Trees, Subtrees and Isomorphism: A Revisit

Chen Shaoyuan

ACM-ICPC Training Team, Nanjing University

June 9, 2019

Definitions

(Unrooted) tree : a connected acyclic graph.

Rooted tree : an orientation of a tree, where exactly one vertex, called the root of the tree, has no incoming edge; every other vertex has exactly one incoming edge. A rooted tree can also be recursively defined as a finite multiset of rooted trees.

Graph isomorphism : two graphs (V_1, E_1) and (V_2, E_2), are called isomorphic, if there exists a bijection $\phi : V_1 \rightarrow V_2$, such that, for every vertex pair $u, v \in V_1$, $uv \in E_1$ iff $\phi(u)\phi(v) \in E_2$.

Subtree : a subtree of an unrooted tree T is either T itself, or one of the two components obtained by removing any edge of T.
A subtree rooted at u of a rooted tree T, is the rooted tree containing u all descendants of u.

Definitions

Generalized subtree (g-subtree) A generalized subtree of unrooted tree T is a connected subgraph of T.
A generalized subtree of rooted tree T is a subgraph of T which is also a rooted tree.

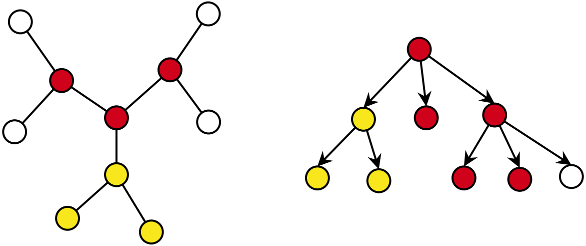


Figure: The yellow part is a subtree of the whole graph; the red part is a g-subtree of the whole tree.

Rooted Tree Isomorphism

Given two rooted trees T_1 and T_2, decide if they are isomorphic.

Solution (unique representation via sorting)

For every node, we may sort its child subtrees in some order. The resulting ordered trees are unique for two isomorphic trees.

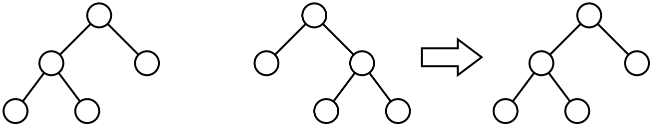


Figure: Canonicalization of trees via sorting.

We can also use tree hashing to avoid sorting.

Unrooted Tree Isomorphism

But how can we decide the isomorphism of two unrooted trees?

Theorem

We can solve the unrooted tree isomorphism problem based on rooted tree isomorphism algorithm in the same asymptotic time.

Basic idea: root the trees at their centroids. A tree has either one or two centroids. In case that a tree has two centroids, try both possibilities.

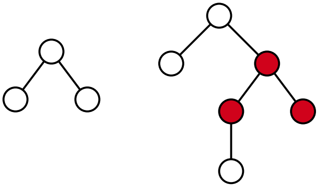
centroid The centroid of a graph G is defined as

$$\arg\min_{u \in V(G)} \max_{v \in V(G)} d(u, v)$$

The centroids of a tree can be found in linear time by dynamic programming or running BFS twice.

Subtree Isomorphism, Rooted

Given rooted trees S and T, decide if there exists a **g-subtree** of T isomorphic to S.



There might be exponentially many g-subtrees in a given tree!

Subtree Isomorphism, Rooted

Solution (dynamic programming)

We may use dynamic programming to solve this problem. Let for $u \in S, v \in T$, define $M[u, v] = 1$ if S_u is isomorphic to some g-subtree of T rooted at v; otherwise $M[u, v] = 0$. $M[u, v]$ can be computed from $M[x, y]$ where $x \in C(u), y \in C(v)$.

We may define a bipartite graph $C(u) \cup C(v)$ where $x \in C(u)$ and $y \in C(v)$ are connected by an edge if $M[x, y] = 1$. Note that $M[u, v] = 1$ if and only if the bipartite graph has a $C(u)$-perfect matching.

Subtree Isomorphism, Rooted

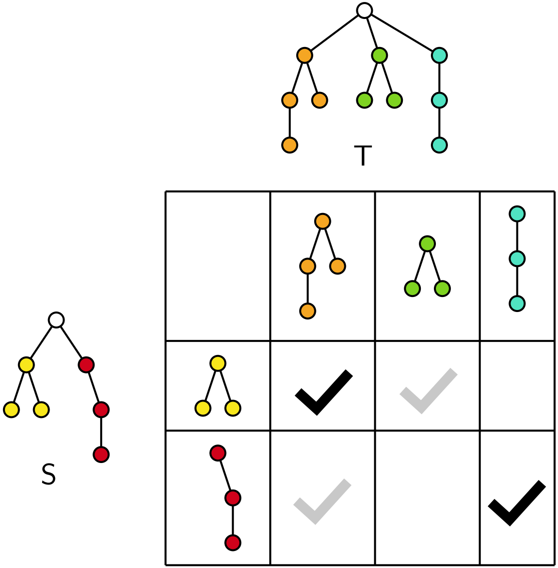
Algorithm 1 Rooted Subtree Isomorphism

Input: two rooted trees S and T

Output: decide if S is isomorphic to some g-subtree of T

initialize $M[\cdot,\cdot]$ with 0
for vertex v in post-order traversal of T **do**
    for each vertex u is S **do**
        build bipartite graph $G = (C(u) \cup C(v), M[C(u), C(v)])$
        set $M[u, v] = 1$ is G has a $C(u)$-perfect matching
    end for
end for
return true if $M[root(S), v] = 1$ for some $v \in T$

Subtree Isomorphism, Rooted



Subtree Isomorphism, Rooted

Assume we use Hopcroft-Karp algorithm (or, equivalently, Dinic's maximum flow) to find a maximum matching in $O(E\sqrt{V})$ time. The total time complexity is

$$\begin{aligned}
time &= \sum_{u\in S}\sum_{v\in T} time(u,v) \\
&= \sum_{u\in S}\sum_{v\in T} \deg(u)\deg(v)\sqrt{\deg(u)+\deg(v)} \\
&\le \sum_{u\in S}\sum_{v\in T} \deg(u)\deg(v)\sqrt{n_s+n_t} \\
&\le n_s n_t \sqrt{n_s+n_t} \\
&= O(n_s n_t^{3/2})
\end{aligned}$$

Subtree Isomorphism, Rooted

About Maximum Bipartite Matching

Algorithm 2 Hopcroft-Karp Algorithm, $O(E\sqrt{V})$

Input: a bipartite graph $U \cup V$

Output: a maximum matching $\mathscr{M}$

```
set 𝓜 = ∅
while there exists augmenting path do
    let A be a maximal set of vertex-disjoint shortest augmenting paths
    set 𝓜 = 𝓜 ⊕ A
end while
return 𝓜
```

Lemma

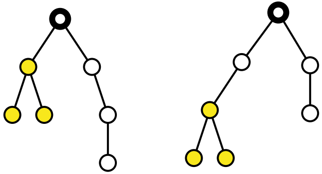
The length of shortest augmenting path strictly increases after each iteration of Hopcroft-Karp algorithm.

- Each iteration of Hopcroft-Karp can be done in $O(E)$;
- After the first $\sqrt{V}$ iterations the length of shortest augmenting path is at least $\sqrt{V}$;
- Now the symmetric difference of the current matching and the maximum matching is the disjoint union of several augmenting paths and alternating cycles; Since each augmenting path has length at least $\sqrt{V}$, there are at most $\sqrt{V}$ augmenting paths;
- Conclusion: Hopcroft-Karp terminates after at most $2\sqrt{V}$ iterations.

What if the trees are unrooted?

Just root T arbitrarily and try all possible roots of S. The total time complexity is $O(n_s^2 n_t^{3/2})$

But we can still optimize the algorithm. The basic observation is, there may be common subtrees in different rootings of S.

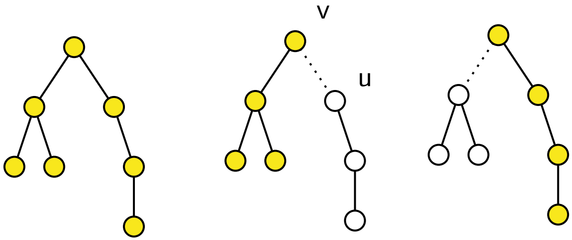


A natural idea is to use hashmap to dedupliate subtrees. However, this helps little in the worst case (consider a star-like graph).

Subtree Isomorphism, Unrooted

Note that all subtrees can be found as follows:

> *Root S at u. The resulting rooted tree S^u itself is a rooted subtree. Also, removing any child subtree of S^u yields a rooted subtree.*

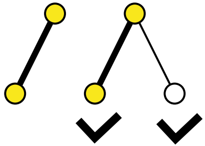


We use $S[u, v]$ to denote the subtree rooted at v after removing child subtree u.

Subtree Isomorphism, Unrooted

We may compute the maximum bipartite matchings for all these trees together. Formally, let U_i denote U removing the i-th element u_i, we want to compute maximum bipartite matching for $U_i \cup V$ for every i. This results in an unrooted subtree isomorphism algorithm in the same time as the rooted case.

Note that removing u_i decreases the size of maximum matching by at most 1. In case that u_i is unmatched, or reachable from any unmatched vertex via alternating path, in any maximum bipartite matching of $U \cup V$, the removal of u_i won't decrease the size of maximum matching.



Subtree Isomorphism, Unrooted

Algorithm 3 Unooted Subtree Isomorphism

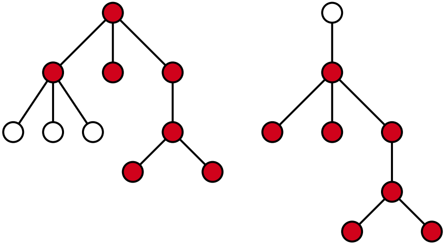
Input: two unrooted trees S and T

Output: decide if S is isomorphic to some g-subtree of T

root T arbitrarily
initialize $M[\cdot,\cdot]$ with 0
for vertex v in post-order traversal of T **do**
  for each vertex u is S **do**
    compute the maximum bipartite matchings of $C(S[w,u]) \cup C(v)$ for each $w \in N(u) \cup \{NIL\}$
  end for
end for
return true if $M[S[NIL,u], root(T)] = 1$ for some $u \in S$

Maximum Common Subtrees, Rooted

Given two rooted trees S, T, find a maximum size g-subtree of S isomorphic to some g-subtree of T.



Consider the rooted case first.

Maximum Common Subtrees, Rooted

Algorithm 4 Rooted Maximum Common Subtrees

Input: two rooted trees S and T

Output: the size of maximum common g-subtrees of S and T

for vertex v in post-order traversal of T **do**
    for each vertex u is S **do**
        build a weighted bipartite graph

$$G = (C(u) \cup C(v), M[C(u), C(v)])$$

        let $M[u, v]$ be the maximum weight matching of G, plus 1
    end for
end for
return $\max_{u \in S, v \in T} M[u, v]$

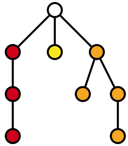
Maximum Common Subtrees, Rooted

Solution (dynamic programming)

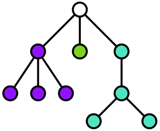
We may again use dynamic programming to solve this problem. Let for $u \in S, v \in T$, define $M[u, v]$ as the size of maximum common subtrees rooted at u and v. The value of $M(u, v)$ can be computed from $M[x, y]$ $(x \in C(u), y \in C(v))$.

We may similarly construct a bipartite graph. The value $M[u, v]$ is computed from maximum weight bipartite matching (instead of maximum cardinality bipartite matching).

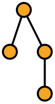
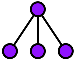
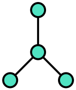
Maximum Common Subtrees, Rooted



T



S

			
	2	1	3
	1	1	1
	3	1	3

Maximum Common Subtrees, Rooted

By using a careful implementation of Kuhn-Munkres algorithm, the maximum weight bipartite matching of $U \cup V$ ($|U| < |V|$) can be solved in $O(|U||V|^2)$ time.

Applying an analysis similar to the subtree isomorphism problem, the total time complexity is $O(n^3)$.

Maximum Common Subtrees, Unrooted

What about unrooted case?

It is easy to obtain an $O(n^4)$ algorithm by trying all possible roots of one tree.

The same technique in subtree isomorphism may apply here to obtain a cubic time algorithm, by not taking maximum weight bipartite matching as black box.

The maximum weight perfect matching can be written as a linear program:

The primal program:

$$
\begin{aligned}
\max_{x_{uv}} \quad & \sum_{u,v} w_{uv} x_{uv} \\
\text{s.t.} \quad & \sum_{u} x_{uv} = 1 \quad \forall v \\
& \sum_{v} x_{uv} = 1 \quad \forall u \\
& x_{uv} \geq 0
\end{aligned}
$$

The dual program:

$$
\begin{aligned}
\min_{y_u, y_v} \quad & \sum_{u} y_u + \sum_{v} y_v \\
\text{s.t.} \quad & y_u + y_v \geq w_{uv} \quad \forall u, v
\end{aligned}
$$

Let $\bar{x}_{uv}$ be optimal solution to primal program, $\bar{y}_u, \bar{y}_v$ be optimal solution to dual program.

Theorem (Strong duality theorem)

$$w_{uv}\bar{x}_{uv} = \sum_u \bar{y}_u + \sum_v \bar{y}_v \quad \forall u, v$$

Theorem (Complementary slackness theorem)

$$\sum_{uv} \bar{x}_{uv}(\bar{y}_u + \bar{y}_v - w_{uv}) = 0$$

This means only saturated edges can be in maximum matching.

Maximum Common Subtrees, Unrooted

About Maximum Weight Perfect Bipartite Matching

$$
\begin{array}{c|ccc} & 0 & 4 & 3 \\ \hline 0 & 0 & 3 & 0 \\ 0 & 0 & 4 & 2 \\ 0 & 0 & 0 & 3 \end{array}
\rightarrow
\begin{array}{c|ccc} & 0 & 4 & 3 \\ \hline 0 & \mathbf{0} & 3 & 0 \\ 0 & \mathbf{0} & \mathbf{4} & 2 \\ 0 & \mathbf{0} & 0 & \mathbf{3} \end{array}
\rightarrow
\begin{array}{c|ccc} & 0 & 4 & 3 \\ \hline 0 & \boxed{\mathbf{0}} & 3 & 0 \\ 0 & \mathbf{0} & \boxed{\mathbf{4}} & 2 \\ 0 & \mathbf{0} & 0 & \boxed{\mathbf{3}} \end{array}
$$

After removing a vertex:

$$
\begin{array}{c|ccc} & 0 & 4 & 3 \\ \hline 0 & 0 & 3 & 0 \\ 0 & 0 & 4 & 2 \\ 0 & 0 & 0 & \underline{0} \end{array}
\rightarrow
\begin{array}{c|ccc} & 0 & 4 & 3 \\ \hline 0 & \mathbf{0} & 3 & 0 \\ 0 & \mathbf{0} & \mathbf{4} & 2 \\ 0 & \mathbf{0} & 0 & 0 \end{array}
\rightarrow
\begin{array}{c|ccc} & 0 & 4 & \underline{2} \\ \hline 0 & \mathbf{0} & 3 & 0 \\ 0 & \mathbf{0} & \mathbf{4} & \mathbf{2} \\ 0 & \mathbf{0} & 0 & 0 \end{array}
\rightarrow
$$

$$
\begin{array}{c|ccc} & 0 & \underline{3} & \underline{1} \\ \hline 0 & \mathbf{0} & \mathbf{3} & 0 \\ \underline{1} & 0 & \mathbf{4} & \mathbf{2} \\ 0 & \mathbf{0} & 0 & 0 \end{array}
\rightarrow
\begin{array}{c|ccc} & 0 & 3 & 1 \\ \hline 0 & \mathbf{0} & \boxed{\mathbf{3}} & 0 \\ 1 & 0 & \mathbf{4} & \boxed{\mathbf{2}} \\ 0 & \boxed{\mathbf{0}} & 0 & 0 \end{array}
$$

Overview

- Three fundamental graph theory problems:

	Trees	General Graphs
Graph Isomorphism	$O(n)$	$O(n^{\text{ploy} \log n})$
Subgraph Isomorphism	P	NP-Complete
Maximum Common Subgraphs	P	NP-Hard

- One problem solving framework: turns the original problem into several instances of matching problem.
- Two auxiliary algorithms:
  - Edmonds-Karp algorithm: solves maximum cardinality bipartite matching in $O(E\sqrt{V})$ time;
  - Kuhn-Munkres algorithm: solves maximum weight bipartite matching in $O(V^3)$ time.

References

- Matula, D. W. (1978). Subtree isomorphism in O(nˆ5/2). Annals of Discrete Mathematics, 2, 91-106.
- Shamir, R. , & Tsur, D. (1999). Faster subtree isomorphism. Journal of Algorithms, 33(2), 267-280.
  (this paper gives a slightly faster subtree isomorphism algorithm, but is of less practical significance)
- Droschinsky, A. , Kriege, N. M. , & Mutzel, P. Faster algorithms for the maximum common subtree isomorphism problem. 41st International Symposium on Mathematical Foundations of Computer Science (MFCS'16).
  (this paper describes a cubic algorithm for unrooted maximum common subtrees in detail)