Algorithms for the Edit Distance Between Trees

Shaofeng Jiang

Western University

sjian7@uwo.ca

December 5, 2017

Overview

Ordered Labelled Tree

An ordered labelled tree is a tree in which the nodes are labeled and the left-to-right order among siblings is significant.

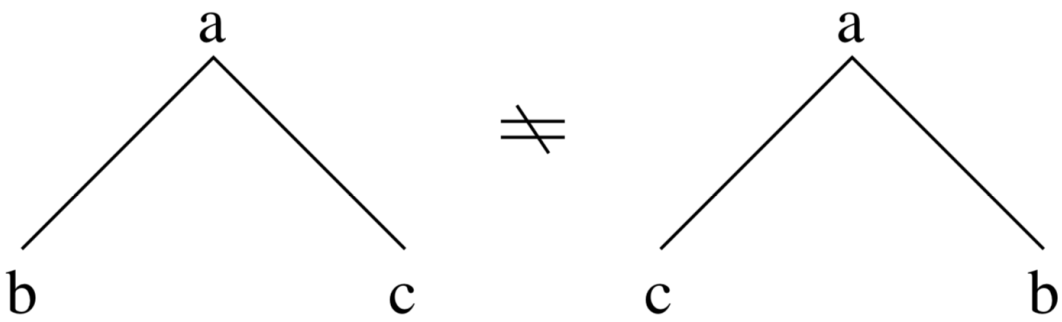


Figure: Ordered Labelled Tree

The ordered labelled tree can represent:

- an XML document
- a natural language parse
- RNA secondary structure
- and so on...

Pattern Matching in Trees

Given a Pattern tree and a Data tree, we may want to match the pattern tree to the data tree.

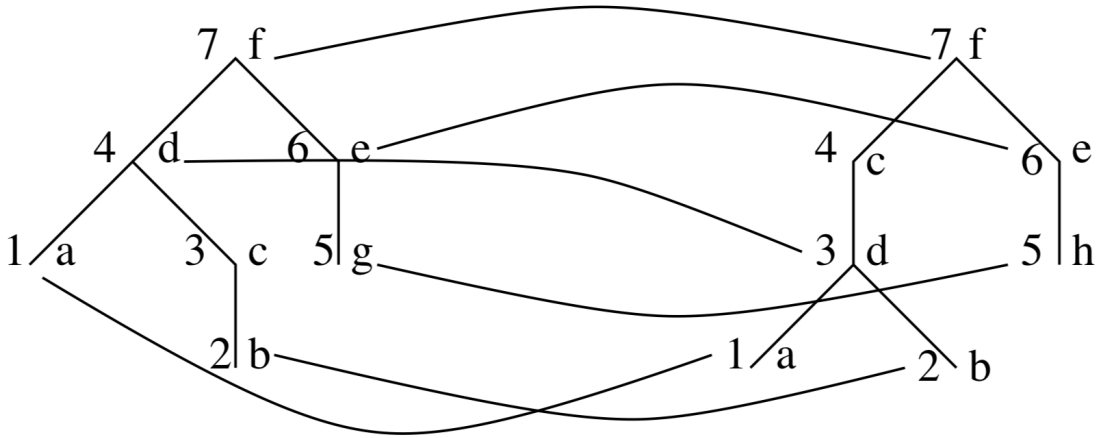
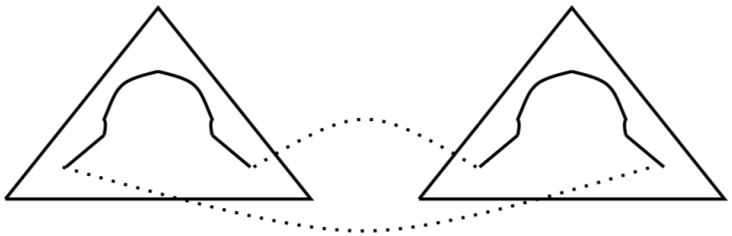


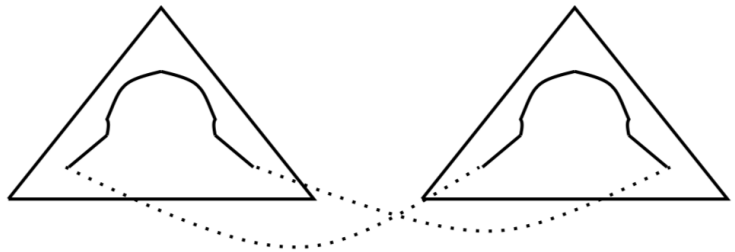
Figure: Pattern Matching in Trees

Pattern Matching in Trees

- **Input:** Two labelled ordered trees T and P.
- **Constraints:**
  - One-to-one relationship
  - Sibling order is preserved.

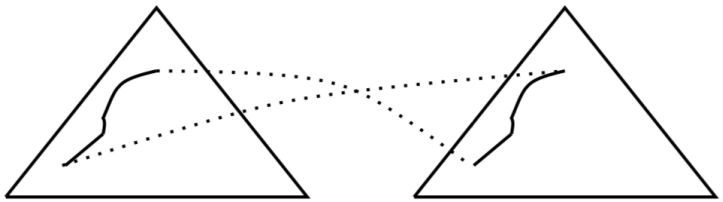


not valid

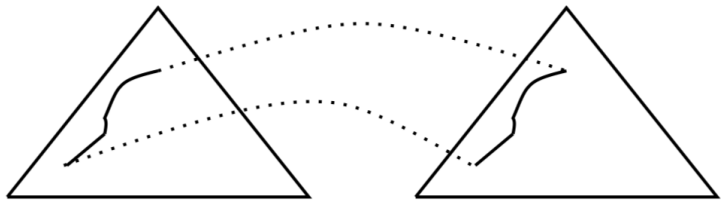


valid

  - Ancestor order is preserved.



not valid

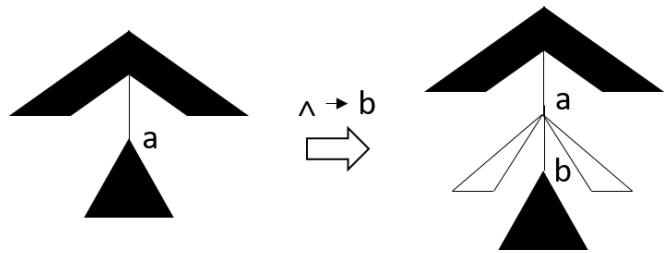


valid

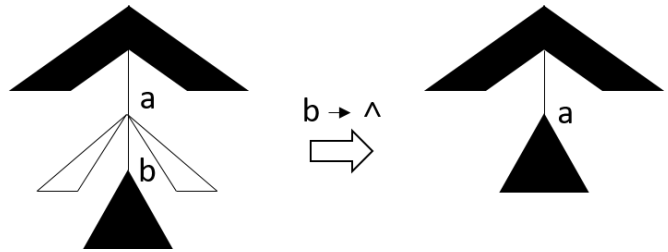
- **Output:** Mapping between T and P.

Edit Operations

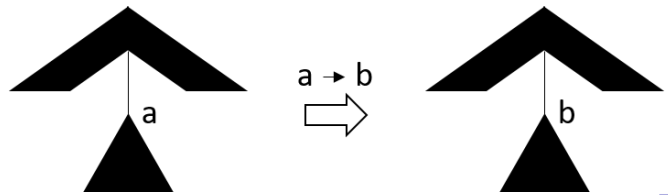
- Mappings on trees can be simplified into sequences of edit operations to transform one tree to another.
- Three valid edit operations supported:
  - **Insertion.** insertion of a node.



  - **Deletion.** deletion of a node

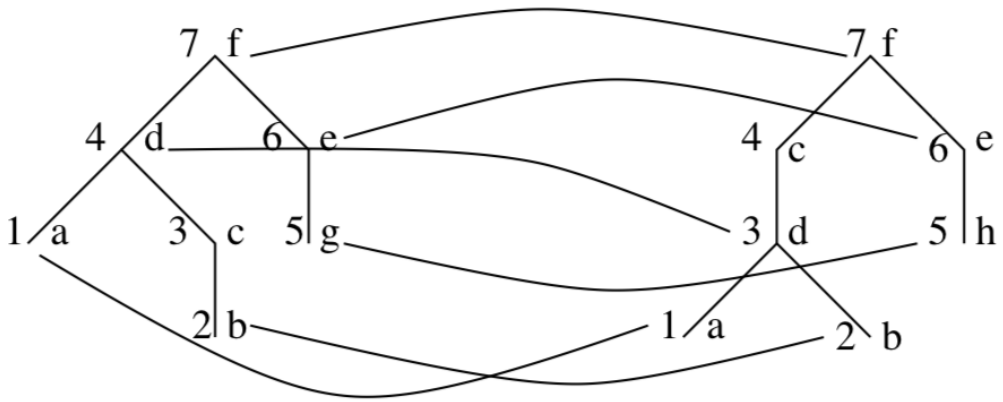


  - **Substitution.** replace a label of a node by another label.



Cost of Mapping and the Edit Distance

- The cost of a mapping is the sum of its insertion, deletion and substitutions.



Edit sequence: $(c \to \lambda), (g \to h), (\lambda \to c)$

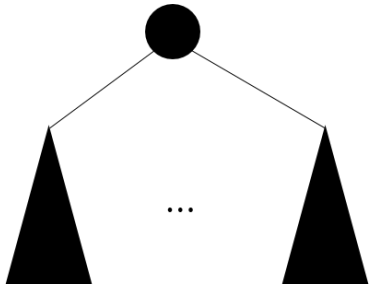
Figure: The Edit Sequence and its Cost

- The edit distance between two trees P and D is the minimum cost to change P to D via a sequence of basic edit operations.

Notation

- **Tree**
  - $l(f)$

  - $l(A_1 \circ A_2 \circ \cdots \circ A_n)$



Let T be a tree,

- $r(T)$ the root of T
- T° $T - r(T)$
- $lr(T)$ the root of the leftmost tree in the forest T°
- $rr(T)$ the root of the rightmost tree in the forest T°

Notation

- **Forest**
  - $l(f) \circ t$

  - $t \circ l(f)$

Let F be a forest,

- $|F|$ the number of nodes in the forest F
- $\#leaves(F)$ the number of leaves in the forest F
- $F(i)$ the sub-tree of F rooted at node i
- $F - i$ the forest after deleting node i
- $lr(F)$ the root of the leftmost tree in the forest F
- $rr(F)$ the root of the rightmost tree in the forest F

String Edit Distance and its Recursive Solution

- String edit distance is the minimum cost to change one string to another via a sequence of basic edit operations.
- The recursive solution of the string edit problem $d(S_1, S_2)$ is

$$d(S_1, S_2) = min \begin{cases} d(S_1 - u, S_2) + \delta(u, \varnothing) \\ d(S_1, S_2 - v) + \delta(\varnothing, v) \\ d(S_1 - u, S_2 - v) + \delta(u, v) \end{cases}$$

String Edit Distance and its Recursive Solution

- If u and v is both the first element of string S_1 and S_2, then it is a leftmost decomposition.

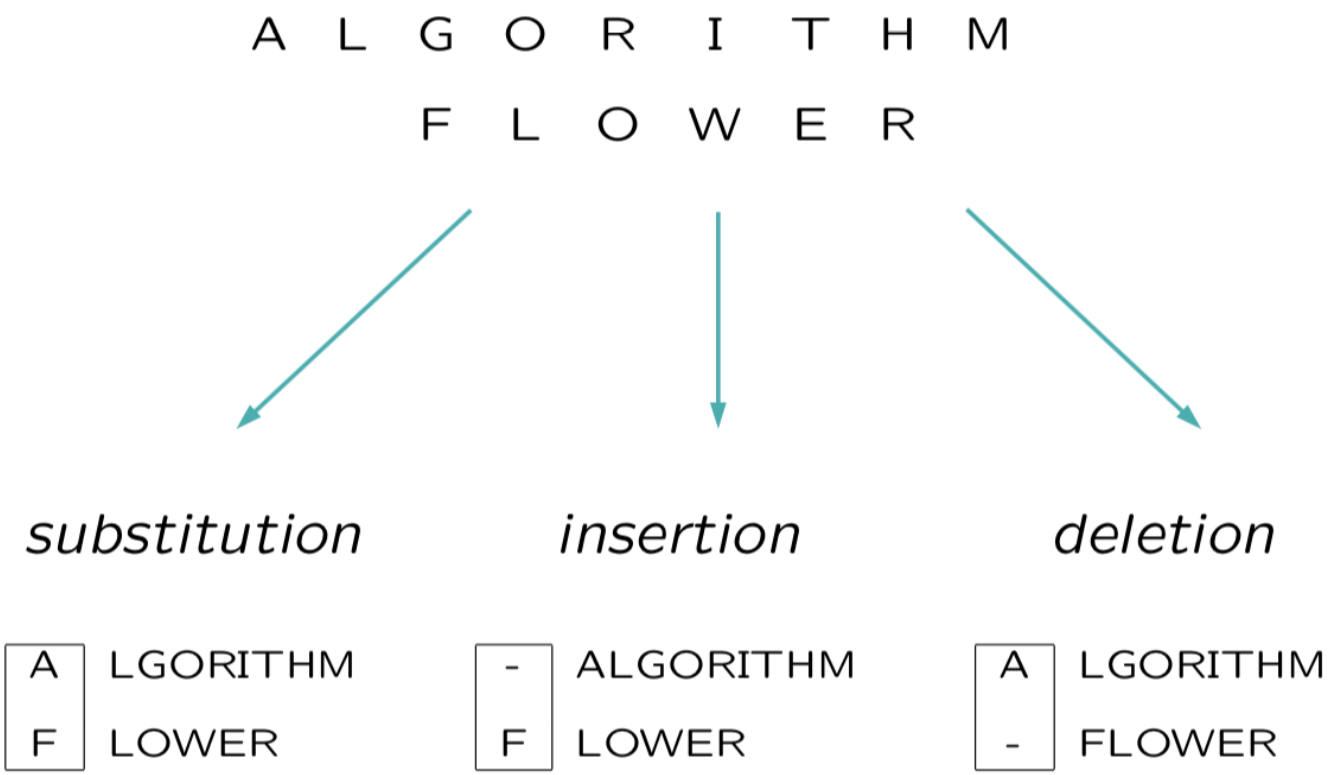


Figure: String Leftmost Decomposition

String Edit Distance and its Recursive Solution

- If u and v is both the last element of string S_1 and S_2, then it is a rightmost decomposition.

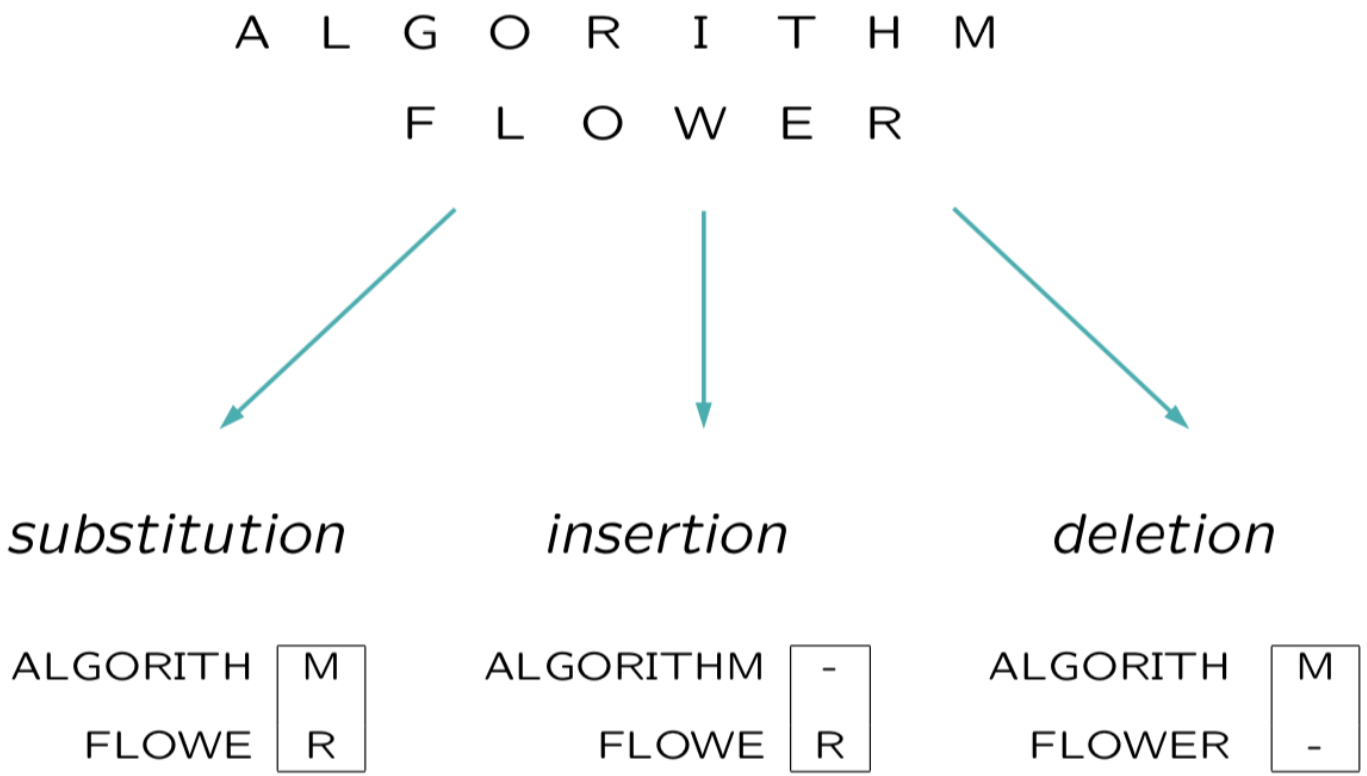


Figure: String Rightmost Decomposition

Tree Edit Distance and its Recursive Solution

For tree-to-tree distance, Let T_1 and T_2 are both trees of the form $l(f)$ and $l'(f')$,

$$d(l(f), l'(f')) = min \begin{cases} d(f, l'(f')) + \delta(l, \varnothing) \\ d(l(f), l'(f')) + \delta(\varnothing, l') \\ d(f, f') + \delta(l, l') \end{cases}$$

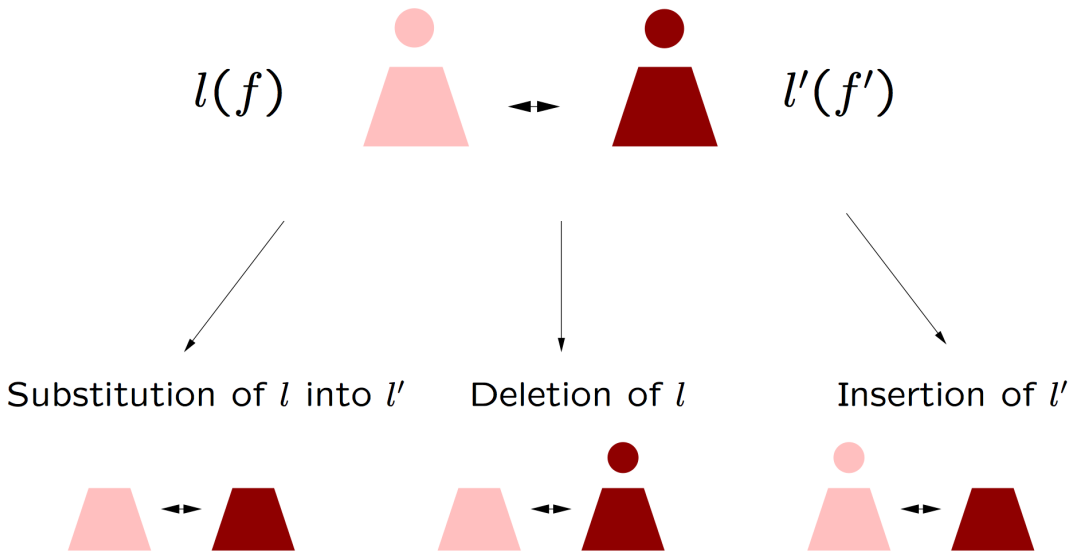


Figure: Tree-to-Tree Edit Distance

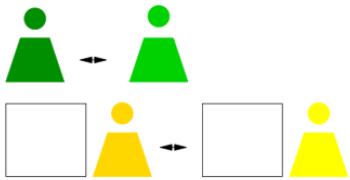
Tree Edit Distance and its Recursive Solution

For forest-to-forest distance, Let F_1 and F_2 are both trees of the form $l(f) \circ t$ and $l'(f') \circ t'$,

$$d(l(f) \circ t, l'(f') \circ t') = min \begin{cases} d(f \circ t, l'(f') \circ t') + \delta(l, \varnothing) \\ d(l(f) \circ t, f' \circ t') + \delta(\varnothing, l') \\ d(f \circ t, f' \circ t') + \delta(l, l') \end{cases}$$

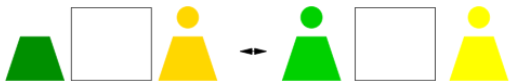
Leftmost decomposition

Substitution of l into l'



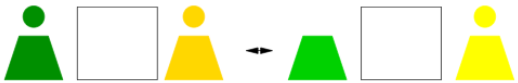
$\texttt{Distance}(l(f), l'(f'))$
$+$
$\texttt{Distance}(t, t')$

Deletion of l



$\texttt{Distance}(f \circ t, l'(f') \circ t') + \texttt{del}(l)$

Insertion of l'



$\texttt{Distance}(l(f) \circ t, f' \circ t') + \texttt{ins}(l')$

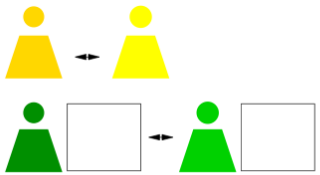
Tree Edit Distance and its Recursive Solution

For forest-to-forest distance, Let F_1 and F_2 are both trees of the form $l(f) \circ t$ and $l'(f') \circ t'$,

$$d(t \circ l(f), t' \circ l'(f')) = min \begin{cases} d(t \circ f, t' \circ l'(f')) + \delta(l, \varnothing) \\ d(t \circ l(f), t' \circ f') + \delta(\varnothing, l') \\ d(t \circ f, t' \circ f') + \delta(l, l') \end{cases}$$

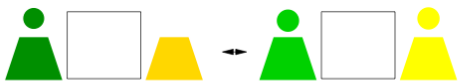
Rightmost decomposition

Substitution of l into l'



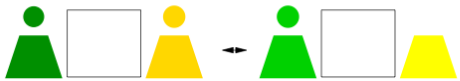
$\texttt{Distance}(l(f), l'(f'))$ + $\texttt{Distance}(t, t')$

Deletion of l



$\texttt{Distance}(t \circ f, t' \circ l'(f'))+\texttt{del}(l)$

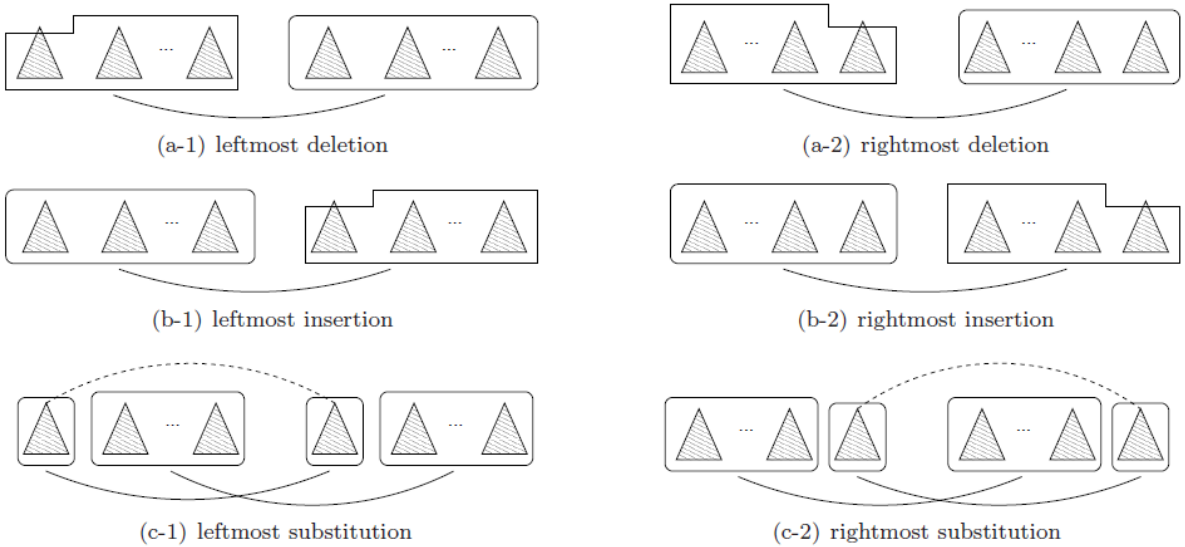
Insertion of l'



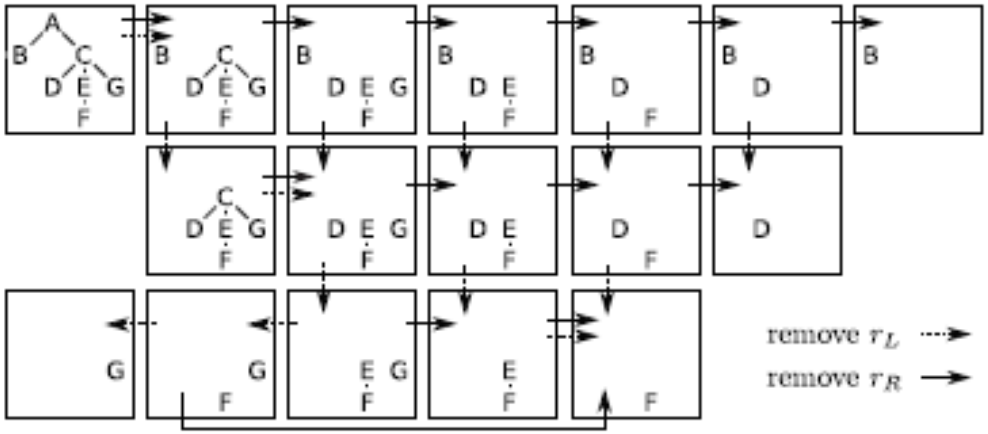
$\texttt{Distance}(t \circ l(f), t' \circ f')+\texttt{ins}(l')$

Relevant Sub-forests and Full Decomposition

- Relevant sub-forests

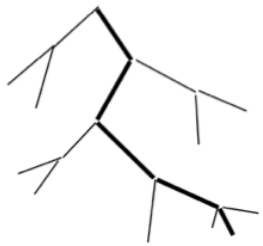


- The full decomposition

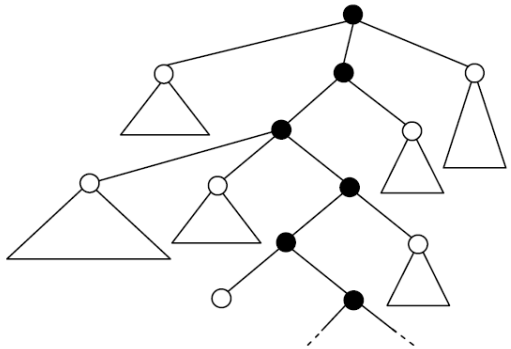


Root-leaf Path and Relevant Sub-trees

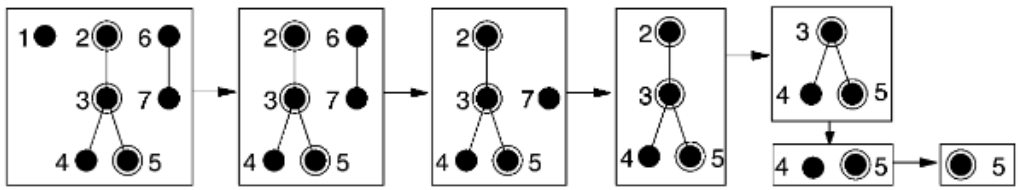
- Root-leaf path



- Relevant sub-trees

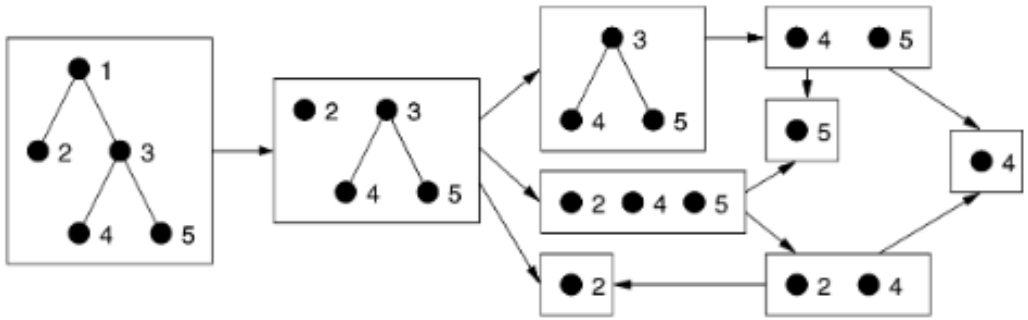


- Path decomposition

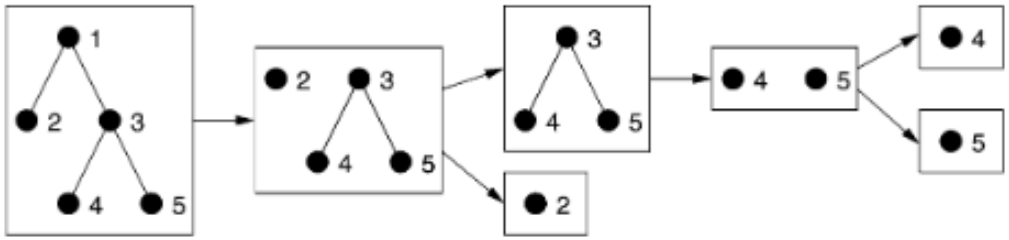


Path Decomposition

- Leftmost path decomposition



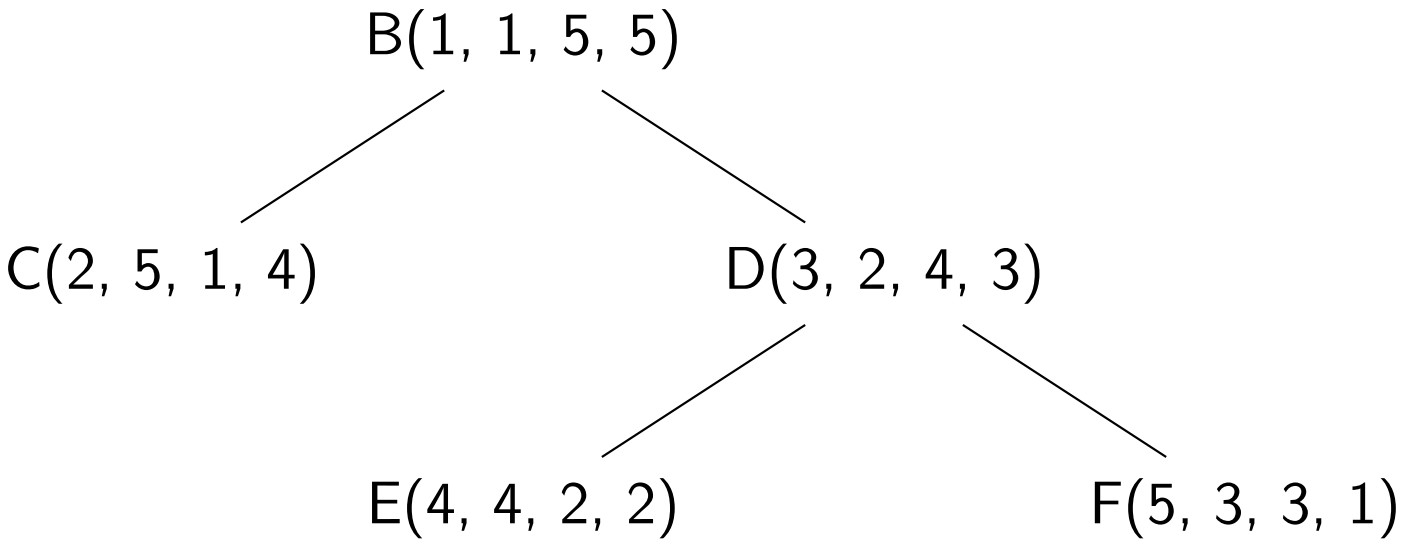
- Rightmost path decomposition



Tree Indexing

- $preOrder\_left\_to\_right \rightarrow preL$
- $preOrder\_right\_to\_left \rightarrow preR$
- $postOrder\_left\_to\_right \rightarrow postL$
- $postOrder\_right\_to\_left \rightarrow postR$

Consider a tree T(preL, preR, postL, postR)



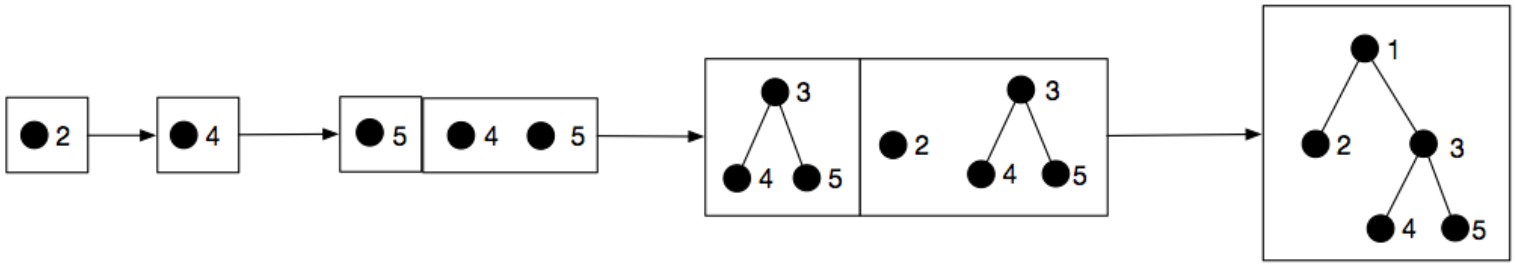
Note:

- $preL + postR = treeSize$
- $preR + postL = treeSize$

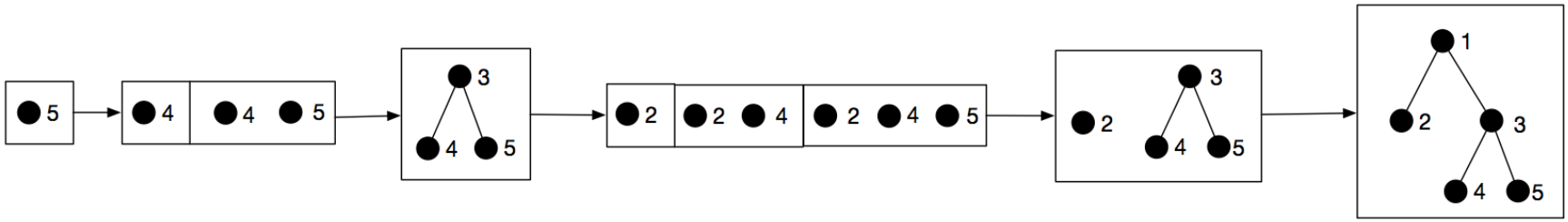
Bottom-up Enumeration

Two orders of the enumeration of the path decomposition:

- LR-postorder enumeration



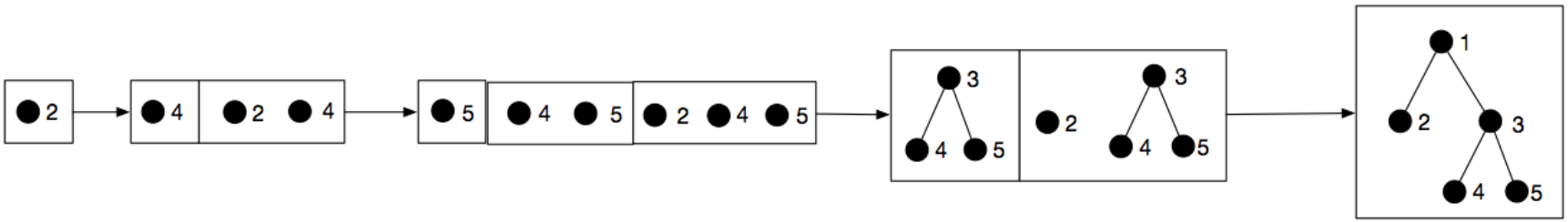
- RL-postorder enumeration



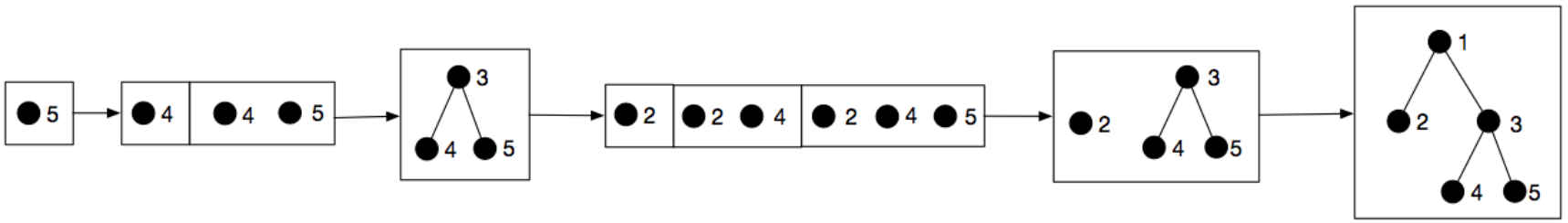
Bottom-up Enumeration

Two orders of the enumeration of the full decomposition:

- Prefix-suffix order



- Suffix-prefix order



A Simple Algorithm

Main Idea:

Enumerate each pairs of sub-forests in tree A and B in prefix-suffix post order or suffix-prefix post order.

Pseudocode:

Algorithm 1: Compute tree edit distance by enumerating all pairs in $O(m^2n^2)$ time.

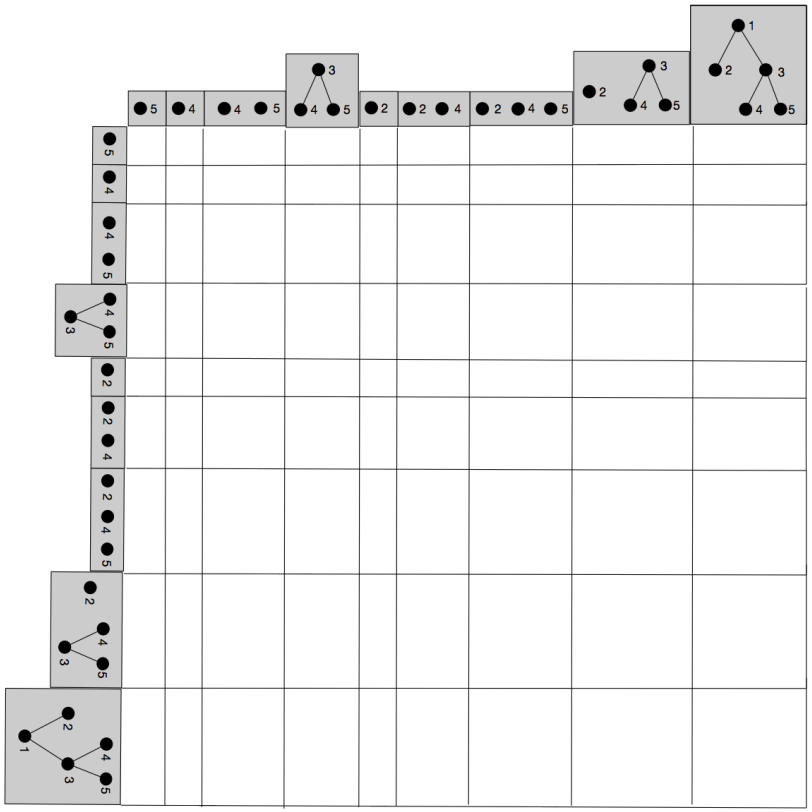
inputs : (T_1, T_2), with $|T_1| = m$ and $|T_2| = n$
output: $d(T_1[i], T_2[j])$ for $1 \leq i \leq m$ and $1 \leq j \leq n$

```
1 L1 ← POSTORDER(T1);
2 L2 ← POSTORDER(T2);
3 for i = 1 to |L1| do
4 |   for j = 1 to |L2| do
5 |   |   compute d(L1[i], L2[j]) as in Equation 2.3
6 |   end
7 end
8 d(T1[i], T2[j]) ← d(L1[|L1|][|L2|]);
9 return d(T1[i], T2[j]);
```

A Simple Algorithm

Implementation:

- Implemented in dynamic programming.
- Runs in $\mathcal{O}(m^2n^2)$ time using $\mathcal{O}(m^2n^2)$ space.

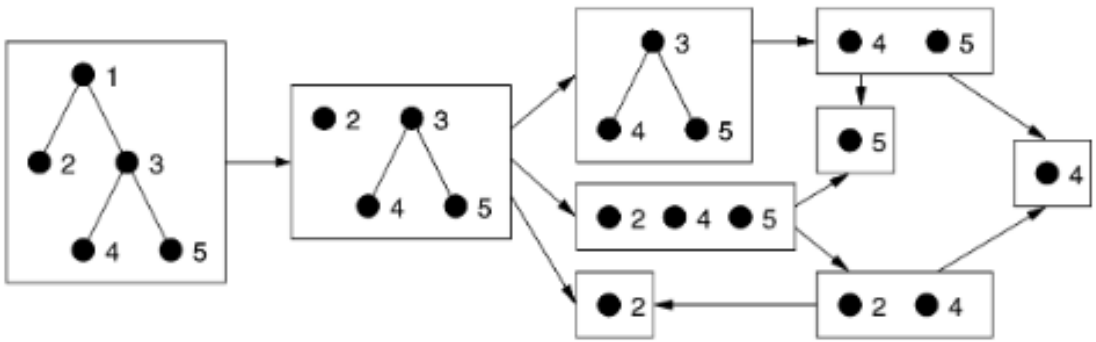


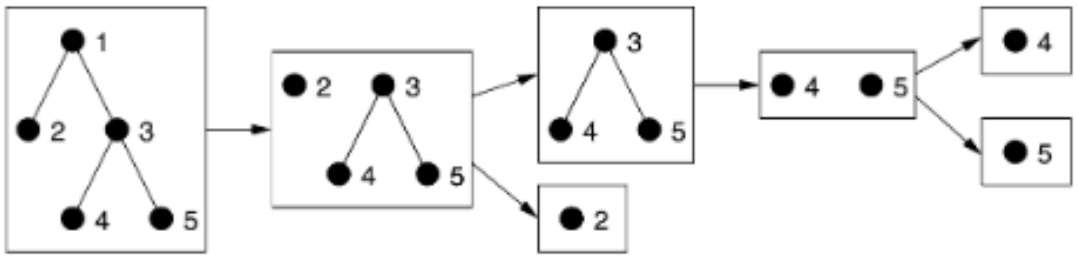
- The upper bound of the post-order enumeration method for the tree edit distance problem is $\mathcal{O}(n^4)$

Improved Algorithmic Path Strategies

Key Ideas:

- Different path decomposition creates different number of relevant sub-forests.
- Take advantage of the overlap among sub-forests that are contained in the same sub-tree, and the overlap of sub-trees as well.





Zhang and Shasha's Algorithm

Key Ideas:

- Fixed direction in each recursive calls.
  - Recursive right decomposition - leftmost paths.
  - Recursive left decomposition - rightmost paths.

(a) leftmost paths

(b) rightmost paths

Zhang and Shasha's Algorithm

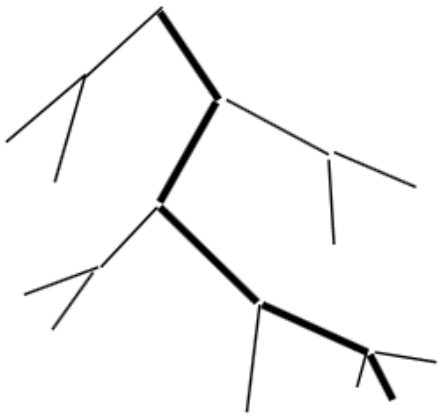
Implementation:

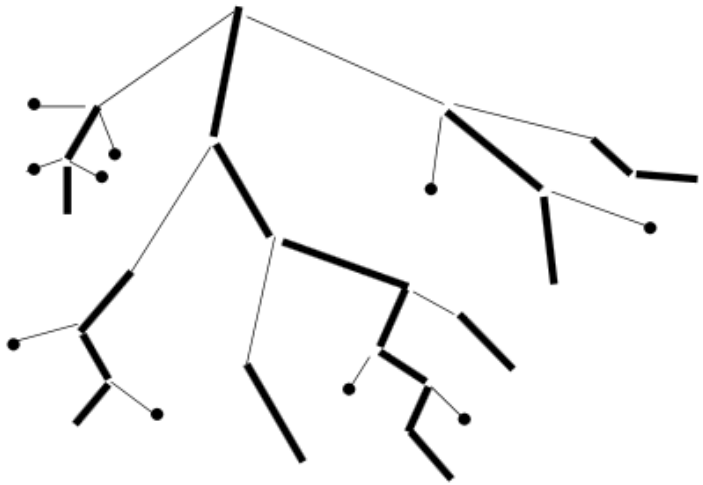
- Implemented via dynamic programming.
- Enumerates sub-trees pairs rooted at key roots.
  - Leftmost paths: either tree root or has a left sibling
  - Rightmost paths: either tree root or has a right sibling
- Uses temporary tables to store forest-forest distance and a permanent tables to store tree-tree distance.
- Runs in $\mathcal{O}(|A| * |B| * min\{depth(A), leaves(A)\} * min\{depth(B), leaves(B)\})$ time and $\mathcal{O}(|A| * |B|)$ space.

Klein's Algorithm

Key Ideas:

- Zhang and Shasha's algorithm depends on shapes of trees.
- Heavy paths generate the least number of sub-forests on one tree.





Klein's Algorithm

Implementation:

- Implemented via dynamic programming.
- applies heavy paths to the larger tree and fully decomposes the smaller tree.
- Uses temporary tables to store forest-forest distance and a permanent tables to store tree-tree distance.
- Runs in $\mathcal{O}(\log(|A|) * |B|^2)$ time using $\mathcal{O}(|A| * |B|)$ space.

Demaine's Algorithm

Key Ideas:

- Klein's algorithm consider the shape of one tree with no consideration on the other tree.
- Heavy paths can be applied on both trees.
- Always applies heavy paths on the larger tree.

Implementation:

- Implemented via dynamic programming.
- Runs in $\mathcal{O}(|A|^2 |B| (1 + \log(\frac{|B|}{|A|}))$ time using $\mathcal{O}(|A| * |B|)$ space.

Conclusion

	Time	**Space**	**Comments**
Simple Algorithm	$\mathcal{O}(n^4)$	$\mathcal{O}(n^4)$	first algorithm
Zhang	$\mathcal{O}(n^2 \log^2(n))$	$\mathcal{O}(n^2)$	efficient for balanced trees
Klein	$\mathcal{O}(n^3 \log(n))$	$\mathcal{O}(n^2)$	no consider smaller trees
Demaine	$\mathcal{O}(n^3)$	$\mathcal{O}(n^2)$	worse case is frequent

Table: State-of-the-Art Algorithms in the Tree Edit Distance Problem

Time Complexity(Naive Algorithm)

Prerequisite of the Analysis

- Lemma 1 implies dynamic programming.
- Each new subproblem can be computed in constant time.
- Time complexity is bounded by $\#subproblems\ of\ F_1 * \#subproblems\ of\ F_2$.

Count the subproblems of F

- $\#subproblem\ of\ F = \#(i,j) - deleted\ subforest$, $0 \leq i + j \leq |F|$
- $\#(i,j) - deleted\ subforest = \sum_{k=0}^{|F|} k = \mathcal{O}(|F|^2)$, since for each i there are $|F| - i$ choices for j
- numerous redundant subforests.

Conclusion

Time complexity is bounded by $\mathcal{O}(|F_1|^2 |F_2|^2)$.

Key Idea(Zhang and Shasha's Algorithm)

LEMMA 2

For tree-tree edit distance

$$\delta(l(f), l'(f')) = min \begin{cases} \delta(f, l'(f')) + \gamma(l \to \lambda) \\ \delta(l(f), f') + \gamma(\lambda \to l') \\ \delta(f, f') + \gamma(l \to l') \end{cases} \quad (1)$$

For forest-forest edit distance

$$\delta(l(f) \circ t, l'(f') \circ t') = min \begin{cases} \delta(f \circ t, l'(f') \circ t') + \gamma(l \to \lambda) \\ \delta(l(f) \circ t, f' \circ t') + \gamma(\lambda \to l') \\ \delta(l(f), l'(f')) + \delta(t, t') \end{cases} \quad (2)$$

Key Idea(Zhang and Shasha's Algorithm)

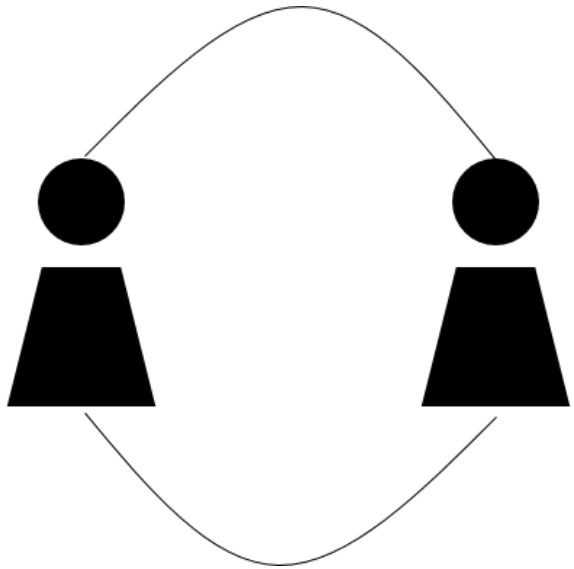
The lemma 2 is correct.

Proof.

For tree-tree distance, formula

$$\delta(I(f), I'(f')) \leq \delta(f, f') + \gamma(I, I')$$

represents a particular mapping of $\delta(I(f), I'(f'))$



Key Idea(Zhang and Shasha's Algorithm)

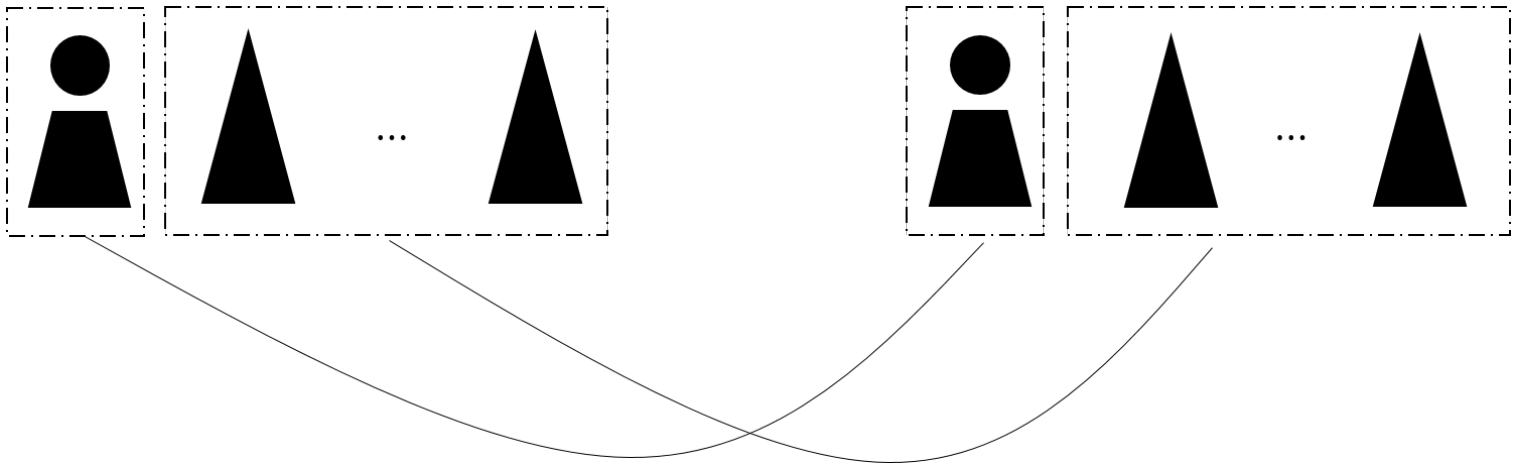
The lemma 2 is correct.

Proof.Continue

For forest-forest distance, formula

$$\delta(l(f) \circ t, l'(f') \circ t') \leq \delta(l(f), l'(f')) + \delta(t, t')$$

represents a particular mapping of $\delta(l(f) \circ t, l'(f') \circ t')$



Key Idea(Zhang and Shasha's Algorithm)

The lemma 2 is correct.

Another Proof.

Naive Algorithm

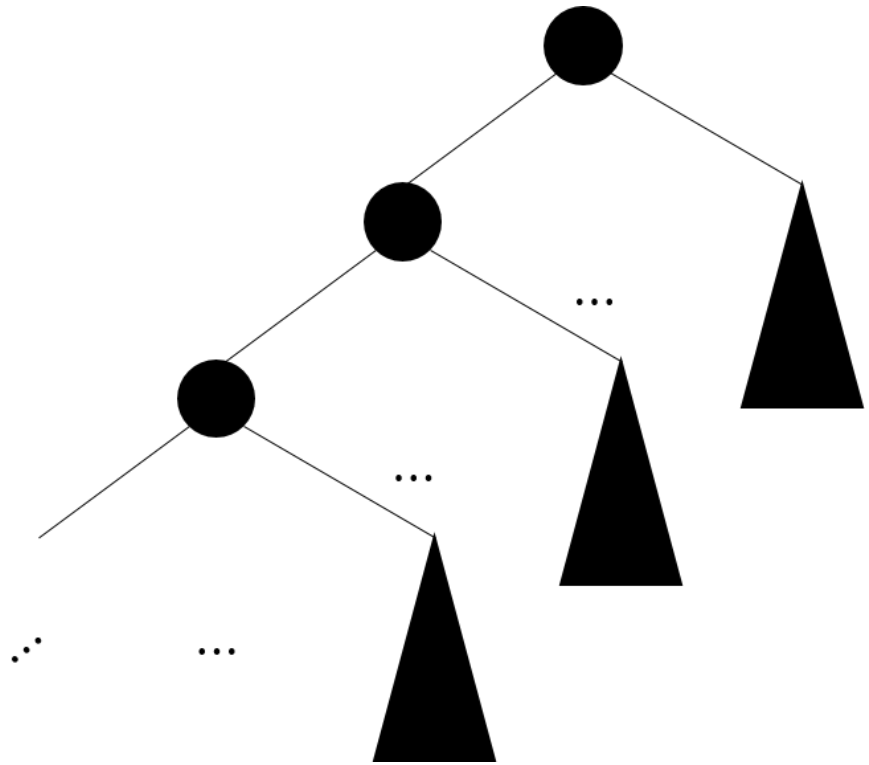
$$\delta(l(f) \circ t, l'(f') \circ t') = min \begin{cases} \delta(f \circ t, l'(f') \circ t') + \gamma(l \to \lambda) \\ \delta(l(f) \circ t, f' \circ t') + \gamma(\lambda \to l') \\ \delta(f, f') + \delta(t, t') + \gamma(l \to l') \end{cases}$$

Zhang-Shasha's Algorithm

$$\delta(l(f) \circ t, l'(f') \circ t') = min \begin{cases} \delta(f \circ t, l'(f') \circ t') + \gamma(l \to \lambda) \\ \delta(l(f) \circ t, f' \circ t') + \gamma(\lambda \to l') \\ \delta(l(f), l'(f')) + \delta(t, t') \end{cases}$$

Key Idea(Zhang and Shasha's Algorithm)

- dynamic programming style algorithm
- bottom-up procedure
- compute tree-tree distance between subtrees rooted at keyroots of both tree to avoid redundant computation.



Key Concept(Zhang and Shasha's Algorithm)

We define keyroot of T as

$$keyroots(T) = \{k \mid k \text{ is either } root(T) \text{ or } k \text{ has at least one siblings}\}.$$

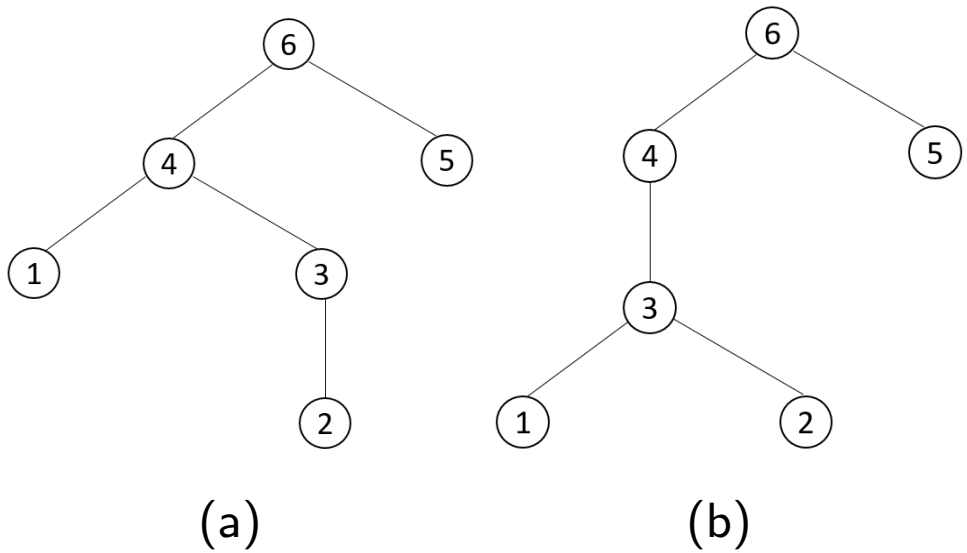


Figure: compute the edit distance between tree(a) and tree(b)

keyroot(T_1) = {3, 5, 6}
keyroot(T_2) = {2, 5, 6}

Time Complexity(Zhang and Shasha's Algorithm)

Prerequisite of the Analysis

- The time complexity is bounded by

$$\sum_{i=1}^{i=|keyroots(T_1)|} |T_1(keyroots(T_1)[i])| * \sum_{i=1}^{i=|keyroots(T_2)|} |T_2(keyroots(T_2)[i])|$$

Time Complexity(Zhang and Shasha's Algorithm)

Prerequisite of the Analysis

- Define for each node i,

$$cdepth(i) = |anc(i) \cap keyroots(T)|$$

Define for a tree T,

$$cdepth(T) = max\ cdepth(i)$$

- **Lemma 3**

$$|keyroot(T)| \leq |leaves(T)|$$

, since each leaf is the leftmost descendant of at most one member of keyroot(T).

- From definition and Lemma 3 $cdepth(i) \leq min(depth(T), leaves(T))$ for $1 \leq i \leq |T|$. Hence, $cdepth(T) \leq min(depth(T), leaves(T))$.

Time Complexity(Zhang and Shasha's Algorithm)

Prerequisite of the Analysis

- **Lemma 4**

$$\sum_{i=1}^{i=|keyroots(T)|} |T(keyroots(T)[i])| = \sum_{j=1}^{j=|T|} cdepth(j)$$

, each node j is counted cdepth(j) times.

Time Complexity(Zhang and Shasha's Algorithm)

Conclusion

- Time Complexity

$$\sum_{i=1}^{i=|keyroots(T_1)|} \sum_{j=1}^{j=|keyroots(T_2)|} T_1(keyroots(T_1)[i]) * T_2(keyroots(T_2)[j])$$

$$= \sum_{i=1}^{i=|T_1|} cdepth(i) * \sum_{j=1}^{j=|T_2|} cdepth(j)$$

$$\leq |T_1| * cdepth(T_1) * |T_2| * cdepth(T_2)$$

- Hence we can conclude that the time complexity is

$$\mathcal{O}(|T_1|*|T_2|*min(depth(T_1), leaves(T_1))*min(depth(T_2), leaves(T_2)))$$

Definition(Klein's Algorithm)

Leftmost decomposition

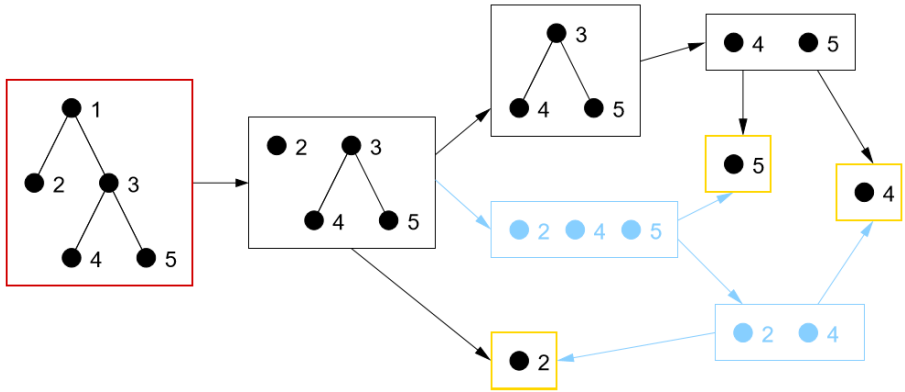
$$\delta(t \circ l(f), t' \circ l'(f')) = min \begin{cases} \gamma(l \to \lambda) + \delta(t \circ f, t' \circ l'(f')) \\ \gamma(\lambda \to l') + \delta(t \circ l(f), t' \circ f') \\ \delta(l(f), l'(f')) + \delta(t, t') \end{cases}$$

Rightmost decomposition

$$\delta(l(f) \circ t, l'(f') \circ t') = min \begin{cases} \gamma(l \to \lambda) + \delta(f \circ t, l'(f') \circ t') \\ \gamma(\lambda \to l') + \delta(l(f) \circ t, f' \circ t') \\ \delta(l(f), l'(f')) + \delta(t, t') \end{cases}$$

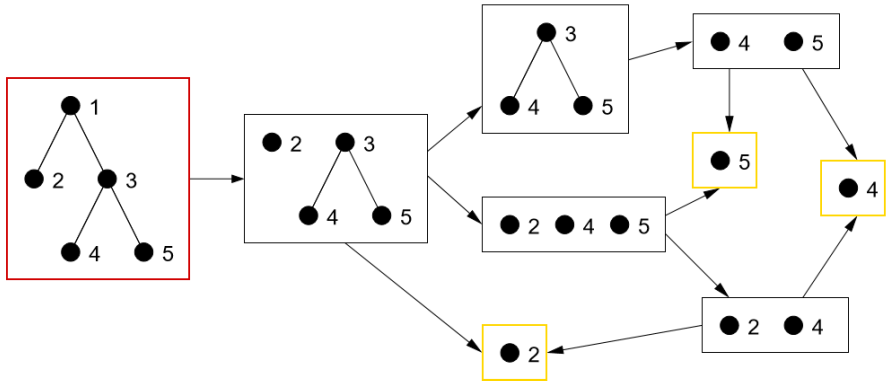
Observation(Klein's Algorithm)

Left Decomposition



gives 7 sub-forests.

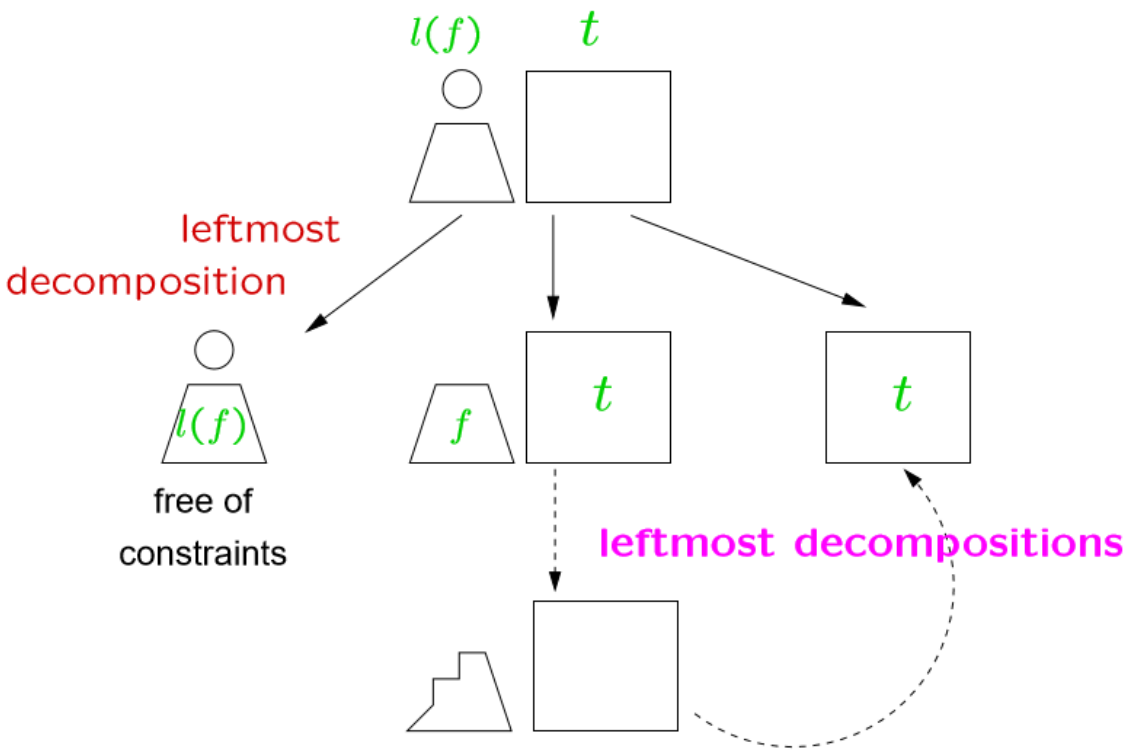
Right Decomposition



gives 9 sub-forests.

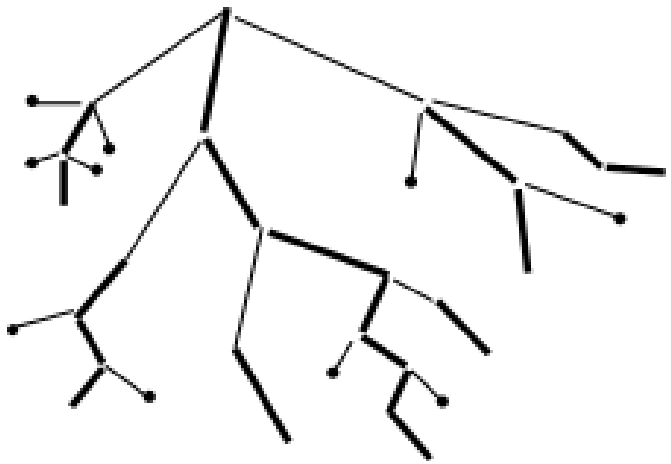
Key Idea(Klein's Algorithm)

- Optimize the subproblems of a forest($l(f) \circ t$) by eliminating nodes of f in $f \circ t$, so that $f \circ t$ and t share relevant forests as most as possible.
- However, this method also loses the ability to control the second forest($l'(f') \circ t'$)



Key Concept(Klein's Algorithm)

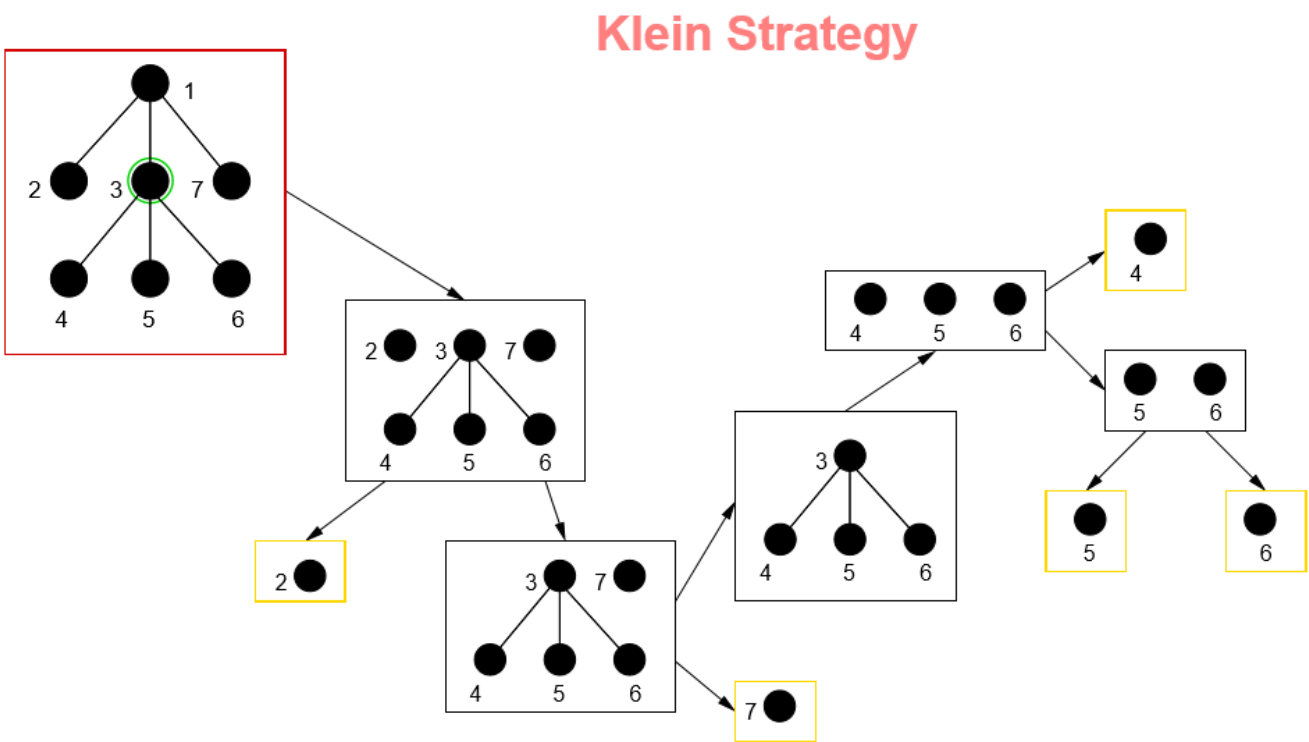
- **heavy child.** $heavy(i)$ denote the child of i having greatest weight.
- **heavy path.** The sequence of nodes $i, heavy(i), heavy(heavy(i)), \cdots$ defines a descending path which is called the heavy path.



Algorithm(Klein's Algorithm)

uses the heavy path as a guide for the decomposition of the forest. Let $I(f) \circ t$ be the forest to be decomposed. compute the heavy path P for $I(f) \circ t$

1) if I belongs to P, apply $t \circ I(f)$. Otherwise, apply $I(f) \circ t$.
2) apply this scheme recursively to all sub-forests of $I(f) \circ t$.



Time Complexity(Klein's Algorithm)

Prerequisite of the Analysis

- Define one of node i 's children with maximal number of descendants and marks it as *heavy*, and we mark all the other children of i as *light*
- The time complexity is bounded by

$$\sum_{i \in light(T_1)} |T_1(i)| * |T_2|^2$$

Time Complexity(Klein's Algorithm)

Prerequisite of the Analysis

- Define ldepth(i) to be the number of light nodes that are **proper ancestors** of i in F.
- **Theorem 1**(Sleator and Tarjan, 1983)
  Let v be any vertex of tree F. There is at most one heavy edge entering v, and there are at most $log_2 |F|$ light edges on the tree path from v to root(v).
- According to theorem 1, for each node i in tree F, we have

$$ldepth(i) \leq log_2 |F|$$

Time Complexity(Klein's Algorithm)

Conclusion

- Subforests of T_1

$$\begin{aligned}
&\sum_{i \in light(T_1)} |T_1(i)| \\
&\leq \sum_{i \in T_1} (1 + ldepth(i)) \\
&\leq \sum_{i \in T_1} (log_2 |T_1| + \mathcal{O}(1)) \\
&= \mathcal{O}(|T_1| \, log_2 |T_1|)
\end{aligned}$$

- Hence the time complexity is $\mathcal{O}(|T_1| \, log_2 |T_1| * |T_2|^2)$

Decomposition Strategies

Lemma 5

- if F or F' is an empty forest:
  $R(\epsilon, l(f) \circ t) = (\epsilon, l(f) \circ t) \cup R(\epsilon, f \circ t)$ *whenever* $\phi(\epsilon, l(f) \circ t) = left$,
  $R(\epsilon, t \circ l(f)) = (\epsilon, t \circ l(f)) \cup R(\epsilon, t \circ f)$ *whenever* $\phi(\epsilon, t \circ l(f)) = right$,
  $R(l(f) \circ t, \epsilon) = (l(f) \circ t, \epsilon) \cup R(f \circ t, \epsilon)$ *whenever* $\phi(l(f) \circ t, \epsilon) = left$,
  $R(t \circ l(f), \epsilon) = (t \circ l(f), \epsilon) \cup R(t \circ f, \epsilon)$ *whenever* $\phi(t \circ l(f), \epsilon) = right$,
- *if* $(F, F') = (l(f) \circ t, l'(f') \circ t')$ *and* $\phi(F, F') = left$, *then*
  $R(F, F') = (F, F') \cup R(f \circ t, F') \cup R(F, f' \circ t') \cup R(l(f), l'(f')) \cup R(t, t')$
- *if* $(F, F') = (t \circ l(f), t' \circ l'(f'))$ *and* $\phi(F, F') = right$, *then*
  $R(F, F') = (F, F') \cup R(t \circ f, F') \cup R(F, t \circ f) \cup R(l(f), l'(f')) \cup R(t, t')$

Lower Bound of Relevant Forests

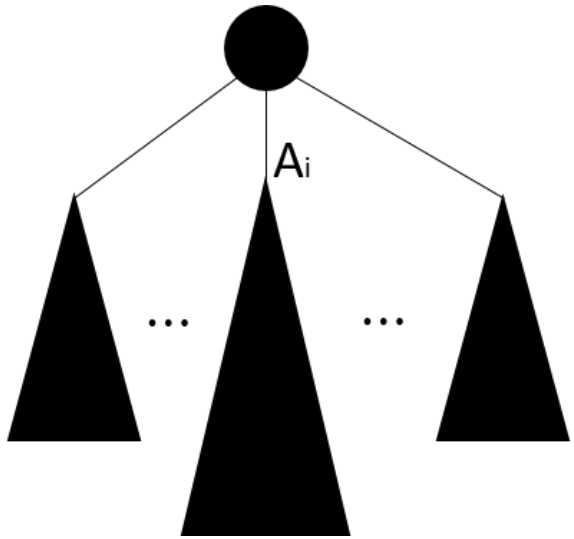
Lemma 6

- $R(l(f)) = \{l(f)\} \cup R(f)$, no matter what the direction is.
  That is to say, $\#ref(l(f)) \geq |l(f)|$
- $R(l(f) \circ t) = \{(l(f) \circ t)\} \cup R(f \circ t) \cup R(l(f)) \cup R(t)$, if the direction is left.
  That is to say, $\#ref(l(f) \circ t) \geq \#ref(l(f)) + \#ref(t)$
- $R(t \circ l(f)) = \{(t \circ l(f))\} \cup R(t \circ f) \cup R(l(f)) \cup R(t)$, if the direction is right.

Lower Bound of Relevant Forests

Lemma 7

Given a tree A=$l(A_1 \circ A_2 \circ \cdots \circ A_n)$, for any strategy we have,
$\#rel(A) \geq |A| - |A_i| + \#rel(A_1) + \cdots + \#rel(A_n)$
, where i $\in$ [1 $\cdots$ n] is such that size of A_i is maximal.



Lower Bound of Relevant Forests

Lemma 7

Given a tree A=$I(A_1 \circ A_2 \circ \cdots \circ A_n)$, for any strategy we have,
$\#rel(A) \geq |A| - |A_i| + \#rel(A_1) + \cdots + \#rel(A_n)$
, where i $\in$ [1 $\cdots$ n] is such that size of A_i is maximal.

Proof.

From lemma 6, $R(I(f)) = I(f) \cup R(f)$. That is to say,
$\#rel(I(f)) = 1 + \#rel(f)$.

Then let $F = A_1 \circ \cdots \circ A_n$. To prove lemma 7, we first prove that
$\#ref(F) \geq |F| - |A_i| + \#ref(A_1) + \cdots + \#ref(A_n)$

Lower Bound of Relevant Forests

Lemma 7

Given a tree A=$l(A_1 \circ A_2 \circ \cdots \circ A_n)$, for any strategy we have,
$\#rel(A) \geq |A| - |A_i| + \#rel(A_1) + \cdots + \#rel(A_n)$
, where i $\in$ [1 $\cdots$ n] is such that size of A_i is maximal.

Proof.

For case n = 1, $\#ref(A_1) \geq |A_1| - |A_1| + \#ref(A_1)$

For case $n > 1$, let $F = l(g) \circ A_2 \cdots A_n$, **t be** $A_2 \circ \cdots \circ A_n$ then by lemma 5,
$R(F) = \{F\} \cup R(g \circ t) \cup R(A_1) \cup R(t)$
From lemma 6, we know,
$R(g \circ t) \geq |g| + |t| \geq min\{|g|, |t|\}$
By applying the inequality, we have
$\#ref(F) \geq 1 + \#ref(A_1) + \#ref(t) + min\{|g|, |t|\}$

Lower Bound of Relevant Forests

Lemma 7

Given a tree A=$l(A_1 \circ A_2 \circ \cdots \circ A_n)$, for any strategy we have,
$\#rel(A) \geq |A| - |A_i| + \#rel(A_1) + \cdots + \#rel(A_n)$
, where i $\in [1 \cdots$ n] is such that size of A_i is maximal.

Proof.

Applying induction hypothesis for $t = A_2 \circ \cdots \circ A_n$
$\#ref(F) \geq 1 + \#ref(A_1) + \cdots + \#ref(t) + |t| - |A_j| + min\{|g|, |t|\}$

Then we need to prove that, $1 + |t| - |A_j| + min\{|g|, |t|\} \geq |F| - |A_i|$, where A_i is the size of the largest sub-tree in forest $A_1 \circ \cdots \circ A_n$, while A_j is the size of the largest sub-tree in forest $A_2 \circ \cdots \circ A_n$

Lower Bound of Relevant Forests

Lemma 7

Given a tree A=$l(A_1 \circ A_2 \circ \cdots \circ A_n)$, for any strategy we have,
$\#rel(A) \geq |A| - |A_i| + \#rel(A_1) + \cdots + \#rel(A_n)$
, where i $\in$ [1 $\cdots$ n] is such that size of A_i is maximal.

Proof.

In the case when $|g| \leq |t|$, then we have $1 + |t| + |g| = |F|$. Since $|A_j| \leq |A_i|$, it follows that $1 + |t| - |A_j| + min\{|g|, |t|\} \geq |F| - |A_i|$

In the case when $|g| > |t|$, this implies that A_1 is the largest subtree of A. So $i = 1$ and $|F| - |A_i| = |t|$, that proves
$1 + |t| - |A_j| + min\{|g|, |t|\} \geq |F| - |A_i|$

Lower Bound of Relevant Forests

Lemma 7

Given a tree A=$I(A_1 \circ A_2 \circ \cdots \circ A_n)$, for any strategy we have,
$\#rel(A) \geq |A| - |A_i| + \#rel(A_1) + \cdots + \#rel(A_n)$
, where i $\in$ [1 $\cdots$ n] is such that size of A_i is maximal.

Proof.

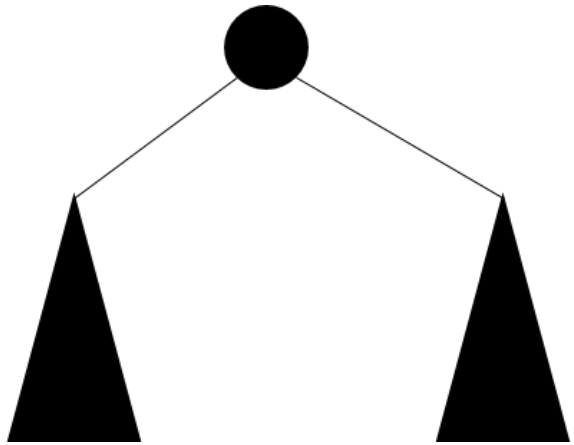
To sum up, the number of relevant sub-forest of a tree A is

$$\begin{aligned}\#ref(A) &= 1 + \#rel(F) \\ &\geq 1 + |F| - |A_i| + \#rel(A_1) + \cdots + \#rel(A_n) \\ &= |A| - |A_i| + \#rel(A_1) + \cdots + \#rel(A_n)\end{aligned}$$

Lower Bound of Relevant Forests

Lemma 8 $\#rel(A)$ has a lower bound in $\mathcal{O}(nlog(n))$

Proof. Let T_n be a complete balanced binary tree of size n, we prove that $\#ref(T_n) \geq \frac{(n+1)log_2(n+1)}{2}$



For case n $= 1$, $rel(T_n) = 1$, consistent with lemma 8.

Lower Bound of Relevant Forests

Lemma 8 $\#rel(A)$ has a lower bound in $\mathcal{O}(nlog(n))$

Proof. Let T_n be a complete balanced binary tree of size n, we prove that $\#ref(T_n) \geq \frac{(n+1)log_2(n+1)}{2}$
For case $n > 1$, by lemma 7, we have

$$\#rel(T_n) \geq n - m + 2 * \#ref(T_m)$$

, where $m = (n-1)/2$. T_n is of the form $l(A \circ B)$, where A and B are two complete balanced tree of size m. By induction hypothesis for T_m, it converts to

$$\begin{aligned}
\#ref(T_n) &\geq n - m + (m+1)log_2(m+1) \\
&= \frac{(n+1)(log_2(m+1)+1)}{2} \\
&= \frac{(n+1)(log_2(2m+1)+1)}{2} \\
&= \frac{(n+1)(log_2(n+1))}{2}
\end{aligned}$$

Lower Bound of Relevant Forests

Lemma 9

Let A and B be two trees of size n. For any decomposition strategy, $\#rel(A, B)$ has a lower bound in $\mathcal{O}(n^2 log^2(n))$

Cover Strategies

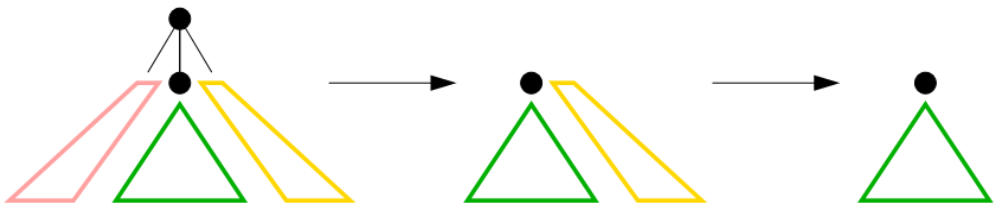
Cover

Let F be a tree, for each node i in F

- if $deg(i) = 0$ or $deg(i) = 1$, then $r(i) \in \{right, left\}$.
- if $deg(i) > 1$, then r(i) is a favourite child of i.

Cover Strategies

- if $deg(i) = 0$ or $deg(i) = 1$, then $\phi(A(i), G) = r(i)$, for each forest G of B.
- Otherwise,let A(i) is of the form $I(A_1 \circ \cdots A_p \circ \cdots \circ A_n)$
  - $\phi(T \circ A_p \cdots \circ A_n, G) = left$, for each forest T of $A_1 \circ \cdots \circ A_{p-1}$
  - $\phi(A_p \circ T, G) = right$, for each forest T of $A_{p+1} \circ \cdots \circ A_n$



leftmost decompositions rightmost decompositions

Cover Strategies

Zhang-Shasha Algorithms Cover Strategies

- for any node of degree 0 or 1, the direction is right($t \circ l(f)$).
- for any other node, the favorite child is the leftmost child. That is to say, the direction is right($t \circ l(f)$).

Klein's Algorithm Cover Strategies

- for any node of degree 0 or 1, the direction is left($l(f) \circ t$).
- for any other node, the favorite child is the root of the heaviest subtree.

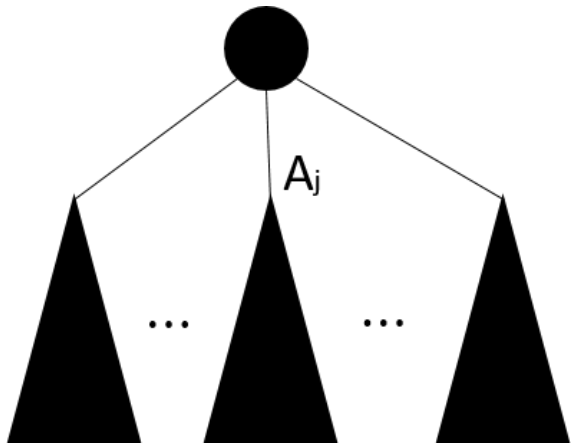
(a) (b)

Exact Number of Relevant Forests

Lemma 10

Let $A = l(A_1 \circ \cdots \circ A_n)$ be a cover tree such that $n = 1$ or the root of A_j is the favourite child.

$$\#rel(A) = |A| - |A_j| + \#rel(A_1) + \cdots + \#rel(A_n)$$



Upper Bound of Relevant Forests

Lemma 11

Let $A = I(A_1 \circ \cdots \circ A_n)$ be a cover tree

$$\#rel(A) \leq \frac{n(n+3)}{2} - \sum_{i \in A} |A(i)|$$

In average

$$\frac{1}{2}n^2 + \frac{\sqrt{\pi}}{2}n^{\frac{3}{2}} + \mathcal{O}(n)$$

Happens of a Cover Strategy on a tree to Another Tree

Lemma 12

Given a pair of tree (A, B) provided with a cover for A, all relevant forests of A fall within three categories:

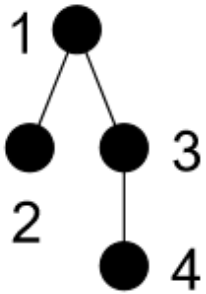
- α those that are compared with all rightmost forests of B,
- β those that are compared with all leftmost forests of B,
- γ those that are compared with all forests of B.

- $\#left(B)$: number of leftmost subforests of B
- $\#right(B)$: number of rightmost subforests of B
- $\#speical(B)$: number of all subforests of B

Note: $\#left(B)$, $\#right(B)$, $\#special(B)$ are known, depended by the structure of B

Happens of a Cover Strategy on a tree to Another Tree

- $\#left(B)$: number of leftmost subforests of B
- $\#right(B)$: number of rightmost subforests of B
- $\#speical(B)$: number of all subforests of B



$$\#\mathtt{right}(B) = 4 - 2 + 1 + 2 = 5$$
$$\#\mathtt{left}(B) = 4 - 1 + 1 + 2 = 6$$
$$\#\mathtt{special}(B) = 4 * 7/2 - 4 - 1 - 2 - 1 = 6$$

Status of Nodes in a Cover Tree

- **Free** : nodes that do not receive anything from the parent
- **Left** : nodes that inherit leftmost forests of B.

$$Left(A(i)) = (A(i), all\ leftmost\ forests\ of\ B)$$

- **Right** : nodes that inherit rightmost forests of B.

$$Right(A(i)) = (A(i), all\ rightmost\ forests\ of\ B)$$

- **All** : nodes that inherit all subforests of B from the parent

$$All(A(i)) = (A(i), all\ subforests\ of\ B)$$

Note1: The status of a node depends of the direction and of the heritage from parent.
Note2: By definition, $\#rel(A, B)$ *equals* $Free(A)$.

Number of Relevant Forests of A Pair of Tree

LEMMA 13

Let (A, B) be a pair of trees. A being a cover tree.

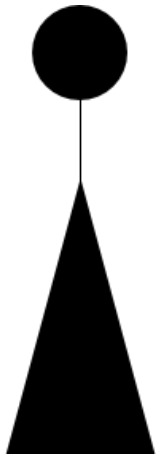
1) If A is reduced to a single node whose direction is right.
   - Free(A) = Left(A) = $\#left(B)$
   - All(A) = Right(A) = $\#special(B)$
2) If A is reduced to a single node whose direction is left.
   - Free(A) = Right(A) = #right(B)
   - All(A) = Left(A) = #special(B)

Number of Relevant Forests of A Pair of Tree

LEMMA 13

Let (A, B) be a pair of trees. A being a cover tree.

3) If $A = l(A')$ and the direction of l is right
   - Free(A) = Left(A) = $\#left(B) + Right(A')$
     ?? Free(A) = Left(A) = $\#left(B) + Left(A')$
   - All(A) = Right(A) = $\#special(B) + All(A')$
4) If $A = l(A')$ and the direction of l is left
   - Free(A) = Right(A) = $\#right(B) + Left(A')$
     ?? Free(A) = Right(A) = $\#left(B) + Right(A')$
   - All(A) = Left(A) = $\#special(B) + All(A')$

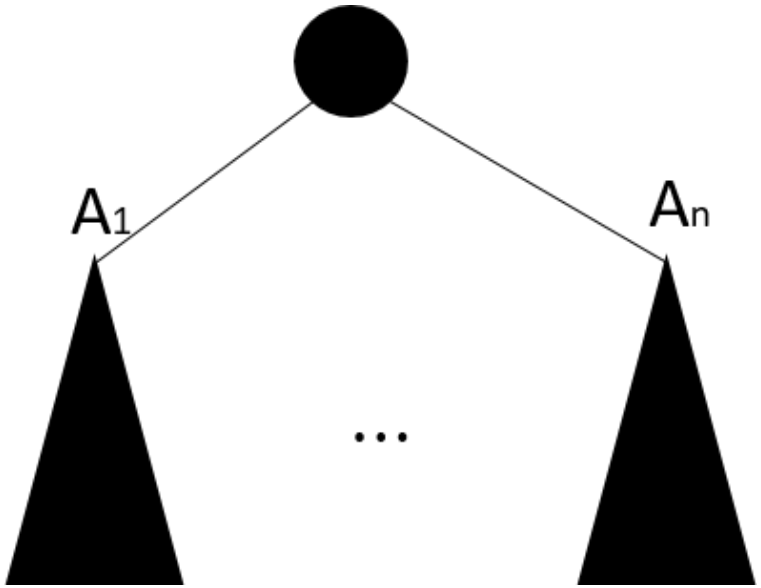


Number of Relevant Forests of A Pair of Tree

LEMMA 13

Let (A, B) be a pair of trees. A being a cover tree.

5) If $A = l(A_1 \circ \cdots \circ A_n)$ and the favourite child is A_1
   - Free(A) = Left(A) = $\sum_{i>1} Free(A_i) + Left(A_1) + \#left(B)(|A| - |A_1|)$
   - All(A) = Right(A) = $\sum_{i>1} Free(A_i) + All(A_1) + \#special(B)(|A| - |A_1|)$
6) If $A = l(A_1 \circ \cdots \circ A_n)$ and the favourite child is A_n
   - Free(A) = Left(A) = $\sum_{i<n} Free(A_i) + Left(A_n) + \#left(B)(|A| - |A_n|)$
   - All(A) = Right(A) = $\sum_{i<n} Free(A_i) + All(A_n) + \#special(B)(|A| - |A_n|)$

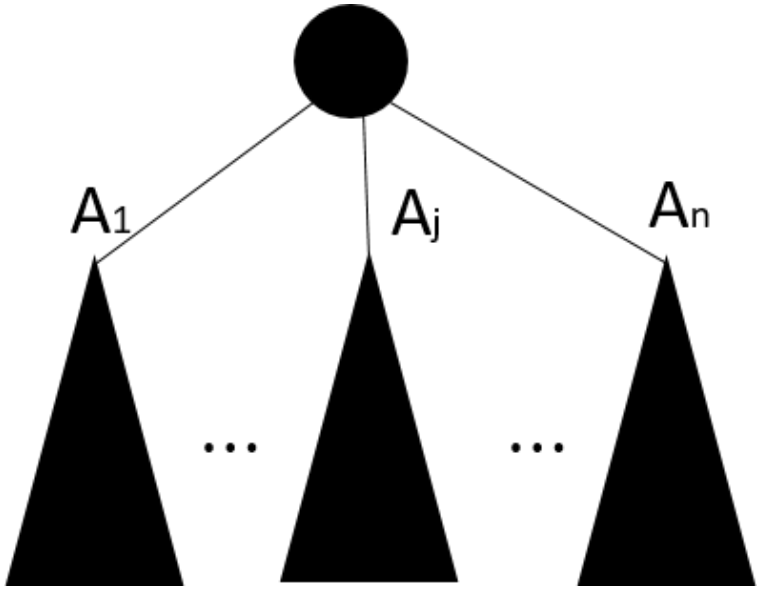


Number of Relevant Forests of A Pair of Tree

LEMMA 13

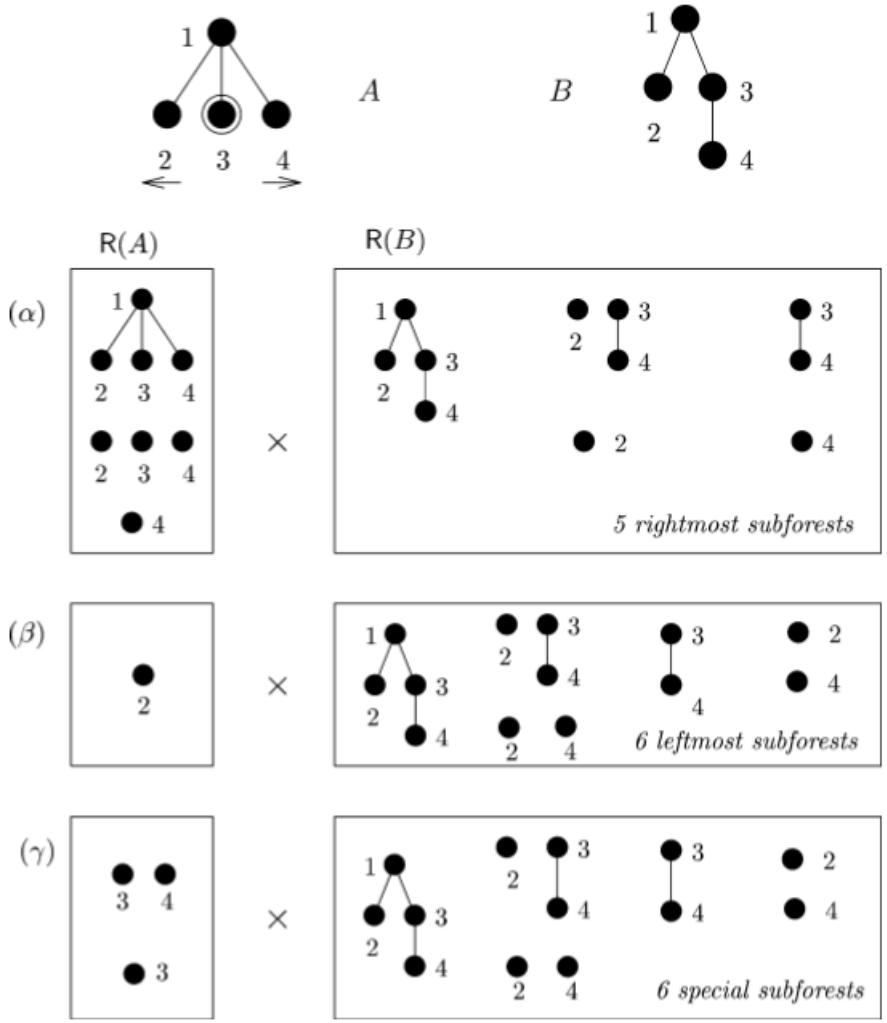
Let (A, B) be a pair of trees. A being a cover tree.

7) If $A = l(A_1 \circ \cdots \circ A_n)$ and the favourite child is A_1
   - Free(A) = Right(A) = $\sum_{i \neq j} Free(A_i) + All(A_j) + \#right(B)(1 + |A_1 \circ \cdots \circ A_{j-1}|) + \#special(B)\,|A_{j+1} \circ \cdots \circ A_n|$
   - All(A) = Left(A) = $\sum_{i \neq j} Free(A_i) + All(A_j) + \#special(B)(|A| - |A_j|)$



Number of Relevant Forests of A Pair of Tree

Example of LEMMA 13



gives 33 pairs of subforest.

Number of Relevant Forests of A Pair of Tree

Example of LEMMA 13

$$
\begin{aligned}
Free(A) &= Free(2) + Free(4) + All(3) + \#special(B) * 1 + \#right(B) * 2 \\
&= \#left(B) + \#right(B) + \#special(B) + \#special(B) \\
&\quad + \#right(B) * 2 \\
&= 33
\end{aligned}
$$

Number of Rightmost, Leftmost and Subforest.

LEMMA 14

Let A be a tree,

$$\#right(A) = \sum(|A(i)|, i \in A) - \sum(|A(j)|, j \text{ is a rightmost child})$$

$$\#left(A) = \sum(|A(i)|, i \in A) - \sum(|A(j)|, j \text{ is a leftmost child})$$

Proof.

From lemma 10, let A be $I(A_1 \circ \cdots \circ A_n)$, then

$$\#right(A) = \sum_i \#right(A_i) + |A| - |A_n|$$

Number of Rightmost, Leftmost and Subforest.

LEMMA 15

Let F be a forest of size n,

$$\#special(F) = \frac{n(n+3)}{2} - \sum_{i \in F} |F(i)|$$

Proof

For case $n = 0$, then $\#special(F) = 0$, which is consistent with lemma 15.

For case $n > 1$, let $F = l(g) \circ t$. There are two kinds of special sub-forest of F to be considered:

- those containing the node l: there are $|t| + 1$ such sub-forest
- those not containing the node l:there are $\#special(g \circ t)0$ such sub-forest

Number of Rightmost, Leftmost and Subforest.

LEMMA 15

Let F be a forest of size n,

$$\#special(F) = \frac{n(n+3)}{2} - \sum_{i \in F} |F(i)|$$

Proof

This allows that

$$\#special(F) = |t| + 1 + \#special(g \circ t)$$

On one hand, $|t| + 1 = n - |l(g)| + 1$

On the other hand, by applying the hypothesis on $g \circ t$, whose size is n-1

$$\#special(g \circ t) = \frac{(n-1)(n+2)}{2} - \sum_{i \in g \circ t} |g \circ t(i)|$$

Since $g \circ t$ is a sub-forest of F, this implies

$$\#special(g \circ t) = \frac{(n-1)(n+2)}{2} - \sum_{i \in g \circ t} |F(i)|$$

Number of Rightmost, Leftmost and Subforest.

LEMMA 15

Let F be a forest of size n,

$$\#special(F) = \frac{n(n+3)}{2} - \sum_{i \in F} |F(i)|$$

Proof

It allows that

$$\begin{aligned}\#special(F) &= n - |l(g)| + 1 + \frac{(n-1)(n+2)}{2} - \sum_{i \in g \circ t} |F(i)| \\ &= n + 1 + \frac{(n-1)(n+2)}{2} - \sum_{i \in F} |F(i)| \\ &= \frac{n(n+3)}{2} - \sum_{i \in F} |F(i)|\end{aligned}$$

This concludes the proof.

Number of Rightmost, Leftmost and Subforest.

LEMMA 16

For Zhang and Shasha's algorithm, $\#rel(A, B) = \#left(A) * \#left(B)$.

Proof

Applying lemma 13, case 1, 3, 5 and lemma 14.This concludes the proof.

Key Idea(A New Optimal Cover Strategy)

- An optimal cover strategy is a strategy that minimizes the total number of relevant forest.
- Define four dynamic programming tables **Right**, **Left**, **Free** and **All** indexed by nodes A.
- The favourite child is chosen to be the child that minimizes the number of relevant forests.

Algorithm(A New Optimal Cover Strategy)

Let A $= I(A_1 \circ \cdots \circ A_n)$ then

$$Free(A) = \sum_{i \geq i} Free(A_i)$$

$$+ min \begin{cases} Left(A_1) - Free(A_1) + \#left(B) * (|A| - |A_1|), \\ All(A_j) - Free(A_j) + \#special(B)\,|A_{j+1} \circ \cdots \circ A_n| \\ \quad +\#right(B)(1 + |A_1 \circ \cdots \circ A_{j-1}|), 1 < j < n \\ Right(A_n) - Free(A_n) + \#right(B)(|A| - |A_n|). \end{cases}$$

Note 1 The favourite child is selected to be the root of the subtree A_j so that A_j gives the minimal value.
Note 2 The optimal cover is then built up by tracing back from Free(A).

Time Complexity(A New Optimal Cover Strategy)

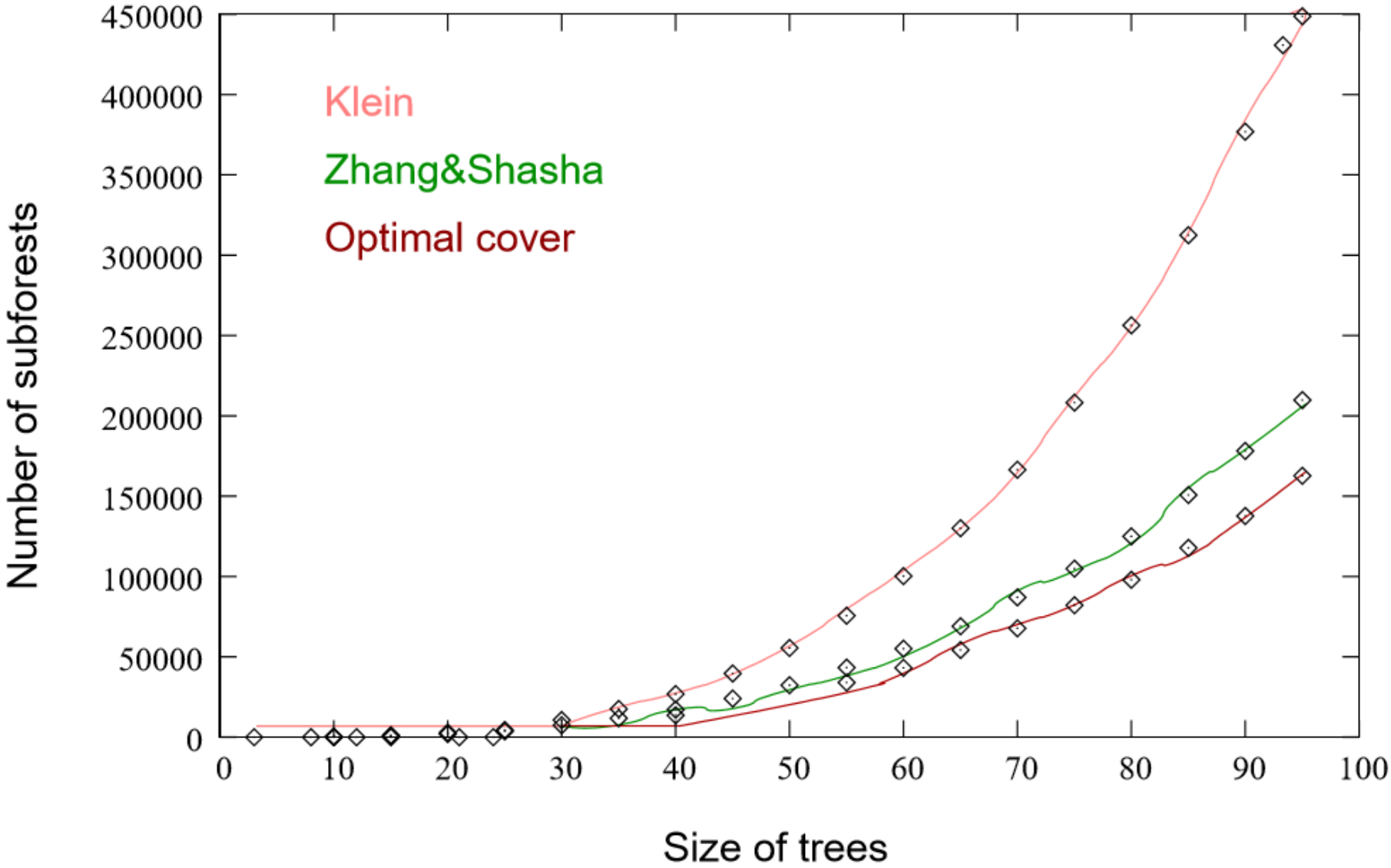
Prerequisite of the Analysis

- Two main steps in preprocessing
  (1) the computation of $\#right(B)$, $\#left(B)$ and $\#special(B)$.
  (2) the computation of array of **Right**, **Left**, **Free** and **All** indexed by nodes A.
- Time complexity is bounded by $\mathcal{O}((1)+(2))$
- (1) can be computed by applying lemma 10 and 11 in $\mathcal{O}(|B|)$
- Each array in (2) can be computed in $\mathcal{O}(\sum_{i\in A} deg(i))$, that is in $\mathcal{O}(|A|)$

Conclusion

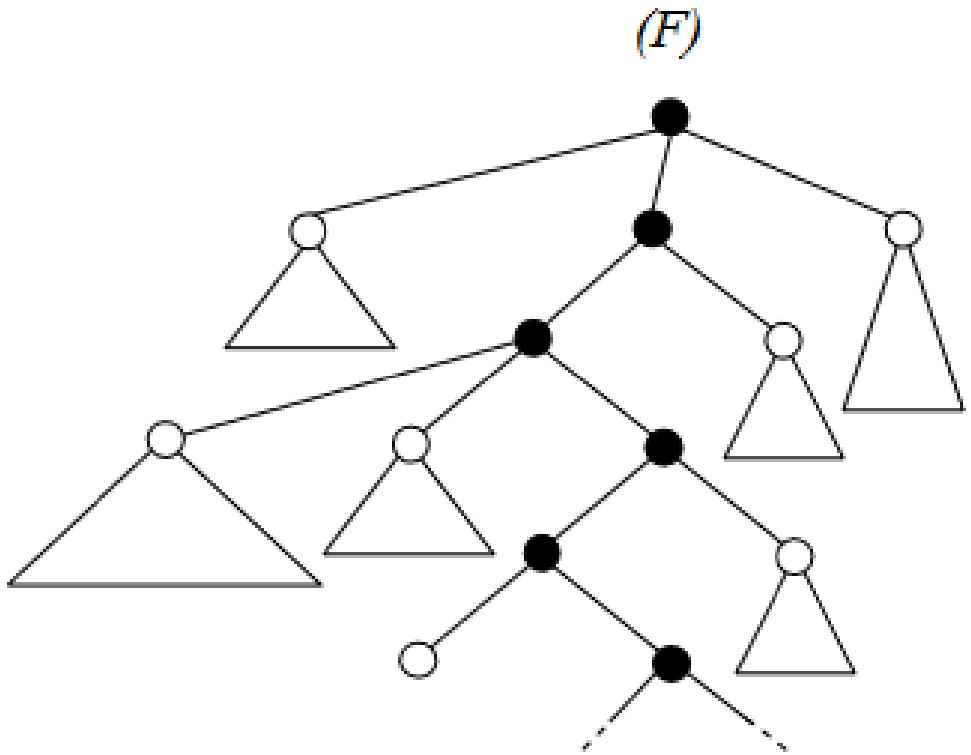
The time complexity of preprocessing is $\mathcal{O}(|A|+|B|)$

Performance(A New Optimal Cover Strategy)



Key Concept(An Optimal Decomposition Algorithm)

- set **TopLight(F)** is the set of *light nodes* with *ldepth* 1 in F.



Key Idea(An Optimal Decomposition Algorithm)

- For all $v \in TopLight(F)$, Klein's strategy solves $\delta(F(v), G)$ by determining the direction according to F(v), even if $|F(v)| < |G|$
- In such case, making decision according to the larger forest can do better.

Algorithm(An Optimal Decomposition Algorithm)

We compute $\delta(F, G)$ recursively as follows:

(1) if $|F| < |G|$, compute $\delta(G, F)$ instead.

(2) Recursively compute $\delta(F(v), G)$ for all $v \in \mathit{TopLight}(F)$.

(3) Compute $\delta(F.G)$ using the following decomposition strategy:

- S(F', G')=left, if F' is a tree, or the root of leftmost tree is not the heavy child of its parent.
- Otherwise, S(F', G')=right.

Time Complexity(An Optimal Decomposition Algorithm)

Prerequisite of the Analysis

- $\sum_{v \in TopLight(F)} |F(v)| \leq |F|$.
- $|F(v)| < \frac{|F|}{2}$ for each $v \in TopLight(F)$. Otherwise v would be a heavy node.
- This algorithm consists of two main steps(step 2 and 3). Then algorithm is bounded by $\mathcal{O}(\mathcal{O}(Step2) + \mathcal{O}(Step3))$
- In step 3, a node v in the heavy path of F cannot be matched or deleted until the remaining subforest is a tree. That is to say only $|F|$ subforest need to be considered. This gives $\mathcal{O}(|F||G|^2)$
- In step2, the complexity is bound by the number of subforest $\sum_{v \in TopLight(G)} R(F(v), G)$

Time Complexity(An Optimal Decomposition Algorithm)

Analysis

We have established that if $|F| \geq |G|$

$$R(F,G) \leq |G|^2 |F| + \sum_{v \in TopLight(F)} R(F(v), G)$$

Otherwise

$$R(F,G) \leq |F|^2 |G| + \sum_{w \in TopLight(G)} R(F, G(w))$$

Time Complexity(An Optimal Decomposition Algorithm)

Analysis

LEMMA 17

$$R(F,G) \leq 4(|F||G|)^{\frac{3}{2}}$$

Proof. By applying the induction hypothesis on $R(F(v),G)$

$$\begin{aligned}
R(F,G) &\leq |G|^2|F| + \sum_{v\in TopLight(F)} 4(|F(v)||G|)^{\frac{3}{2}} \\
&= |G|^2|F| + 4|G|^{\frac{3}{2}} \sum_{v\in TopLight(F)} |F(v)|^{\frac{3}{2}} \\
&\leq |G|^2|F| + 4|G|^{\frac{3}{2}} \sum_{v\in TopLight(F)} |F(v)|\, max_{v\in TopLight(F)}\sqrt{|F(v)|} \\
&\leq |G|^2|F| + 4|G|^{\frac{3}{2}}|F|\sqrt{\frac{|F|}{2}} = |G|^2|F| + \sqrt{8}(|F||G|)^{\frac{3}{2}} \\
&\leq 4(|F||G|)^{\frac{3}{2}}
\end{aligned}$$

Algorithm

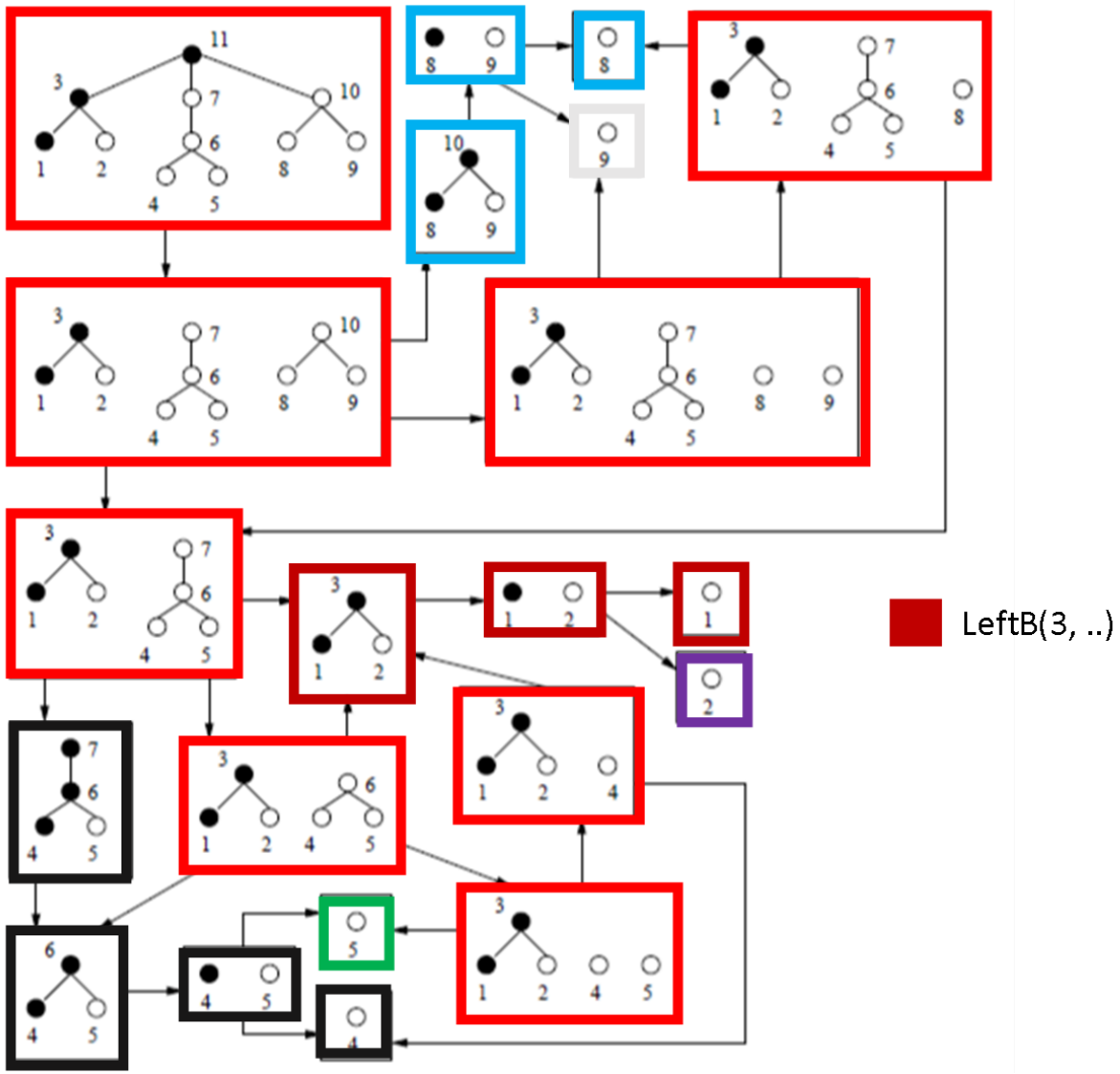
For tree-tree edit distance

$$\delta(l(f), l'(f')) = min \begin{cases} \delta(f, l'(f')) + \gamma(l \to \lambda) \\ \delta(l(f), f') + \gamma(\lambda \to l') \\ \delta(f, f') + \gamma(l \to l') \end{cases}$$

For forest-forest edit distance

$$\delta(l(f) \circ t, l'(f') \circ t') = min \begin{cases} \delta(f \circ t, l'(f') \circ t') + \gamma(l \to \lambda) \\ \delta(l(f) \circ t, f' \circ t') + \gamma(\lambda \to l') \\ \delta(l(f), l'(f')) + \delta(t, t') \end{cases}$$

Decomposition Example



Definition

- **Free** : the number of sub-problems of tree A compared with tree B.
- **LeftA** : the number of sub-problems created by **right** deletions of tree B(favorite child is the **leftmost** root of tree B), which are compared with all **leftmost** forest of tree A.
- **RightA** : the number of sub-problems created by **left** deletions of tree B(favorite child is the **rightmost** root of tree B), which are compared with all **rightmost** forest of tree A.
- **AllA** : the number of sub-problems after a series of left and right deletions of tree B(favorite child is neither rightmost nor leftmost root of tree B), which are compared with all **special** forest of tree A.

Note

- The definitions of **LeftB**, **RightB**, **AllA** are analogy with **LeftA**, **RightB**, **AllA**.
- The size of each matrix is $|A| * |B|$.

Number of Rightmost, Leftmost and Subforest.

Let A be a tree of size n,

$$\#right(A) = \sum(|A(i)|, i \in A) - \sum(|A(j)|, j \text{ is a rightmost child})$$

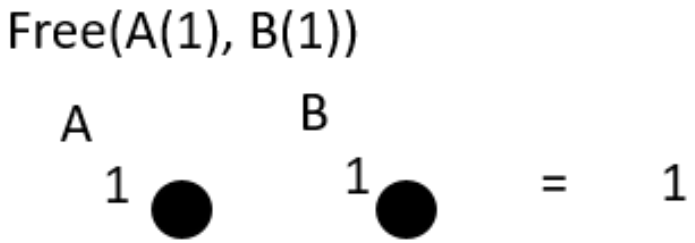
$$\#left(A) = \sum(|A(i)|, i \in A) - \sum(|A(j)|, j \text{ is a leftmost child})$$

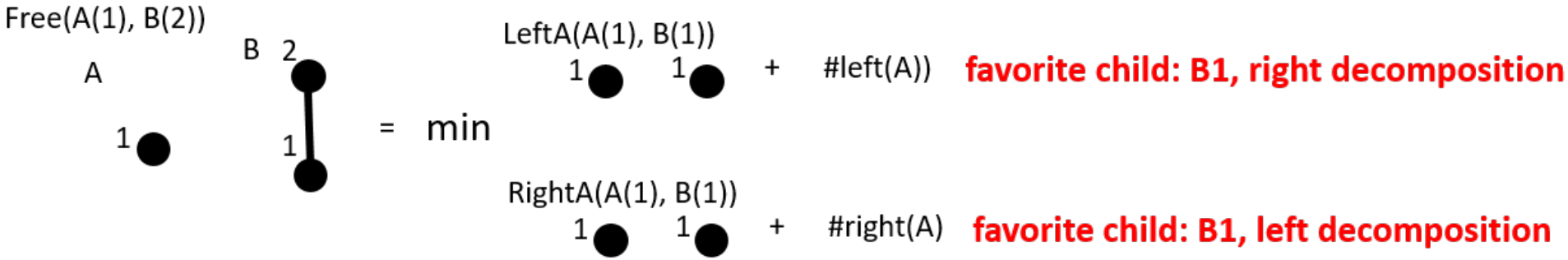
$$\#special(A) = \frac{n(n+3)}{2} - \sum_{i \in A} |A(i)|$$

Time Complexity

To compute the number of rightmost, leftmost and special forests for each subtree whose each root is a node in tree A and tree B. This can be made in $\mathcal{O}(|A| + |B|)$

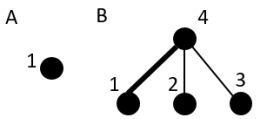
Computation of Free





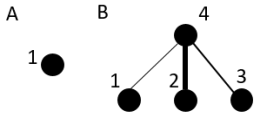
Computation of Free

Free(A(1), B(4))



= Free(A(1), B(2)) + Free(A(1), B(3)) + LeftA(A(1), B(1)) + #left(A) * (|B| - |B(1)|)

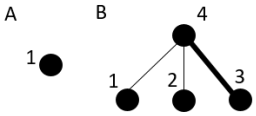
favorite child: B1, right decomposition



= Free(A(1), B(1)) + Free(A(1), B(3)) + AllA(A(1), B(2)) + min

#right(A) * (1+|B(1)|) + #special(A) * (|B(3)|) **favorite child: B2, left decomposition**

#left(A) * (1+|B(3)|) + #special(A) * (|B(1)|) **favorite child: B2, right decomposition**

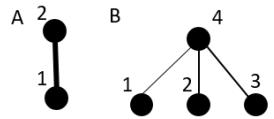


= Free(A(1), B(1)) + Free(A(1), B(2)) + RightA(A(1), B(2)) + #right(A) * (|B| - |B(3)|)

favorite child: B3, left decomposition

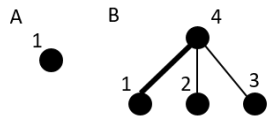
Computation of Free

Free(A(2), B(4))



=

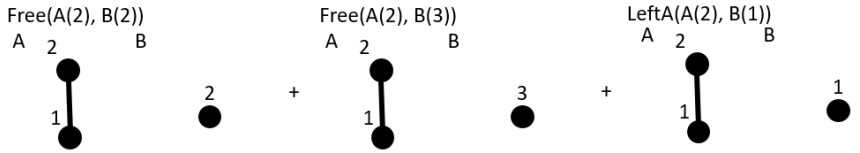
LeftB(A(1), B(4))



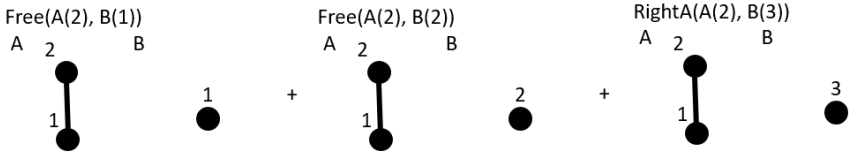
#left(B) **favorite child: A1, right decomposition**

+ min

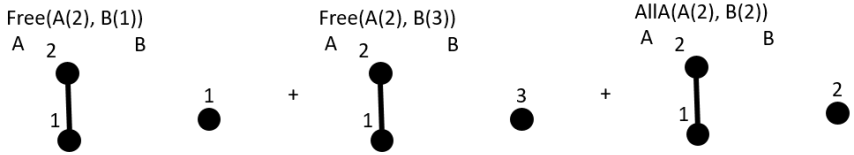
#right(B) **favorite child: A1, left decomposition**



\+ #left(A) * (|B| - |B(1)|) **favorite child: B1, right decomposition**



\+ #right(A) * (|B| - |B(3)|) **favorite child: B3, left decomposition**

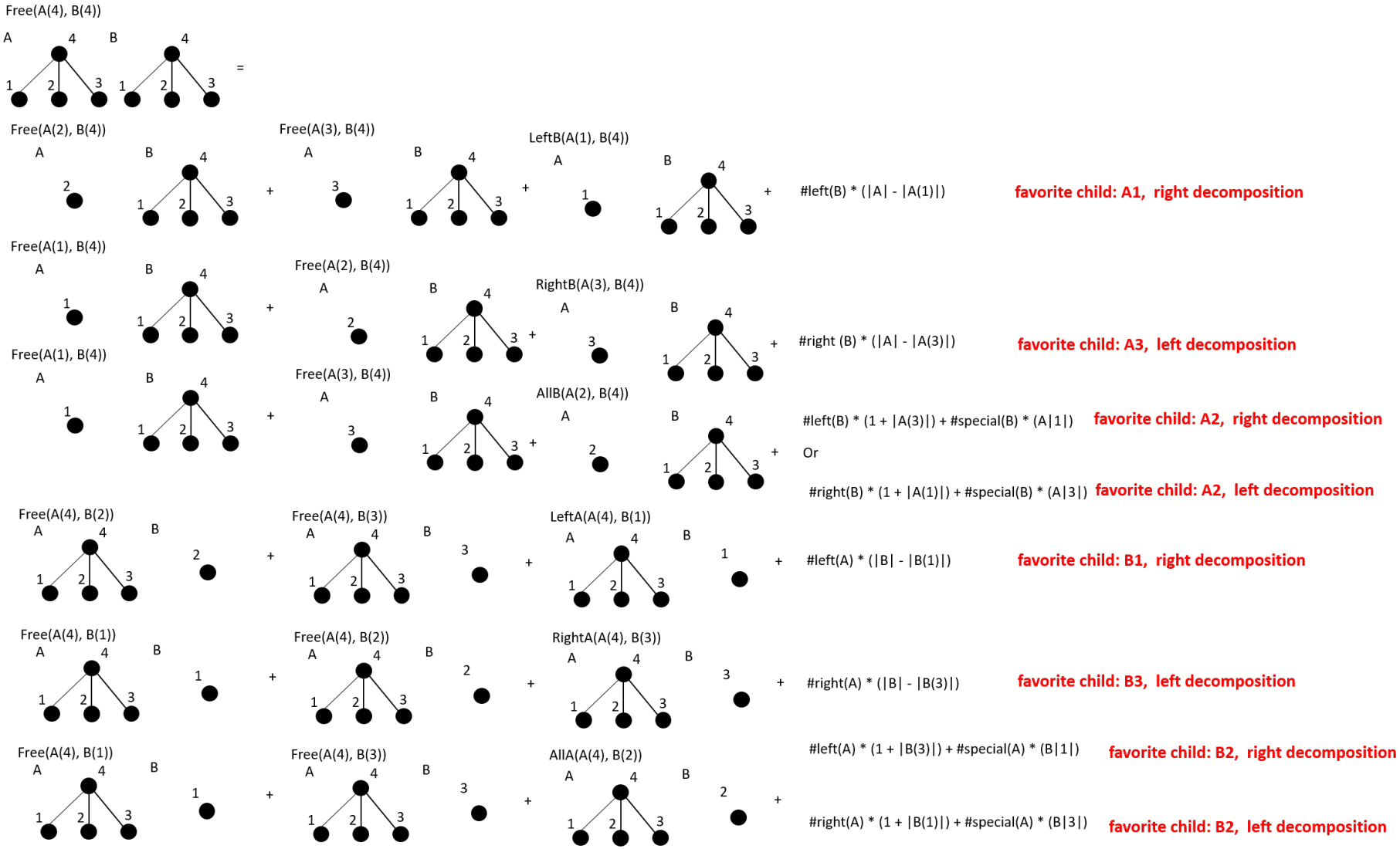


\+ min

#left(A) *(1 + |B(1)|) + #special(A) *(|B(3)|) **favorite child: B2, right decomposition**

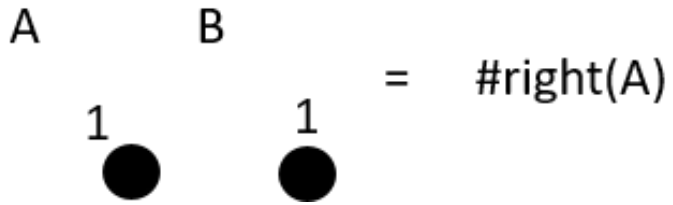
#right(A) *(1 + |B(3)|) + #special(A) *(|B(1)|) **favorite child: B2, left decomposition**

Computation of Free



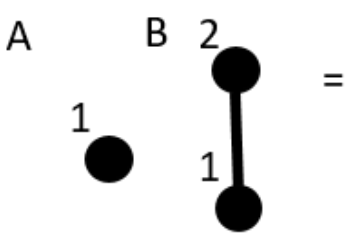
Computation of RightA

RightA(A(1), B(1))

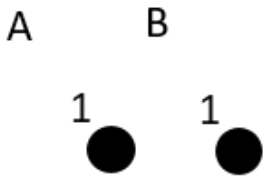


= #right(A)

RightA(A(1), B(2))

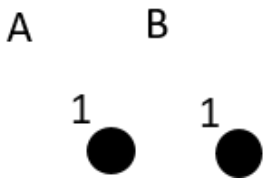


= RightA(A(1), B(1))



+ #right(A) **favorite child: B1, left decomposition**

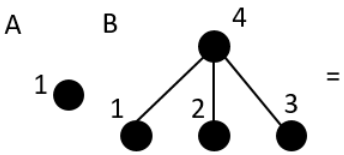
AllA(A(1), B(1))



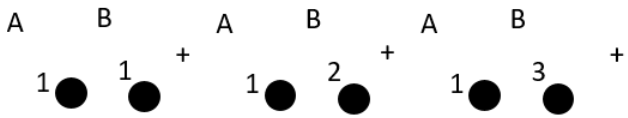
+ #special(A) **favorite child: B1, right decomposition**

Computation of RightA

RightA(A(1), B(4))

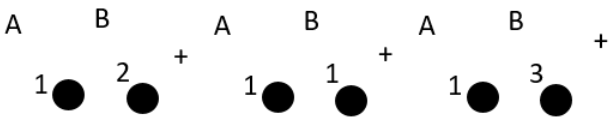


AllA(A(1), B(1)) Free(A(1), B(2)) Free(A(1), B(3))



\+ #special(A)*(|A|-|A(1)|) **favorite child: B1, right decomposition**

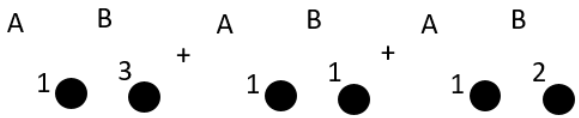
AllA(A(1), B(2)) Free(A(1), B(1)) Free(A(1), B(3))



#right(A)*(1+|A(1)|) + #special(A)*(|A(3)|) **favorite child: B2, left decomposition**

#special(A)*(|A|-|A(2)|) **favorite child: B2, right decomposition**

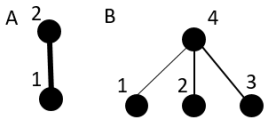
RightA(A(1), B(3)) Free(A(1), B(1)) Free(A(1), B(2))



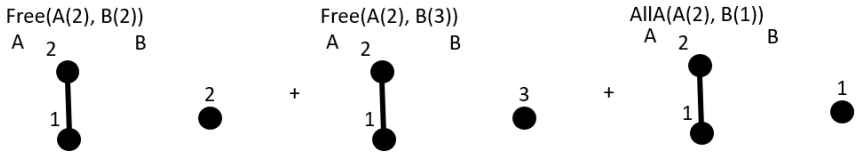
\+ #right(A)*(|A|-|A(3)|) **favorite child: B3, right decomposition**

Computation of RightA

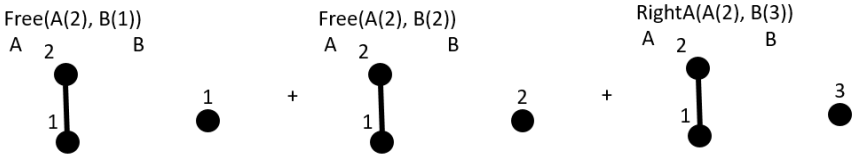
RightA(A(2), B(4))



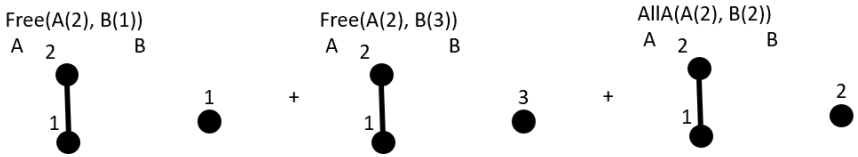
=



\+ #special(A) * (|B| - |B(1)|) **favorite child: B1, right decomposition**



\+ #right(A) * (|B| - |B(3)|) **favorite child: B3, left decomposition**

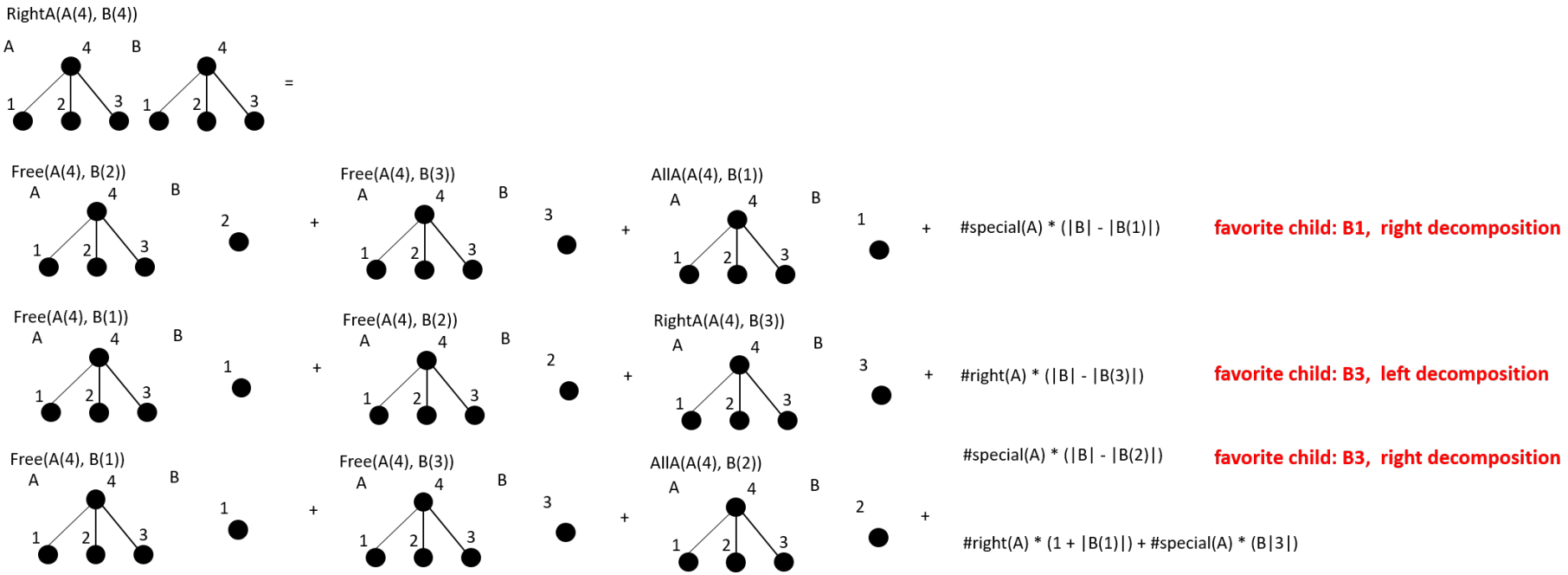


\+ #special(A) * (|B| - |B(2)|) **favorite child: B2, right decomposition**

\+ min

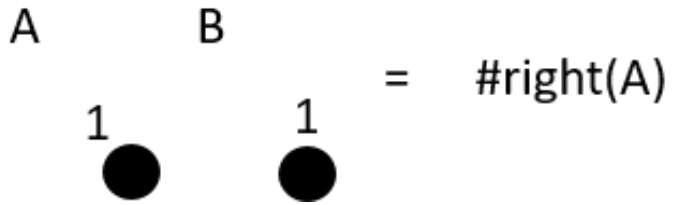
\+ #right(A) *(1 + |B(1)|) + #special(A) *(|B(3)|) **favorite child: B2, left decomposition**

Computation of RightA



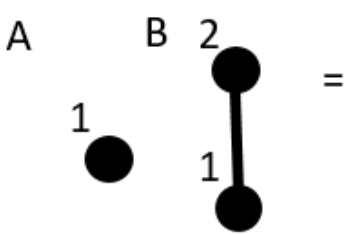
Computation of RightA

RightA(A(1), B(1))

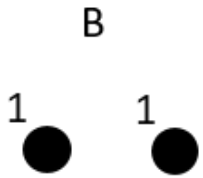


= #right(A)

RightA(A(1), B(2))

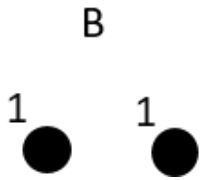


= RightA(A(1), B(1))



+ #right(A) **favorite child: B1, left decomposition**

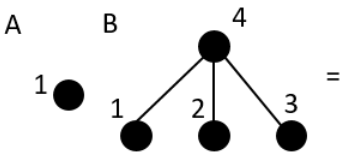
AllA(A(1), B(1))



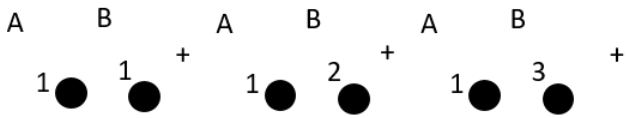
+ #special(A) **favorite child: B1, right decomposition**

Computation of RightA

RightA(A(1), B(4))



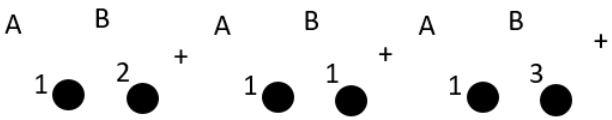
AllA(A(1), B(1)) Free(A(1), B(2)) Free(A(1), B(3))



#special(A)*(|A|-|A(1)|) **favorite child: B1, right decomposition**

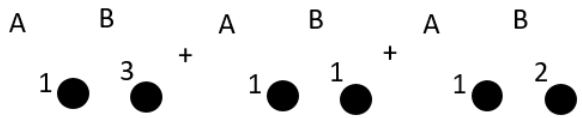
AllA(A(1), B(2)) Free(A(1), B(1)) Free(A(1), B(3))

#right(A)*(1+|A(1)|) + #special(A)*(|A(3)|) **favorite child: B2, left decomposition**



#special(A)*(|A|-|A(2)|) **favorite child: B2, right decomposition**

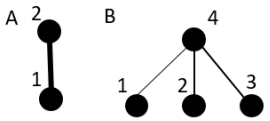
RightA(A(1), B(3)) Free(A(1), B(1)) Free(A(1), B(2))



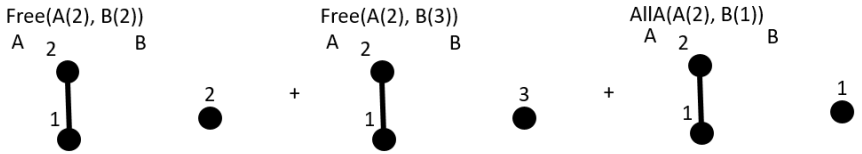
+ #right(A)*(|A|-|A(3)|) **favorite child: B3, right decomposition**

Computation of RightA

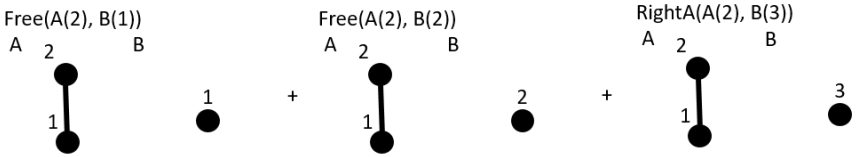
RightA(A(2), B(4))



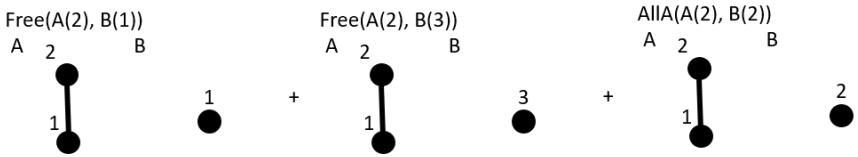
=



\+ #special(A) * (|B| - |B(1)|) **favorite child: B1, right decomposition**



\+ #right(A) * (|B| - |B(3)|) **favorite child: B3, left decomposition**

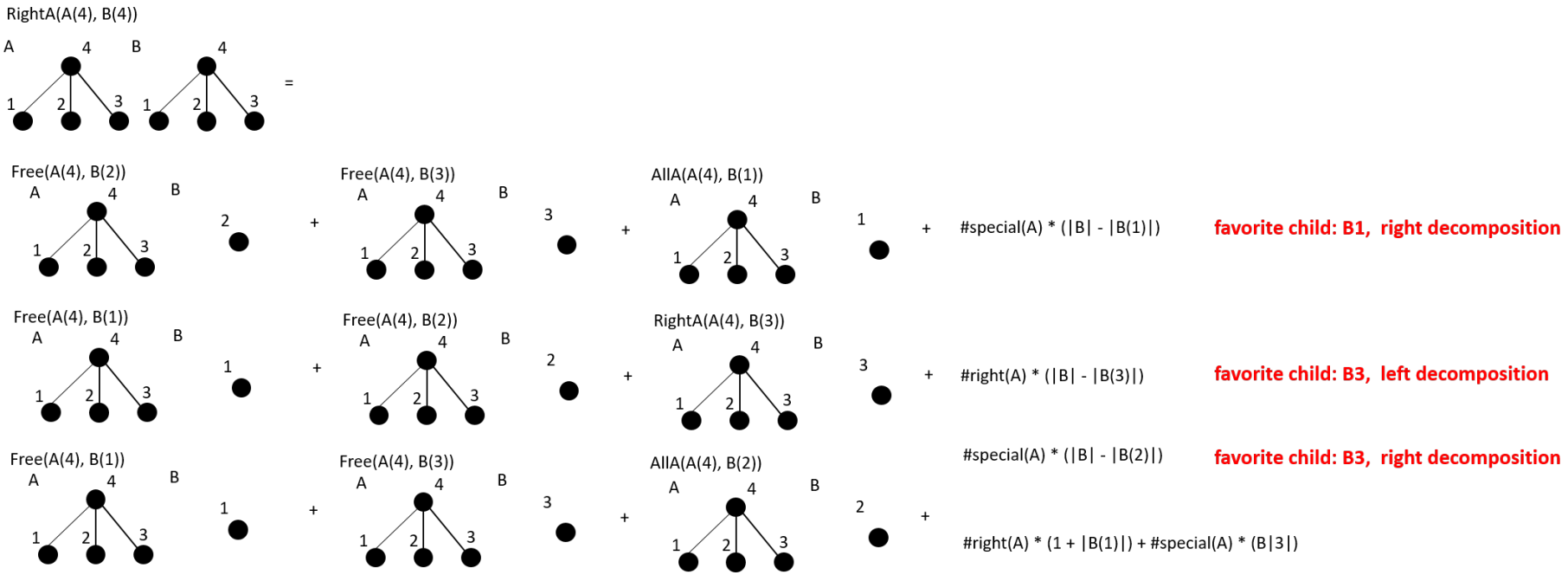


\+ #special(A) * (|B| - |B(2)|) **favorite child: B2, right decomposition**

\+ min

\+ #right(A) *(1 + |B(1)|) + #special(A) *(|B(3)|) **favorite child: B2, left decomposition**

Computation of RightA



Tree Indexing

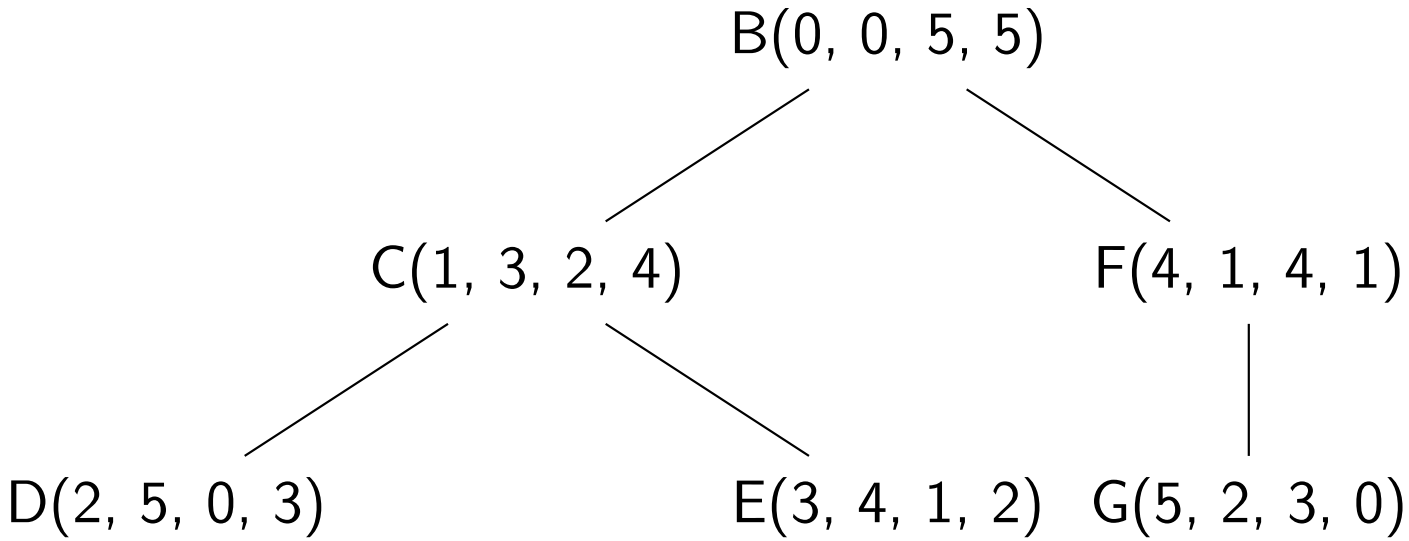
- $preOrder\_left\_to\_right \rightarrow preL$
- $preOrder\_right\_to\_left \rightarrow preR$
- $postOrder\_left\_to\_right \rightarrow postL$
- $postOrder\_right\_to\_left \rightarrow postR$

Note:

- $preL + postR + 1 = treeSize$
- $preR + postL + 1 = treeSize$

Tree Indexing

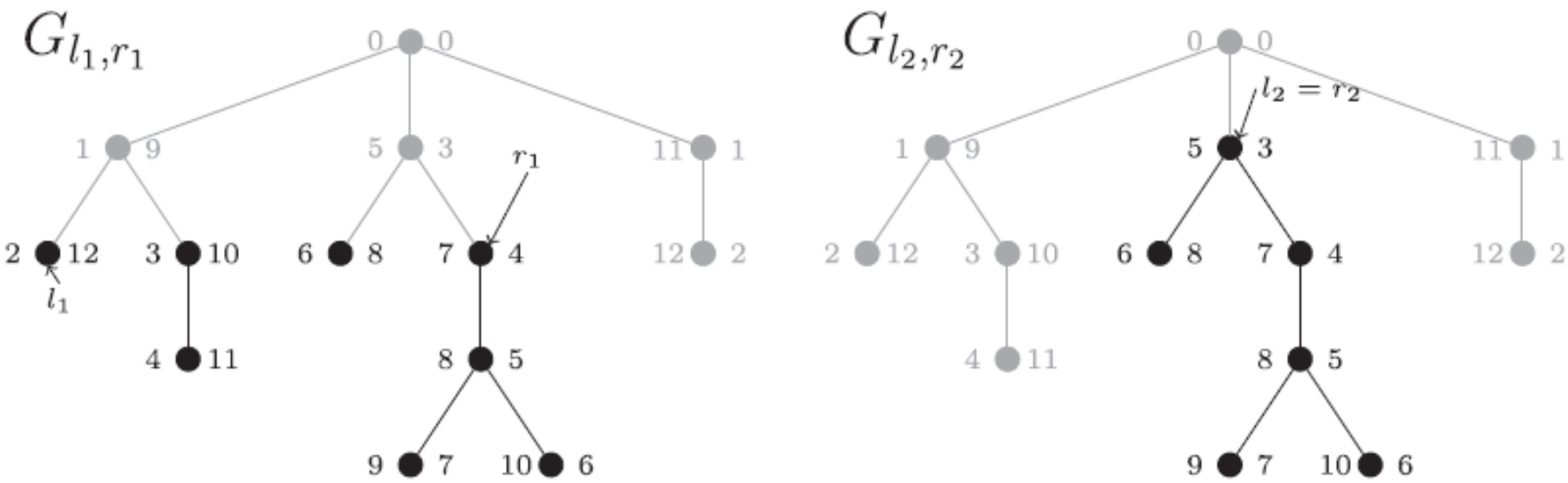
A sequence **"(B(C(D)(E))(F(G)))"** can be built to a tree F.
Consider a tree F(preL, preR, postL, postR)



Note:

- $preOrder\_left\_to\_right \rightarrow preL \rightarrow lF$
- $preOrder\_right\_to\_left \rightarrow preR \rightarrow rF$

Tree Indexing

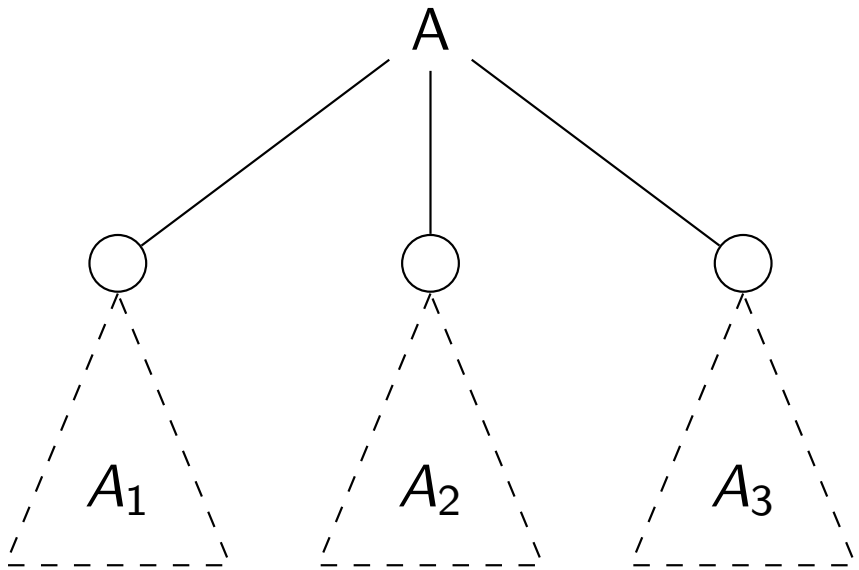


Definition:

$F_{lF,rF}$ is the subforest of F with nodes $N(F_{lF,rF}) = \{lF, rF\} \cup \{x{:}x \in F,$ x succeeds lF in left-to-right preorder and x succeed rF in right-to-left preorder$\}$ and edges $E(F_{lF,rF}) = \{(v, w) \in E(F){:}\ v \in F_{lF,rF} \cap w \in F_{lF,rF}\}$

Computation of Rightmost, Leftmost and Special Forest

Let A be a tree of size n,



$$\#right(l(A_1, \cdots, A_n)) = \sum \#right(A_i) + |A| - |A_n|$$

$$\#left(l(A_1, \cdots, A_n)) = \sum \#left(A_i) + |A| - |A_1|$$

$$\#special(A) = \frac{n(n+3)}{2} - \sum_{i \in A} |A(i)|$$

Computation of Rightmost, Leftmost and Special Forest

Let A be a tree of size n,

$$\#right(l(A_1, \cdots, A_n)) = \sum \#right(A_i) + |A| - |A_n|$$

$$\#left(l(A_1, \cdots, A_n)) = \sum \#left(A_i) + |A| - |A_1|$$

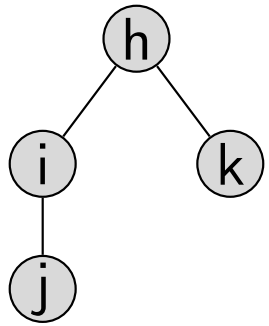
$$\#special(A) = \frac{n(n+3)}{2} - \sum_{i \in A} |A(i)|$$

Note: Each node marks

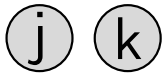
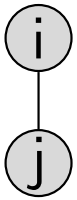
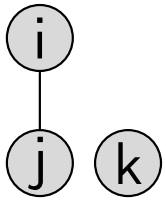
- the size of subtree rooted at that node
- the sum of the size of subtree of its chilren
- the number of leftmost forest of subtree rooted at that node
- the number of rightmost forest of subtree rooted at that node
- the number of special forest of subtree rooted at that node

Special Forest Enumeration

Let G be a tree,

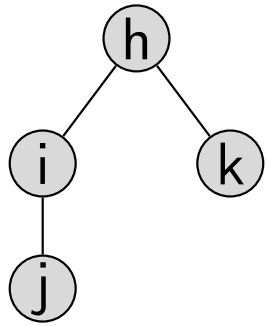


Then the **left-to-right** enumeration would be

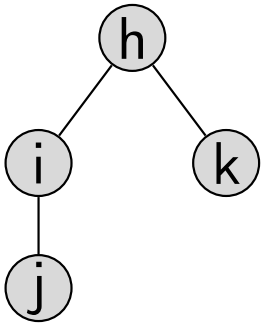
- IG = k
   (rG = k)
- IG = j
   (rG = j)
   (rG = k)
- IG = i
   (rG = i)
   (rG = k)

Special Forest Enumeration

Let G be a tree,



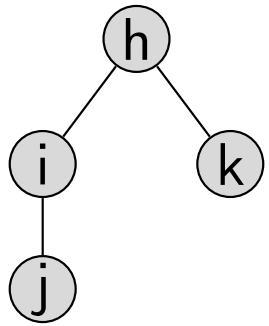
- IG = h



(rG = h)

Special Forest Enumeration

Let G be a tree,

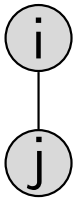


Then the **right-to-left** enumeration would be

- rG = j

  (IG = j)
- rG = i

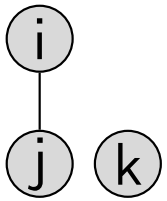
  (IG = i)
- rG = k

  (IG = k)

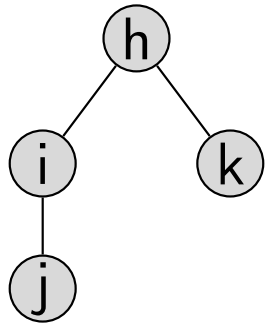
  (IG = j)

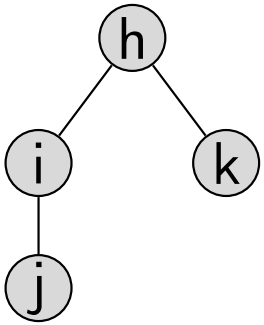
  (IG = i)

Special Forest Enumeration

Let G be a tree,



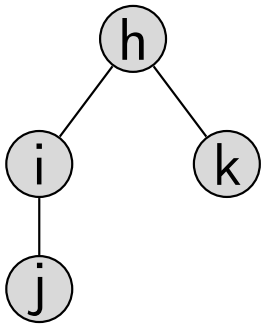
- $rG = h$

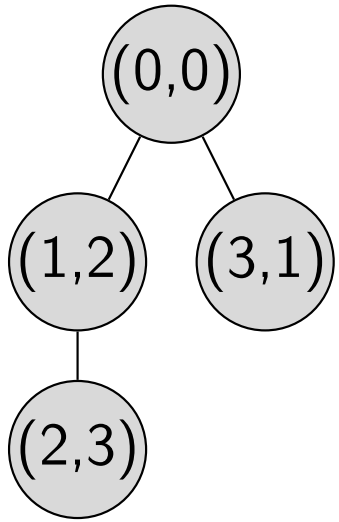


$(lG = h)$

Special Forest Enumeration

Let G be a tree,

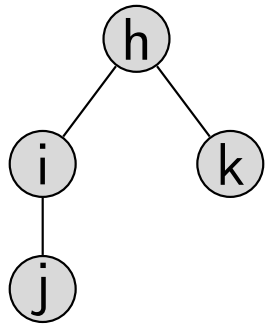




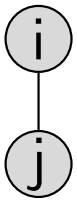
- left-to-right
  - lG enumerate in reverse left to right pre-order.
  - rG enumerate from subset right(G, lG) $\cup$ lG in reverse right to left pre-order.
- right-to-left
  - rG enumerate in reverse right to left pre-order.
  - lG enumerate from subset left(G, rG) $\cup$ rG in reverse left to right pre-order

Rightmost Forest Enumeration - right path decomposition

Let G be a tree,



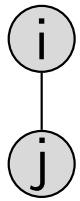
- rG = j
  

  (IG = j)
- rG = i
  

  (IG = i)
- rG = k

(IG = k)

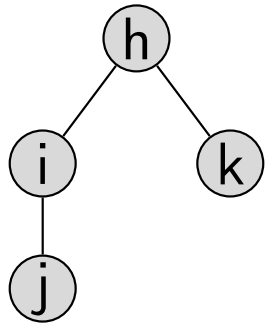
(IG = j)



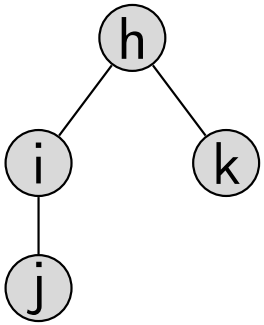
(IG = i)

Rightmost Forest Enumeration - right path decomposition

Let G be a tree,



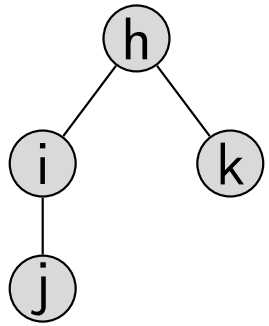
- $IG = h$



$(rG = h)$

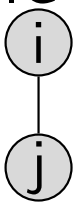
Leftmost Forest Enumeration - left path decomposition

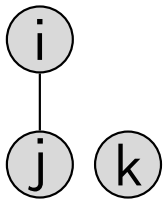
Let G be a tree,



- $lG = k$
  

  $(rG = k)$
- $lG = j$
  

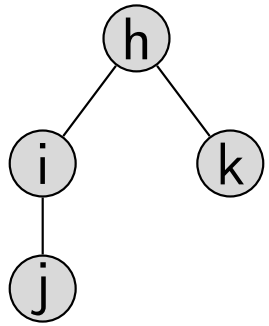
  $(rG = j)$
- $lG = i$
  

  $(rG = i)$
  

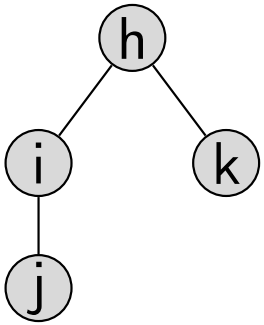
  $(rG = k)$

Leftmost Forest Enumeration - left path decomposition

Let G be a tree,



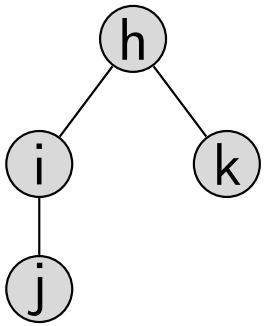
- rG = h

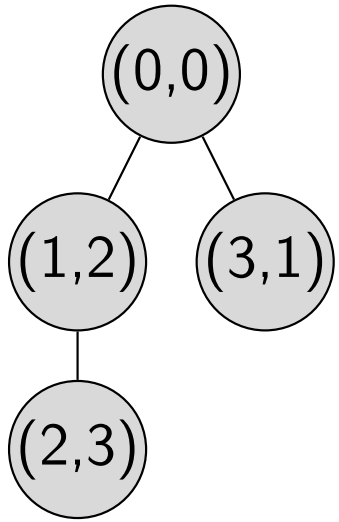


(lG = h)

Leftmost/Rightmost Forest Enumeration

Let G be a tree,

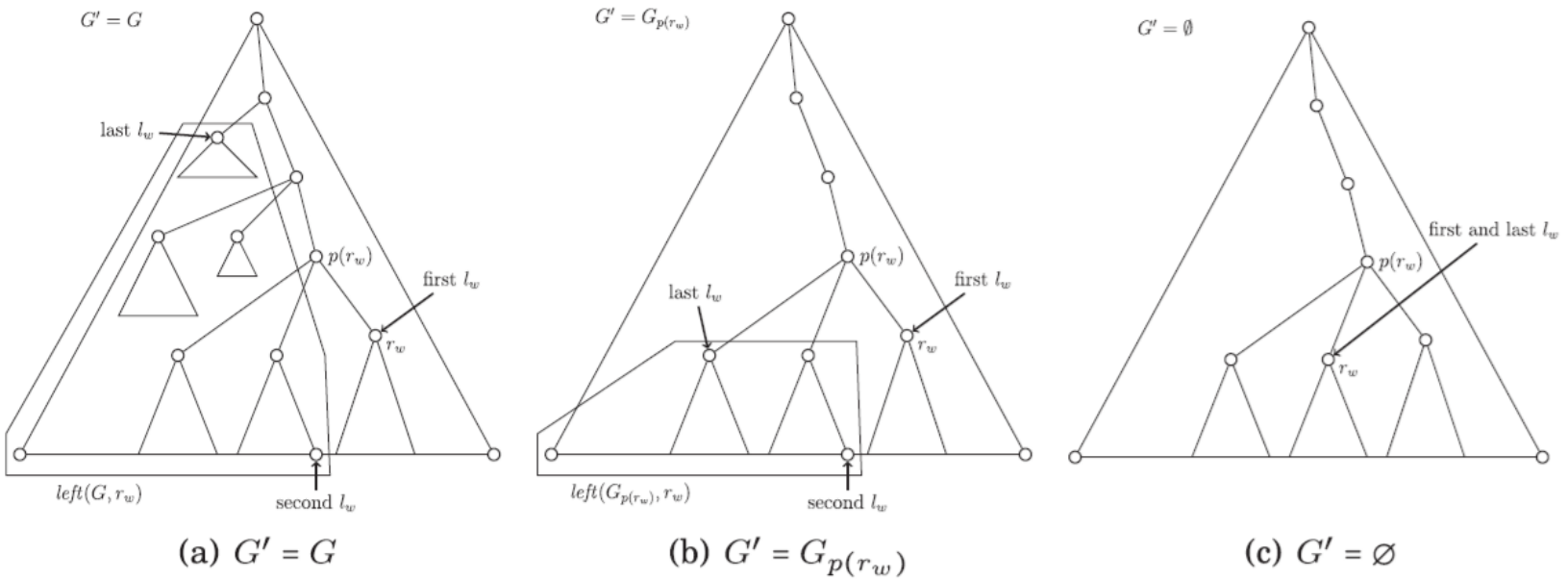




- leftmost forest enumeration
  - lG enumerate in reverse left to right pre-order.
  - rG enumerate from subset right(G', lG) $\cup$ lG in reverse right to left pre-order.
- rightmost forest enumeration
  - rG enumerate in reverse right to left pre-order.
  - lG enumerate from subset left(G', lG) $\cup$ lG in reverse left to right pre-order.

Leftmost/Rightmost Forest Enumeration

G'

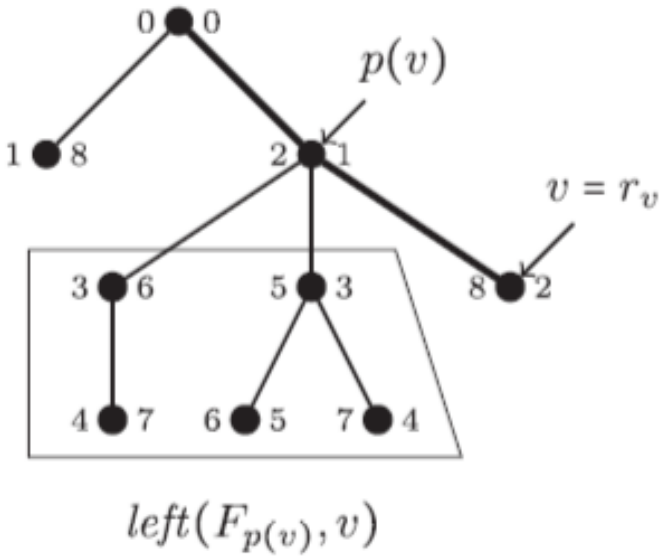


(a) $G' = G$ (b) $G' = G_{p(r_w)}$ (c) $G' = \varnothing$

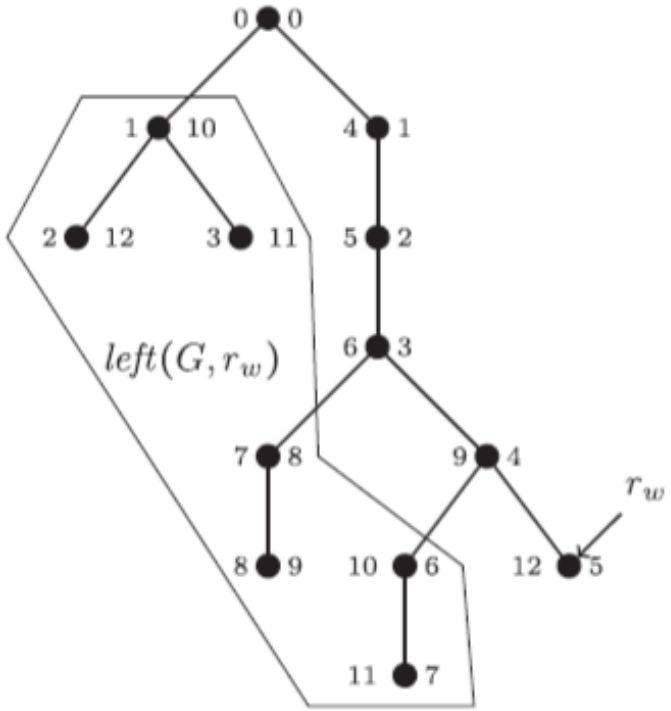
- special forest enumeration G' = G
- leftmost forest enumeration
  - IG is the leftmost child of its parent G' = $G_p(lG)$
  - otherwise G'= $\varnothing$
- rightmost forest enumeration
  - rG is the rightmost child of its parent G' = $G_(rG)$
  - otherwise G' = $\varnothing$

Leftmost/Rightmost Forest Enumeration

left($F_{p(v)}$, v)



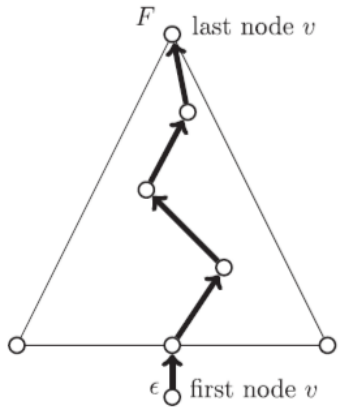
left(G, w)



Sub Problem Enumeration

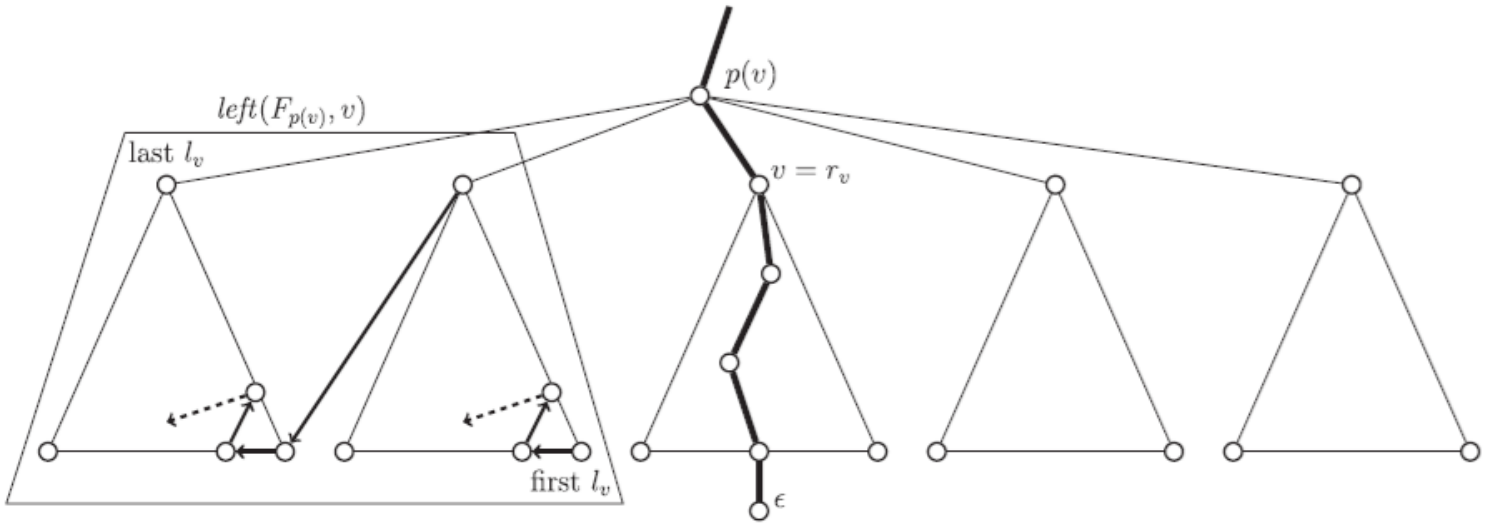
Enumerate quadruples(IF, rF, IG, rG) in loops (A(BCD)(B'C'D)).

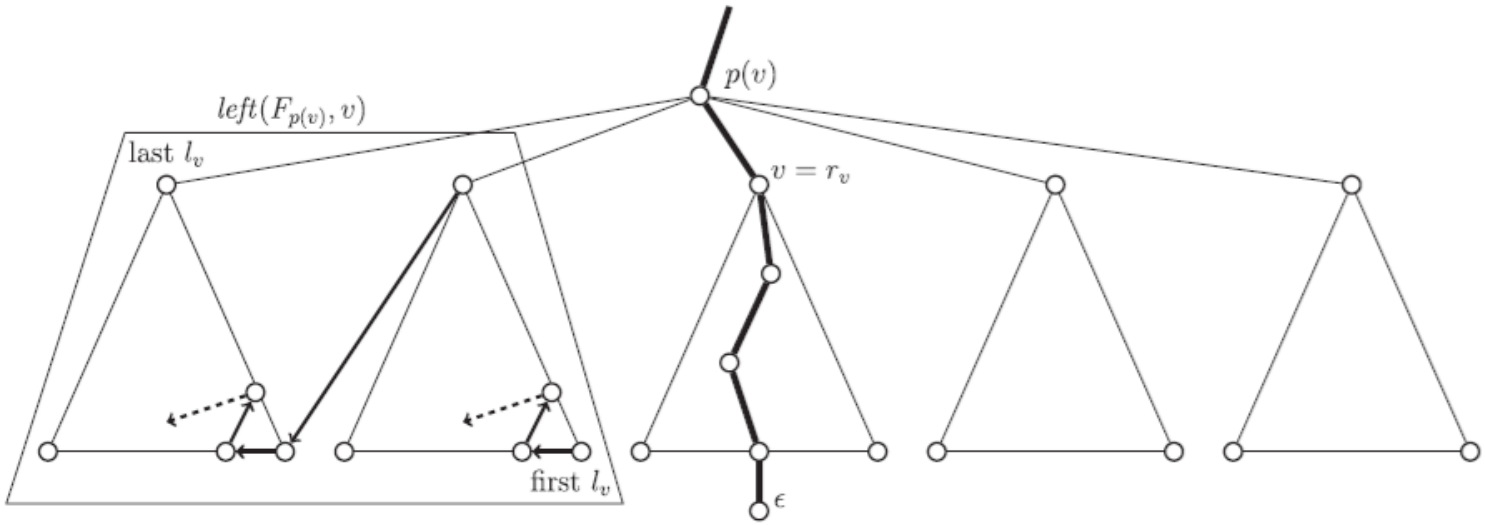
- Loop A is the outermost loop and iterates bottom-up over the nodes of path.



- Loop C iterates over the nodes to the left of path , in particular, over the nodes left(Fp(v), v) for a given node v, where v is the index variable of loop A.
- Loop C' is symmetric to loop C and iterates over the nodes right(Fp(v), v) ∪ p(v).

Subproblem Enumeration(Tree F Enumeration)

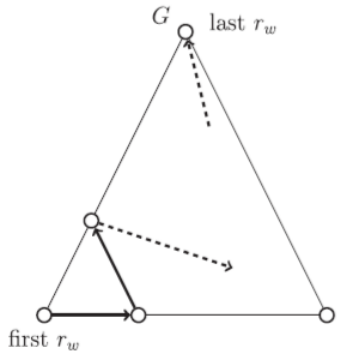


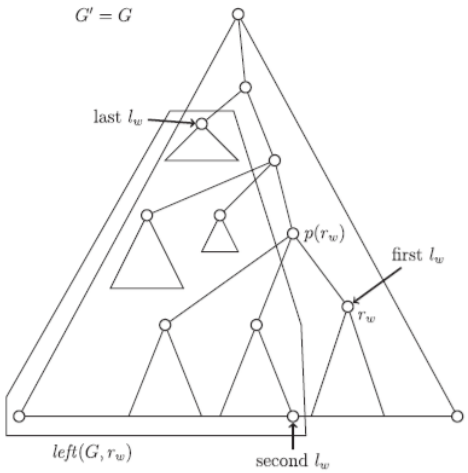


Subproblem Enumeration

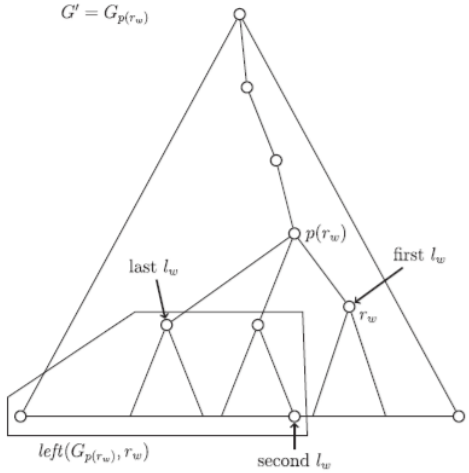
Enumerate quadruples(IF, rF, IG, rG) in loops (A(BCD)(B'C'D)).

- Loops B and D define subforests of the right-hand tree G.

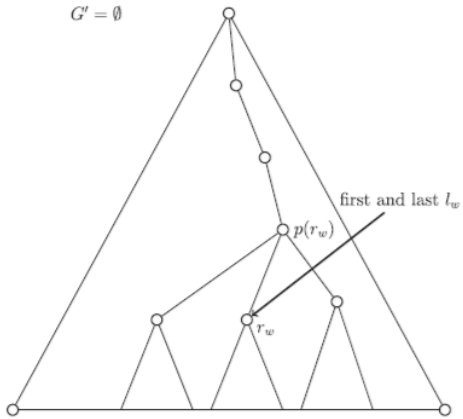




(a) $G' = G$



(b) $G' = G_{p(r_w)}$



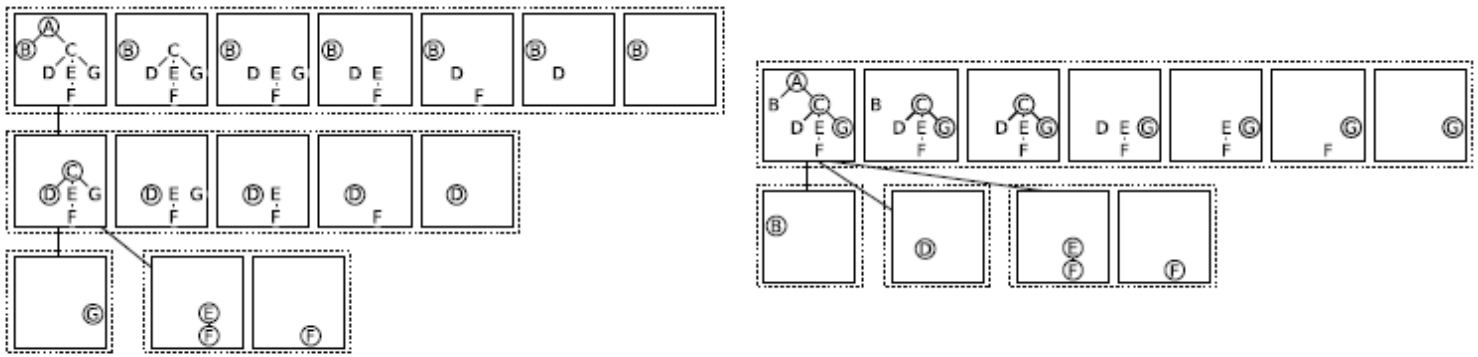
(c) $G' = \emptyset$

Efficient Space Complexity

- Time complexity in average $\mathcal{O}(n^3)$:
  - The path decomposition in F create n sub forests.
  - The all decomposition in G create n^2 sub forests.
- The naive implement leads to the space complexity in average $\mathcal{O}(n^3)$, but can be reduced to $\mathcal{O}(n^2)$

The intermediate subforest enumeration

The intermediate subforest enumeration with respect to $F(V_{p-1})$ and $F(V_p)$ is the sequence of forests $F(V_{p-1}) = F_0, F_1 \cdots F_5 = F(V_p)$



(a) left path decomposition (15 relevant subforests)

(b) right path decomposition (11 relevant subforests)

The intermediate subforest enumeration

$$\delta_L(F_{lF,rF}, G_{lG,rG}) = min \begin{cases} \delta(F_{lF,rF} - lF, G_{lG,rG}) + \gamma(lF \to \lambda) \\ \delta(F_{lF,rF}, G_{lG,rG} - lG) + \gamma(\lambda \to lG) \\ \delta(F_{lF,rF} - F_{lF}, G_{lG,rG} - F_{lG}) \\ \quad +\delta(F_{lF} - lF, G_{lG} - lG) + \gamma(lF \to lG) \end{cases}$$

$\mathcal{O}(n^2)$ Space Complexity Implement

Four Memorization tables: D, S, T, and Q

- D(of size $|F||G|$) stores distance values which are needed across different calls of the single-path function. Subproblems in D are **pairs of subtrees without their root nodes**.
- S(of size $|F||G|$) stores all subproblems computed in the two innermost loops C(IF in left decompose) and D(IG in left decompose) or C'(rF in right decompose) and D'(rG in right decompose) for specific nodes in the outer loops.$\rightarrow$ maintain in each round in B and B'.
- T(of size $|G|^2$) stores distances between **a specific subforest in F** and **each relevant subforest of G**. The subforest is either a subtree rooted on the path or a subforest $F_{l'_v,v}$ where l'_v is the leftmost child of p(v).
- Q(of size $|F|$) stores distances between **each subforest of F defined in loop C**(path decomposition) and **a specific subforest** in G of the form $G_{p(r_G)-p(r_G)}$, that **is a subtree in G without root node**. used for the next C loop.

$\mathcal{O}(n^2)$ Space Complexity Implement

Tables indexing:

For $\delta_L(F_{lF,rF}, G_{lG,rG})$

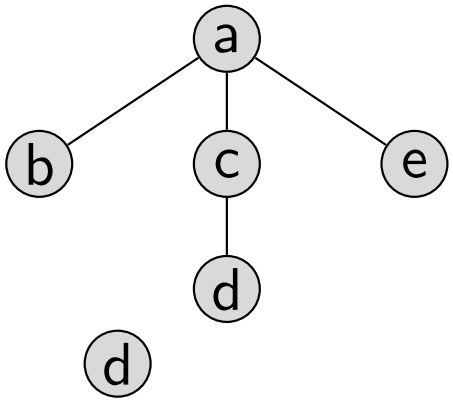
- D: $D[preR\_to\_preL(rF), preR\_to\_preL[rG]]$
- S: $S[lF, lG]$
- T: $T[lG, rG]$
- Q: $Q[lF]$

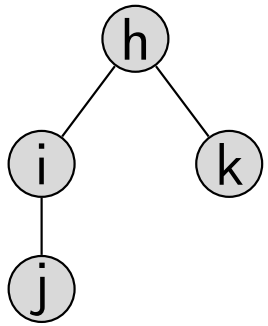
For $\delta_R(F_{lF,rF}, G_{lG,rG})$

- D: $D[lF, rF]$
- S: $S[rF, rG]$
- T: $T[lG, rG]$
- Q: $Q[rF]$

$\mathcal{O}(n^2)$ Space Complexity Implement - Example

Let F and G be two trees





T[d] k