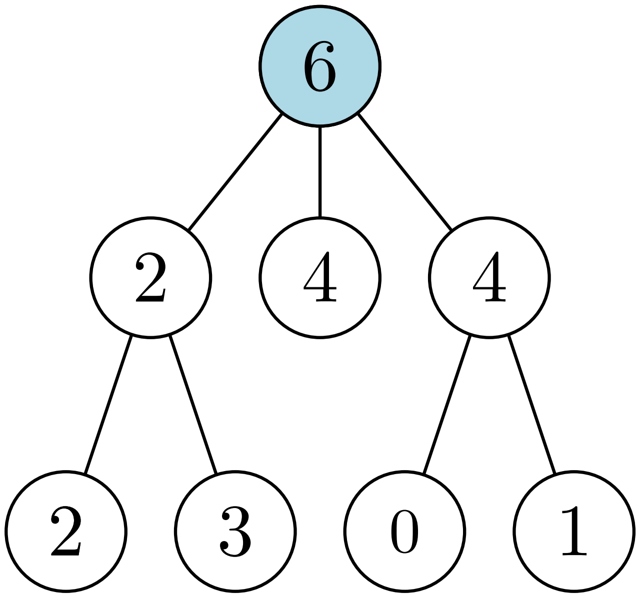
```
$ ./max_weight_is < nasty_instance.in
$ Segmentation fault
```

Max-Weight Independent Set on Trees
+ Budget Constraints

Budgeted Max-Weight IS on Trees

Input: A tree T with integer weights on its vertices, a *budget* $B \in \mathbb{N}$.

Output: The maximum weight of an independent set S of T such that $|S| \leq B$.

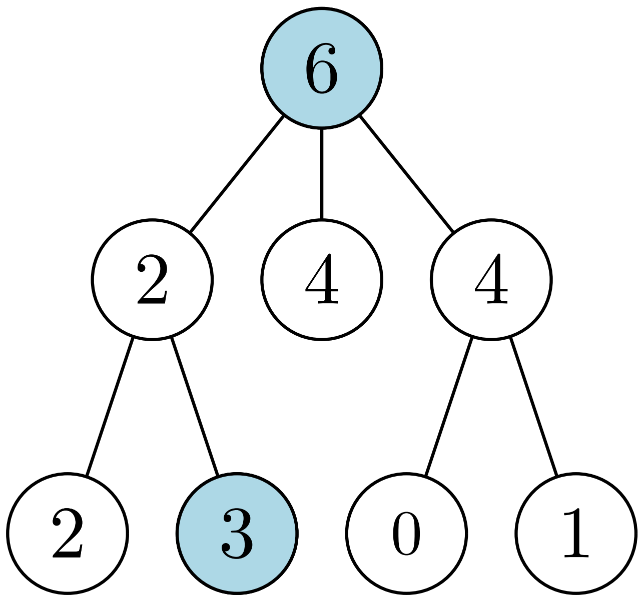


$B = 1$

Budgeted Max-Weight IS on Trees

Input: A tree T with integer weights on its vertices, a *budget* $B \in \mathbb{N}$.

Output: The maximum weight of an independent set S of T such that $|S| \leq B$.

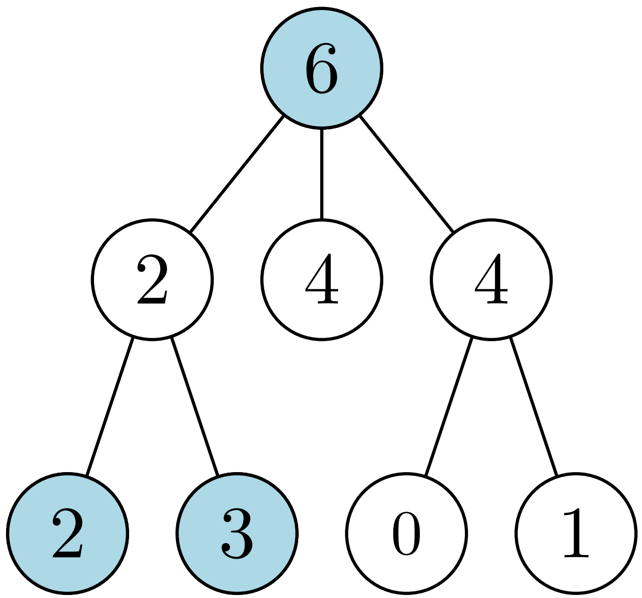


$B = 2$

Budgeted Max-Weight IS on Trees

Input: A tree T with integer weights on its vertices, a *budget* $B \in \mathbb{N}$.

Output: The maximum weight of an independent set S of T such that $|S| \leq B$.

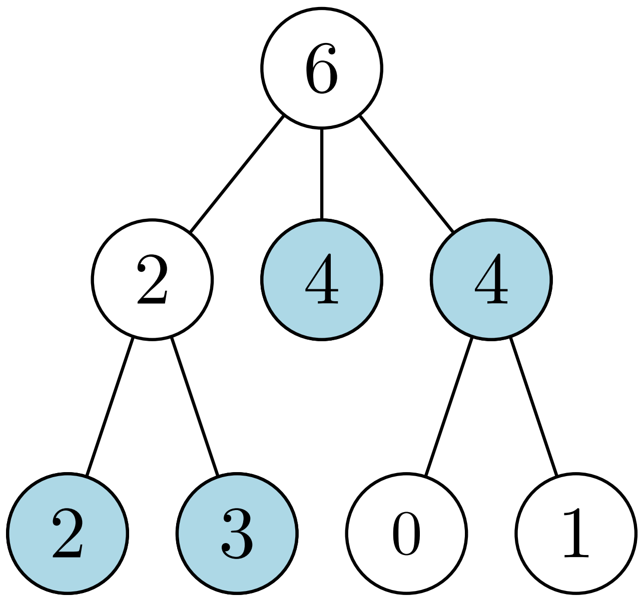


$B = 3$

Budgeted Max-Weight IS on Trees

Input: A tree T with integer weights on its vertices, a *budget* $B \in \mathbb{N}$.

Output: The maximum weight of an independent set S of T such that $|S| \leq B$.

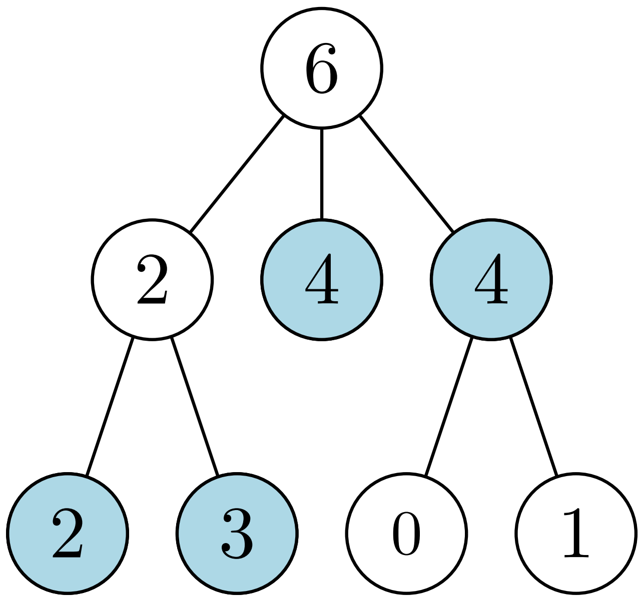


$B = 4$

Budgeted Max-Weight IS on Trees

Input: A tree T with integer weights on its vertices, a *budget* $B \in \mathbb{N}$.

Output: The maximum weight of an independent set S of T such that $|S| \leq B$.

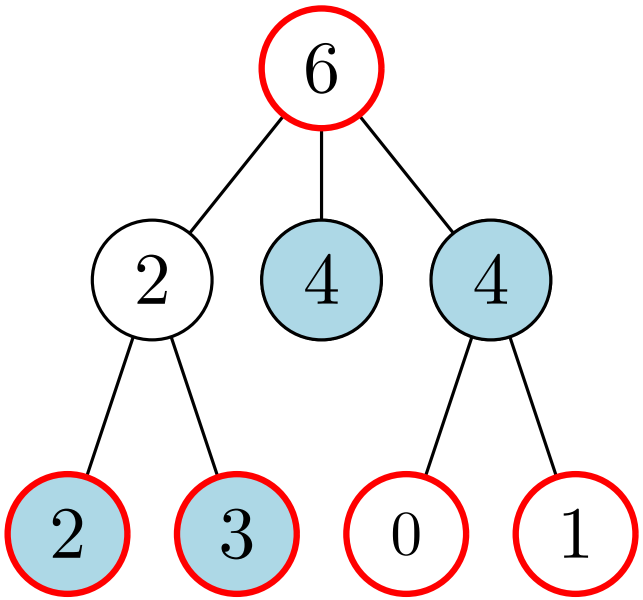


$B = 5$

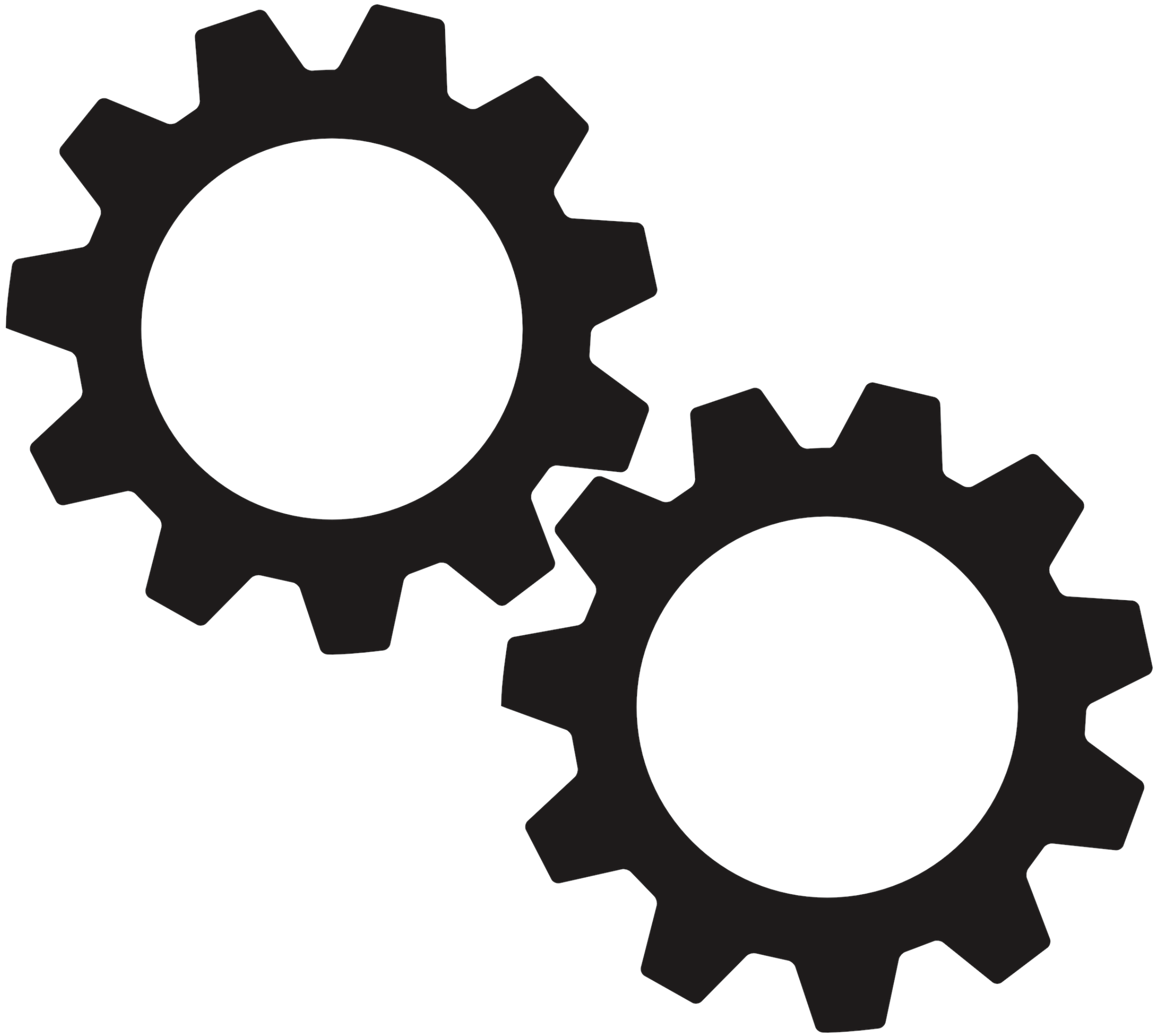
Budgeted Max-Weight IS on Trees

Input: A tree T with integer weights on its vertices, a *budget* $B \in \mathbb{N}$.

Output: The maximum weight of an independent set S of T such that $|S| \leq B$.



$B = 5$



Budgeted Max-Weight IS on Trees

Subproblem definition:

$OPT^+[v,b] =$ Maximum Weight of an IS of T_v that contains v and has size at most b.

$OPT^-[v,b] =$ Maximum Weight of an IS of T_v that does not contain v and has size at most b.

Base cases: v is a leaf of T.

$$OPT^+[v,b] = \begin{cases} w(v) & \text{if } b \geq 1 \\ -\infty & \text{if } b = 0 \end{cases}$$

Constrains can't be satisfied!

$$OPT^-[v,b] = 0$$

Max-Weight IS on Trees w. Budget

Recursive Formula:

- Let's consider $OPT^+[v, b]$.
- If $b = 0$, then $OPT^+[v, b] = -\infty$.
- If $b > 0$, we need to "distribute" $b - 1$ units of budget among $C(v) = \{u_1, u_2, \ldots, u_k\}$
- We want to choose $b_1, b_2, \ldots, b_k \in \mathbb{N}$ such that $b_1 + \cdots + b_k \leq b - 1$ and they maximize:

$$OPT^-[u_1, b_1] + OPT^-[u_2, b_2] + \cdots + OPT^-[u_k, b_k]$$

Max-Weight IS on Trees w. Budget

Recursive Formula (First Attempt):

- "Guess" the correct combination of $b_1, b_2, \ldots, b_k$:

$$OPT^+[v, b] = w(v) + \max_{\substack{b_1, b_2, \ldots, b_k \in \mathbb{N} \\ b_1 + \cdots + b_k \leq b-1}} \sum_{i=1}^{k} OPT^-[u_i, b_i]$$

Max-Weight IS on Trees w. Budget

Recursive Formula (First Attempt):

- "Guess" the correct combination of $b_1, b_2, \ldots, b_k$:

$$OPT^+[v, b] = w(v) + \max_{\substack{b_1, b_2, \ldots, b_k \in \mathbb{N} \\ b_1 + \cdots + b_k \le b - 1}} \sum_{i=1}^{k} OPT^-[u_i, b_i]$$

Will this work?

Max-Weight IS on Trees w. Budget

Recursive Formula (First Attempt):

- "Guess" the correct combination of $b_1, b_2, \ldots, b_k$:

$$OPT^+[v,b] = w(v) + \max_{\substack{b_1,b_2,\ldots,b_k \in \mathbb{N} \\ b_1+\cdots+b_k \le b-1}} \sum_{i=1}^{k} OPT^-[u_i, b_i]$$

Will this work? Yes!

Max-Weight IS on Trees w. Budget

Recursive Formula (First Attempt):

- "Guess" the correct combination of $b_1, b_2, \ldots, b_k$:

$$OPT^+[v,b] = w(v) + \max_{\substack{b_1, b_2, \ldots, b_k \in \mathbb{N} \\ b_1 + \cdots + b_k \le b-1}} \sum_{i=1}^{k} OPT^-[u_i, b_i]$$

Will this work? Yes!

How long will this take?

Stars and Bars!

- How many possible choices of $b_1, b_2, \ldots, b_k \in \mathbb{N}$ such that $b_1 + \cdots + b_k = x$?

Stars and Bars!

- How many possible choices of $b_1, b_2, \ldots, b_k \in \mathbb{N}$ such that $b_1 + \cdots + b_k = x$?

- How many different ways to arrange x stars and $k-1$ bars?

$$\star\star \mid \star\star\star \mid \quad \mid \star\star\star\star$$

Stars and Bars!

- How many possible choices of $b_1, b_2, \ldots, b_k \in \mathbb{N}$ such that $b_1 + \cdots + b_k = x$?

- How many different ways to arrange x stars and $k-1$ bars?

$$\underbrace{\star\star}_{b_1=2} \mid \underbrace{\star\star\star}_{b_2=3} \mid \underbrace{\quad}_{b_3=0} \mid \underbrace{\star\star\star\star}_{b_4=4}$$

Stars and Bars!

- How many possible choices of $b_1, b_2, \ldots, b_k \in \mathbb{N}$ such that $b_1 + \cdots + b_k = x$?

- How many different ways to arrange x stars and $k-1$ bars?

$$\underbrace{\star\star}_{b_1=2} \mid \underbrace{\star\star\star}_{b_2=3} \mid \underbrace{\quad}_{b_3=0} \mid \underbrace{\star\star\star\star}_{b_4=4}$$

$$\frac{(x+k-1)!}{x!(k-1)!} = \binom{x+k-1}{k-1} = \Omega\left(\left(\frac{x}{k}\right)^k\right)$$

Stars and Bars!

- How many possible choices of $b_1, b_2, \ldots, b_k \in \mathbb{N}$ such that $b_1 + \cdots + b_k = x$?

Too slow!

$$\frac{(x+k-1)!}{x!(k-1)!} = \binom{x+k-1}{k-1} = \Omega\left(\left(\frac{x}{k}\right)^k\right)$$

Recursive Formula: Second Attempt

Let's consider a more abstract problem.

- **Input:** $f_1, \ldots, f_k : \mathbb{N} \to \mathbb{R}$ and $B \in \mathbb{N}$.

- **Output:** $x_1, \ldots, x_k \in \mathbb{N}$ such that $\sum_i x_i \leq B$ and $\sum_i f_i(x_i)$ is maximized.

(Assume that each f_i can be evaluated in constant time).

Recursive Formula: Second Attempt

Let's consider a more abstract problem.

- **Input:** $f_1, \ldots, f_k : \mathbb{N} \to \mathbb{R}$ and $B \in \mathbb{N}$.

- **Output:** $x_1, \ldots, x_k \in \mathbb{N}$ such that $\sum_i x_i \leq B$ and $\sum_i f_i(x_i)$ is maximized.

(Assume that each f_i can be evaluated in constant time).

How do we solve this problem?

Recursive Formula: Second Attempt

Let's consider a more abstract problem.

- **Input:** $f_1, \ldots, f_k : \mathbb{N} \to \mathbb{R}$ and $B \in \mathbb{N}$.
- **Output:** $x_1, \ldots, x_k \in \mathbb{N}$ such that $\sum_i x_i \leq B$ and $\sum_i f_i(x_i)$ is maximized.

(Assume that each f_i can be evaluated in constant time).

How do we solve this problem?

Dynamic Programming!

Distributing Budget Optimally

Subproblem Idea

$D[j, b]$ = Best way to distribute b units of budget among the first j functions.

More Formally:

$$D[j, b] = \max_{\substack{x_1, \ldots, x_j \in \mathbb{N} \\ x_1 + \cdots + x_j \leq b}} \sum_{i=1}^{j} f_i(x_i)$$

Base Case: If $j = 1$, explictly check the $b + 1$ possible choices.

$$D[1, b] = \max\{f_1(0), f_1(1), f_1(2), \ldots, f_1(b)\}$$

Distributing Budget Optimally

Recursive Formula

"Guess" how much budget b' will be assigned to f_j.

$$D[j, b] = \max_{b' \in \{0,\ldots,b\}} \{D[j-1, b-b'] + f_j(b')\}$$

At most $O(B)$ choices.

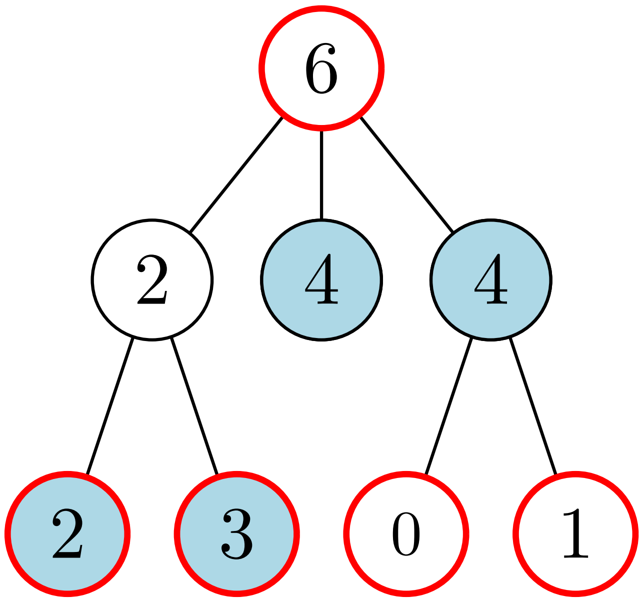
Time Complexity: $k(B+1) \cdot O(B) = O(kB^2)$

(Value of the) Optimal Solution: $D[k, B]$

Back to the Original Problem

Input: A tree T with integer weights on its vertices, a *budget* $B \in \mathbb{N}$.

Output: The maximum weight of an independent set S of T such that $|S| \leq B$.



$B = 5$

Max-Weight IS on Trees w. Budget

Base cases: v is a leaf of T.

$$OPT^+[v,b] = \begin{cases} w(v) & \text{if } b \geq 1 \\ -\infty & \text{if } b = 0 \end{cases}$$

$$OPT^-[v,b] = 0$$

Recursive formula for $OPT^+[v,b]$

- Let $C(v) = \{u_1, \ldots, u_k\}$.
- Compute $D[k, b-1]$ for $f_i(x) = OPT^-[u_i, x]$.

$$OPT^+[v,b] = w(v) + D[k, b-1]$$

Max-Weight IS on Trees w. Budget

Base cases: v is a leaf of T.

$$OPT^{+}[v,b] = \begin{cases} w(v) & \text{if } b \geq 1 \\ -\infty & \text{if } b = 0 \end{cases}$$

$$OPT^{-}[v,b] = 0$$

Nested DP!

Recursive formula for $OPT^{+}[v,b]$

- Let $C(v) = \{u_1, \ldots, u_k\}$.
- Compute $D[k, b-1]$ for $f_i(x) = OPT^{-}[u_i, x]$.

$$OPT^{+}[v,b] = w(v) + \underline{D[k, b-1]}$$

Max-Weight IS on Trees w. Budget

Base cases: v is a leaf of T.

$$OPT^+[v,b] = \begin{cases} w(v) & \text{if } b \geq 1 \\ -\infty & \text{if } b = 0 \end{cases}$$

$$OPT^-[v,b] = 0$$

Recursive formula for $OPT^-[v,b]$

Nested DP!

- Let $C(v) = \{u_1, \ldots, u_k\}$.
- Compute $D[k,b]$ for

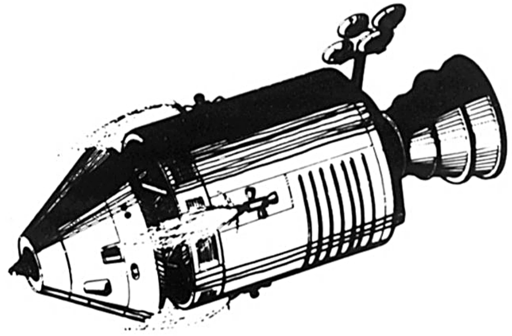
$$f_i(x) = \max\{OPT^-[u_i,x], OPT^+[u_i,x]\}.$$

$$OPT^-[v,b] = \underline{D[k,b]}$$

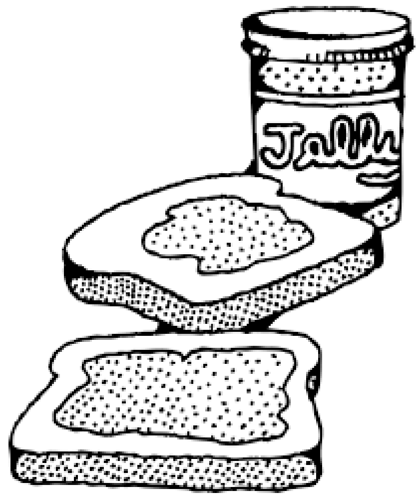
Edit Distance

Edit Distance

"Next NASA mission is going to land on ~~toast~~ mars"



- Autocorrect / Spell checking
- Unix `diff`
- Bioinformatics (DNA alignment)
- Plagiarism detection
- Speech recognition
- ...



Edit Distance

Input: Two strings $S = s_1 s_2 \ldots s_n$, and $T = t_1 t_2 \ldots t_m$.

Output: The *edit distance* between S and T.

Definition: The *edit distance* between S and T is the minimum number of *edits* required to turn S into T, where an edit is one of:

- **Insertion:** Inserting a new character at some position of S.

  `MARS` $\rightarrow$ `MARKS`

- **Deletion:** Removing one of the characters in S.

  `MARS` $\rightarrow$ `MAS`

- **Substitution:** Replacing one character of S with another.

  `MARS` $\rightarrow$ `CARS`

A Dynamic Programming Algorithm

Subproblem definition. For $0 \leq i \leq n$ and $0 \leq j \leq m$:

$OPT[i, j] =$ Edit distance between $S^{(i)} = s_1, \ldots, s_i$ and $T^{(j)} = t_1, \ldots, t_j$.

Note: $S^{(0)} = T^{(0)} = \varepsilon$, where ε is the empty string.

Base case:

$OPT[0, 0] =$ Minum number of operations needed to transform $S^{(0)} = \varepsilon$ into $T^{(0)} = \varepsilon$.

$OPT[0, 0] = 0$

A Dynamic Programming Algorithm

Recursive formula

If $i, j > 0$:

$$OPT[i,j] = \min \begin{cases} 1 + OPT[i-1,j] & \text{(deletion)} \\ 1 + OPT[i,j-1] & \text{(insertion)} \\ \mathbb{1}_{(s_i \neq t_j)} + OPT[i-1,j-1] & \text{(substitution)} \end{cases}$$

If $i = 0$ or $j = 0$:

$$OPT[i,j] = \begin{cases} 1 + OPT[0,j-1] & \text{if } i = 0 \\ 1 + OPT[i-1,0] & \text{if } j = 0 \end{cases} = \max\{i,j\}$$

Example

$\texttt{MARS} \rightarrow \texttt{TOAST}$

		0	1	2	3	4	5
	$i \backslash j$	ε	T	O	A	S	T
0	ε	0					
1	M						
2	A						
3	R						
4	S						

Example

$\texttt{MARS} \rightarrow \texttt{TOAST}$

		0	1	2	3	4	5
	$i \backslash j$	ε	T	O	A	S	T
0	ε	0	1	2	3	4	5
1	M	1					
2	A	2					
3	R	3					
4	S	4					

Example

$$\texttt{MARS} \rightarrow \texttt{TOAST}$$

		0	1	2	3	4	5
	$i \backslash j$	ε	T	O	A	S	T
0	ε	0	1	2	3	4	5
1	M	1					
2	A	2					
3	R	3					
4	S	4					

Example

$$\texttt{MARS} \to \texttt{TOAST}$$

	0	1	2	3	4	5
$i \backslash j$	ε	T	O	A	S	T
0 ε	0	1	2	3	4	5
1 M	1					
2 A	2					
3 R	3					
4 S	4					

Example

$$\texttt{MARS} \rightarrow \texttt{TOAST}$$

		0	1	2	3	4	5
	$i \backslash j$	ε	T	O	A	S	T
0	ε	0	1	2	3	4	5
1	M	1	1				
2	A	2					
3	R	3					
4	S	4					

Example

MARS $\rightarrow$ TOAST

		0	1	2	3	4	5
	$i \backslash j$	ε	T	O	A	S	T
0	ε	0	1	2	3	4	5
1	M	1	1				
2	A	2					
3	R	3					
4	S	4					

Example

$\texttt{MARS} \rightarrow \texttt{TOAST}$

		0	1	2	3	4	5
	$i \backslash j$	ε	T	O	A	S	T
0	ε	0	1	2	3	4	5
1	M	1	1	2			
2	A	2					
3	R	3					
4	S	4					

Example

MARS $\rightarrow$ TOAST

		0	1	2	3	4	5
	$i \backslash j$	ε	T	O	A	S	T
0	ε	0	1	2	3	4	5
1	M	1	1	2	3		
2	A	2					
3	R	3					
4	S	4					

Example

$$\texttt{MARS} \rightarrow \texttt{TOAST}$$

		0	1	2	3	4	5
	$i \backslash j$	ε	T	O	A	S	T
0	ε	0	1	2	3	4	5
1	M	1	1	2	3		
2	A	2	2				
3	R	3					
4	S	4					

Example

$$\texttt{MARS} \rightarrow \texttt{TOAST}$$

		0	1	2	3	4	5
	$i \backslash j$	ε	T	O	A	S	T
0	ε	0	1	2	3	4	5
1	M	1	1	2	3		
2	A	2	2	2			
3	R	3					
4	S	4					

Example

$\texttt{MARS} \rightarrow \texttt{TOAST}$

		0	1	2	3	4	5
	$i \backslash j$	ε	T	O	A	S	T
0	ε	0	1	2	3	4	5
1	M	1	1	2	3		
2	A	2	2	2	2		
3	R	3					
4	S	4					

Example

$$\texttt{MARS} \rightarrow \texttt{TOAST}$$

		0	1	2	3	4	5
	$i\backslash j$	ε	T	O	A	S	T
0	ε	0	1	2	3	4	5
1	M	1	1	2	3		
2	A	2	2	2	2		
3	R	3					
4	S	4					

Example

$$\texttt{MARS} \to \texttt{TOAST}$$

		0	1	2	3	4	5
	$i \backslash j$	ε	T	O	A	S	T
0	ε	0	1	2	3	4	5
1	M	1	1	2	3	4	5
2	A	2	2	2	2	3	4
3	R	3	3	3	3	3	4
4	S	4	4	4	4	3	4

Example

$$\texttt{MARS} \rightarrow \texttt{TOAST}$$

		0	1	2	3	4	5
	$i \backslash j$	ε	T	O	A	S	T
0	ε	0	1	2	3	4	5
1	M	1	1	2	3	4	5
2	A	2	2	2	2	3	4
3	R	3	3	3	3	3	4
4	S	4	4	4	4	3	4

Edit distance: 4

Example

MARS $\rightarrow$ TOAST

		0	1	2	3	4	5
	$i \backslash j$	ε	T	O	A	S	T
0	ε	0	1	2	3	4	5
1	M	1	1	2	3	4	5
2	A	2	2	2	2	3	4
3	R	3	3	3	3	3	4
4	S	4	4	4	4	3	4

Edit distance: 4 **Time:** ?

Example

$$\texttt{MARS} \rightarrow \texttt{TOAST}$$

	0	1	2	3	4	5
$i \backslash j$	ε	T	O	A	S	T
0 ε	0	1	2	3	4	5
1 M	1	1	2	3	4	5
2 A	2	2	2	2	3	4
3 R	3	3	3	3	3	4
4 S	4	4	4	4	3	4

Edit distance: 4 **Time:** $O(nm)$

A Possible Implementation

```
int edit_distance(std::string &s, std::string &t)
{
    std::array<std::array<int, t.size()+1>, s.size()+1> OPT;

    for(int i=0; i<=s.size(); i++) OPT[i][0] = i;
    for(int j=1; j<=t.size(); j++) OPT[0][j] = j;

    for(int i=1; i<=s.size(); i++)
        for(int j=1; j<=t.size(); j++)
            OPT[i][j] = std::min({OPT[i-1][j]+1, OPT[i][j-1]+1,
                              OPT[i-1][j-1] + ((s[i]==t[j])?0:1)});

    return OPT[s.size()][t.size()];
}
```

Reconstructing Solutions

- **Option 1:** Retrace optimal choices backwards.

	ε	T	O	A	S	T
ε	0	1	2	3	4	5
M	1	1	2	3	4	5
A	2	2	2	2	3	4
R	3	3	3	3	3	4
S	4	4	4	4	3	4

Reconstructing Solutions

- **Option 1:** Retrace optimal choices backwards.

	ε	T	O	A	S	T
ε	0	1	2	3	4	5
M	1	1	2	3	4	5
A	2	2	2	2	3	4
R	3	3	3	3	3	4
S	4	4	4	4	3	4

Reconstructing Solutions

- **Option 1:** Retrace optimal choices backwards.

	ε	T	O	A	S	T
ε	0	1	2	3	4	5
M	1	1	2	3	4	5
A	2	2	2	2	3	4
R	3	3	3	3	3	4
S	4	4	4	4	3	4

Reconstructing Solutions

- **Option 1:** Retrace optimal choices backwards.

	ε	T	O	A	S	T
ε	0	1	2	3	4	5
M	1	1	2	3	4	5
A	2	2	2	2	3	4
R	3	3	3	3	3	4
S	4	4	4	4	3	4

Reconstructing Solutions

- **Option 1:** Retrace optimal choices backwards.

	ε	T	O	A	S	T
ε	0	1	2	3	4	5
M	1	1	$\boxed{2}$	3	4	5
A	2	2	2	2	3	4
R	3	3	3	3	3	4
S	4	4	4	4	3	4

Reconstructing Solutions

- **Option 1:** Retrace optimal choices backwards.

	ε	T	O	A	S	T
ε	0	1	2	3	4	5
M	1	1	2	3	4	5
A	2	2	2	2	3	4
R	3	3	3	3	3	4
S	4	4	4	4	3	4

Reconstructing Solutions

- **Option 1:** Retrace optimal choices backwards.

	ε	T	O	A	S	T
ε	0	1	2	3	4	5
M	1	1	2	3	4	5
A	2	2	2	2	3	4
R	3	3	3	3	3	4
S	4	4	4	4	3	4

Reconstructing Solutions

- **Option 1:** Retrace optimal choices backwards.

	ε	T	O	A	S	T
ε	$\boxed{0}$	1	2	3	4	5
M	1	1	2	3	4	5
A	2	2	2	2	3	4
R	3	3	3	3	3	4
S	4	4	4	4	3	4

```
int i=s.size(), j=t.size();
while(i!=0 || j!=0)
{
    //Do something
    if(i>0 && OPT[i][j]==OPT[i-1][j]+1) i--; //Deletion
    else if(j>0 && OPT[i][j]==OPT[i][j-1]+1) j--; //Insertion
    else { i--; j--; } //Substitution
}
```

Reconstructing Solutions

- **Option 1:** Retrace optimal choices backwards.

	ε	T	O	A	S	T
ε	0	1	2	3	4	5
M	1	1	2	3	4	5
A	2	2	2	2	3	4
R	3	3	3	3	3	4
S	4	4	4	4	3	4

Reconstructing Solutions

- **Option 1:** Retrace optimal choices backwards.

	ε	T	O	A	S	T
ε	0	1	2	3	4	5
M	1	1	2	3	4	5
A	2	2	2	2	3	4
R	3	3	3	3	3	4
S	4	4	4	4	3	4

	MARS
• Change M to T	TARS
• Insert O	TOARS
• (Leave A unchanged)	TOARS
• Delete R	TOAS
• (Leave S unchanged)	TOAS
• Insert T	TOAST

Reconstructing Solutions

- **Option 2:** Esplicitly store (any of) the optimal choice(s) for each subproblem while filling the table.

		0	1	2	3	4	5
	$i\backslash j$	ε	T	O	A	S	T
0	ε	0	1	2	3	4	5
1	M	1	1	2	3	4	5
2	A	2	2	2	2	3	4
3	R	3	3	3	3	3	4
4	S	4	4	4	4	3	4

Reconstructing Solutions

- **Option 2:** Esplicitly store (any of) the optimal choice(s) for each subproblem while filling the table.

		0	1	2	3	4	5
	$i \backslash j$	ε	T	O	A	S	T
0	ε	0	1	2	3	4	5
1	M	1	1	2	3	4	5
2	A	2	2	2	2	3	4
3	R	3	3	3	3	3	4
4	S	4	4	4	4	3	4