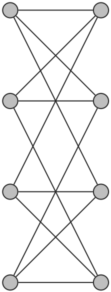
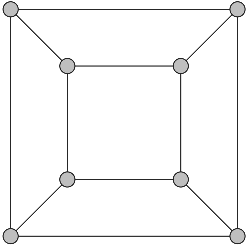
Graph Isomorphism and Related Problems

Graph Isomorphism

Graph Isomorphism

Two graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ are *isomorphic* if there is a bijection $\varphi\colon V_1 \to V_2$ such that, for all $u, v \in V_1$, $uv \in E_1$ if and only if $\varphi(u)\varphi(v) \in E_2$. Isomorphism of two graphs G_1 and G_2 is denoted as $G_1 \simeq G_2$.





Graph Isomorphism – Related Problems

Subgraph Isomorphism

- Determine if G is isomorphic to a subgraph of H.

Largest Common Subgraph

- For two given graphs G_1 and G_2, find the largest graph H such that H is isomorphic to a subgraph of G_1 and to a subgraph of G_2.

Theorem

There is (probably) no polynomial time algorithm to solve the Subgraph Isomorphism problem.

Isomorphism of Trees

Isomorphism of Trees

Centers

- A vertex v is *center* of G, if $\mathrm{ecc}(v) = \mathrm{rad}(G)$.
- A tree has at most two centers which can be found in linear time. In case of two, both are adjacent.

Finding Ceters in Trees

- Pick an arbitrary vertex v.
- Find a vertex x with $d(v, x) = \mathrm{ecc}(v)$.
- Find a vertex y with $d(x, y) = \mathrm{ecc}(x)$.
- Then, $\mathrm{ecc}(x) = \mathrm{ecc}(y) = \mathrm{diam}(T)$ and the vertices in the middle of the path from x to y are centers of T (two vertices if $d(x, y)$ is odd, one vertex if $d(x, y)$ is even).

Note: There is a linear time algorithm to compute the eccentricity of every vertex in a tree.

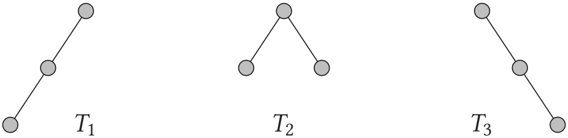
Isomorphism of Trees

Rooted Tree

We say (V, E, r) is a *tree with root* r if (V, E) is a tree and $r \in V$.

Isomorphism of Rooted Trees

Two rooted trees $T_1 = (V_1, E_1, r_1)$ and $T_2 = (V_2, E_2, r_2)$ are *isomorphic* if there is a bijection $\varphi\colon V_1 \longrightarrow V_2$ such that, for all $u, v \in V_1$, $uv \in E_1$ if and only if $\varphi(u)\varphi(v) \in E_2$ and $\varphi(r_1) = \varphi(r_2)$.



T_1 and T_3 are isometric. However, T_2 is not isometric to T_1 and T_3.

Isomorphism of Trees

Theorem

If there is an $\mathcal{O}(f(n))$ time algorithm $\mathcal{A}$ to check if two rooted trees are isomorphic, then there is an $\mathcal{O}(f(n))$ time algorithm $\mathcal{A}^*$ to check if two ordinary trees are isomorphic.

Proof: We design $\mathcal{A}^*(T_1, T_2)$ as follows: Find the centers of T_1 and T_2. Then, there are three cases.

C1 Both have only one center (c_1 and c_2).
$\textsc{Retrun}\ \mathcal{A}(T_1, c_1, T_2, c_2)$

C2 Both have two centers (c_1, c_1', c_2, and c_2').
$\textsc{Retrun}\ \mathcal{A}(T_1, c_1, T_2, c_2) \vee \mathcal{A}(T_1, c_1', T_2, c_2)$

C3 Different number of centers.
$\textsc{Return False}$ □

Therefore, it is enough if we can check rooted trees.

Aho-Hopcroft-Ullman-Algorithm

Let T_1 and T_2 be two rooted trees with corresponding roots r_1 and r_2. With $F_{i,j}$, we denote the forest created by removing all vertices $v \in T_i$ with $d(v, r_i) \leq j$.

Theorem

T_1 is isometric to T_2 if and only if, for all i, $F_{1,i}$ is isometric to $F_{2,i}$.

Algorithm Idea

- Check bottom-up if $F_{1,i}$ is isometric to $F_{2,i}$.
- Note: The lowest layer contains only leaves.
- Use that $F_{1,i+1}$ is isometric to $F_{2,i+1}$ when checking $F_{1,i}$ and $F_{2,i}$.

Aho-Hopcroft-Ullman-Algorithm

Input: Two rooted trees T_1 and T_2 with the corresponding roots r_1 and r_2 and with height h.

- Compute the depth of each vertex in T_1 and T_2.
- To each leaf, assign the integer 0.
- **For** $i = h - 1$ **DownTo** 0
  - $\langle$Assign integers to all nodes with depth i.$\rangle$
- T_1 and T_2 are isomorphic if and only if r_1 and r_2 have the same integer assigned.

Aho-Hopcroft-Ullman-Algorithm

Procedure *AssignIntegers(i)*

- **For Each** $v \in V_i$ (the set of non-leaf vertices with depth i)
  - Create a sorted tuple t_v containing the integers assigned to the children of v.
- Let S_1 and S_2 be the sets of tuples created for vertices in T_1 and T_2 with depth i.
- Sort S_1 and S_2 lexicographically and compare them. If $S_1 \neq S_2$, **Stop**. T_1 and T_2 are not isomorphic.
- Assign integers, continuously and beginning with 1, to vertices in V_i such that two vertices have the same integer assigned if and only if they have equal tuples.