Structuri de date

Laboratorul 2

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

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

1 Tipuri de recursivitate

1.1 Introducere

In timpul rularii, un program utilizeaza o zona de memorie, denumita stiva, cu informatiile de care are nevoie, in diferite momente. Dupa cum am vazut inca din laboratorul trecut, la fiecare apel de functie, stiva creste. La fiecare revenire dintr-un apel, cu o anumita valoare, stiva se reduce. Cand recursivitatea este foarte adanca - exista multe apeluri de functie realizate, din care nu s-a revenit inca - stiva devine foarte mare.

Observatie

Acest lucru influenteaza:

- ***memoria*** - in unele situatii, trebuie retinute foarte multe informatii.
- ***timpul*** - in locul redimensionarii stivei, ar putea fi utilizat pentru alte calcule, programul rezultat fiind, astfel, mai rapid.

1.2 Recursivitatea pe stiva

O functie este recursiva pe stiva daca apelul recursiv este parte a unei expresii mai complexe, fiind necesara retinerea de informatii, pe stiva, pe avansul in recursivitate.

Cod sursa C

```c
int factorial(int nr) {
    if(nr == 0)
        return 1;
    else
        nr * factorial(nr - 1);
}
```

1.3 Recursivitatea pe coada

O functie este recursiva pe coada daca valoarea intoarsa de apelul recursiv constituie valoarea de retur a apelului curent, nefiind necesara retinerea de informatie pe stiva.

In general, pentru a trece din recursivitatea pe stiva in cea pe coada, se uziteaza o variabila de acumulare.

⚠ IMPORTANT !

⚠ Aparitia unei variabile de acumulare, in cazul unei functii recursive, nu reprezinta un indicator al recursivitatii pe coada, ci doar forma procesului generat de apelurile recursive.

Cod sursa C

```c
int tail_recursion(int nr, int acumulator) {
  if(nr == 0)
    return acumulator;
  else
    return tail_recursion(nr - 1, nr * acumulator);
}

int factorial(int nr) {
  return tail_recursion(nr, 1);
}
```

1.4 Recursivitatea arborescenta

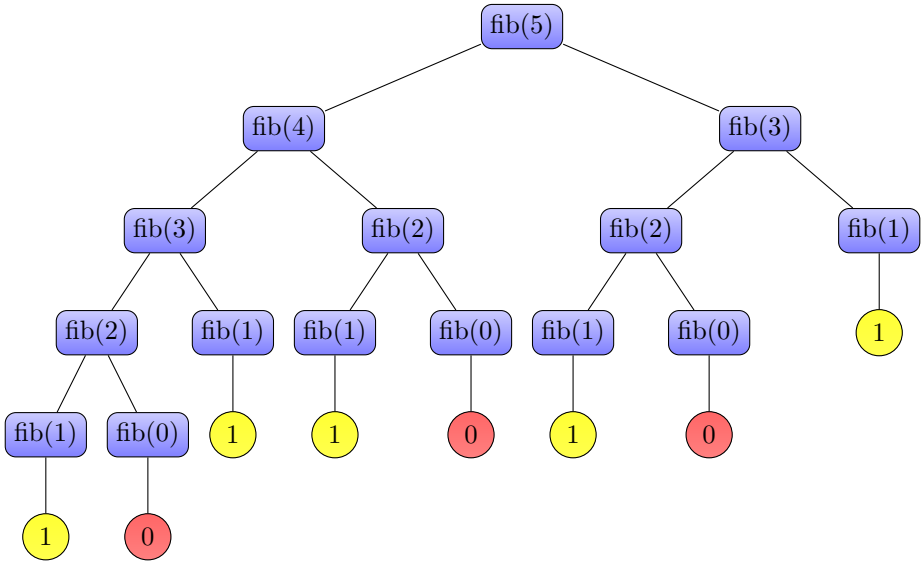
Recursivitatea arborescenta apare in cazul functiilor care contin, in implementare, cel putin doua apeluri recursive care se executa neconditionat.

In laboratorul precedent, ati avut de implementat o functie pentru acest tip de recursivitate.

Cod sursa C

```c
int fibonacci(int nr) {
  if(nr == 0) {
    return 0;
  } else if(nr == 1) {
    return 1;
  } else {
    return fibonacci(nr - 1) + fibonacci(nr - 2);
  }
}
```

Pentru o intelegere mai buna a acestui tip de recursivitate, se recomanda analizarea desenului de mai jos care evidentiaza forma procesului generat de apelurile recursive pentru functia definita mai sus.



2 Divide et Impera

2.1 Introducere

Divide et impera se bazeaza pe principiul descompunerii problemei in doua sau mai multe subprobleme (mai usoare), care se rezolva, iar solutia pentru problema initiala se obtine combinand solutiile subproblemelor.

De multe ori, subproblemele sunt de acelasi tip si pentru fiecare din ele se poate aplica aceeasi tactica a descompunerii in (alte) subprobleme, pana cand (in urma descompunerilor repetate) se ajunge la probleme care admit rezolvare imediata.

2.2 Cautarea binara

Problema: Se citeste un vector cu ***nr*** elemente **sortate crescator** si o valoare ***x***. Sa se verifice daca valoarea ***x*** se afla in vectorul citit sau nu.

Solutie: Algoritmul functioneaza pe baza tehnicii ***divide et impera***. Valoarea cautata este comparata cu cea a elementului din mijlocul vectorului. Daca este egala cu cea a acelui element, algoritmul se termina. Daca este mai mare decat acea valoare, algoritmul se reia, de la mijlocul vectorului pana la sfarsit, iar daca

e mai mica, algoritmul se reia pentru elementele de la inceputul vectorului pana la mijloc.

2.3 Turnurile din Hanoi

Problema: Se dau 3 tije simbolizate prin ***a***, ***b***, ***c***. Pe tija ***a*** se gasesc discuri de diametre diferite, asezate in ordine descrescatoare a diametrelor privite de jos in sus. Se cere sa se mute de pe tija ***a*** pe ***c***, uzitand ca tija intermediara tija ***b***, respectand urmatoarele reguli:

- la fiecare pas se muta un singur disc;
- nu este permis sa se aseze un disc cu diametrul mai mare peste un disc cu diametrul mai mic.

Solutie: Notam cu ***ab*** o mutare valida a unui disc de pe tija ***a*** pe tija ***b*** si cu ***n*** numarul de discuri.

1. Daca $n = 1$ se face mutarea ***ac***.
2. Daca $n = 2$ se fac mutarile ***ab***, ***ac***, ***bc***.
3. Daca $n > 2$ problema se complica. Notam cu ***H(n, a, b, c)*** sirul mutarilor celor ***n*** discuri de pe tija ***a*** pe tija ***c***, utilizand ca tija intermediara tija ***b***. Conform strategiei ***Divide et Impera***, incercam sa descompunem problema in alte doua subprobleme de acelasi tip, urmand apoi combinarea solutiilor. In acest sens, observam ca mutarea celor n discuri de pe tija a pe tija c este echivalenta cu:
   - mutarea a ***n-1*** discuri de pe tija ***a*** pe tija ***b***, utilizand ca tija intermediara tija ***c***;
   - mutarea discului ramas pe tija ***c***;
   - mutarea a ***n-1*** discuri de pe tija ***b*** pe tija ***c***, utilizand ca tija intermediara tija ***a***.

References

[1] Clara Ionescu, Adina Balan. *Informatica pentru grupele de performanta.* (Clasa a X-a) , *Editura Dacia Educational*, Cluj-Napoca, 2004.

[2] Laborator - Paradigme de programare, `http://elf.cs.pub.ro/pp/`

3 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.

3.1 Probleme standard

Problema 1 - 1 puncte

Implementati o functie **recursiva pe coada** care calculeaza suma cifrelor unui număr natural.

```
Uzitarea unui acumulator
Calcul realizat pe avansul in recursivitate
```

Problema 2 - 2 puncte

Scrieti o functie recursiva care cauta un anumit numar intr-un sir de numere intregi, ordonat crescator, si returneaza **pozitia primei aparitii** a numarului in sirul crescator, asigurand o implementare optima.

```
Cautare binara
```

Problema 3 - 2 puncte

Scrieti o functie recursiva care sa determine elementul minim si elementul maxim dintr-un vector cu elemente intregi, uzitand tehnica Divide et Impera.

```
Pair min_max(int *v, int start, int end);
```

Algorithm 1 Min Max - Divide et Impera

```
1:  procedure MinMax(A)
2:      if |A| = 1 then
3:          min = max = A[0]
4:      else
5:          {A_1, A_2} = A
6:          (min_1, max_1) ← MinMax(A_1)
7:          (min_2, max_2) ← MinMax(A_2)
8:          if min_1 ≤ min_2 then
9:              min ← min_1
10:         else
11:             min ← min_2
12:         if max_1 ≥ max_2 then
13:             max ← max_1
14:         else
15:             max ← max_2
16:     return (min, max)
```

Problema 4 - 3 puncte

Sa se implementeze functia recursiva ***quick_sort*** care sorteaza valorile vectorului **v** aflate intre pozitiile **first** si **last**.

Sortarea se va face uzitand algoritmul ***Quick Sort***, descris in pseudocodul de mai jos.

Exemplu

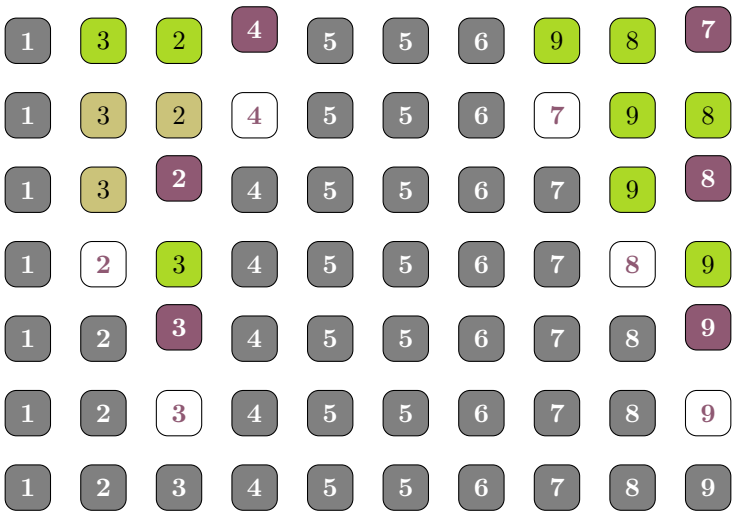
Pas 1: Alegere pivot

Pas 2: Valorile mai mici in stanga, cele mai mari sau egale in dreapta

Pas 3: Repeta pasul 1 pentru cele doua subintervale

Pas 4: Repeta pasul 2 pentru cele doua subintervale

Pas 5: Tot asa pana la final



Algorithm 2 QuickSort

```
1: procedure QUICK_SORT(V, start, end)
2:     if start < end then
3:         pivot ← partition(V, start, end)
4:         quick_sort(V, start, pivot − 1)
5:         quick_sort(V, pivot + 1, end)
```

Algorithm 3 Partition

```
1: procedure PARTITION(V, start, end)
2:     pivot ← start
3:     index ← start
4:     for i ← start + 1 to end do
5:         if V[i] < V[pivot] then
6:             index ← index + 1
7:             swap(V[i], V[index])
8:     swap(V[start], V[index])
9:     return index
```

Problema 5 - 2 puncte

Asupra unui vector cu n elemente se executa succesiv operatii de taiere astfel: vectorul v se injumatateste, iar daca lungimea lui este un numar impar, atunci se elimina elementul din mijloc. Acest proces se repeta asupra fiecarei jumatati pana la obtinerea unui vector cu un singur element. Sa se calculeze suma tuturor elementelor ce raman in vector dupa aplicarea acestui proces.

Exemplu: [1, 2, 3, 4, 5, 6, 7] => [1, 3, 5, 7] => 16

3.2 Probleme bonus

Problema 6 - 4 puncte

Intr-un depozit compartimentat conform unui caroiaj a izbucnit focul in **m** locuri diferite. Magazionerul, aflat in depozit in momentul izbucnirii incendiului, ar vrea sa iasa din cladire. Stiind ca iesirea se afla in coltul in dreapta-jos a depozitului, sa se determine toate drumurile posibile spre iesire.

In anumite compartimente, corespunzatoare unor patratele din caroiaj, se afla obiecte peste care magazionerul nu va putea trece sau sari. De asemenea, pe drumul sau spre iesire, magazionerul nu poate trece prin compartimentele vecine orizontal, vertical sau pe diagonala celor cuprinse de incendiu. Magazionerul va putea sa traverseze depozitul trecand printr-o succesiune de compartimente vecine orizontal sau vertical.

Depozitul este descris printr-o matrice cu **n** linii si **n** coloane, un element al matricei putand lua urmatoarele valori: **0** - spatiu liber, **1** - obstacol, **2** - foc, **3** - pozitia magazionerului la izbucnirea incendiului.

Implementati o functie recursiva care sa determine solutia problemei. Aceasta se va afisa sub forma unui tablou bidimensional avand **n** linii si **n** coloane. Acest tablou va contine traseul magazionerului. Pentru fiecare valoare **k** (cuprinsa intre **1** si **lungimea traseului**) in tablou va exista un singur element avand valoarea **k**, corespunzator pasului **k** al magazionerului, in rest, tabloul va contine elemente egale cu 0.

`Backtracking`

Exemplu

M			F	
	O			
			O	
F				I

Figure 1: Careul initial

M		O	F	O
	O	O	O	O
O	O		O	
F	O			I

Figure 2: Careul prelucrat

Problema 7 - 2 puncte

Statistici de ordine: se da un vector de numere intregi neordonate. Scriind o functie de partitionare, folositi **Divide et Impera** pentru a determina a **k-lea element** ca marime din vector.

Exemplu: Pentru vectorul {0, 1, 2, 4, 5, 7, 6, 8, 9}, al 3-lea element ca ordine este 2.

Problema 8 - 2 puncte

Implementati o functie recursiva care sa testeze daca o expresie este parantezata corect.

Exemplu:

Parantezare corecta: *(())()()*

Parantezare gresita: *(()(()))(()()((((((((((()))))))))))*

4 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.

4.1 Probleme interviu

Problema 1: Se da un sir de **n** numere intregi. Sa se scrie o functie recursiva care plaseaza intre aceste numere **n - 1** operatori aritmetici (+, **-**, ***, /), astfel incat valoarea expresiei obtinute sa fie egala cu un numar intreg dat m.

Problema 2: Se considera un sir de litere, fiecare reprezentand numele unei functii sau numele unui argument al acesteia. Cunoscand aritatea (numarul argumentelor) fiecarei functii, sa se implementeze un program care scrie corect, cu paranteze si virgule, functia si argumentele ei.

Problema 3: Sa se realizeze, in mod recursiv, o functie care calculeaza suma elementelor de deasupra diagonalei principale a unei matrice patratice de dimensiune n cu elemente numere intregi.

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!