Documentație-Tema Divide et Impera-Problema 9

Pentru a nu mai fi nevoit să atribui elementelor din matrice valori de 0 m-am gândit că este util să aloc memoria dinamic matricei folosind functia calloc. Astfel rămane doar de găsit o metodă pentru a putea atribui unde este necesar valoarea 1 elementelor din tablou atunci când reducem matricea la cazul de bază, o matrice de un element. Pentru a împărți recursiv în cadrane matricea este evident că vom avea nevoie de patru autoapeluri la fiecare pas de împărțire. De asemenea trebuie identificată o conditie de oprire a apelurilor recursive. În acest sens ne folosim de o observație pe care o putem face legată de dimensiunile matricelor obținute prin împărțirea în cadrane. La fiecare împărțire dimesiunea se reduce de la la . Astfel, obținem că atunci când matricea se reduce la dimensiunea de un element n va avea valoarea 0. Putem impune deci n=0 ca fiind condiția de oprire. Pentru a cunoaște în permanență nevoia de a completa cadranul din dreapta sus doar cu valori de 1 am ales transmiterea prin autoapel și a unui parametru numit „tot”. Dacă acest parametru primește valoarea 1 atunci în cadrul următorului apel cele 4 autoapeluri necesare împărțirii în cadrane se vor face atribuind de asemenea variabilei tot care urmează a fi transmisă valoarea 1. Dacă parametrul „tot” primește valoarea 0 atunci dintre cele 4 autoapeluri, unul va atribui valoarea 1 parametrului „tot” necesar atribuirii de valori 1 în cadranul dreapta sus, iar celelalte 3 autoapeluri vor atribui valoarea 0 parametrului „tot” pentru a ști următoarele apeluri că trbuie făcute cele patru autoapeluri în același mod. Un alt element cheie în împărțirile matricei îl reprezintă cunoașterea poziției de la care începe parcurgerea unei submatrici și cunoașterea limitelor maxime pe care le pot lua indicii de linie și de coloană, care, vor oferi astfel poziția la care s-ar termina parcurgerea submatricei. În nevoia de a cunoaște aceste limite am încercat să observ legături între limitele inferioare și superioare curente și dimensiunea submatricelor. Astfel, am remarcat faptul că atunci când limita superioară trebuie să se înjumătățească trebuie să scad din valoarea curentă a limitei superioare valoarea dimensiunii submatricei ale cărei limite trebuie aflate.

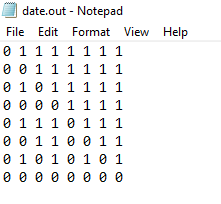
|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 00 | 01 | 02 | 03 | 04 | 05 | 06 | 07  lmax- cmax, i, (j+cmax)/2+1 dreapta sus  lmax- ,cmax- , i, j stanga sus  lmax, cmax- ,(i+lmax)/2+1, jstanga jos  lmax,cmax,(i+lmax)/2+1,(j+cmax)/2+1 dreapta jos |
| 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 |
| 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 |
| 30 | 31 | 32 | 33 | 34 | 35 | 36 | 37 |
| 40 | 41 | 42 | 43 | 44 | 45 | 46 | 47 |
| 50 | 51 | 52 | 53 | 54 | 55 | 56 | 57 |
| 60 | 61 | 62 | 63 | 64 | 65 | 66 | 67 |
| 70 | 71 | 72 | 73 | 74 | 75 | 76 | 77 |

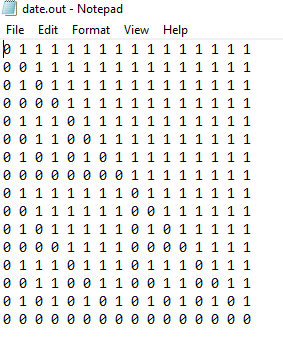
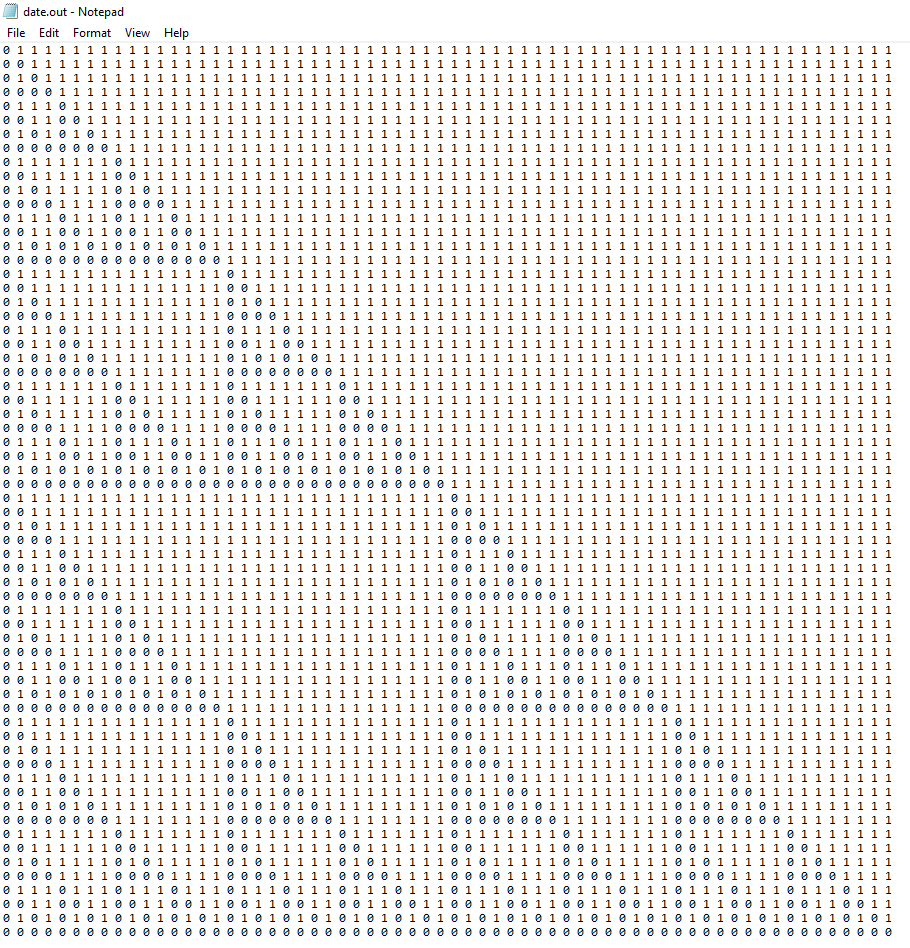
Am inserat un tabel cu indicii pozițiilor elementelor din matrice în cazul în care n=3 pentru a se putea observa mai ușor formulele așa cum se văd și pe hârtie. Acel +1 din cadrul calculului lui i și j apare datorită faptului că indicii de început ai matricei inițiale sunt 0 si 0. O altă variabilă pe care am utilizat-o în funcția de împărțire este „ok”. Utilizarea acesteia a fost necesară pentru a trata cazul în care n-ul citit din fișier este 0. Cum în funcție este atribuită valoarea 1 elementelor matricei dacă „tot” are valoarea 1, în cazul apelului din funcția main, „tot” are valoarea 0 iar „ok” are valoarea 1, iar dacă n-ul din fișier este 0 nu vor mai exista autoapeluri pentru a putea atribui valoarea 1 lui „tot” și astfel este necesară existența lui „ok” care este ca un fel de „tot” pentru primul apel.

Dacă dimensiunea matricei este de atunci dimensiunea datelor va fi . Legat de complexitatea algoritmului , o problemă se descompune în patru subprobleme deci a=4, dimensiunea datelor de intrare este dată aproximativ de mărimea datelor submatricei de unde rezultă b=. Complexitatea împărțirii problemei curente în subprobleme și combinarea soluțiilor subproblemelor este doar o sumă de complexități O(1) deci f(n) se pate aproxima la valoarea 1. Din toate acestea rezultă că algoritmul de tip Divide et Impera are o complexitate T()=4·T()+1=4[4·T()+1]+1=·T()+4+1=·T()++4+1=...=·T()++..++4+1=+...+4+1=S

4S=+...++4

Rezulta că S=, deci complexitatea algoritmului este O()=O()=O(n)

Ca și exemple am testat să văd până la ce valoare a lui n compilează cu succes programul și am observat că n=14 este ultima valoare care compilează cu succes. În acest caz în timpul rulării era utilizat 1GB de Ram cu o durată a rulării de 73 de sec la o frecvență de aproximativ 3,3 GHz a procesorului.

Voi introduce și niște capturi de ecran ale câtorva exemple care încap în întregime pe ecran din punct de vedere al dimensiunilor. 

n=4 n=3

n=6