Tema 1 - PA

Mitrofan Alexandru

Facultatea de Automatica si Calculatoare 323CD

1 Tehnica Divide et Impera

1.1 Enuntul problemei

Se consideră un șir cu n elemente, numere naturale. Folosind metoda Divide et Impera, determinați suma elementelor acestui șir.

1.2 Descrierea solutiei problemei

Adun prin metoda Divide et impera prima jumatate din sir cu a doua jumatate din sir. Si cand "left" o sa fie egal cu "right" inseamna ca s-a ajuns in punctul in care mai avem un singur element(in submultimea de la elemental de pe pozitia "left" la elemental de pe pozitia "right") si il putem returna.

1.3 Pseudocod

Algorithm 1 Sume Elementelor unui Vector

```
1: procedure SUM(V,LEFT,RIGHT)
2:     mid ← (left + right)/2
3:     if left >= right then
4:         return 0
5:     else
6:         return sum(v,left,mid)+ sum(v,mid+1,right)
```

1.4 Complexitate

T(1)=1
T(n)= 2 * T(n/2) + 1 $\Rightarrow$ T(n/2)= 2 * T(n/4) +1
$\Rightarrow$ T(n)= 4 * T(n/2) + 2 +1

....(dupa k termeni)

$T(n) = 2^k * T(n/2^k) + (2^{k-1} + \ldots + 2 + 1) = 2^k * T(n/2^k) + 2^k - 1$
$n/2^k = 1 \Rightarrow 2^k = n \Rightarrow k = log2n$

$\Rightarrow T(n) = n * T(1) + 2^k$–1 $= n * 1 + n$–1 $\Rightarrow T(n) = 2n - 1 = O(n)$

1.5 Analiza succinta a complexitatii

Algoritmul este la fel de optim ca si varianta clasica de while in care adunam element cu element, adica O(n), cea mai buna complexitate care se poate obtine pentru suma unei multimi.

1.6 Exemplificarea aplicării algoritmului propus

V= 1 2 3 4
Left =1, right =4
Avem sum(v,1,4)=sum(v,1,2)+sum(v,3,4)=sum(v,1,1)+sum(v,2,2)+sum(v,3,3)+sum(v,4,4)= v[1]+v[2]+v[3]+v[4]=1+2+3+4=10 (egalul se poate inlocui si cu return)

2 Tehnica Greedy

2.1 Enuntul problemei

Se dau o multime A cu m numere intregi nenule si o multime B cu $n \geq m$ numere intregi nenule. Se cere sa se selecteze un sir cu m elemente din B,x1,x2,...,xm astfel incat expresia urmatoare sa fie maxima: E=a1x1+a2x2+...+amxm unde, a1, a2,...,am sunt elemente ale multimii A intr-o anumita ordine pe care trebuie sa o determinati.

2.2 Descrierea solutiei problemei

Elementele multimii A si elementele multimii B sunt sortate , ideea problemei este ca daca elem a apartine lui A este negativ, trebuie luat cu cel mai mic element b care apartine lui B (nefolosit deja) pentru ca a*b sa fie maximala. Daca daca am ajunsla primul element pozitiv din A ,toate elementele de acolo trebuie inmultite pe rand cu ultimele elemente din B, (pana se ajunge ca ultimul element din A sa se inmulteasca cu ultimul element din B),pentru ca suma elementelor inmultite sa fie maximala.

2.3 Pseudocod

Algorithm 2 Suma maxima E cu algoritm Greedy cu prelucrare

```
1: procedure suma_maxima(A, m, B, n)
2:     Sort(A)
3:     Sort(B)
4:     j ← 1
5:     flag ← 0
6:     sol ← ∅
7:     for i=1,i<=m do
8:         if A[i] > 0 ∧ flag == 0 then
9:             j ← j + n − m
10:            flag ← 0
11:        sol ← sol ∪ B[j]
12:        j ← j + 1
       return sol
```

2.4 Complexitate

Complexitate din perspectiva timpului: O(n * logn)

2.5 Analiza succinta de obtinere a optimului global

Indeplineste proprietatea de alegere de tip Greedy si Proprietatea de substructura optimala.
Pentru cel mai mic a ∈A negativ (neutilizat) trebuie sa alegem cel mai mic b ∈B (neutilizat) ca a*b sa fie maximal,iar pentru cel mai mare a ∈A pozitiv (neutilizat) trebuie sa alegem cel mai mare b ∈B (neutilizat) ca a*b sa fie maximal

2.6 Exemplificarea aplicării algoritmului propus

A=[6,-2,4,-1,2]
B=[1,-2,3,2,4,6,5,7]
Prima data sortez A, B
A = [-2,-1,2,4,6]
B = [-2,1,2,3,4,5,6,7]
Apoi intrand prin for(primele 2 elemente din A sunt negative si ultimele 3 sunt pozitive)
Deci sol o sa ia primele 2 valori din B, si ultimele 3 numere din B(datorita if-ului din for)
Deci solutia care o sa fie returnata [-2,1,5,6,7] care esta maxim global

3 Programare dinamica

3.1 Enuntul problemei

Se da o matrice NxM care este drumul cu suma maximala din coltul stanga sus in coltul dreapta jos, avand voie sa parcurgi mergand doar la dreapta sau in jos.

3.2 Descrierea solutiei problemei

"Mat" este matricea data, iar fiecare element din "mat" este schimbat pe rand mergand pe fiecare linie cu drumul maximal pana la acesta. Matricea este bordata cu 0, pentru a functiona if-ul in cazul elementelor de pe prima linie sau prima coloana.

3.3 Pseudocod

Algorithm 3 Max sum drum Programare dinamica

```
1: procedure drum_maxim(n, m, mat)
2:     Bordam_cu_0(mat)
3:     for i ← 1, n do
4:         for j ← 1, m do
5:             if mat[i − 1][j] > mat[i][j − 1] then
6:                 mat[i][j] ← mat[i][j] + mat[i − 1][j]
7:             else
8:                 mat[i][j] ← mat[i][j] + mat[i][j − 1]
       return mat[n] [m]
```

3.4 Complexitate

Complexitate din perspectiva timpului: NxM
Complexitate din perspectiva memorieii: NxM

3.5 Gasire recurenta

Caz de baza Cand matricea are un singur element,acesta este drumul maximal.

Cazul general Raspuns = mat[i] [j] + max(mat[i-1] [j]+ mat[i][j-1]), daca i==n si j==m
mat[i] [j] = mat[i] [j]+ max(mat[i-1] [j]+ mat[i][j-1]),altfel
Functioneaza parcurgand matricea o singura data pe linie

3.6 Exemplificarea aplicării algoritmului propus

N=2,M=2
mat= $\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$
Dupa prima intrare prin for se alege else-ul din for si pt ca matricea este bordata,mat o sa fie: $\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$
Doupa a doua intrare prin for se alege tot else ul iar mat devine: $\begin{bmatrix} 1 & 3 \\ 3 & 4 \end{bmatrix}$
Dupa a treia intrare prin for se respecta conditia din if, iar mat o sa fie: $\begin{bmatrix} 1 & 3 \\ 4 & 4 \end{bmatrix}$
Dupa a patra intrare prin for se intra pe else, iar mat ajunge: $\begin{bmatrix} 1 & 3 \\ 4 & 8 \end{bmatrix}$
Drumul maxim este mat[n][m] adica 8.

4 Backtracking

4.1 Enuntul problemei

Se dă un număr natural n Generați toate șirurile de 2 * n paranteze rotunde care se închid corect.

4.2 Descrierea solutiei problemei

In variabilele “deschise” si “inchise” retin numarul de paranteze deschise respective inchise puse pana la pozitia “poz” ,iar pana se incheie sirul de paranteze (corect inchise), “deschise” trebuie sa fie mai mare sau egal cu “inchise (conditia 1) si “deschise” nu trebuie sa treaca de n(conditia 2). Se ajunge la o varianta corecta de siruri de paranteze in momentul in care numarul de paranteze “inchise” este egal cu n, pentru ca stim ca pe parcursul programului s au respectat conditiile 1 si 2 ,deci din aste rezulta ca sunt 2*n paranteze in sirul creeat de noi deci am gasit un sir correct de paranteze inchise de lungime 2*n .Folosind metoda backtracking ne intoarcem tot timpul intr-un moment inapoi unde subsirul era correct si continuam sa punem paranteze inntr-un mod diferit ,dar respectand conditiile 1 si 2.

4.3 Pseudocod

Algorithm 4 Sir paranteze

```
1: procedure paranteze(n, poz, deschise, inchise, str)
2:     if inchise == n then
3:         Print (str)
4:     else
5:         if deschise > inchise then
6:             str[poz] ← ")"
7:             paranteze(n,poz+1,deschise,inchise+1,str)
8:         if deschise < n then
9:             str[poz] ← "("
10:            paranteze(n,poz+1,deschise+1,inchise,str)
```

4.4 Complexitate

Complexitate temporala : $O(2^n)$

4.5 Analiză succinta asupra eficient,ei algoritmului propus

Complexitatea spatiala s-a redus la O(n) pentru ca am folosit vectorul str pentru a printa,in loc sa fac o matrice cu toate sirurile inchise corect si apoi sa o printez.

4.6 Exemplificarea aplicării algoritmului propus

(pentru simplitatea problemei if- ul cu inchise == n easte if ul 1 iar celelalte 2 sunt if-urile 2 ,respectiv3)
n=2
Prima data se intra prin if-ul 3 deci str=" (" (starea 1)
Dupa se intra prin if-ul 2(pt ca deschise=1 inchise =0) deci str= "( )" (starea 2)
Dupa se intra prin if-ul 3(pt ca deschise=1 inchise =1) deci st= "( )("(starea 3)
Dupa se intra prin if-ul 2(pt ca deschise=2 inchise =1) deci str "( )( )"(starea 4)

Dupa se intra prin if-ul 1(pt ca inchise =2) deci se afiseaza str= "( )( )"
Dupa se ajunge in starea 4 si nu se intra in if-ul 3(deschise=2=n)
Dupa se ajunge in starea 3 si se iese direct

Dupa se ajunge in starea 2 si se intra in if-ul 3(pt ca deschise=1 ¡ n=2) deci st= "(("(starea 5)
Dupa se intra prin if-ul 2(pt ca deschise=2 inchise =0)deci str="(( )"(starea 6)
Dupa se intra prin if-ul 2(pt ca deschise=2 inchise =1) deci str="(( ))" (starea 7)

Dupa se intra prin if-ul 1(pt ca inchise =2) deci se afiseaza str= "(( ))"
Dupa se ajunge in starea 7 si nu se intra in if-ul 3(deschise=2=n)
Dupa se ajunge in starea 6 si nu se intra in if-ul 3(deschise=2=n)
Dupa se ajunge in starea 5 si se iese direct

Dupa se ajunge in starea 1 si se iese direct si programul se termina
La output vom avea "( ) ( )" si "(( ))"

5 Analiza comparativa

5.1 Precizarea tipurilor de probleme pentru care fiecare tehnică poate fi aplicată

Tehnica Divide et Impera se poate aplica numai pe probleme care se pot imparti inn subprobleme.
Tehnica Greedy se poate aplica pe problemele unde trebuie sa afisam o secventa de actiuni,avand mai multe posibilitati.
Tehnica Programarii Dinamice se poate aplicatanumai pe probleme care se pot imparti inn subprobleme mai mici,iar subproblemele au elemente comune.
Tehnica Backtracking se poate aplica pe algoritmii care ofera mai multe solutii si are ca rezultat obtinerea tuturor solutiilor probleme.

5.2 Problema aleasa

Problema aleasa este cea de la programare dinamica (avand enuntul problemei scris mai sus) si o voi transforma in Tehnica Divide et Impera.

5.3 Explicarea algoritmului

Am parcurs cu ajutorul Tehnicii Divide et Impera toate drumurile posibile(mergand doar la dreapta sau in jos), care ajung in colrul dreapta jos si am returnat drumul maxim.

5.4 Pseudocod

Algorithm 5 Max sum drum Divide et Impera

```
1: procedure divide(mat, i, j, n, m, suma)
2:     suma ← suma + mat[i][j]
3:     if i == n ∧ j == m then return suma;
4:     if i == n then return divide(mat,i,j+1,n,m,suma)
5:     if j == m then return divide(mat,i+1,j,n,m,suma)
       return max(divide(mat,i+1,j,n,m,suma),divide(mat,i,j+1,n,m,suma))
```

5.5 Complexitate

T(n, m) = O(1) (pentru i == n ∧ j == m)
T(n, m) = T(n, m-1) + O(1) (pentru i == n)
T(n, m) = T(n-1, m) + O(1) (pentru j == m)
T(n, m) = T(n-1, m) + T(n, m-1) + O(1) (pentru ultimul return)
Complexitatea este foarte mare(exponentiala)

5.6 Avantaje si dezavantaje

Ambele metode afiseaza varianta corecta,adica cea de optim global,fata de o posibila varianta Greedy ,dar varianta cu Divide et Impera are complexitatea de timp mult mai mare(exponentiala) fata de cea rezolvata cu Programare dinamica.

5.7 Exemplificarea aplicării algoritmului propus

N=2,M=2
mat= $\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$
Prima data se face divide(mat,1,1,2,2,0) si se ajunge la ultimul return,sunde se alege maximul dintre 2 functii (max(divide(mat,2,1,2,2,1),divide(mat,1,2,2,2,1)))

divide(mat,2,1,2,2,1) intra pe al doilea if returnand divide(mat,2,2,2,2,4) care mai apoi intra pe primul if si returneaza 4+mat[2][2]=4 + 4 = 8

divide(mat,1,2,2,2,1) intra pe al treilea if returnand divide(mat,2,2,2,2,3) care mai apoi intra pe primul if si returneaza 3+mat[2][2]=3 + 4 = 7

Deci functia divide(mat,1,1,2,2,0) intoarce max(8,7)=8

References

1. https://www.pbinfo.ro/probleme/categorii/94/divide-et-impera
2. https://cardosraul.weebly.com/metoda-greedy.html
3. https://info.mcip.ro/?cap=Programare
4. https://www.pbinfo.ro/probleme/344/paranteze