Tree Isomorphism and Related Problems

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

under different notions of

- tree
  - rooted or unroteed trees
  - ordered or unordered trees
  - evolutionary or phylogenetic trees
- isomorphism (embedding relation)
  - graph isomorphism
  - topological embedding
  - minor containment
- subgraph
  - tree
  - connected graph
  - forest

Tree Isomorphism and Related Problems

Let us recall the different embedding relations, from the most restrictive to the most general one.

- 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.
- 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.
- 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.

Tree 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.

Tree 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 be a list of the leaves of T_1 at level 0
5: let L_2 be a list of the leaves of T_2 at level 0
6: **for all** levels i starting from 1 **do**
7: $\langle\langle$*assign integers to all nodes at level* $i\rangle\rangle$
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 procedure**

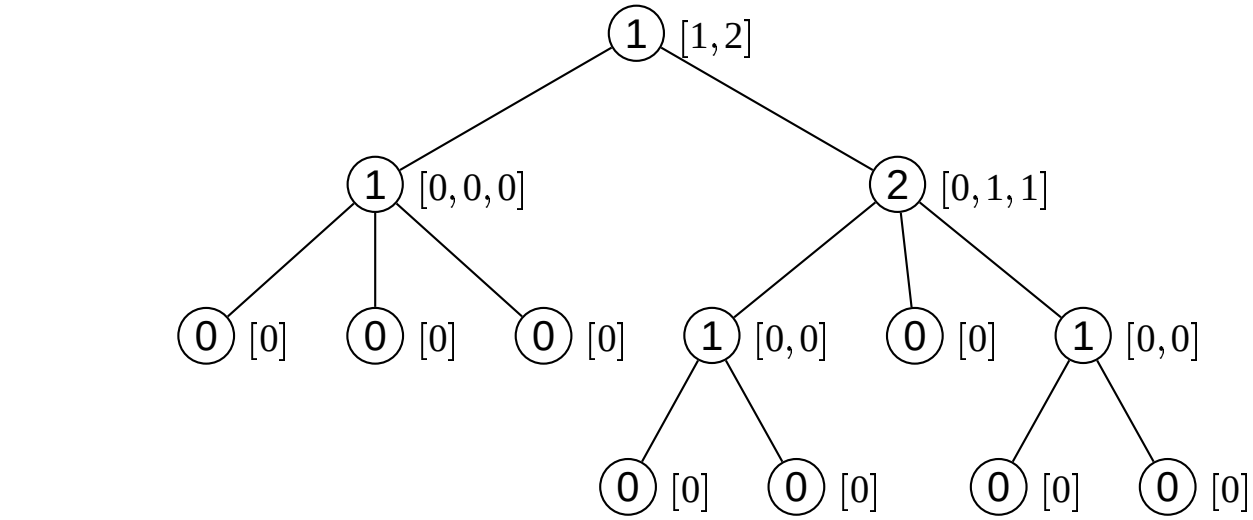
Tree Isomorphism and Related Problems

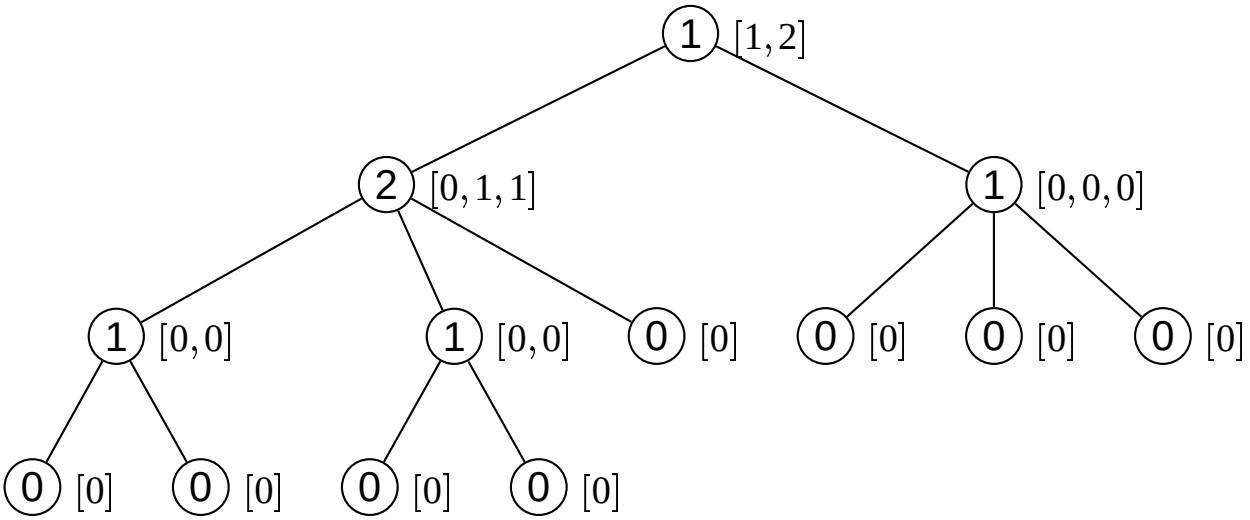
$\langle\langle$*assign integers to all nodes at level* $i\rangle\rangle \equiv$

1: **for all** nodes v on list L_1 **do**
2: assign to the next component of the tuple associated with the parent of v the integer assigned to v
3: let S_1 be the sequence of tuples created for the nonleaves of T_1 on level i
4: let S_2 be the corresponding sequence of tuples of T_2
5: bucket sort S_1
6: bucket sort S_2
7: **if** $S_1 \neq S_2$ **then**
8: T_1 and T_2 are not isomorphic
9: **else**
10: let L_1 be an empty list of nodes
11: **for all** k from 1 to the number of distinct tuples on S_1 **do**
12: **for all** nodes v of T_1 on level i represented by the kth distinct tuple on S_1 **do**
13: assign to node v the integer k
14: append node v to L_1
15: append to the front of L_1 all leaves of T_1 on level i
16: let L_2 be the corresponding list of nodes of T_2

Tree Isomorphism and Related Problems

Numbers assigned by the tree isomorphism algorithm.





Tree 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. 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.

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 $t_2[j]$, and a node $t_2[j]$ with no pair $(i, j) \in M$ is inserted into T_2.

Tree 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.

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.

Tree 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)

```
1: procedure lcs([a_1, ..., a_m], [b_1, ..., 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:     return M[m, n]
9: end procedure
```

Tree 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. 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 $b_1, \ldots, b_j$.

Tree 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.

```
1:  procedure match(A, B)
2:      if root(A) and root(B) contain distinct elements then
3:          return 0
4:      else
5:          let m be the number of first-level subtrees of A
6:          let n be the number of first-level subtrees of B
7:          let M[i, 0] be 0 for all i from 0 to m
8:          let M[0, j] be 0 for all j from 0 to n
9:          for all i from 1 to m do
10:             for all j from 1 to n do
11:                 let A_i be the ith first-level subtree of A
12:                 let B_j be the jth first-level subtree of B
13:                 let W[i, j] be match(A_i, B_j)
14:                 let M[i, j] be
                    max(M[i, j − 1], M[i − 1, j], M[i − 1, j − 1] + W[i, j])
15:         return M[m, n] + 1
16: end procedure
```

In order to account for the fact that the roots of the trees A and B match, 1 is added to $M[m, n]$ on line 15.

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

A 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.

Tree 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.

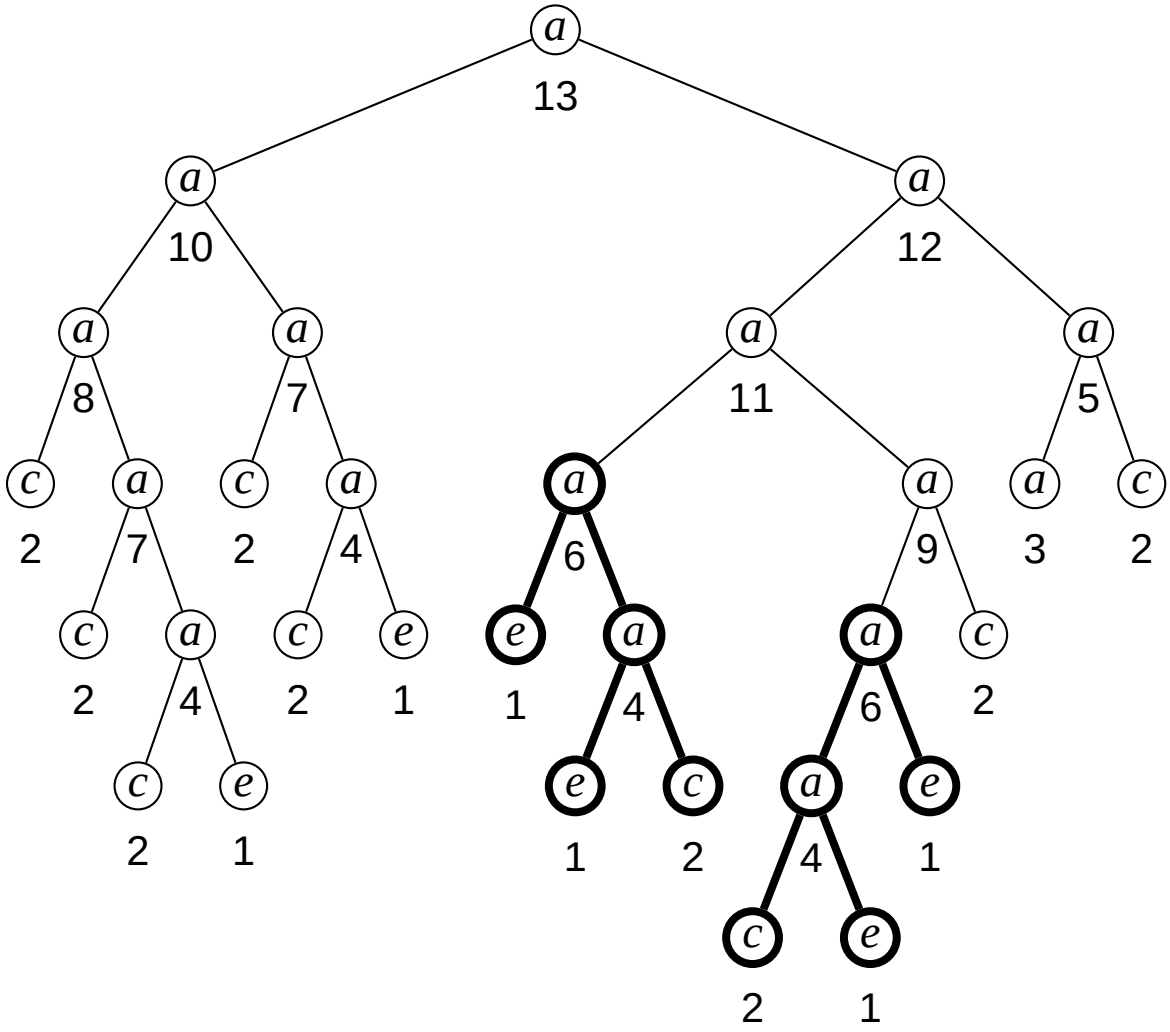
The common subexpression problem was introduced in

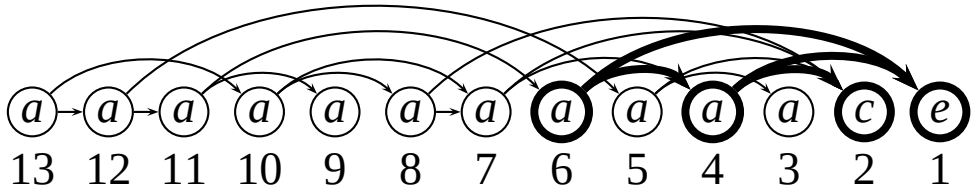
- P. J. Downey, R. Sethi, and R. E. Tarjan. Variations on the common subexpression problem. *J. ACM*, 27(4):758–771, 1980.
- P. Flajolet, P. Sipala, and J.-M. Steyaert. Analytic variations on the common subexpression problem. In *Automata, Languages, and Programming*, volume 443 of *Lecture Notes in Computer Science*, pages 220–234. Springer-Verlag, 1990.

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

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

Tree Isomorphism and Related Problems





Tree 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.

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.

Tree 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:      set count to zero
10:     repeat
11:         dequeue node v from q
12:         ⟨⟨assign integer to subtree rooted at node v⟩⟩
13:         if node v is not the root of a tree in the forest then
14:             increment size[parent[v]] by size[v]
15:             decrement children[parent[v]] by one
16:             if children[parent[v]] = 0 then
17:                 enqueue node parent[v] into q
18:     until the queue q is empty
19: end procedure
```

Tree 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:  bucket sort L
6:  insert label[v] at front of L
7:  lookup L in dictionary D
8:  if found then
9:      set integer[v] to the value found
10: else
11:     increment count by one
12:     insert ⟨L, count⟩ in dictionary D
13:     set integer[v] to count
```

Tree Isomorphism and Related Problems

The algorithm allows to solve several tree comparison problems in linear time

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

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.

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

with the help of a simple data structure, which can be sorted in linear time using bucket sort according to different criteria.