Crețu Laurențiu Vasile
CTI, Grupa 161

Tema 1- Divide et impera

Problema 4

Enunț:

Pe o suprafata este trasată o curbă închisă. Suprafata este memorata ca o matrice cu M linii si N coloane, cu valori 0, iar punctele apartinand curbei sunt notate cu 1. Dandu-se un punct din interiorul zonei marginite de curba, se cere sa se umpla cu 1 această zonă.

Date de intrare: fișierul date.in conține:

- pe prima linie doua numere natural, m și n;
- pe urmatoarele m linii elementele unei matrice cu m linii si n coloane, cu semnificatia din enunt.
- pe ultima linie doua numere l0,c0, reprezentand linia, respectiv coloana pe care se află punctul.

Date de ieșire: fișierul date.out conține matricea transformata.

Exemplu:

date.in	date.out
10 8 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 0 0 0 1 1 0 0 1 0 0 1 1 0 0 0 1 1 0 1 0 0 0 0 0 1 0 1 1 0 1 1 1 1 0 0 1 0 1 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 5 4	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 0 0 0 1 1 1 1 1 0 0 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 0 0 1 1 1 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0

Varianta 1 de rezolvare: Divide et impera

Această variantă are ca și principiu de rezolvare împărțirea matricei m×n în linii. Când am extras o linie din matrice, verificăm dacă aceasta conține „1”, în caz afirmativ fixăm prima și ultima apariție a unei valori de „1” în linia curentă, apoi parcurgem linia între cele două extremități și setăm toate elementele cu valoarea „1”. Algoritmul se oprește atunci când toate liniile matricei au fost parcurse și verificate.

```
int linieUnu(int *v,int n){
    for (int i=0; i<n; i++)
        if (v[i]==1)
            return 1;
    return 0;
}
```

```
int* transformareUnu(int *v,int n){
    int prim = -1;
    int ultim = -1;
    for (int i=0; i<=n; i++){
        if (v[i] == 1 && prim == -1)
            prim = i;
        if (v[n-i] == 1 && ultim == -1)
            ultim = n-i;
        if (prim!=-1 && ultim!=-1)
            break;
    }
    for (int k=prim; k<=ultim; k++)
        v[k] = 1;
    return v;
}
void divide(int **matrice,int i,int m,int n){
    if (i == m){
        if (linieUnu(matrice[m],n)==1)
            matrice[m] = transformareUnu(matrice[m],n);
        return;
    }
    int mij= (m+i)/2;
    divide(matrice,i,mij,n);
    divide(matrice,mij+1,m,n);
}
```

Complexitatea timpului de executare:

Relația de recurență: $T(m,n) = a \cdot T\left(\frac{m}{b}, n\right) + f(n)$, unde:

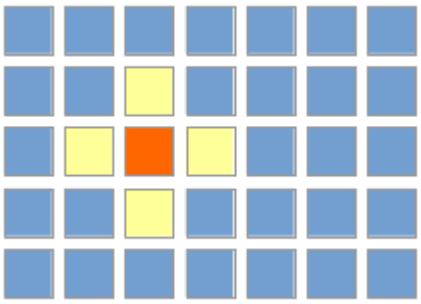
- a = 2, reprezintă numărul de subprobleme în care se împarte problema inițială
- b = 2, reprezintă numărul de subprobleme folosite în rezolvarea problemei inițiale
- f(n) = n, reprezintă complexitatea de rezolvare a unei subprobleme direct rezolvabilă

Deci, $T(m,n) = 2 \cdot T\left(\frac{m}{2}, n\right) + n$.

$\left.\begin{array}{l} \log_b a = \log_2 2 = 1 \\ f \in \mathcal{O}(n^p) \Rightarrow p = 1 \end{array}\right\} \; p = \log_b a = 1 \Rightarrow T(m,n) \in \mathcal{O}(n \cdot \log_2 m)$

Varianta 2 de rezolvare: Algoritm de fill

Această variantă are ca și principiu de rezolvare parcurgerea vecinilor unui element atât timp cât ne aflăm în interiorul curbei de „1”. Pornim de la elementul de coordonate (x_start,y_start) și parcurgem vecinii acestuia în direcțiile (-1,0), (0,1), (1,0) și (0,-1), la fiecare pas setăm valoarea elementului curent cu „1”. Complexitatea în timp a acestei metode este $\mathcal{O}(n \cdot m)$.



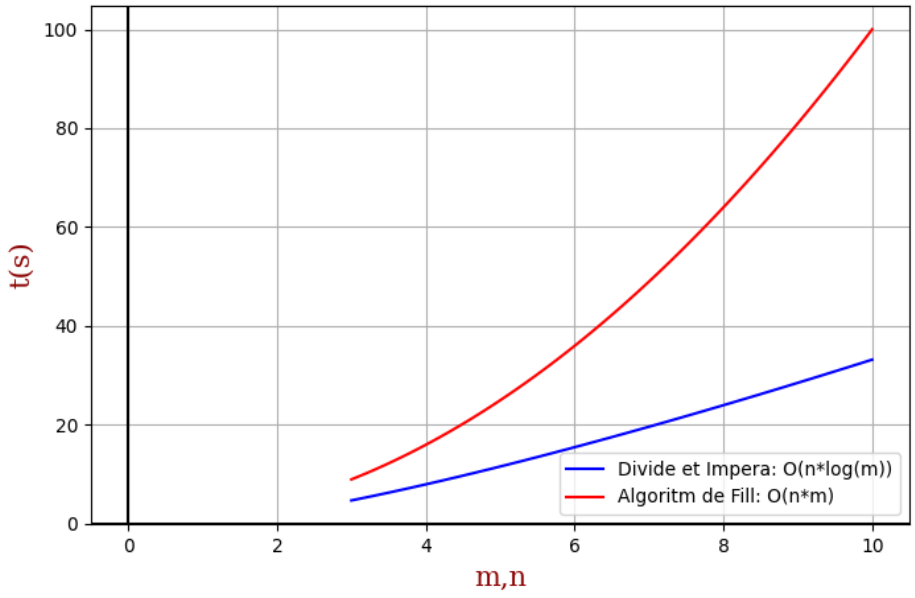
Coordonate vecini:

- Nord: (-1,0)
- Est: (0,1)
- Sud: (1,0)
- Vest: (0,-1)

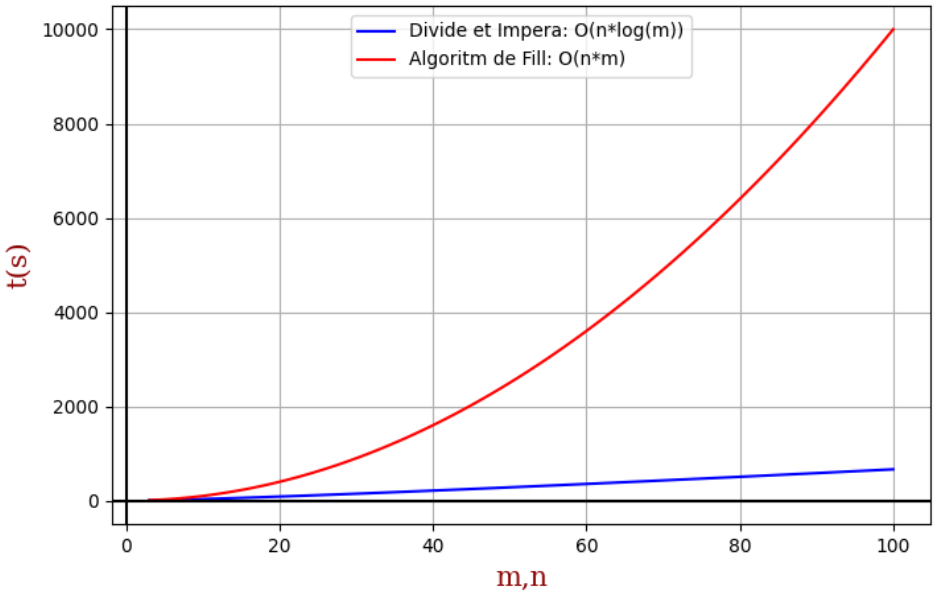
```
void fill(int **matrice,int m,int n,int x_curent,int y_curent){
    if (matrice[x_curent][y_curent]==1)
        return;
    matrice[x_curent][y_curent] = 1;
    fill(matrice,m,n,x_curent-1,y_curent);
    fill(matrice,m,n,x_curent,y_curent+1);
    fill(matrice,m,n,x_curent+1,y_curent);
    fill(matrice,m,n,x_curent,y_curent-1);
}
```

Compararea complexității timpului de executare a celor două metode

- Pentru m,n ∈ [3,10]:



- Pentru m,n ∈ [3,100]:



- Pentru m,n ∈ [3,10^6]: