

RECURSIVITATE DIVIDE ET IMPERA BACKTRACKING

-Admitere UBB 2022-

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 jumatați ale acestuia au toate elementele 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 $n \cdot \log_2 n$ – 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 $\log_2 n$, 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

```
1  #include <iostream>
2  using namespace std;
3
4  // generarea permutarilor
5
6  int n; // elementele vor fi 1, 2, 3, ... , n -> n este in limita a 10 elemente
7  int x[11], P[11];
8
9  void afis(){
10     for(int i = 1; i <= n; ++i)
11         cout << x[i] << ' ';
12     cout << "\n";
13 }
14
15 void back(int k){ // complexitate: n * n!
16     for(int i = 1; i <= n; ++i)
17         if(!P[i]){
18             P[i] = 1;
19             x[k] = i;
20             if(k == n)
21                 afis();
22             else back(k + 1);
23             P[i] = 0;
24         }
25 }
26
27 int main(){
28     cin >> n;
29     back(1);
30     return 0;
31 }
```

COMBINARI

```
1  #include <iostream>
2  using namespace std;
3
4  // generarea combinarilor de n luate cate k
5
6  int n, k; // datele de intrare
7  int x[11];
8
9  void afis(){
10     for(int i = 1; i <= k; ++i)
11         cout << x[i] << ' ';
12     cout << '\n';
13 }
14
15 void back(int pas){
16     for(int i = x[pas - 1] + 1; i <= n - (k - pas); ++i){
17         x[pas] = i;
18         if(pas == k)
19             afis();
20         else back(pas + 1);
21     }
22 }
23
24 int main(){
25     cin >> n >> k;
26     back(1);
27     return 0;
28 }
```

DRUM IN MATRICE

```
1  #include <iostream>
2  using namespace std;
3
4  // problema soarece
5
6  int n, m, xs, ys, xb, yb, cnt_drumuri;
7  bool a[11][11], P[11][11];
8
9  int dx[]={0,0,-1,1};
10 int dy[]={-1,1,0,0};
11
12 bool inmat(int x, int y){
13     return x >= 1 && x <= n && y >= 1 && y <= m;
14 }
15
16 void back(int x, int y){
17     if(x == xb && y == yb)
18         cnt_drumuri ++;
19     for(int d = 0; d <= 3; ++d){
20         int xnou = dx[d] + x;
21         int ynou = dy[d] + y;
22         if(inmat(xnou, ynou) && P[xnou][ynou] == 0 && a[xnou][ynou] == 0)
23             P[xnou][ynou] = 1, back(xnou, ynou), P[xnou][ynou] = 0;
24     }
25 }
```

```
27 int main(){
28     cin >> n >> m;
29     for(int i = 1; i <= n; ++i)
30         for(int j = 1; j <= m; ++j)
31             cin >> a[i][j];
32     cin >> xs >> ys >> xb >> yb;
33     P[xs][ys] = 1;
34     back(xs, ys);
35     cout << cnt_drumuri;
36     return 0;
37 }
```