Semestrální práce z předmětu 36TI

Martin Tůma, 3/50

20.12.2004

Obsah

1 Zadání

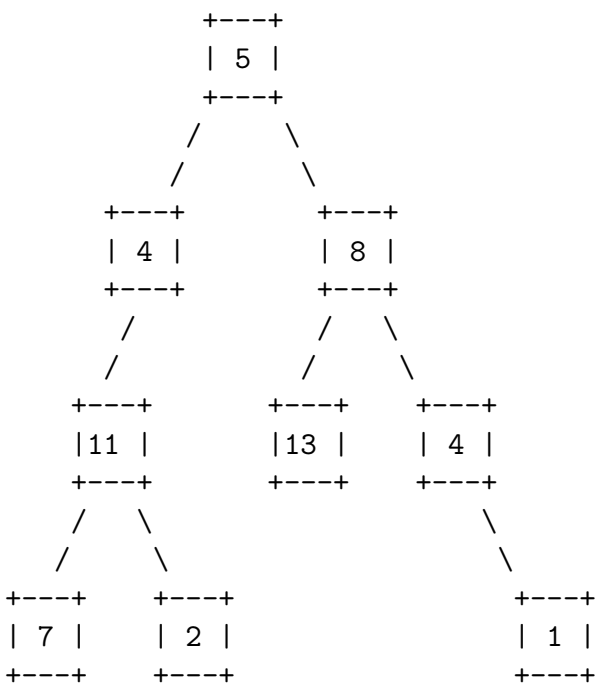
1.1 Pozadí

LISP was one of the earliest high-level programming languages and, with FORTRAN, is one of the oldest languages currently being used. Lists, which are the fundamental data structures in LISP, can easily be adapted to represent other important data structures such as trees.

This problem deals with determining whether binary trees represented as LISP S-expressions possess a certain property.

1.2 Problém

Given a binary tree of integers, you are to write a program that determines whether there exists a root-to-leaf path whose nodes sum to a specified integer. For example, in the tree shown below there are exactly four root-to-leaf paths. The sums of the paths are 27, 22, 26, and 18.



Binary trees are represented in the input file as LISP S-expressions having the following form.

```
empty tree  ::= ()
tree        ::= empty tree | (integer tree tree)
```

The tree diagrammed above is represented by the expression

```
(5 (4 (11 (7 () ()) (2 () ()) ) ()) (8 (13 () ()) (4 () (1 () ()) ) ) )
```

Note that with this formulation all leaves of a tree are of the form

```
(integer () () )
```

Since an empty tree has no root-to-leaf paths, any query as to whether a path exists whose sum is a specified integer in an empty tree must be answered negatively.

1.3 Vstup

The input consists of a sequence of test cases in the form of integer/tree pairs. Each test case consists of an integer followed by one or more spaces followed by a binary tree formatted as an S-expression as described above. All binary tree S-expressions will be valid, but expressions may be spread over several lines and may contain spaces. There will be one or more test cases in an input file, and input is terminated by end-of-file.

1.4 Výstup

There should be one line of output for each test case (integer/tree pair) in the input file. For each pair I,T (I represents the integer, T represents the tree) the output is the string yes if there is a root-to-leaf path in T whose sum is I and no if there is no path in T whose sum is I.

1.5 Příklad

Sample Input

```
22 (5(4(11(7()())(2()()))()) (8(13()())(4()(1()()))))
20 (5(4(11(7()())(2()()))()) (8(13()())(4()(1()()))))
10 (3
     (2 (4 () () )
        (8 () () ) )
     (1 (6 () () )
        (4 () () ) ) )
5 ()
```

Sample Output

```
yes
no
yes
no
```

2 Řešení

2.1 Popis algoritmu

Daný problém zjištění existence cest určité hodnoty binárního stromu je řešen pomocí rekurzivního prohledávání stromu do hloubky – konkrétně preorder variantou.

Implementovaný algoritmus je díky zadanému formátu dat a jejich jednoduššímu zpracování dvouprůchodový. Nejdříve je celý strom načten funkcí *nacti* do jednoho normalizovaného[1] řetězce a teprve poté jsou vyhodnoceny cesty pomocí procedury *projdi*. Toto dvojí zpracování však nemá na asymptotickou složitost algoritmu vliv, viz důkaz v sekci 2.3.

2.2 Implementace

Struktura použitých dat pro implementaci je velice jednoduchá - jedná se prakticky o jeden objekt *Tstrom*, který obsahuje potřebné proměnné pro reprezentaci stromu i obě funkce pro jeho vytvoření a zpracování. Význam proměnných by měl být zřejmý již z jejich názvu.

Normalizovaný řetězec umožňuje simulovat stromovou strukturu v samotném řetězci, pro konstrukci stromu tedy není nutné konstruovat jinou datovou strukturu.

Funkce *nacti* funguje následovně: Nejdříve načte číslo na začátku řádky reprezentující požadovanou sumu cesty a následně v „nekonečném“ cyklu načítá po řádkách další znaky, ze kterých sestavuje normalizovaný řetězec reprezentující strom. V tomto cyklu funkce za každou "(" o jedničku zvýší čítač, za každou ")" jej o 1 sníží. V okamžiku, kdy je hodnota čítače rovna 0, je strom kompletní a cyklus je přerušen.

Na takto vytvořený strom je aplikována procedura *projdi*, která jej rekurzivně projde. V každém uzlu je snížena hodnota sumy o hodnotu aktuálního uzlu. V případě, že je uzel listem a hodnota uzlu se shoduje s hodnotou cesty, byla nalezena cesta požadované hodnoty a algoritmus končí.

Více informací o implementaci lze vyčíst z komentářů ve zdrojovém kódu a z kódu samotného.

2.3 Složitost

Je zřejmé, že načtení celého výrazu do normalizovaného řetězce má asymptotickou složitost $O(n)$ (jako n je uvažován počet uzlů reprezentovaného binárního stromu).

Taktéž rekurzivní průchod stromem do hloubky má asymptotickou složitost $O(n)$. Že složitost nalezení cesty od kořene k listu dané hodnoty nemůže být lepší, vyplývá ze samotné definice stromu. Jelikož *strom je souvislý graf bez kružnic* a list je pouze takový *uzel, který inciduje právě s jednou hranou*, není možné při cestách do všech listů neprojít také všechny zbývající uzly. Že je v nejhorším případě nutné projít všechny cesty k listům, je žřejmé, neboť požadovanou sumu může dávat libovolná z cest.

[1] Řetězec bez mezer obsahující kompletní S-LISP výraz reprezentující binární strom

Obě dvě části algoritmu tedy mají složitost $O(n)$. Z definice maximální asymptotické složitosti pak plyne, že celková složitost algoritmu je $O(n)$, neboť $O(n) + O(n) = O(n)$. Uvedená rovnost také mimo jiné umožňuje použít dva průchody bez újmy na asymptotické složitosti.

K tomuto rozboru je potřeba podotknout, že výše uvedené platí pouze, pokud na zadaná data nahlížíme jako na graf a složitost vyjadřujeme pomocí velikosti množiny uzlů (případně hran, které jsou ale u stromu prakticky stejně mohutné). V případě, že zadaný soubor dat budeme považovat za množinu znaků, výše uvedené neplatí a složitost algoritmu je $O(n_1 + n_2)$, kde n_1 je velikost množiny znaků včetně mezer a n_2 velikost množiny znaků normalizovaného řetězce.

3 Závěr

Výsledný program prošel úspěšně jak testem na přiloženém datovém souboru, tak testem v online vyhodnovacím systému ACM (http://acm.uva.es)[2].

Nutno přiznat, že reálné výsledky nejsou nijak oslňivé – moje řešení se pohybuje u konce „startovního pole“ se skutečnou časovou složitostí přibližně 8x horší než je střední hodnota všech úspěšných řešení v systému. U reálné aplikace by tedy stálo za to, kromě asymptotické složitosti zlepšit i skutečnou časovou složitost tím, že by se výrazy procházely jenom jednou, tzn. bez předchozí úpravy (a to ještě díky možnosti předčasně vyhodnocování ukončit při nalezení hledané sumy ne vždy celé).

Použitá literatura

[1] Kolář, J.: *Teoretická informatika.* Česká informatická společnost 2004

[2] Hudec, B.: *Programovací techniky.* ČVUT 2004

[2]Pro použití v tomto systému je třeba program mírně upravit. Z kódu je třeba odstranit všechny funkce pracující s externímy soubory (assign,fopen) a nahradit ukazatel fp za standartní vstup input