KSI 2014/2015

Úloha 2-3: Umělecká chvilka

Jan Horáček
Gymnázium, Brno, Vídeňská 47; jan.horacek@seznam.cz

13. prosince 2014

1 Popis algoritmu

K řešení tohoto problém jsem využil, jak už hint napovídá, průchodu binárního stromu do hloubky. Problém je řešený pomocí rekurzivního volání funkce `is_christmas_tree`. Zde bych rád podotknul, že zásobníku, který je nutný pro procházení do hloubky, jsem se "zbavil"jen na oko — ve skutečnosti v programu je — je totiž reprezentován zásobníkem volání funkcí.

Samotná funkce `is_christmas_tree` pro každého existujícího následníka vrcholu `node` zkontroluje rozdílnost jeho barvy od barvy své. V případě, že existují oba následníci, zkontroluje také rozdílnost jejich barem navzájem. Pokud narazí na shodu barev, vrátí `False`, jinak zavolá sama sebe pro každého existujícího následníka. Pokud nenastane kolize barev, vrátí `True`.

Implementace zajišťuje, že pokud dojde v "hlubokém zanoření"k vrácení `False`, tato logická hodnota okamžitě probublá až do návratové hodnoty prvotního volání funkce. Viz zdrojový kód.

2 Poznámky

1. Pro přehlednost jsem parametr `tree` přejmenoval na `node`.

2. V programu jsou (na první pohled poměrně zběsile) využity logické operátory v podmínkách, které staví na tom, že Python vyhodnocuje podmínky líně. Což naštěstí dělá.

3. Časová složitost algoritmu je v nejhorším případě (tedy v tom případě, že je strom správně obarvený) $O(n)$, kde n je počet vrcholů. Důvodem je to, že program jednoduše musí projít všechny vrcholy. Při nalezení shody barev je průchod ihned ukončen a časová složitost tím pádem klesá.

   Maximální paměťová složitost programu, tedy maximální velikost dat za zásobníku, je v nejhorším případě — binárním stromu degradovaném na lineární seznam — $O(n)$. Tento případ ale v praxi obvykle nenastává. V případě vyváženého binárního stromu je paměťová složitost $O(log(n))$.