Subtree Isomorphism and Related Problems

Gabriel Valiente
Technical University of Catalonia
Department of Software
valiente@lsi.upc.es

Rīga, March 28–April 13, 2001

Subtree Isomorphism and Related Problems

- Problems related to tree isomorphism
- Basic tree isomorphism algorithm
- Mappings and subtree isomorphism
- Top-down subtree isomorphism
  - Worst-case quadratic time algorithm
- Bottom-up subtree isomorphism
  - Expected linear time algorithm
  - Worst-case linear time algorithm

Subtree Isomorphism and Related Problems

PROBLEMS

RELATED

TO

TREE

ISOMORPHISM

Subtree Isomorphism and Related Problems

- Tree isomorphism
- Subtree isomorphism
- Largest common subgraph
- Smallest common supergraph

under different notions of tree, embedding, and subgraph.

Subtree Isomorphism and Related Problems

Different notions of tree

- rooted or unroteed trees
- ordered or unordered trees
- evolutionary (phylogenetic) trees

Subtree Isomorphism and Related Problems

Different notions of embedding

- graph isomorphism
- topological embedding
- minor containment

Subtree Isomorphism and Related Problems

Different notions of embedding

- graph isomorphism

  There is a subgraph isomorphism of S into T if there is a subgraph of T isomorphic to S, that is, if the nodes of S can be mapped to nodes of T in such a way that the edges of S map to edges in T.

Subtree Isomorphism and Related Problems

Different notions of embedding

- topological embedding

  There is a topological embedding of S into T if a tree isomorphic to S can be obtained from T by a series of contractions of simple paths, that is, if the nodes of S can be mapped to nodes of T in such a way that the edges of S map to node-disjoint paths in T.

Subtree Isomorphism and Related Problems

Different notions of embedding

- minor containment

  There is a minor embedding of S into T if a tree isomorphic to S can be obtained from T by a series of node and edge deletions and edge contractions.

Subtree Isomorphism and Related Problems

Different notions of subgraph

- tree
  - top-down
  - bottom-up
- connected graph
- forest
  - top-down
  - bottom-up

Subtree Isomorphism and Related Problems

BASIC

TREE

ISOMORPHISM

ALGORITHM

Subtree Isomorphism and Related Problems

Tree isomorphism is the basis of naïve solutions to the more general problems of subtree isomorphism, largest common subtree, and perhaps also smallest common supertree.

- A. V. Aho, J. E. Hopcroft, and J. D. Ullman. *The Design and Analysis of Computer Algorithms*. Addison-Wesley, 1974.

The following algorithm determines whether two rooted unordered trees T_1 and T_2 with n nodes are isomorphic in $O(n)$ time.

The algorithm assigns integers to the nodes of the two trees, starting with the leaves and working up towards the roots, in such a way that the trees are isomorphic if and only if their roots are assigned the same integer.

Subtree Isomorphism and Related Problems

1: **procedure** *isomorphic* (T_1, T_2)
2: assign level numbers to all nodes of T_1 and T_2
3: assign to all leaves of T_1 and T_2 the integer 0
4: let L_1 and L_2 be lists of the leaves of T_1 and T_2 at level 0
5: **for all** levels i starting from the previous to last **do**
6: $\langle\langle$*assign integers to all nodes at level* $i\rangle\rangle$
7: **end for**
8: **if** the roots of T_1 and T_2 are assigned the same integer **then**
9: T_1 and T_2 are isomorphic
10: **else**
11: T_1 and T_2 are not isomorphic
12: **end if**
13: **end procedure**

Subtree Isomorphism and Related Problems

$\langle\langle$*assign integers to all nodes at level* $i\rangle\rangle \equiv$

1: **for all** nodes v on list L_1 and w on list L_2 **do**
2: assign to the next component of the tuple associated with the parent of v and w the integer assigned to v and w
3: **end for**
4: let S_1 and S_2 be the sequences of tuples created for the nonleaves of T_1 and T_2 on level i
5: bucket sort S_1 and S_2
6: **if** $S_1 \neq S_2$ **then**
7: T_1 and T_2 are not isomorphic
8: **else**
9: $\langle\langle$*build lists of leaves at level* $i\rangle\rangle$
10: **end if**

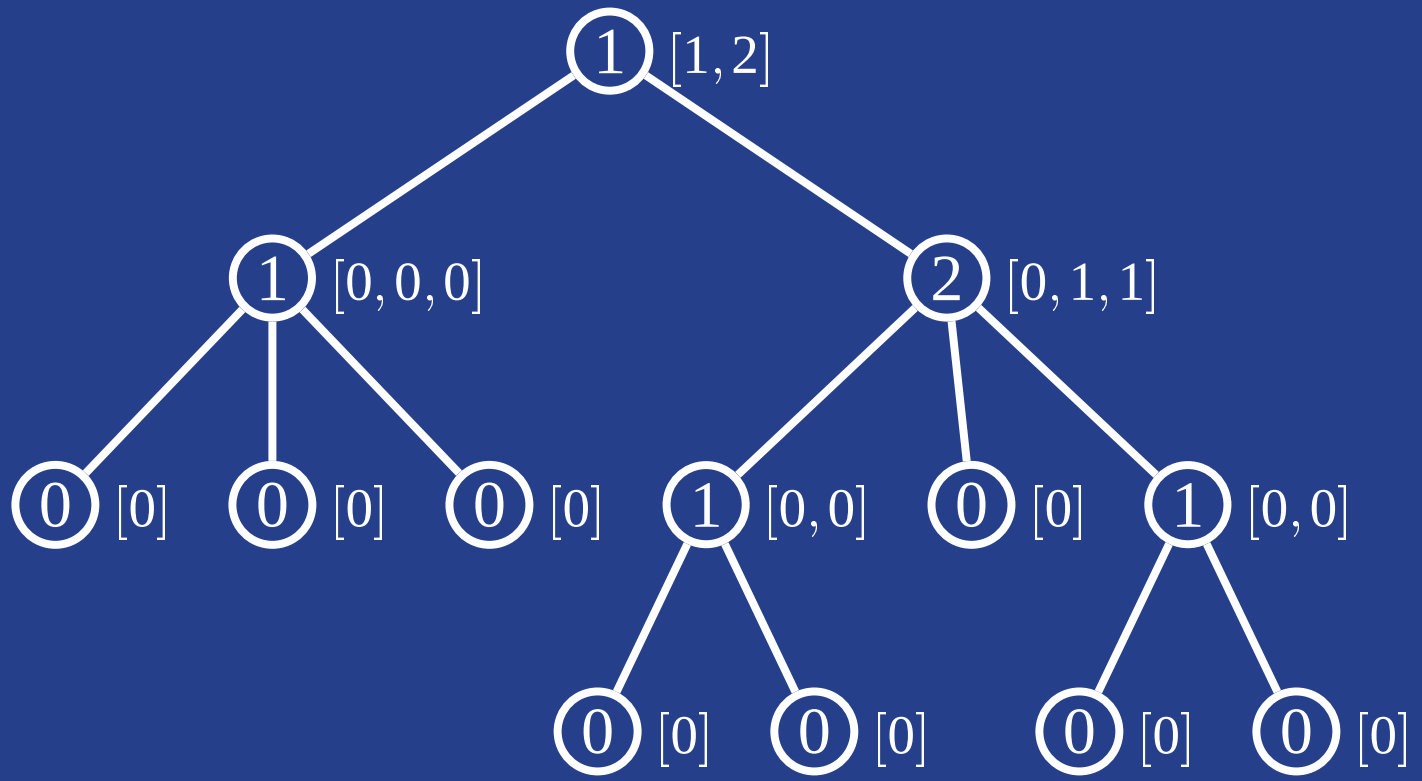
Subtree Isomorphism and Related Problems

$\langle\langle$*build lists of leaves at level* $i\rangle\rangle \equiv$

1: let L_1 be an empty list of nodes
2: **for all** k from 1 to the number of distinct tuples on S_1 **do**
3: **for all** nodes v of T_1 on level i represented by the kth distinct tuple on S_1 **do**
4: assign to node v the integer k
5: append node v to L_1
6: **end for**
7: **end for**
8: append to the front of L_1 all leaves of T_1 on level i
9: let L_2 be the corresponding list of nodes of T_2

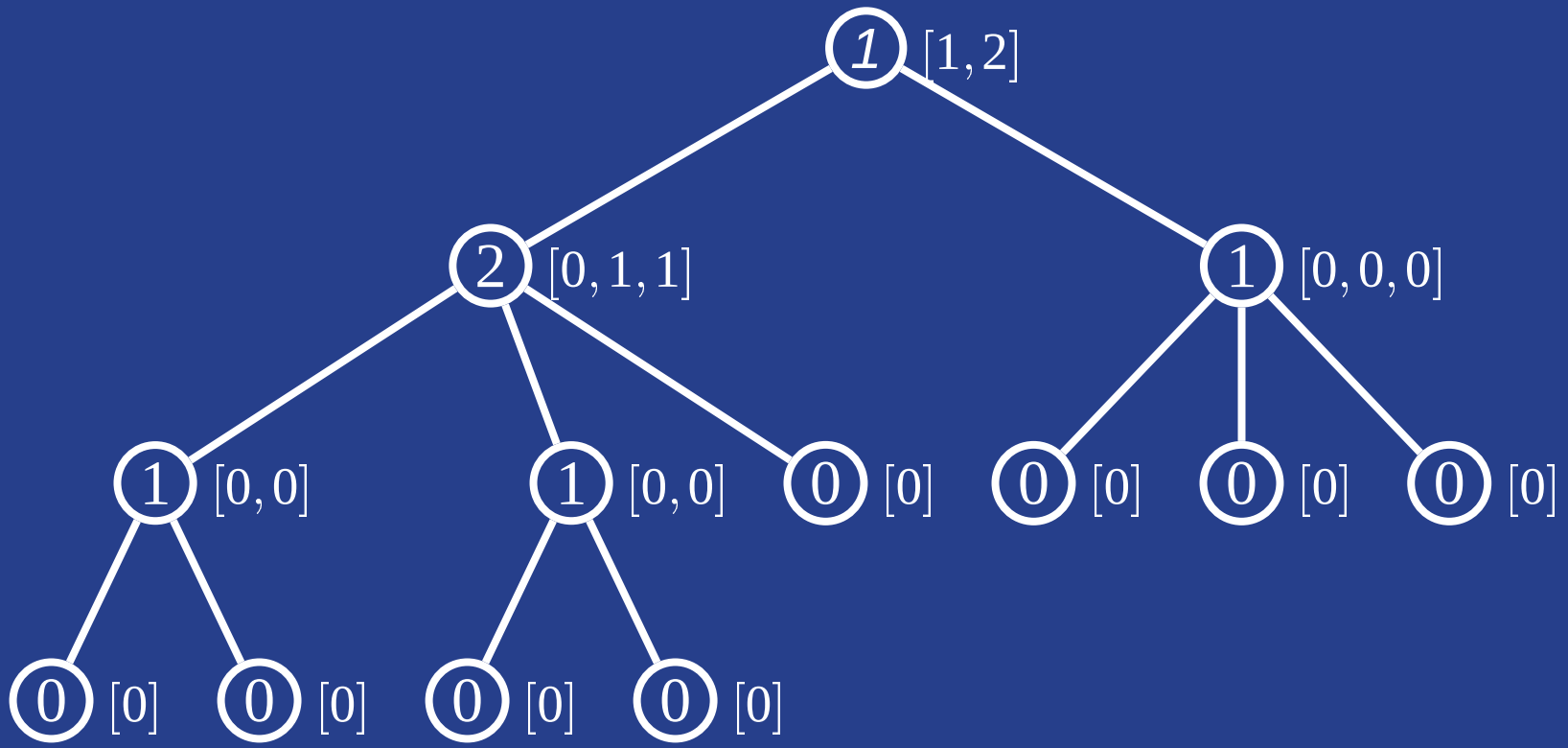
Subtree Isomorphism and Related Problems

Example. *Numbers assigned by the tree isomorphism algorithm.*



Subtree Isomorphism and Related Problems

Example. *Numbers assigned by the tree isomorphism algorithm.*



Subtree Isomorphism and Related Problems

MAPPINGS

AND

SUBTREE

ISOMORPHISM

Subtree Isomorphism and Related Problems

A mapping establishes a one-to-one correspondence between the nodes of two ordered trees which preserves the order of siblings and ancestors.

Mappings were introduced in

- K.-C. Tai. The tree-to-tree correction problem. *J. ACM*, 26(3):422–433, 1979.

in order to describe how a sequence of edit operations transforms a tree into another one.

Subtree Isomorphism and Related Problems

A mapping from a tree T_1 to a tree T_2 is a set M of ordered pairs of integers (i, j), $1 \leqslant i \leqslant n_1$, $1 \leqslant j \leqslant n_2$ such that

- $i_1 = i_2$ if, and only if, $j_1 = j_2$
- $t_1[i_1]$ is to the left of $t_1[i_2]$ if, and only if, $t_2[j_1]$ is to the left of $t_2[j_2]$
- $t_1[i_1]$ is an ancestor of $t_1[i_2]$ if, and only if, $t_2[j_1]$ is an ancestor of $t_2[j_2]$

for all $(i_1, j_1), (i_2, j_2) \in M$, where $t[i]$ denotes the node of T whose position in the postorder traversal of T is i.

Subtree Isomorphism and Related Problems

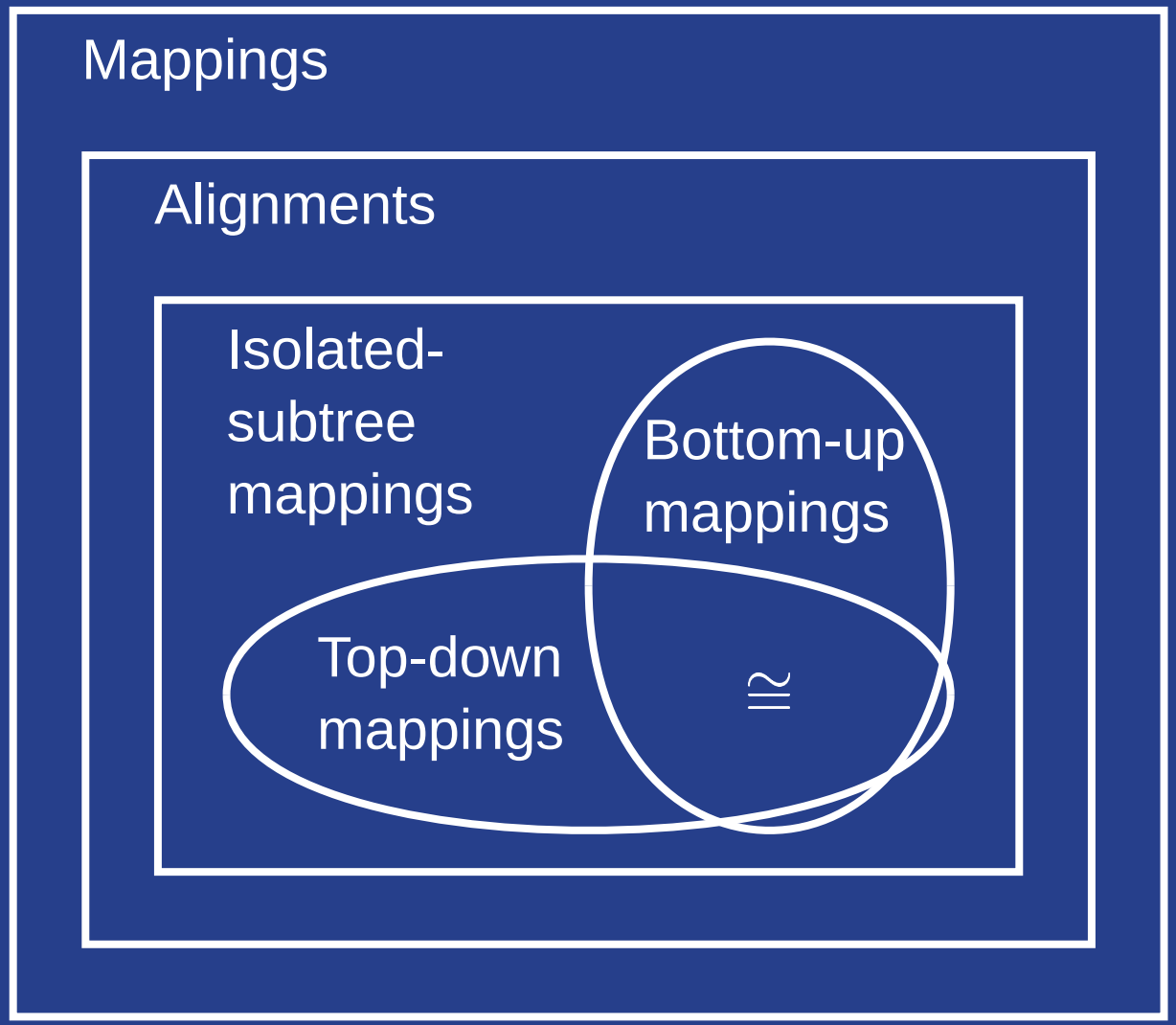
A mapping from a tree T_1 to a tree T_2 describes the edit operations that allow to transform T_1 into T_2.

- A node $t_1[i]$ with no pair $(i, j) \in M$ is deleted from T_1.
- A pair $(i, j) \in M$ indicates the substitution of node $t_1[i]$ by node $t_2[j]$.
- A node $t_2[j]$ with no pair $(i, j) \in M$ is inserted into T_2.

Subtree Isomorphism and Related Problems

- A mapping is an alignment if it can be extended to an isomorphism between the underlying unlabeled trees after inserting a least number of nodes into the two trees
- A mapping is isolated-subtree if it maps disjoint subtrees to disjoint subtrees
- A mapping is top-down if the parents of nodes in the mapping are also in the mapping
- An isolated-subtree mapping is bottom-up if the children of nodes in the mapping are also in the mapping

Subtree Isomorphism and Related Problems



Subtree Isomorphism and Related Problems

TOP-DOWN

SUBTREE

ISOMORPHISM

Subtree Isomorphism and Related Problems

Top-down subtree isomorphism was introduced in

- S. M. Selkow. The tree-to-tree editing problem. *Inform. Process. Lett.*, 6(6):184–186, 1977.
- W. Yang. Identifying syntactic differences between two programs. *Software—Practice and Experience*, 21(7):739–755, 1991.

where an algorithm was given to compute the distance between two trees T_1 and T_2 in $O(n_1 n_2)$ time.

In a top-down mapping, the parents of nodes in the mapping are also in the mapping.

Subtree Isomorphism and Related Problems

A mapping M from a tree T_1 to a tree T_2 is top-down if it satisfies the following condition:

- if $(i, j) \in M$ then $(par(i), par(j)) \in M$

for all i, j such that $t_1[i]$ and $t_2[j]$ are not the root of T_1 and T_2, respectively, where $par(i)$ denotes the postorder number of the parent of node $t[i]$.

The top-down distance from tree T_1 to tree T_2 is the cost of a least-cost top-down mapping between T_1 and T_2.

Subtree Isomorphism and Related Problems

Consider first the dynamic programming algorithm introduced in

- D. S. Hirschberg. A linear space algorithm for computing maximal common subsequences. *Commun. ACM*, 18(6):341–343, 1975.

which is based on the observation that the longest common subsequence of the two sequences $[a_1, \ldots, a_m]$ and $[b_1, \ldots, b_n]$ can be computed from the three longest common subsequences of

- $[a_1, \ldots, a_m]$ and $[b_1, \ldots, b_{n-1}]$ (deletion)
- $[a_1, \ldots, a_{m-1}]$ and $[b_1, \ldots, b_n]$ (insertion)
- $[a_1, \ldots, a_{m-1}]$ and $[b_1, \ldots, b_{n-1}]$ (substitution)

Subtree Isomorphism and Related Problems

1: **procedure** $lcs([a_1, \ldots, a_m], [b_1, \ldots, b_n])$
2: let $M[i, 0]$ be 0 **for all** i from 0 to m
3: let $M[0, j]$ be 0 **for all** j from 0 to n
4: **for all** i from 1 to m **do**
5: **for all** j from 1 to n **do**
6: let $W[i, j]$ be $[a_i = b_j]$
7: let $M[i, j]$ be $max(M[i, j-1], M[i-1, j], M[i-1, j-1] + W[i, j])$
8: **end for**
9: **end for**
10: return $M[m, n]$
11: **end procedure**

Subtree Isomorphism and Related Problems

In the longest common subsequence algorithm,

- $W_{i,j}$ is either 0 or 1, depending on whether a_i and b_j are identical elements
- $M_{i,j}$ denotes the length of a longest common subsequence of the two prefixes $[a_1, \ldots, a_i]$ and $[b_1, \ldots, b_j]$.

Sequences can be seen as ordered trees whose height is 2.

Subtree Isomorphism and Related Problems

The longest common subsequence algorithm can be generalized to find the number of pairs in a largest matching of two trees, by extending the meaning of the weight matrix W.

- $W_{i,j}$ denotes the number of pairs in a largest matching of the subtrees rooted at a_i and b_j.

- $M_{i,j}$ denotes the number of pairs in a largest matching between the two forests of trees rooted at $a_1, \ldots, a_i$ and $b_1, \ldots, b_j$.

The algorithm recursively finds the number of pairs in a largest matching between first-level subtrees of A and B.

Subtree Isomorphism and Related Problems

If the roots of A and B contain distinct elements, then the two trees for not match at all.

If the roots contain identical elements, then the algorithm recursively finds the number of pairs in a largest matching between first-level subtrees of A and B.

Subtree Isomorphism and Related Problems

```
1:  procedure match(A, B)
2:      if root(A) and root(B) contain distinct elements then
3:          return 0
4:      else
5:          let m and n be the number of first-level subtrees of A and B
6:          let M[i, 0] be 0 for all i from 0 to m
7:          let M[0, j] be 0 for all j from 0 to n
8:          ⟨⟨fill-in mapping matrix⟩⟩
9:          return M[m, n] + 1
10:     end if
11: end procedure
```

Subtree Isomorphism and Related Problems

$\langle\langle$*fill-in mapping matrix*$\rangle\rangle \equiv$

```
1: for all i from 1 to m do
2:     for all j from 1 to n do
3:         let A_i be the ith first-level subtree of A
4:         let B_j be the jth first-level subtree of B
5:         let W[i, j] be match(A_i, B_j)
6:         let M[i, j] be max(M[i, j−1], M[i−1, j], M[i−1, j−1]+W[i, j])
7:     end for
8: end for
```

Subtree Isomorphism and Related Problems

BOTTOM-UP

SUBTREE

ISOMORPHISM

Subtree Isomorphism and Related Problems

Bottom-up subtree isomorphism was introduced in

- G. Valiente. Simple and efficient subtree isomorphism. Technical Report LSI-00-72-R, Technical University of Catalonia, Department of Software, 2000.

- G. Valiente. Simple and efficient tree comparison. Technical Report LSI-01-1-R, Technical University of Catalonia, Department of Software, 2001.

where an algorithm was given to compute the distance between two trees T_1 and T_2 in expected $O(n_1 + n_2)$ time.

Subtree Isomorphism and Related Problems

In a bottom-up mapping, the children of nodes in the mapping are also in the mapping.

An isolated-subtree mapping M from a tree T_1 to a tree T_2 is bottom-up if it satisfies the following condition:

- if $(i, j) \in M$ then $(i_1, j_1), \ldots, (i_k, j_k) \in M$

where $t_1[i_1], \ldots, t_1[i_k]$ are the children of node $t_1[i]$ and $t_2[j_1], \ldots, t_2[j_k]$ are the children of node $t_2[j]$.

The bottom-up distance from tree T_1 to tree T_2 is the cost of a least-cost bottom-up mapping between T_1 and T_2.

Subtree Isomorphism and Related Problems

The algorithm is based on a reduction of the tree pattern matching problem to the extension to forests of the common subexpression problem:

Represent a rooted tree in a maximally compact form as a directed acyclic graph, where common (isomorphic) subtrees are factored and shared.

Subtree Isomorphism and Related Problems

The common subexpression problem was introduced in

- P. J. Downey, R. Sethi, and R. E. Tarjan. Variations on the common subexpression problem. *J. ACM*, 27(4):758–771, 1980.

- P. Flajolet, P. Sipala, and J.-M. Steyaert. Analytic variations on the common subexpression problem. In *Automata, Languages, and Programming*, volume 443 of *Lecture Notes in Computer Science*, pages 220–234. Springer-Verlag, 1990.

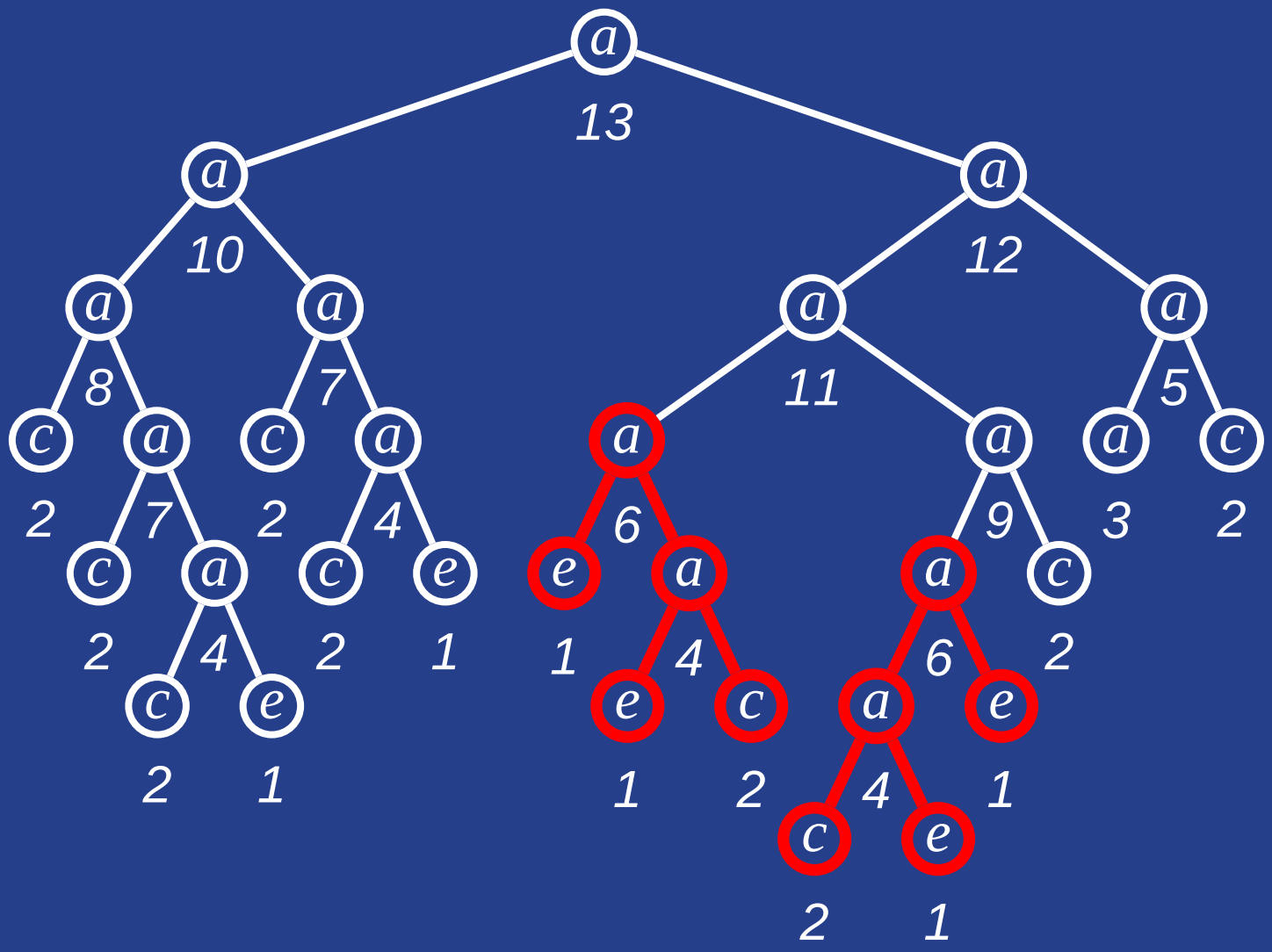
Subtree Isomorphism and Related Problems

A rooted, oriented, random tree of size n has a compacted form of expected size $O(n/\sqrt{\log n})$.

- P. Flajolet and J.-M. Steyaert. A complexity calculus for recursive tree algorithms. *Math. Syst. Theory*, 19(4):301–331, 1987.

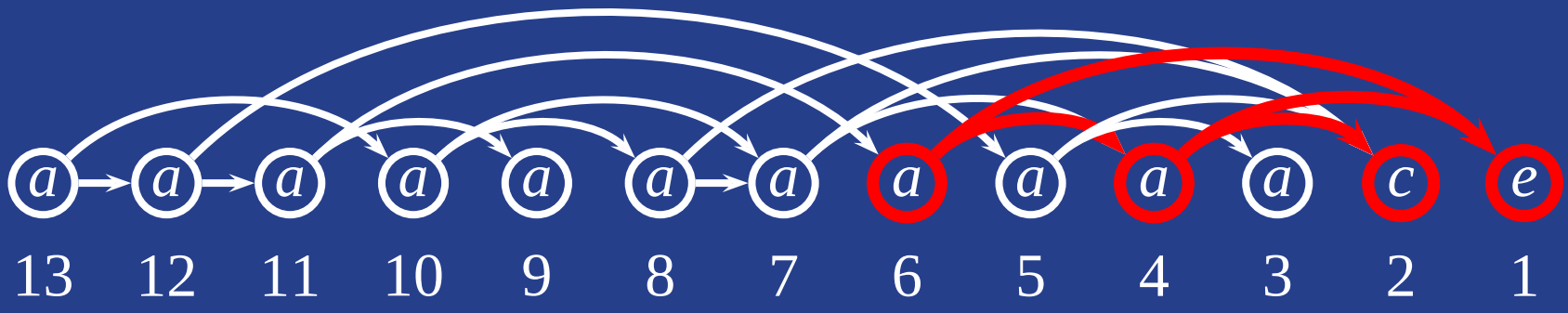
Subtree Isomorphism and Related Problems

Example.



Subtree Isomorphism and Related Problems

Example.



Subtree Isomorphism and Related Problems

The algorithm assigns integers to the nodes of a forest, in such a way that any two nodes have the same integer assigned if, and only if, the subtrees rooted at them are isomorphic.

The set of rooted subtrees of the forest is thus partitioned into isomorphism equivalence classes.

Subtree Isomorphism and Related Problems

The algorithm improves previous algorithms for partitioning a rooted tree into isomorphism equivalence classes.

- R. Grossi. On finding common subtrees. *Theor. Comput. Sci.*, 108(2):345–356, 1993.
- Y. Dinitz, A. Itai, and M. Rodeh. On an algorithm of Zemlyachenko for subtree isomorphism. *Inform. Process. Lett.*, 70(3):141–146, 1999.

Subtree Isomorphism and Related Problems

Bottom-up traversal of a forest is equivalent to performing a topological sort on the graph obtained by reversing the direction of all directed edges in the graph that represents the forest.

Subtree Isomorphism and Related Problems

```
1:  procedure bottom-up (F)
2:      let q be an empty queue of nodes
3:      for all nodes v in forest F do
4:          let parent[v] be the parent of node v
5:          let children[v] be the degree of node v
6:          if children[v] = 0 then
7:              enqueue node v into q
8:          end if
9:      end for
10:     ⟨⟨main loop⟩⟩
11: end procedure
```

Subtree Isomorphism and Related Problems

$\langle\langle$*main loop*$\rangle\rangle \equiv$

```
1: repeat
2:     dequeue node v from q
3:     if node v is not the root of a tree in the forest then
4:         decrement children[parent[v]] by one
5:         if children[parent[v]] = 0 then
6:             enqueue node parent[v] into q
7:         end if
8:     end if
9: until the queue q is empty
```

Subtree Isomorphism and Related Problems

```
1:  procedure isomorphism(F)
2:      let q be an empty queue of nodes
3:      for all nodes v in forest F do
4:          let parent[v] be the parent of node v
5:          set size[v] to one
6:          let children[v] be the degree of node v
7:          if children[v] = 0 then
8:              enqueue node v into q
9:          end if
10:     end for
11:     set count to zero
12:     ⟨⟨main loop⟩⟩
13: end procedure
```

Subtree Isomorphism and Related Problems

```
⟨⟨main loop⟩⟩ ≡
repeat
    dequeue node v from q
    ⟨⟨assign integer to subtree rooted at node v⟩⟩
    if node v is not the root of a tree in the forest then
        increment size[parent[v]] by size[v]
        decrement children[parent[v]] by one
        if children[parent[v]] = 0 then
            enqueue node parent[v] into q
        end if
    end if
until the queue q is empty
```

Subtree Isomorphism and Related Problems

$\langle\langle$*assign integer to subtree rooted at node* $v\rangle\rangle \equiv$

1: let D be a dictionary of lists of integers
2: let L be an empty list of integers
3: **for all** edges (v, w) in the forest **do**
4: append $integer[w]$ to L
5: **end for**
6: bucket sort L
7: insert $label[v]$ at front of L
8: $\langle\langle$*set integer for node* $v\rangle\rangle$

Subtree Isomorphism and Related Problems

$\langle\langle$*set integer for node* $v\rangle\rangle \equiv$

1: lookup L in dictionary D
2: **if** found **then**
3: set $integer[v]$ to the value found
4: **else**
5: increment *count* by one
6: insert $\langle L, count\rangle$ in dictionary D
7: set $integer[v]$ to *count*
8: **end if**

Subtree Isomorphism and Related Problems

The algorithm allows to solve several tree comparison problems with the help of a simple data structure, which can be sorted in linear time using bucket sort according to different criteria.

1. Subtree isomorphism.

   Find all the subtrees in a given forest which are isomorphic to the subtree rooted at a given node.

Subtree Isomorphism and Related Problems

The algorithm allows to solve several tree comparison problems with the help of a simple data structure, which can be sorted in linear time using bucket sort according to different criteria.

2. Largest common subtree.

   Find all the largest common subtrees in a given forest.

   More in general, find all the k-th largest or the k-th smallest common subtrees in the given forest.

Subtree Isomorphism and Related Problems

The algorithm allows to solve several tree comparison problems with the help of a simple data structure, which can be sorted in linear time using bucket sort according to different criteria.

3. Most often repeated subtree.

   Find all the subtrees in a given forest that are repeated most often.

   More in general, find in the given forest all the k-th most often or the k-th least often repeated subtrees.

Subtree Isomorphism and Related Problems

The following improved algorithm performs a bottom-up traversal of a given forest F of rooted unlabeled trees and builds the compacted directed acyclic graph representation G in time linear in the number of nodes.

- Trees in a given forest are manipulated and compared with the compacted directed acyclic graph representation of the forest by means of a node mapping, instead of computing isomorphism codes.
- A simple, practical algorithm is obtained, running in worst-case time linear in the number of nodes, and with small hidden constants.

Subtree Isomorphism and Related Problems

```
1:  procedure subtree isomorphism (F: forest, G: graph)
2:      set G to an empty directed graph
3:      add a new node z to G
4:      set height[z] to one
5:      let map be a dynamic map of nodes of F to nodes of G
6:      let Q be an empty queue of nodes
7:      for all nodes v in F do
8:          let parent[v] be the parent of node v
9:          let children[v] be the degree of node v
10:         if children[v] = 0 then
11:             enqueue node v into Q
12:         end if
13:     end for
```

```
14:     repeat
15:         dequeue node v from Q
16:         ⟨⟨set height of node v⟩⟩
17:         if the degree of node v is zero then
18:             set map[v] to z
19:         else
20:             set found to false
21:             ⟨⟨find corresponding node⟩⟩
22:             if not found then
23:                 add a new node w to G
24:                 set map[v] to w
25:                 set height[w] to height[v]
26:                 for all children u of node v do
27:                     add a new arc in G from node w to node map[u]
```

```
28:             end for
29:         end if
30:     end if
31:     if node v is not the root of a tree in F then
32:         decrement children[parent[v]] by one
33:         if children[parent[v]] = 0 then
34:             enqueue node parent[v] into Q
35:         end if
36:     end if
37: until the queue Q is empty
38: end procedure
```

Subtree Isomorphism and Related Problems

$\langle\langle$*set height of node* $v\rangle\rangle \equiv$

1: set max to one
2: **for all** children u of node v **do**
3: set max to the maximum between max and $height[u]$
4: **end for**
5: increment $height[v]$ by max

Subtree Isomorphism and Related Problems

$\langle\langle$*find corresponding node*$\rangle\rangle \equiv$

```
1:  for all nodes w in G in reverse order do
2:      if height[v] ≠ height[w] or degree[v] ≠ outdegree[w] then
3:          break
4:      end if
5:      let V be an empty list of nodes
6:      for all children u of node v do
7:          append map[u] to V
8:      end for
9:      let W be an empty list of nodes
10:     for all children u of node w do
11:         append w to W
12:     end for
```

```
13:     if V = W then
14:         set map[v] to w
15:         set found to true
16:         break
17:     end if
18: end for
```

Subtree Isomorphism and Related Problems

- The algorithm can be used to find all occurrences of a pattern tree in a text tree, or even all occurrences of every subtree of a pattern in a text, in time linear in the total number of nodes.
- The algorithm handles multiple pattern trees and also multiple text trees, and deals with ordered and unordered labeled trees.

Subtree Isomorphism and Related Problems

- Dealing with unordered trees just requires sorting the list V of nodes in the graph which the children of node v in the forest have been mapped to, and the list W of children of node w in the graph, right before the $V = W$ equality test.

  Sorting these lists of nodes can be made in time linear in the length of the lists (the degree of node v in the forest, same as the outdegree of node w in the graph) using bucket sort, for instance on the node number in the actual representation of the graph.

Subtree Isomorphism and Related Problems

- Dealing with labeled trees requires solving the additional problem of partitioning the set of leaves in the forest into classes of nodes having the same label, instead of mapping all leaves to a same node in the graph.

  Assuming all labels are integers in the range $[1, n]$, where n is the number of nodes in the forest, the partitioning can be realized in $O(n)$ time using bucket sort followed by a traversal of the sorted list of nodes.