Laborator 8 - Arbori și Arbori Binari

Responsabili

- Cristian Crețeanu [mailto:mailto:cristiancreteanu06@gmail.com]
- Cosmin Petrișor [mailto:mailto:cosmin.ioan.petrisor@gmail.com]

Obiective

În urma parcurgerii articolului, studentul va fi capabil să:

- înţeleagă noţiunea de arbore şi structura unui arbore binar
- construiască, în limbajul C++, un arbore binar
- realizeze o parcurgere a structurii de date prin mai multe moduri
- citească o expresie matematică şi să-i construiască arborele binar asociat
- evalueze o expresie matematică dată printr-un arbore binar.

Noțiuni teoretice

Noțiunea de arbore. Arbori binari

Matematic, un arbore este un graf neorientat conex aciclic.

În ştiinţa calculatoarelor, termenul de **arbore** este folosit pentru a desemna o structură de date care respectă definiţia de mai sus, însă are asociate un nod rădăcină şi o orientare înspre sau opusă rădăcinii.

Arborii sunt folosiţi în general pentru a modela o **ierarhie de elemente**.

Astfel, fiecare element (**nod**) poate deţine un număr de unul sau mai mulţi descendenţi, iar în acest caz nodul este numit **părinte** al nodurilor descendente.

Fiecare nod poate avea un **singur nod părinte**. Un nod fără descendenţi este un **nod terminal**, sau **nod frunză**.

În schimb, există un singur nod fără părinte, iar acesta este întotdeauna **rădăcina arborelui** (**root**).

Un **arbore binar** este un caz special de arbore, în care fiecare nod poate avea maxim **doi descendenţi**:

- nodul stâng
- nodul drept.

În funcţie de elementele ce pot fi reprezentate în noduri şi de restricţiile aplicate arborelui, se pot crea structuri de date cu proprietăţi deosebite: heap-uri, arbori AVL, arbori roşu-negru, arbori Splay şi multe altele. O parte din aceste structuri vor fi studiate la curs şi în laboratoarele viitoare.

În acest articol ne vom concentra asupra unei utilizări comune a arborilor binari, şi anume pentru a reprezenta şi evalua expresii logice.

Reprezentarea arborilor binari

Arborii binari pot fi reprezentați în mai multe moduri. Structura din spatele acestora poate fi un simplu vector, alocat dinamic sau nu, sau o structură ce folosește pointeri, așa cum îi vom reprezenta în acest articol.

BinaryTree.h

```
#ifndef __BINARY_TREE_H__
#define __BINARY_TREE_H__

#include <cstdio>
#include <cstdlib>

template <typename T>
class BinaryTree
{
public:
    BinaryTree();
    ~BinaryTree();

private:
    BinaryTree<T> *leftNode;
    BinaryTree<T> *rightNode;

    T *pData;
};

#endif // __BINARY_TREE_H__
```

Structura nodului de mai sus este clară:

- pointer către fiul stâng
- pointer către fiul drept
- pointer către date

Pentru toți membrii unui nod, trebuie să alocați memorie **dinamic**, dar nu in constructor! Alocați memoria **doar** atunci când aveți nevoie de ea.

De asemenea, dezalocarea memoriei se va face recursiv, dar numai atunci cand este **necesar**.

Pentru a ne reaminti cum alocăm memorie:

```
BinaryTree<T> *node = new BinaryTree<T>();
delete node;

T *pData = new T;
delete pData;
```

Parcurgerea arborilor

Se implementeaza foarte usor recursiv:

Preordine

- Se parcurge **rădăcina**
- Se parcurge subarborele **stâng**
- Se parcurge subarborele **drept**

Exemplu:

```
PreorderTraverse(BinaryTree<T> *node)
{
    Process(node->pData);
    PreorderTraverse(node->leftNode);
    PreorderTraverse(node->rightNode);
}
```

Inordine

- Se parcurge subarborele **stâng**
- Se parcurge **rădăcina**
- Se parcurge subarborele **drept**

Postordine

- Se parcurge subarborele **stâng**
- Se parcurge subarborele **drept**
- Se parcurge **rădăcina**

Lățime

Se folosește o coadă, iar la fiecare pas se extrage din această coadă câte un nod și se adăugă înapoi în coadă nodul stâng, respectiv drept al nodului scos. Acest algoritm continuă până când coada devine goală.

Nodurile frunză nu au descendenţi → nodul stâng şi nodul drept pointează la **NULL** şi nu trebuie adăugate în coadă.

Arbori asociați expresiilor

O expresie matematică este un şir de caractere compus din:

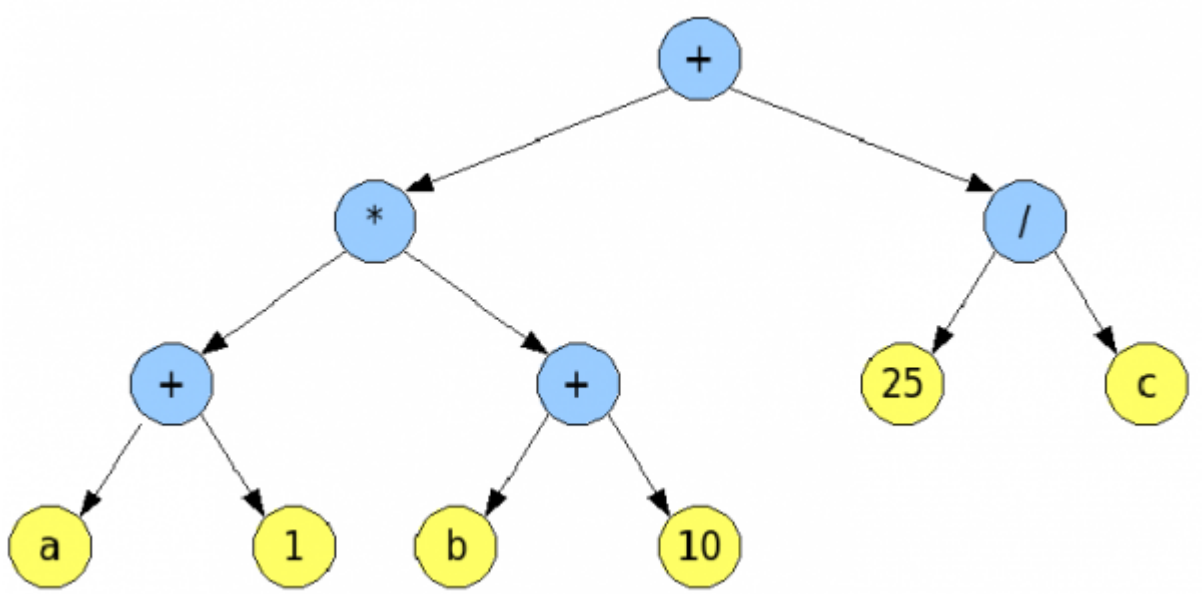
- variabile
- constante
- operatori
- paranteze (eventual).

Fiecărei expresii i se poate asocia un arbore binar, în care:

- nodurile interioare reprezintă **operatorii**
- frunzele reprezintă **constantele** şi/sau **variabilele**.

În terminologia limbajelor formale şi a compilatoarelor, acest arbore se mai numeşte şi **Abstract Syntax Tree (AST)**.

Pentru expresia **(a+1)*(b+10)+25/c** , arborele asociat este prezentat mai jos:



Evaluarea expresiilor

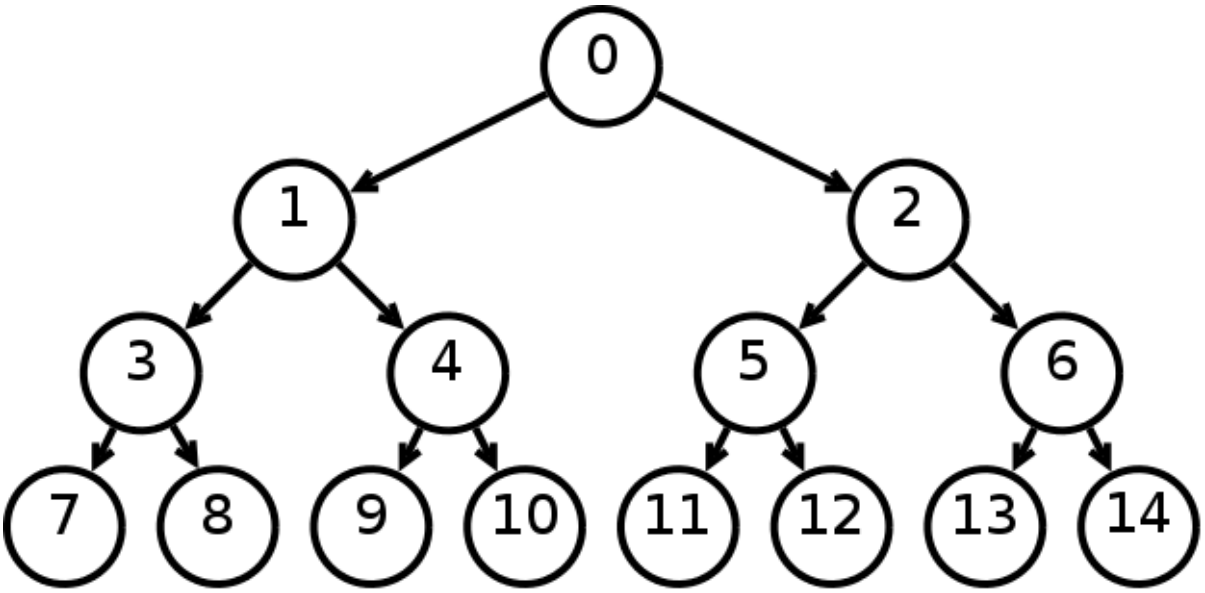
Următorul pseudo-cod reprezintă în linii mari algoritmului de evaluare a expresiilor reprezentate sub formă de arbori binari:

```
Evalueaza(Node nod) {
    // Daca nu este nod terminal...
    if (nod->left || nod->right) {
        // Evaluam expresiile subarborilor...
        res1 = Evalueaza(nod->left);
        res2 = Evalueaza(nod->right);

        // ... si combinam rezultatele aplicand operatorul
        return AplicaOperator(nod->op, res1, res2);
   } else {
        // Daca nodul terminal contine o variabila, atunci intoarcem valoarea variabilei
        if (nod->var) {
            return Valoare(nod->var);
        } else { // Avem o constanta
            return nod->val;
        }
   }
}
```

Cel mai mic strămoș comun

O problemă importantă în analiza arborilor este determinarea celui mai mic strămoș comun (LCA - Lowest Common Ancestor). LCA-ul a două noduri, u si v, este nodul cel mai depărtat de rădăcină care îi are pe u și v ca descendenți.



Spre exemplu, cel mai mic strămoș comun al nodurilor 1 și 12 este 0, în timp ce pentru nodurile 4 și 7, acesta este 1.

Schelet

Schelet

Exerciții

Fiecare laborator va avea unul sau doua exerciții publice si un pool de subiecte ascunse, din care asistentul poate alege cum se formeaza celelalte puncte ale laboratorului.

În cadrul arhivei, aveți la dispoziție un parser pentru expresii logice sub **forma normal disjunctivă - DNF**:

```
(Expresie) := (Termen1) | (Termen2) | (Termen3) | ... | (TermenN), N >= 1
(Termen) := (Literal1) & (Literal2) & ... & (LiteralM), M >= 1
```

Câteva exemple de expresii logice valide: E1 = a & b & !c E2 = a & b | c & !a

În cazul expresiilor logice considerate în forma de mai sus şi ţinând cont de precedenţa convenabilă a operatorilor, arborele expresiilor se generează destul de uşor, de exemplu dupa regulile următoare (folosită în implementarea din laborator - de remarcat ca sunt mai multe posibilităţi de generare a acestui arbore):

Se genereaza un nod pentru prima disjuncţie (|) întâlnită:

- (Termen1) ca subarbore stâng şi restul expresiei ca subarbore drept
- Pentru subarborii dreapta se aplică recursiv regula de la pasul precedent, expandându-se ca subarbori stânga toţi termenii până la (TermenN-1), şi având un singur subarbore dreapta cu (TermenN)

Pentru fiecare subarbore asociat unui termen se generează un nod pentru prima conjuncţie întâlnită (&):

- cu (Literal1) ca subarbore stânga şi restul termenului ca subarbore dreapta, aplicându-se recursiv această regula şi pentru ceilalţi literali, similar cazului disjuncţiilor.

Negarea este reținută direct în nodul literal, deci pentru a trata acest caz trebuie să verificați primul caracter al nodului.

1) [**5p**] Implementați (și **compilați!**) următoarele funcții pentru un arbore binar:

- [**1p**] (BinaryTree.h) Constructor / Destructor (eliberați memorie doar dacă este cazul) (**TODO 1.1**)
- [**2p**] (BinaryTree.h) Implementați metoda de a seta datele reținute de un nod, posibilitatea de a returna și a seta arborii. (**TODO 1.2**)
- [**2p**] (BinaryTree.h) Implementați metodele de inserare recursivă într-un nod din arbore. Presupuneți că se inserează în subarborele stâng sau drept, în mod aleator. (**TODO 1.3**)

2) [**5p**] Implementaţi (şi **compilaţi!**) următoarele funcţii pentru un arbore binar:

- [**1p**] (BinaryTree.h) Realizați o parcurgere `inordine` în displayTree. Puteți să folosiți sau nu variabila de indentare care este dată ca argument. (**TODO 2.1**)
- [**2p**] (BinaryTree.h) Calculaţi înălţimea arborelui în getTreeHeight. (**TODO 2.2**)
- [**2p**] (BinaryTree.h) Găsiți LCA-ul a două noduri. (**TODO 2.3**)

Folosiţi-vă de proprietăţile de bază ale unui arbore (datele pe care le puteţi obţine de la subarborele stâng / drept).

3) [**6p**] Implementați (și **compilați!**) următoarele funcții pentru un arbore binar:

- [**2p**] (BinaryTree.h) Implementați una dintre cele 3 funcții: top/bottom/side view. (**TODO 3.1**)
- [**2p**] (BinaryTree.h) Verificați dacă doi arbori sunt identici/simetrici. (**TODO 3.2**)
- [**2p**] (BinaryTree.h) Afișați un anumit nivel al arborelui. (**TODO 3.3**)

4) [**3p**] (ast.cpp) Terminați de implementat parser-ul, actualizând și populând conținutul nodurilor din arbore.

- [**1p**] Folosiți drept model parseExpression și terminați parseTerm (**TODO 4.1**)
- [**1p**] Folosiți drept model parseExpression și terminați parseLiteral (**TODO 4.2**)
- [**1p**] Verificați că expresia afișată este chiar cea pe care ați dat-o ca parametru (trebuie să faceți parcurgere în inordine și fără indentare) (**TODO 4.3**)

Pentru testarea acestui exercițiu, folosiți o expresie fără variabile, de exemplu: 0 & 1 | 1 & !0 | !1 | 1 & 1 & 1

5) [**4p**] (ast.cpp) Implementați evaluarea unei expresii în evaluateAST()

6) [**2p**] Folosiți un hashtable pentru a ține evidența valorilor variabilelor. Variabilele sunt declarate la început, folosind atribuiri `variabila = valoare`. Pentru fiecare astfel de linie citită, parsați-o și introduceți variabila împreună cu valoarea ei într-un hashtable (puteți folosi clasa unordered_map [http://www.cplusplus.com/reference/unordered_map/unordered_map/] din STL.) Pentru evaluarea expresiei, de fiecare dată când intâlniți o variabilă, vedeți ce valoare îi este atribuită în hashtable și folosiți acea valoare pentru evaluarea expresiei.

Interviu

Această secțiune nu este punctată și încearcă să vă facă o oarecare idee a tipurilor de întrebări pe care le puteți întâlni la un job interview (internship, part-time, full-time, etc.) din materia prezentată în cadrul laboratorului.

- Ce este un arbore?
- Cum poate fi reprezentat un nod dintr-un arbore binar?
- Daţi exemplu de un tip (mai multe tipuri) de parcurgere al arborilor binari. Descrieţi modul de funţionare al acestuia (acestora).
- Daţi exemplu de un mod de utilizare al arborilor binari.

- Ce complexitate medie / worst-case au funcţiile de inserare / ştergere / căutare pentru un arbore binar, BST, AVL, etc. (mai multe despre complexitatea algoritmilor şi structurilor de date veţi învăţa în anul 2: Analiza Algoritmilor şi Proiectarea Algoritmilor).

Bibliografie

1. Binary Tree [http://en.wikipedia.org/wiki/Binary_tree]
2. AVL [http://en.wikipedia.org/wiki/AVL_tree]
3. Red-Black Tree [http://en.wikipedia.org/wiki/Red%E2%80%93black_tree]
4. Splay Tree [http://en.wikipedia.org/wiki/Splay_tree]
5. AST [http://en.wikipedia.org/wiki/Abstract_syntax_tree]
6. DNF [http://en.wikipedia.org/wiki/Disjunctive_normal_form]

sd-ca/2018/laboratoare/lab-08.txt · Last modified: 2019/02/01 13:25 by teodora.serbanescu