Algoritmi recursivi - continuare. Tablouri.

1. **Recursivitate. Continuare.**
2. **Tablouri**

1. Recursivitate. Continuare.

Aşa cum a fost prezentat în primul laborator, algoritmii recursivi sunt ***algoritmi care se autoapelează***.

Executia unei functii care implementeaza un algoritm recursiv se realizează astfel:

- se creeaza in stiva de recursie o "inregistrare de activare" in care sunt memorati:
  - parametrii de apel;
  - adresa instructiunii de retur (cu care va continua programul dupa terminarea executiei functiei);
- se rezerva spatiu pentru variabile locale.
- se executa instructiunile functiei care folosesc pentru parametri si variabile locale parametrii memorati in "inregistrarea de activare";
- se scoate din stiva "înregistrarea de activare" (decrementarea vârfului stivei), stiva fiind ordonată; se continuă cu instrucțiunea dată de adresa de retur memorată în "inregistrarea de activare".

Asadar, *variabilele globale (statice)* sunt memorate intr-o zona de memorie fixă, mai exact în segmentele de date. *Variabilele automate (locale)* se memorează in stivă, iar *variabilele dinamice* in "heap"-uri (cu malloc in C şi cu new in C++).

Consumul de memorie al unui algoritm recursiv este proporțional cu numărul de apeluri recursive ce se fac. Variabilele recursive consumă mai multa memorie decât cele iterative. La prelucrarea unei liste, dacă primul element nu este vid, se prelucrează acesta, urmând apoi ca restul listei sa fie considerat ca o noua lista mai mica, etc.

Aplicații

Şirul lui Fibonacci

Folosind o metodă recursivă, să se determine primii *n* termeni ai şirului lui Fibonacci, F(n) pe baza relatiei de recurentã: F(n) = F(n-2)+F(n-1) si primele 2 numere din şir F(0)=F(1)=1.

```
int Fibonacci(int n)
{
        if(n<2)
        return n;
        else
        return Fibonacci(n-1)+Fibonacci(n-2);
}
```

Laborator de Structuri de Date si Algoritmi – Lucrarea nr. 2

Rezolvați problema şi nerecursiv.

Numere prime

Folosind o metodă recursivă, să se stabilească dacă un număr întreg citit de la tastatură este număr prim sau nu.

```
bool isPrime(int p, int i) // p este variabila citită de la tastatură
{
        if(i= =p) return 1;
        if(p%i= =0) return 0;
        return isPrime(p,i+1);
}
```

Care este semnificația argumentului "i" şi ce valoare va avea la apelul funcției din main?

Permutări

O categorie de probleme cu soluții recursive sunt cele care conțin un apel recursiv într-un ciclu, deci un număr variabil (de obicei mare) de apeluri recursive. Aici se încadrează algoritmii de tip "backtracking".
Să se scrie o funcție care generează şi afişează toate permutările posibile ale primelor n numere naturale.

Problema turnurilor din Hanoi

Se dau ***n*** discuri: a_1, a_2, ... , a_n de dimensiuni diferite, cu $d_1 < d_2 < ... < d_n$, d_i - fiind diametrul discului a_i. Discurile respective sunt stivuite pe o tija.
Se cere sa se deplaseze aceasta stiva pe o alta tija, folosind ca manevra o tija auxiliara, respectandu-se conditia: Un disc nu poate fi plasat decat peste un disc cu diametrul mai mare decat al acestuia.
Problema P(n) a deplasarii a ***n*** discuri, se rezolva prin deplasari succesive ale discurilor de pe o tija pe alta. Deplasarea de pe o tija pe alta este echivalenta cu deplasarea a ***n-1*** discuri de pe tija intiala (t_i) pe tija de manevra (t_m), apoi plasarea celui mai mare disc pe tija finala (t_f), pentru ca la sfarsit sa se aduca de pe tija de manevra pe tija finala cele ***n-1*** discuri.

PseudoCod:

```
Hanoi(n, ti, tf, tm)
{
     if(n=1) then muta (ti, tf) //deplaseaza discul superior
                                //de pe ti pe tf
     else  Hanoi(n-1, ti, tm, tf)
             muta(ti, tf)
             Hanoi(n-1, tm, tf, ti)
}
```

Pentru o problema P(1), timpul T(1) = 1 , pentru o mutare.

Pentru P(n) , timpul:

$$T(n)=2*T(n-1)+1 \quad (1)$$

Dorim sa aflam ordinul de complexitate a lui T(n). Asociem relației (1) ecuația caracteristică:

$$x=2x+1 \quad (2)$$

Rezulta $x_0=-1$ şi $T(n)- x_0=2*T(n-1)- x_0$

Dacă se noteaza $f(n)=T(n)- x_0$ se obține

$$f(n)=2*f(n-1= \quad (3)$$

Aplicând (2) rezultă $f(n)=2*f(n-1)=2*2*f(n-2)=...=2^{n-1} * f(1)$.
Inlocuind f(n) cu T(n)+1 şi f(1) cu 2 se obține $T(n)=2^n -1$.

Prin urmare, ordinul de complexitate este ***O(2^n)***, adică o complexitate exponențială.

Soluția nerecursivă se bazează pe analiza stărilor (mutărilor) discurilor: se repetă de $(2^n–1)$ ori funcția "muta" recalculând la fiecare pas numărul tijei sursă şi al tijei destinație (numărul discului nu contează deoarece nu se poate muta decât discul din vârful stivei de pe tija sursă):

```
int n=4, i; // n = numar de discuri
for (i=1; i < (1 << n); i++) // numar de mutari= 1<<n = 2n
printf("Muta de pe %d pe %d \n", (i&i-1)%3,((i|i-1)+1)%3);
```

Problema labirintului

Se dă o matrice cu elemente 0 şi 1 reprezentând "harta" unui labirint (0-spatiu liber; .....1-zid). Se cere să se determine un drum între o poziție inițială şi o poziție finală date. Un drum este format din poziții libere învecinate pe verticală sau orizontală (pe aceeaşi linie sau aceeaşi coloană).

```
  123456789
1 ±±±±±±±±±    ± - zid
2 ±       ± ±
3 ±±±I±±± ±    I - pozitia initiala
4 ±       ±F±  F- pozitia finala
5 ±±± ±    ±
6 ±±±±±±±±±
```

Soluția acestei probleme este un vector de poziții în matrice. În cazul nostru soluția este:

((3,4), (4,4), (4,5), (4,6), (5,6), (5,7), (5,8), (4,8))

Fiecare poziție din drum este rezultatul unei alegeri între mai multe variante de inaintare. De exemplu din pozitia inițială (3,4) se alege între variantele (2,4) şi (4,4).
Metoda de rezolvare se bazează pe încercări. Dintr-o anumită poziție se inaintează atât cât este posibil, pe rând în toate direcțiile libere. Atunci când o pistă se "inchide" fără să se ajungă în poziția finală, se revine urmărind drumul parcurs pentru a încerca alte piste.

Laborator de Structuri de Date si Algoritmi – Lucrarea nr. 2

Matricea labirint va fi o matrice de caractere:

L[i,j]='±' =>zid
L[i,j]=' ' =>pozitie libera.

Pentru a putea indexa variantele de înaintare dintr-o poziţie se foloseste vectorul "***dir***" cu incrementii care trebuie aplicaţi indecşilor de linie şi coloană pentru înaintare într-o direcţie.

Direcţia:	**1**	**2**	**3**	**4**
Increment pt. *x*	0	0	-1	1
Increment pt. *y*	-1	1	0	0

Se poate aplica următoarea strategie recursivă: căutarea drumului pâna la poziţia finală, pornind dintr-o poziţie dată, va însemna căutarea drumului pornind pe rând din toate poziţiile învecinate din care acestă căutare nu a fost făcută anterior.
Să se scrie o variantă de rezolvare a problemei labirintului folosind o funcţie recursivă. Pentru iniţializarea matricii L (labirintul) veţi folosi o funcţie citeste_labirint care citeşte matricea labirint dintr-un fiţier text, al cărui nume îl primeşte drept parametru:

Conţinutul fişierului:

```
6 9
±±±±±±±±±
±       ± ±
±±±I±±± ±
±       ±F±
±±± ±    ±
±±±±±±±±±
```

Problema Jeep-urilor

Un jeep are un rezervor care poate contine ***r*** litri de benzina, consuma ***c*** litri de benzina la 100 km. Initial jeep-ul este parcat intr-o oaza din Sahara si are rezervorul gol. In oaza exista ***n*** canistre, fiecare continind ***r*** litri de benzina. Presupunem ca jeep-ul poate transporta la un moment dat o singura canistra plus benzina din rezervor. Rezervorul poate fi umplut numai atunci cand este golit complet.

Sa se determine cea mai mare distanta fata de oaza (pozitia initiala) care poate fi parcursa de jeep utilizand toate cele ***n*** canistre.

2. Tablouri

Tabloul este o colecție omogenă de date în care fiecare element poate fi identificat pe baza unui index, colecția asigurând ***timp de acces constant pentru fiecare element***. Prin reprezentarea tabloului se înțelege plasarea elementelor în locații succesive de memorie. Locațiile de memorie pot fi numerotate, numărul locației reprezentând indexul elementului.

Probleme

1. Să se scrie un program care să afişeze elemntele unei matrice pătratice de diemnsiune impară, în ordinea parcurgerii acesteia în spirală, începând cu colțul stânga-sus.

Exemplu:

Pentru matricea

$$A = \begin{matrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{matrix}$$

ordinea parcurgerii este a_{11} a_{12} a_{13} a_{23} a_{33} a_{32} a_{31} a_{21} a_{22}.

2. Să se scrie un program care roteşte cu 90^o, în sens trigonometric, o matrice pătratică de dimensiune impară. Drept centru de rotație se va considera elementul din mijloc. **NU se utilizează matrici suplimentare**.

Exemplu:

Daca

$$A = \begin{matrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{matrix}$$

atunci A rotită cu 90 grade este

$$A = \begin{matrix} a_{13} & a_{23} & a_{33} \\ a_{12} & a_{22} & a_{32} \\ a_{11} & a_{21} & a_{31} \end{matrix}$$

Laborator de Structuri de Date si Algoritmi – Lucrarea nr. 2

3. O secțiune a unui tablou ***A[1..n]*** este formată din elementele ***A[i]***, ***A[i+1]***,..., ***A[j]***, *i<=j*. Suma unei secțiuni este suma elementelor sale. Să se scrie un program care calculează secțiunea de sumă maximă.

4. Se consideră un tablou bidimensional ***A*** cu elemente de tip întreg şi cu proprietatea că elementele fiecărei linii şi fiecărei coloane sunt ordonate crescător. Se ştie că elementul ***x*** apare printre elementele tabloului. Să se scrie un program C care să determine i_0,j_0 astfel încât $A[i_0,j_0] = x$. Dacă există mai multe asfel de poziții pe care apare ***x,*** atunci programul va determina una dintre ele (nu are importanță care); se impune însă ca numărul de operații să fie minim.

TEME

Să se implementeze soluțiile tuturor problemelor prezentate în laborator.