Laborator de Structuri de Date si Algoritmi – Lucrarea nr. 8

Arbori de grad oarecare

1. **Reprezentarea standard**
2. **Reprezentarea FIU-FRATE**
3. **Reprezentarea prin noduri de dimensiune variabila**

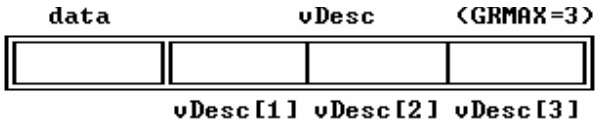
1. Reprezentarea "standard"

In reprezentarea standard in fiecare nod al arborelui, pe linga informatia utila se memoreaza informatii de inlantuire care indica descendentii.

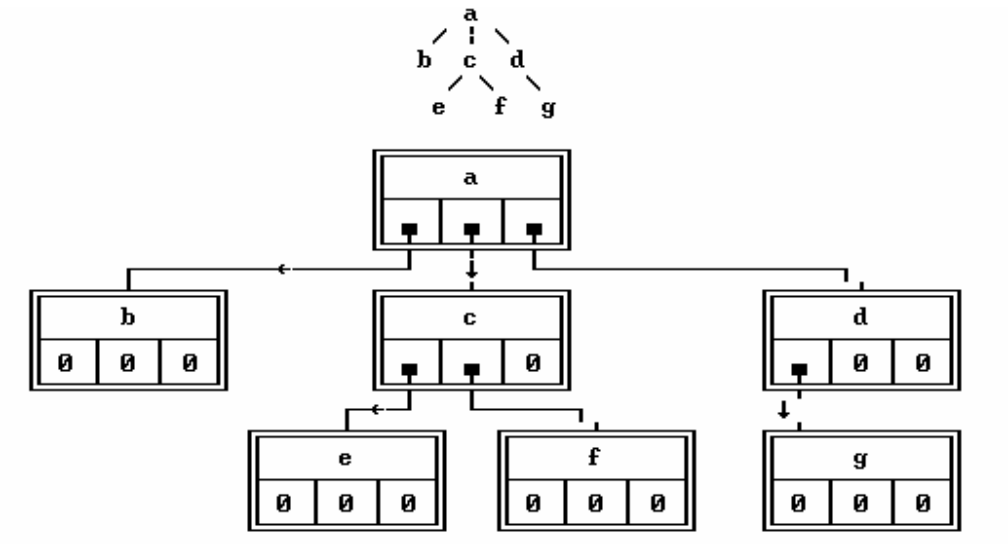
Intr-o prima varianta de reprezentare, fiecare nod este compus din informatia utila si un vector de dimensine fixa, in care se memoreaza legaturilor de tip pointer la descendenti. Dimensiunea acestui vector este data gradul maxim al nodurilor arborelui. Declaratiile de tip folosite de aceasta reprezentare sint:

```
struct Nod{
          Atom data;
          Nod* vDesc[GRMAX];
          };
```

Un nod va ocupa o zona de memorie de dimensiune fixa:



In figura de mai jos am reprezentat inlantuirile pentru arborele:



In aceasta reprezentare, daca **p** este un pointer la un nod, descendentul ***i*** al nodului va fi identificat direct prin pointerul

```
p->vDesc[i]
```

Intr-un arbore cu un numar ***N*** de noduri vor exista ***N-1*** muchii, deci in total ***N-1*** elemente din vectorii de pointeri la descendenti sint ocupate cu informatie utila. Se obtine raportul:

$$\frac{N-1}{N * GRMAX} = \frac{1}{GRMAX}$$

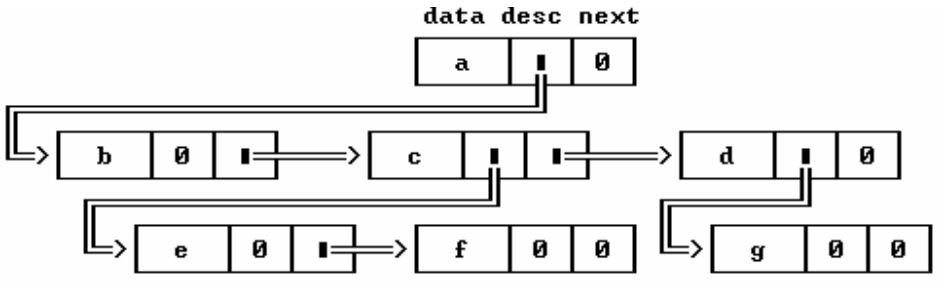
care indica gradul de utilizare eficienta a memoriei. In consecinta aceasta reprezentare este acceptabila numai pentru arbori de grad mic.

2. Reprezentarea "FIU-FRATE"

Pentru a obtine o utilizare mai eficienta a memoriei, pot fi utilizate listele de descendenti. Fiecare nod va contine pe linga informatia utila, doi pointeri: unul va indica lista cu descendentii sai iar cel de-al doilea urmatorul nod din lista de descendenti din care face parte.

```
struct Nod {
       Atom data;
       Nod* desc;
       Nod* next;
       };
```

In aceasta varianta arborele de mai sus va avea urmatoarea reprezentare:



Avind in vedere semnificatia pointerilor continuti intr-un nod aceasta reprezentare se mai numeste "reprezentarea FIU-FRATE".

In aceasta reprezentare, daca ***p*** este un pointer la un nod, identificarea descendentul ***i*** al nodului va necesita parcurgerea listei inlantuite a descendentilor, lista care incepe cu:

```
p->desc
```

3. Reprezentarea prin noduri de dimensiune variabila

O a treia solutie de reprezentare combina economia de memorie cu avantajele accesarii descendentilor pe baza de index. Aceasta solutie se bazeaza pe posibilitatea de a aloca blocuri de memorie de lungime precizata.

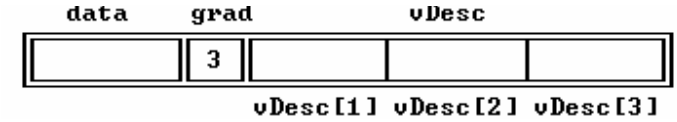
Vom considera aceeasi declaratie pentru tipul ***Nod*** ca si in prima varinata de reprezentare, adugind in plus un cimp in care sa se memoreze gradul nodului.

```
struct Nod{
       Atom data;
       int grad;
       Nod* vDesc[GRMAX];
       };
```

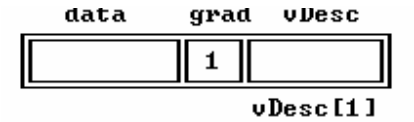
Economia de memorie se va realiza prin alocarea unor zone de memorie de lungime variabila, adaptata gradului fiecarui nod.

Iata cum vor arata:

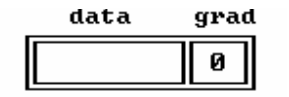
- Un nod de grad 3:



- Un nod de grad 1:



- Un nod terminal:



Pentru a realiza economia de memorie este necesar ca la alocarea spatiului pentru un nod sa se cunoasca numarul de descendenti si in functie de acest numar sa se aloce spatiul necesar pentru vectorul de descendenti. Iata cum trebuie scrisa functia ***make_nod*** care aloca spatiu pentru un nod de grad dat:

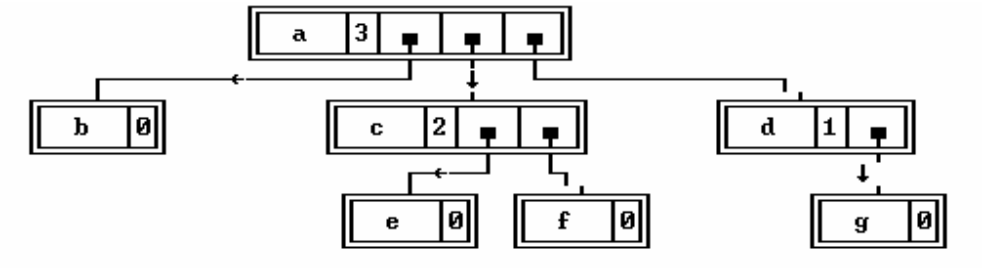
```
Nod* make_nod(int grad)
{
  Nod *p=(Nod*)malloc(sizeof(Nod)-(GRMAX-grad)*sizeof(Nod*));
  p->grad = grad;
  return p;
}
```

Utilizind operatorul ***new***, specific limbajului C++, alocarea va avea forma:

```
Nod* p = (Nod*) new char[sizeof(Nod)-(GRMAX-grad)*sizeof(Nod*)];
```

Iata cum va arata, in aceasta varianta, reprezentarea arborelui dat:

Laborator de Structuri de Date si Algoritmi – Lucrarea nr. 8



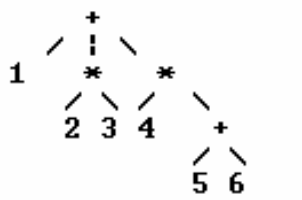
TEMA

1. Functia

```
Nod* creareArbore();
```

din ***Anexa*** citeste de la intrare o expresie aritmetica, cu paranteze, care contine operanzi de o cifra si operatorii + si *, si creaza arborele de grad oarecare asociat expresiei. De exemplu:

1+2*3+4*(5+6) i se asociaza arborele:



Arborele este reprezentat dupa metoda 3, tipul Atom fiind echivalat cu tipul ***int*** (vezi ***Anexa***).

Se cere:
- Sa se determine si sa se afiseze gradul arborelui.
- Sa se afiseze valoarea tuturor operanzilor utilizati in expresie (fara operatori).
- Sa se evalueze expresia si sa se afiseze rezultatul.

2. Se dau doua expresii care contin numai operatorii + si *, operanzi specificati printr-o singura litera si paranteze rotunde. Sa se determine daca cele doua expresii sint identice.

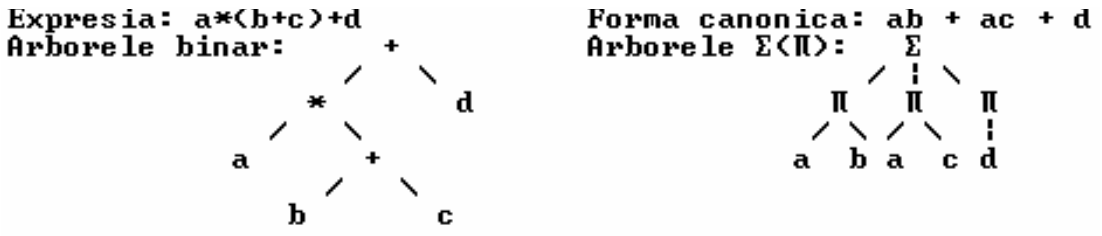
Indicatie: Se aduc cele doua expresii la forma canonica (suma de produse), se sorteaza termenii, si se verifica daca expresiile rezultate sint egale.
Expresia adusa la forma canonica va fi reprezentata sub forma unui arbore cu trei nivele, in care:

Laborator de Structuri de Date si Algoritmi – Lucrarea nr. 8

- radacina este un nod **Σ** - semnificind un nod in care se face insumarea tuturor descendentilor;
- pe nivelul 2 vor fi noduri **π** -semnificind noduri in care se face produsul tuturor dscendentilor;
- pe nivelul 3 vor fi **operanzi**.

Vom numi astfel de arbori ***"arbori Σ(π)"***.

De exemplu:



Pentru a creea arborele **Σ(π)** corespunzator formei canonice vom prelucra arborele initial executind urmatoarele operatii:

[A] - in nodurile terminale: **a** --> **Σ**
|
π
|
a

[B] - in nodurile + : + --> **Σ(π)**
/ \
Σ1(π) Σ2(π)

Nodul **Σ** va avea ca descendenti atat descendentii lui **Σ1** cit si ai lui **Σ2** (listele de descendenti vor fi concatenate).

[C] - nodurile * : * --> **Σ(π)**
/ \
Σ1(π) Σ2(π)

In acest caz arborele **Σ(π)** va contine un numar de descendenti egal cu produsul dintre numarul de descendenti ai nodului **Σ1** si numarul de descendenti ai nodului **Σ2**, obtinuti prin combinarea (inmultirea) fiecare cu fiecare.

Anexa

Fisierul *Arbore.h*

```
#ifndef _ARBORE_H_
#define _ARBORE_H_

#define DIM_EXPR 100
#define GRMAX         20

#define Atom          int

struct Nod
      {
      Atom data;
      int grad;
      Nod* vDesc[GRMAX];
      };

Nod* creareArbore();

#endif
```

Fisierul *Arbore.cpp*

```
#include "arbore.h"
#include <string.h>
#include <ctype.h>
#include <stdlib.h>
#include <conio.h>
#include <stdio.h>
#include <iostream.h>

Nod* parse(char buffer[], int start, int end);

Nod* creareArbore()
{
 char buffer[DIM_EXPR];

 cin >> buffer;
 int length = strlen(buffer);

 Nod* n = parse(buffer, 0, length-1);
 return n;
}
```

Laborator de Structuri de Date si Algoritmi – Lucrarea nr. 8

```
Nod* parse(char buffer[], int start, int end)
{
 int openP = 0;
 int indici[GRMAX];
 int k = 0;

 openP = 0;
 for (int i = start; i <= end; i++)
 {
   if ( buffer[i]=='(' )
     openP ++;
   if ( buffer[i]==')' )
     openP --;
   if ( buffer[i]=='+' )
   {
     if ( openP > 0)
      continue;
     indici[k++] = i;
   }
 }
 if ( k > 0 )
 {
   Nod *p=(Nod*) new char[sizeof(Nod)-(GRMAX-k+1)*sizeof(Nod*)];
   p->grad = k+1;
   p->data='+';
   p->vDesc[0] = parse(buffer, start, indici[0]-1);
   for (int j = 1; j < p->grad - 1; j++)
   {
     p->vDesc[j] = parse(buffer, indici[j-1]+1, indici[j]-1);
   }
   p->vDesc[p->grad-1] = parse(buffer, indici[p->grad-2]+1, end);
   return p;
 }

 openP = 0;
 for (i = start; i <= end; i++)
 {
   if ( buffer[i]=='(' )
     openP ++;
   if ( buffer[i]==')' )
     openP --;
   if ( buffer[i]=='*' )
   {
     if ( openP > 0)
      continue;
     indici[k++] = i;
   }
 }
 if ( k > 0 )
 {
   Nod *p=(Nod*) new char[sizeof(Nod)-(GRMAX-k+1)*sizeof(Nod*)];
   p->grad = k+1;
```

```
    p->data='*';
    p->vDesc[0] = parse(buffer, start, indici[0]-1);
    for (int j = 1; j < p->grad - 1; j++)
    {
      p->vDesc[j] = parse(buffer, indici[j-1]+1, indici[j]-1);
    }
    p->vDesc[p->grad-1] = parse(buffer, indici[p->grad-2]+1, end);
    return p;
  }

  if ( buffer[start] == '(' && buffer[end] == ')' )
    return parse(buffer, start+1, end-1);

  if ( start==end )
    if ( isdigit(buffer[start]) )
      {
      Nod* p = (Nod*) new char[sizeof(Nod)-(GRMAX)*sizeof(Nod*)];
      p->data = buffer[start];
      p->grad = 0;
      return p;
      }

  printf("\nExpresia de intrare este eronata. Apasati o tasta");
  getch();
  exit(1);
}
```