ALGORİTMA ANALİZİ

ÖDEV 1 | SORU 2

GRUP 1

İrem ATILGAN

17061036

07.11.2020

SORU 2

Bir matriste, verilen N değeri dikkate alınarak, “Von Neumann’s Neighborhood” kuralına göre hücrelerin “1” değeri ile işaretlenmesi isteniyor.

ÇÖZÜM

Çözüm aşamaları aşağıdaki gibidir:

1. Kullanıcıdan girdi olarak N sayısı alınır.
2. Oluşacak matristeki 1’ler ile matris sınırı arasında en az bir satır boşluk kalacak şekilde matris boyutları oluşturulur (2*N + 3).
3. **vn_neighborhood** fonksiyonu çağrılarak merkezden itibaren, mevcut hücre “1” ile işaretlenir. Bulunuan hücrenin sağ, sol, üst ve altındaki hücrelerin işareti (1/0) kontrol edilerek hücrelerin tekrar tekrar işaretlenip sonsuz döngüye girmemesi sağlanır. Eğer hücrenin değeri sıfırsa fonksiyon yeniden o hücre için çağrılır. Fonksiyon parametresi olarak merkez noktası (x0,y0 - center) tutularak fonksiyonun bitmesi gerektiği durum kontrol edilir. Eğer Von Neumann’ın Komşuluk Kuralı’nda komşuluk aralığı için belirtilen eşitlik sağlanıyorsa fonksiyon çalışmaya devam eder; Aksi halde durur.

PROGRAM KODLARI

```c
#include <stdio.h>
#include <stdlib.h>

void vn_neighborhood(int**,int,int,int,int);    //Recursive Solution Algorithm (Based on Von Neumann's Neighborhood Rule)
void print_matrix(int**, int);                  //To show result matrix
int** create_matrix(int,int*);                  //Create matrix based on given dimension rule

int main()
{
    int N;      //input from user
    int dims;   //dimensions of matrix
    int center; //location index of center cell in matrix

    printf("\nN = ");
    scanf("%d",&N);
    int** arr = create_matrix(N,&dims); //create matrix and set dimension

    center = dims/2;
    vn_neighborhood(arr,center,center,center,N); //start recursive neighborhood function
    print_matrix(arr,dims);                 //show results

    free(arr);  //free array from memory
    return 0;
}
```

```c
void print_matrix(int** arr, int dims) {
    int i,j;
    int total; //total 1s in a row

    printf("\n");
    for(i = 0; i < dims; i++) {
        total = 0;
        for(j = 0; j < dims; j++) {
            printf("%d  ",arr[i][j]);
            total += arr[i][j];
        }
        printf("\t# of black cells = %d\n",total);
    }
    //Formula for total black cells = 2*r*(r+1)+1
    //dims = 2*r+3, r = (dims-3)/2
    printf("Total number of black cells in matrix = %d",((dims-3)*((dims-3)/2+1)+1));

    return;
}
```

```c
int** create_matrix(int N, int* dims)
{
    int i,j;
    *dims = 2*N+3; //there should be at least one line gap between 1s and matrix
    int** mat = (int**)calloc(*dims,sizeof(int*));
    for(i = 0; i < *dims; i++) {
        mat[i] = (int*)calloc(*dims,sizeof(int));
    }

    return mat;
}

void vn_neighborhood(int** arr, int x, int y, int center,int N) {

    //Check Von Neumann's Neighborhood Rule
    if(abs(x-center) + abs(y-center) <= N)
    {
        arr[x][y] = 1;
        //Call function for cells in the surrounding area if the cell is zero (to not to cause infinite loop)
        if(arr[x-1][y] == 0)
            vn_neighborhood(arr,x-1,y,center,N);
        if(arr[x][y-1] == 0)
            vn_neighborhood(arr,x,y-1,center,N);
        if(arr[x][y+1] == 0)
            vn_neighborhood(arr,x,y+1,center,N);
        if(arr[x+1][y] == 0)
            vn_neighborhood(arr,x+1,y,center,N);



    }

    return;
}
```

EKRAN ÇIKTILARI

```
N = 2

0  0  0  0  0  0  0     # of black cells = 0
0  0  0  1  0  0  0     # of black cells = 1
0  0  1  1  1  0  0     # of black cells = 3
0  1  1  1  1  1  0     # of black cells = 5
0  0  1  1  1  0  0     # of black cells = 3
0  0  0  1  0  0  0     # of black cells = 1
0  0  0  0  0  0  0     # of black cells = 0

--------------------------------
Process exited after 1.426 seconds with return value 0
Press any key to continue . . .
```

```
N = 3

0  0  0  0  0  0  0  0  0       # of black cells = 0
0  0  0  0  1  0  0  0  0       # of black cells = 1
0  0  0  1  1  1  0  0  0       # of black cells = 3
0  0  1  1  1  1  1  0  0       # of black cells = 5
0  1  1  1  1  1  1  1  0       # of black cells = 7
0  0  1  1  1  1  1  0  0       # of black cells = 5
0  0  0  1  1  1  0  0  0       # of black cells = 3
0  0  0  0  1  0  0  0  0       # of black cells = 1
0  0  0  0  0  0  0  0  0       # of black cells = 0

--------------------------------
Process exited after 7.192 seconds with return value 0
Press any key to continue . . .
```

```
N = 5

0  0  0  0  0  0  0  0  0  0  0  0  0   # of black cells = 0
0  0  0  0  0  0  1  0  0  0  0  0  0   # of black cells = 1
0  0  0  0  0  1  1  1  0  0  0  0  0   # of black cells = 3
0  0  0  0  1  1  1  1  1  0  0  0  0   # of black cells = 5
0  0  0  1  1  1  1  1  1  1  0  0  0   # of black cells = 7
0  0  1  1  1  1  1  1  1  1  1  0  0   # of black cells = 9
0  1  1  1  1  1  1  1  1  1  1  1  0   # of black cells = 11
0  0  1  1  1  1  1  1  1  1  1  0  0   # of black cells = 9
0  0  0  1  1  1  1  1  1  1  0  0  0   # of black cells = 7
0  0  0  0  1  1  1  1  1  0  0  0  0   # of black cells = 5
0  0  0  0  0  1  1  1  0  0  0  0  0   # of black cells = 3
0  0  0  0  0  0  1  0  0  0  0  0  0   # of black cells = 1
0  0  0  0  0  0  0  0  0  0  0  0  0   # of black cells = 0

--------------------------------
Process exited after 1.448 seconds with return value 0
Press any key to continue . . .
```