Structuri de date

Laboratorul 1

Mihai Nan
mihai.nan.cti@gmail.com
Grupa 314aCC

Facultatea de Automatică şi Calculatoare
Universitatea Politehnica din Bucureşti
Anul universitar 2017 - 2018

1 Recursivitate

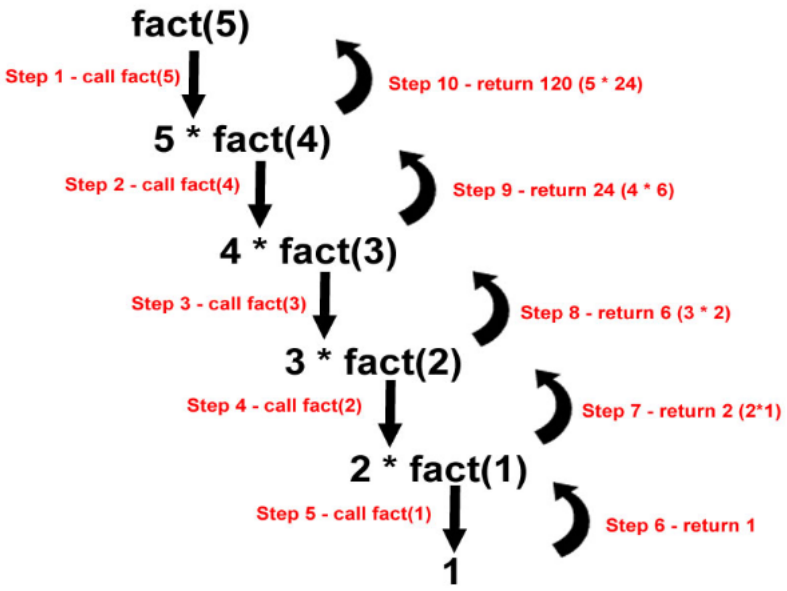
1.1 Introducere

Notiunea de ***recursivitate*** din programare deriva, in mod natural, din notiunea matematica stiuta sub numele de ***recurenta***. In limbajele de programare, ***recursivitatea*** este un concept fundamental care le extinde, in mod esential, puterea de exprimare: recursivitatea permite scrierea unor programare care nu s-ar putea exprima doar cu notiunile fundamentale de ***secventiere*** si ***decizie*** studiate pana acum. Pentru rezolvarea practica a problemelor, recursivitatea este foarte importanta, deoarece permite sa descriem solutia unei probleme complexe folosind una sau mai multe probleme de acelasi tip, dar mai simple. Astfel, aceasta notiune este strans legata de principiul ***descompuerii in subprobleme (divide et impera)***.

Observatie

Definitia unei functii recursive contine doua parti: ***cazul de baza*** si ***relatia recursiva propriu-zisa***. Aceasta sugereaza ca putem folosi operatorul conditional pentru a exprima o definitie recursiva, similar cu functiile, definite anterior, care analizau mai multe cazuri.

1.2 Apelul recursiv



Pentru intelegerea mecanismului apelului recursiv, se recomanda analizarea imaginii de mai sus, in care este prezentata schema apelurilor efectuate pentru calculul factorialului.

1.3 Greseli frecvente in scrierea programelor recursive

O conditie de iesire incorecta din recursivitate va duce, cel mai adesea, la depasirea capacitatii de memorare a stivei, caz in care programul se opreste afisand mesajul: ***Segmentation fault***.

Un autoapel incorect formulat duce la un rezultat incorect sau la umplerea stivei.

In cazul in care se declara parametri formali transmisi prin valoare si/sau variabile locale de tipuri care ocupa mult spatiu de memorie, stiva calculatorului se va umple extrem de repede, ajungandu-se la depasirea spatiului rezervat acesteia chiar si pentru un numar mic de autoapeluri.

1.4 Cand utilizam recursivitatea?

Atunci cand algoritmul care urmeaza sa fie implementat descrie o notiune recurenta sau algoritmul in sine este recursiv, se va lua in considerare oportunitatea descrierii acestuia utilizand tehnica recursivitatii. Se pune insa

problema optimalitatii variantei recursive a algoritmului. Daca acelasi algoritm se poate realiza relativ simplu, utilizand tehnica iterativa (folosind structuri repetitive in locul apelului recursiv), atunci se prefera varianta iterativa datorita faptului ca astfel se vor realiza programe mai rapide. Se evita astfel operatiile mult prea dese de salvarea pe stiva calculatorului, precum si incarcarea acesteia in cazul apelurilor repetate. De asemenea, depanarea programelor recursive este mai anevoioasa decat a celor iterative.

2 Probleme de laborator

Observatie

In arhiva laboratorului, gasiti un schelet de cod pe care il puteti folosi pentru implementarea si testarea problemelor propuse in cadrul acestui laborator.
Punctajul maxim pentru acest laborator este 10 si se va acorda doar daca rezolvarea este insotita si de explicatii.

2.1 Probleme standard

Problema 1 - 2 puncte

Sa se implementeze o functie recursiva care determina elementul maxim dintr-un vector cu **nr** numere naturale.

```
int maxim(int v[], int nr);
```

Problema 2 - 2 puncte

Sa se scrie o functie recursiva care primeste ca parametru un numar natural si returneaza factorialul acelui numar.

```
int factorial(int nr);
```

Problema 3 - 2 puncte

Sa se implementeze o functie care ridica un numar intreg la o putere naturala, folosind cat mai putine inmultiri.

```
Divide et impera
```

Observatie

Pentru calculul puterii naturale a unui numar intreg, se vor uzita formulele:

$$x^n = x^{n/2} * x^{n/2}, \text{n este par} \quad (1)$$

$$x^n = x * x^{(n-1)/2} * x^{(n-1)/2}, \text{n este impar} \quad (2)$$

Problema 4 - 2 puncte

Sa se defineasca o functie recursiva ce va calcula cel mai mare divizor comun a doua numere aplicand algoritmul lui Euclid.

Algorithm 1 Euclid's algorithm

```
1: procedure EUCLID(a, b)
2:     r ← a mod b
3:     while r ≠ 0 do
4:         a ← b
5:         b ← r
6:         r ← a mod b
7:     return b
```

Problema 5 - **2 puncte**

Sa se scrie o functie recursiva care calculeaza valoarea aflata in sirul definit prin relatia de recurenta, de mai jos, pe pozitia ***nr***.

```
X(0) = 0
X(1) = 1
X(nr+1) = 4*X(nr) - 3*X(nr-1) + 1
```

2.2 Probleme bonus

Problema 6 - **2 puncte**

Implementati o functie recursiva care primeste ca argument un sir de caractere si afiseaza toate permutarile posibile ale acestui sir.

```
Backtracking
```

Observatie

Pentru generarea tuturor permutarilor unui sir de caractere se poate folosi urmatoarea formula de recurenta:

$$permutari(S, n) = \bigcup_{k=1}^{n} (s_k + permutari(S - s_k, n - 1)) \tag{3}$$

Problema 7 - **2 puncte**

Scrieti o functie recursiva care primeste ca argument o multime, reprezentata sub forma unui vector de numere intregi, si determina toate submultimile acestei multimi.

Indicatie

Pentru a genera o submultime a unei multimi, trebuie, practic, sa parcurgem elementele multimii, iar, pentru fiecare element, trebuie sa aplicam o alegere binara: *Face sau nu parte din submultimea generata?*. Fiecare dintre aceste N alegeri, unde N reprezinta cardinalul multimii, poate lua una din valorile **DA** (1) sau **NU** (0), iar fiecare configuratie formata din astfel de decizii va genera o submultime. Astfel, putem concluziona ca problema generarii tuturor submultimilor unei multimi este o problema clasica de utilizare a metodei **Backtracking**.

De exemplu, pentru multimea $S = \{1, 2, 3, 4\}$ avem submultimea $S_1 = \{2, 3, 4\}$ care se poate genera folosind secventa de alegeri $0, 1, 1, 1$ sau submultimea $S_2 = \{2, 4\}$ care este generata de $0, 1, 0, 1$.

Astfel, pentru a genera toate submultimile unei multimi, putem uzita urmatorul algoritm:

Algorithm 2 Generare Submulimi

```
1:  procedure SUBSETS(S, index, sequence)
2:      if index > |S| then
3:          PRINT_SUBSET(S, sequence)
4:      else
5:          sequence[index] ← 0
6:          SUBSETS(S, index + 1, sequence)
7:          sequence[index] ← 1
8:          SUBSETS(S, index + 1, sequence)
9:  procedure PRINT_SUBSET(S, sequence)
10:     i ← 1
11:     for each x ∈ S do
12:         if sequence[i] = 1 then
13:             PRINT(S[i])
14:         i ← i + 1
```

3 Interviu

Observatie

Aceasta sectiune este una optionala si incearca sa va familiarizeze cu o serie de intrebari ce pot fi adresate in cadrul unui interviu tehnic. De asemenea, aceasta sectiune poate fi utila si in pregatirea pentru examenul final de la aceasta disciplina.

Cod sursa C

```c
void test1(int n) {
  int i = 0;
  if(n > 1)
    test1(n-1);
  for(i = 0; i < n; i++)
    printf("*");
  printf("\n");
}

int test2(int nr) {
  if(nr == 0 || nr == 1)
    return 7;
  resturn test2(nr - 1) + 2 * test2(nr - 2);
}

int test3(int *v, int nr, int sum) {
  if(nr == 0) {
    sum += v[nr];
    return sum;
  } else {
    return test3(v, nr - 1, sum + v[nr]);
  }
}

int test4(int *v, int nr) {
  if(nr == 0) {
    return v[nr];
  } else {
    return v[nr] + test4(v, nr - 1);
  }
}
```

Intrebari interviu

1. Care este explicatia functionalitatii functiei ***test1*** din blocul de cod de mai jos?
2. Se considera functia recursiva ***test2*** din blocul de cod de mai jos. Cate apeluri recursive vor fi facute pentru $nr = 5$?
3. Se poate determina daca doua numere sunt prime intre ele, folosind o functie recursiva? Propuneti o implementare, daca este posibil, sau oferiti o explicatie pentru care acest lucru nu se poate realiza

printr-o functie recursiva.

4. Se propun spre analiza urmatoarele functii: ***test3*** si ***test4***. Ce puteti spune despre aceste functii, raportandu-va la asemanari si deosebiri?
5. De cate feluri poate fi o functie recursiva? Oferiti cate un exemplu sugestiv pentru fiecare categorie.
6. Definiti conceptul ***Divide et impera*** si enumerati cateva aplicatii practice in care se poate uzita acest principiu.

Feedback

Pentru imbunatatirea constanta a acestui laborator, va rog sa completati formularul de feedback disponibil aici.

De asemenea, va rog sa semnalati orice greseala / neclaritate depistata in laborator pentru a o corecta.

Va multumesc anticipat!