RECURSIVITATE DIVIDE ET IMPERA BACKTRACKING

-Admitere UBB 2022-

Somesan Paul-Ioan

RECURSIVITATE

- Recursivitatea este proprietatea unor notiuni de a se defini prin ele insele. (Pbinfo)
- Exemple de recursivitati cunoscute de la matematica:
    1. Sirul lui Fibonacci
    2. Progresiile matematice (aritmetice si geometrice)
    3. Calculul factorialelor
- La informatica, recursivitatea este implementata pe functii si vom crea in asa fel functiile incat sa functioneze asemenea unei structuri repetitive (vom avea o conditie de oprire si vom calcula rezultate intermediare care ne vor ajuta sa obtinem rezultatul final)

EXEMPLE DE FUNCTII RECURSIVE I

```
6   int factorial(int n){
7       if(n <= 1)
8           return 1;
9       else return n * factorial(n - 1);
10  }
```

```
6   int fibonacci(int n){
7       if(n == 1 || n == 2)
8           return 1;
9       return fibonacci(n-1) + fibonacci(n-2);
10  }
```

```
6   int nr_div(int n, int d){
7       if(n % d == 0 && d * d < n)
8           return 2 + nr_div(n, d+1);
9       if(d * d == n)
10          return 1;
11      if(d < n)
12          return nr_div(n, d + 1);
13      return 0;
14  }
```

```
6   int sum_div(int n, int d){
7       if(n % d == 0 && d * d < n)
8           return n / d + d + sum_div(n, d+1);
9       if(d * d == n)
10          return d;
11      if(d < n)
12          return sum_div(n, d + 1);
13      return 0;
14  }
```

EXEMPLE DE FUNCTII RECURSIVE II

```
6   int sum_cif(int n){
7       if(n == 0)
8           return 0;
9       return n % 10 + sum_cif(n / 10);
10  }
```

```
6   void baza2(int n){
7       if(n == 0)
8           return ;
9       baza2(n / 2);
10      cout << n % 2;
11  }
```

```
6   int sum_vector(int a[], int st, int dr){
7       if(st > dr)
8           return 0;
9       return a[st] + sum_vector(a, st+1, dr);
10  }
```

DIVIDE ET IMPERA

- Divide et Impera este un concept filozofic antic care are in vedere un principiu destul de simplu: divide si stapaneste.
- In informatica, acest lucru va fi folosit sub o forma usor diferita. Acest algoritm se foloseste cel mai mult pentru a determina proprietati ale vectorilor si ale matricilor.
- Ideea este destul de simpla, pentru a stii proprietati ale vectorilor, vom cauta secvential pe acesta si ii vom determina starea.
  - Spre exemplu: Daca ne intereseaza sa stim daca un vector are toate elementele pare, este sufficient pentru a verifica daca ambele jumatati ale acestuia au toate elemtele pare. Divizarea sirului continua pana secventa are un singur element

EXEMPLE DE DIVIDE ET IMPERA I

```
6   int sum_vector(int a[], int st, int dr){
7       if(st == dr)
8           return a[st];
9       int mij = (st + dr) / 2;
10      return sum_vector(a, st, mij) + sum_vector(a, mij + 1, dr);
11  }
```

```
6   bool toate_pare(int a[], int st, int dr){
7       if(st == dr)
8           return a[st] % 2 == 0;
9       int mij = (st + dr) / 2;
10      return toate_pare(a, st, mij) && toate_pare(a, mij + 1, dr);
11  }
```

CAUTARE BINARA CU DIVIDE ET IMPERA

```
6   bool cb(int a[], int st, int dr, int val){
7       if(st == dr)
8           return a[st] == val;
9       int mij = (st + dr) / 2;
10      if(a[mij] == val)
11          return true;
12      if(a[mij] < val)
13          return cb(a, mij + 1, dr, val);
14      return cb(a, st, mij - 1, val);
15  }
```

SORTARE CU DIVIDE ET IMPERA

- Inainte de a vedea metoda de sortare propriu-zisa este important de precizat ca Divide et Impera sta la baza sortarilor in timp n*log2n – cele mai rapide sortari.
- In aceasta lectie vom vorbi despre metoda MERGE sort
- Ideea de la baza acestei metode este sortarea celor 2 jumatati de secvente si interclasarea lor. Astfel, avand in vedere ca lungimile secventelor se injumatatesc constant, numarul de pasi in adancime pe care ii va face recursivitatea este maxim log2*n, iar toate interclasarile vor duce la complexitate finala n (dimensiunea sirului intreg)

```
6   void mergeSort(int st, int dr){
7       if(st == dr)
8           return ;
9       else{
10          int mij = (st + dr) / 2;
11          mergeSort(st, mij);
12          mergeSort(mij + 1, dr);
13          int inda = st, indb = mij + 1, indc = 0;
14          while(inda <= mij && indb <= dr)
15              if(a[inda] <= a[indb])
16                  c[++indc] = a[inda++];
17              else c[++indc] = a[indb++];
18          while(inda <= mij)
19              c[++indc] = a[inda++];
20          while(indb <= dr)
21              c[++indc] = a[indb++];
22          for(int i = 1; i <= indc; ++i)
23              a[st + i - 1] = c[i];
24      }
25  }
```

BACKTRACKING

- Backtracking este un procedeu de programare care pentru a determina rezultatul unei probleme, genereaza toate variantele posibile.
- Este important de precizat ca faptul ca genereaza toate variantele posibile, il face un algoritm foarte lent, care nu poate fi folosit pentru date de intrare foarte mari.
- Informaticienii il folosesc foarte mult pentru matematica, putand sa genereze toate Permutarile, Combinarile, Aranjamentele, Partitiile etc.
- Desi pare un algoritm care se foloseste mult pe siruri, acesta poate fi implementat si pe matrici pentru a determina numarul total de drumuri intre 2 pozitii sau alte probleme de acest tip

PERMUTARI

```cpp
#include <iostream>
using namespace std;

// generarea permutarilor

int n; // elementele vor fi 1, 2, 3, ... , n -> n este in limita a 10 elemente
int x[11], P[11];

void afis(){
    for(int i = 1; i <= n; ++i)
        cout << x[i] << ' ';
    cout << "\n";
}

void back(int k){ // complexitate: n * n!
    for(int i = 1; i <= n; ++i)
        if(!P[i]){
            P[i] = 1;
            x[k] = i;
            if(k == n)
                afis();
            else back(k + 1);
            P[i] = 0;
        }
}

int main(){
    cin >> n;
    back(1);
    return 0;
}
```

COMBINARI

```cpp
#include <iostream>
using namespace std;

// generarea combinarilor de n luate cate k

int n, k; // datele de intrare
int x[11];

void afis(){
    for(int i = 1; i <= k; ++i)
        cout << x[i] << ' ';
    cout << '\n';
}

void back(int pas){
    for(int i = x[pas - 1] + 1; i <= n - (k - pas); ++i){
        x[pas] = i;
        if(pas == k)
            afis();
        else back(pas + 1);
    }
}

int main(){
    cin >> n >> k;
    back(1);
    return 0;
}
```

DRUM IN MATRICE

```cpp
#include <iostream>
using namespace std;

// problema soarece

int n, m, xs, ys, xb, yb, cnt_drumuri;
bool a[11][11], P[11][11];

int dx[]={0,0,-1,1};
int dy[]={-1,1,0,0};

bool inmat(int x, int y){
    return x >= 1 && x <= n && y >= 1 && y <= m;
}

void back(int x, int y){
    if(x == xb && y == yb)
        cnt_drumuri ++;
    for(int d = 0; d <= 3; ++d){
        int xnou = dx[d] + x;
        int ynou = dy[d] + y;
        if(inmat(xnou, ynou) && P[xnou][ynou] == 0 && a[xnou][ynou] == 0)
            P[xnou][ynou] = 1, back(xnou, ynou), P[xnou][ynou] = 0;
    }
}
```

```cpp
int main(){
    cin >> n >> m;
    for(int i = 1; i <= n; ++i)
        for(int j = 1; j <= m; ++j)
            cin >> a[i][j];
    cin >> xs >> ys >> xb >> yb;
    P[xs][ys] = 1;
    back(xs, ys);
    cout << cnt_drumuri;
    return 0;
}
```