LUMEN

Yannis ROCHE

Prep'Isima 1 Année universitaire 2020-2021

Figure 1 – Université Clermont Auvergne

Figure 2 – ISIMA

Figure 3 – Une bougie

À rendre pour le 29/03

Encadrant : Yves-jean DANIEL

Université Clermont Auvergne : 34 Avenue Carnot, 63000 Clermont-Ferrand
ISIMA : 1 Rue de la Chebarde, 63178 Aubière

Table des matières

Table des figures

Références

[1] *CodinGame - LUMEN*, **Sceurpien**

1 Problématique

Le principe de l'exercice *Lumen* créé par *Sceurpien* est de trouver le nombre de cases non éclairés par des bougies dans une pièce de dimension N par N cases.

L'intensité lumineuse de chaque bougie est donnée par un entier L et l'intensité lumineuse décroît de *1* de cases en cases en s'éloignant de la source lumineuse.

Notons que :

$$0 < N \leq 25 \qquad et \qquad 0 < L < 10$$

X	X	X	X	X
X	C	X	X	X
X	X	X	X	X
X	X	X	X	X
X	X	X	X	X

FIGURE 4 – Exemple de pièce possible

2	2	2	1	0
2	3	2	1	0
2	2	2	1	0
1	1	1	1	0
0	0	0	0	0

FIGURE 5 – Luminosité correspondante

2 Résolution de l'exercice

2.1 Principe de résolution

2.1.1 Stockage des informations sur la pièce

La première problématique à laquelle il m'a fallu trouver une solution a été le format d'entrée utilisé par l'utilisateur pour fournir les informations sur la pièce. Pour ce qui est des dimensions, il suffit de stocker la largeur de la pièce (qui est aussi sa longueur) dans une variable N. De même pour l'intensité lumineuse de la source de lumière, stocké dans une variable L.

```
int N, L= 0; // Initialisation des variables N et L a 0
scanf("%d", &N); // Lecture des valeurs de N et L
scanf("%d", &L);
```

Afin de déterminer le nombre de case de la pièce où la luminosité était nulle, il me fallait récupérer toutes les positions des bougies dans la pièce afin de calculer toutes les cases éclairés par ces dîtes bougies et ensuite compter le nombre de cases non éclairées dans la pièce.

Pour pouvoir compter le nombre de cases non éclairés dans la pièce j'ai créer un tableau en deux dimension et de même taille que la pièce, initialisé à *0*. Pour ne pas utiliser plus de mémoire que nécessaires j'ai créé ce tableau par allocation de mémoire et ainsi ne pas avoir de case "inutiles".

```
// Tableau de pointeurs sur des tableau d'entiers (lignes de la piece)
int ** light = (int**) calloc(N, sizeof(int*));
for (int i = 0; i < N; i++) {
    // Tableau de pointeur sur des entiers (cases de la ieme ligne)
    light[i] = (int*) calloc(2*N-1, sizeof(int));
}
```

2.1.2 Écriture des informations de la pièce

Pour mettre en évidence les cases de la pièce qui n'étaient pas éclairé j'ai simplement affecté *1* toutes les cases du tableau qui se trouvaient dans un rayon L d'une bougies tandis que les cases non éclairées ont garder *0* comme valeur. En effet, puisque nous intéressons uniquement aux cases de la pièce aucunement éclairées, il est inutile de prendre en compte la diminution progressive de l'intensité lumineuse (une case très peu éclairée reste une case éclairée).

```
for (int i = 0; i < N; i++) {
    for (int j = 0; j < N; j++) {
        // Lecture des entrees pour toutes les cases de la piece
        char cell[4];
        scanf("%s", cell);
        // Parcours des cases dans un rayon de L autour de la bougie s'il y en a
        if(cell[0] == 'C'){
            for(int k = -L+1; k<L; k++){
                for(int l = -L+1; l<L; l++){
                    if(i+k >= 0 && i+k < N && j+l >= 0 && j+l < N){
                        // Ecriture des cases lumineuses
                        light[i+k][j+l] = 1;
                    }
                }
            }
        }
    }
}
```

2.1.3 Comptage du nombre de cases non éclairées

Finalement pour compter le nombre de cases non éclairés, je re-parcours mon tableau précédemment modifié et j'incrémente une variable *count* à chaque rencontre d'un *0*. Enfin, je peut afficher la valeur de *count* ce qui devrait compléter l'exercice sans oublier de libérer la mémoire attribué à mon tableau de deux dimension pour finir proprement le programme.

```
for (int i = 0; i < N; i++) {
    for (int j = 0; j < N; j++) {
        count += (light[i][j] == 0)?1:0;
    }
}

printf("%d\n",count);

for(int i = 0; i < N; i++){
    free(light[i]);
    light[i] = NULL;
}
free(light);
light = NULL;
```

2.2 Résultats obtenus

Test	Sortie standard	Sortie attendu	Résultat
1	9	9	Success
2	0	0	Success
3	2	2	Success
4	4	4	Success
5	14	14	Success
6	34	34	Success
7	9	9	Success
8	90	90	Success
9	9	9	Success

3 Solution de Amnesix

3.1 Stockage des informations sur la pièce

En ce qui concerne les informations stockés et qui seront étudiés plus tard, Amnesix a créé par allocation mémoire, un tableau de structure. Cette structure comporte un entier correspondant à une ligne de la salle et un entier correspondant à une colonne de la salle.

Ainsi, il stock dans son tableau les coordonnées de toutes les bougies présentent dans la salle, en ne s'intéressant qu'aux coordonnées dont le caractère correspondant (donné par l'utilisateur) est 'C'.

3.2 Étude des informations sur la pièce

Grâce au tableau de coordonnées qu'il a créé, Amnesix n'as plus qu'à pour toutes les coordonnées possibles dans la salle, si ces dîtes coordonnées sont dans une zone non éclairé par les bougies, auquel cas il incrémente un compteur de *1*.

Pour ce faire, Amnesix a créé une fonction *int isZero(struct coord candles[], int n, struct coord c, int light)* qui renvoie *1* si les coordonnées de *c*, ne sont dans aucune des zones éclairées par les bougies. Dans le cas contraire, elle retourne *0*.

```
1 // dist(a, b) renvoie la distance maximun entre a et b
2 int isZero(struct coord candles[], int n, struct coord c, int light)
3 {
4     for (int i=0; i<n; i++)
5         if (dist(candles[i], c) < light)
6             return 0;
7     return 1;
8 }
```

Ainsi, Amnesix peut directement afficher la valeur de son compteur, une fois la fonction *int isZero(...)* executé pour toutes les coordonnées possibles.

3.3 Avis personnel

J'ai choisi de présenter cette solution car bien qu'elle parte d'une même idée de départ que la mienne qui est de stocker des informations avant de les traiter, sa réalisation est quelque peu différente de la mienne. En revanche, je trouve que cette solution utilise beaucoup de d'intermédiaires avant de compter le nombre de cases sombre par rapport à ma solution. De plus Amnesix créer un tableau qui sera toujours de taille N par N pour stocker des coordonnées de bougies,

ce qui est plus gourmand en mémoire. Hors, il serait possible de ne pas utiliser plus d'espace nécessaire en créant un tableau plus petit et en ré-allouant dynamiquement plus d'espace dans son tableau si nécessaire. Cependant, ceci alourdirait le programme qui n'as pas besoin de respecter de contraintes de stockage.

A Annexes

A.1 Programme personnel

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

int main()
{
    int N, L, count = 0;
    scanf("%d", &N);
    scanf("%d", &L);

    int ** light = (int**) calloc(N, sizeof(int*));
    for (int i = 0; i < N; i++) {
        light[i] = (int*) calloc(2*N-1, sizeof(int));
    }

    for (int i = 0; i < 2*N; i++) {
        for (int j = 0; j < N; j++) {
            char cell[4];
            scanf("%s", cell);
            if(cell[0] == 'C'){
                for(int k = -L+1; k<L; k++){
                    for(int l = -L+1; l<L; l++){
                        if(i+k >= 0 && i+k < N && j+l >= 0 && j+l < N){
                            light[i+k][j+l] = 1;
                        }
                    }
                }
            }
        }
    }

    for (int i = 0; i < N; i++) {
        for (int j = 0; j < N; j++) {
            count += (light[i][j] == 0)?1:0;
        }
    }

    printf("%d\n",count);

    for(int i = 0; i < N; i++){
        free(light[i]);
        light[i] = NULL;
    }
    free(light);
    light = NULL;

    return 0;
}
```

A.2 Programme de Amnesix

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

struct coord
{
    int l;
    int c;
};

#define max(a, b) (a > b ? a : b)
#define dist(c1, c2) (max(abs(c1.l-c2.l), abs(c1.c-c2.c)))

// Retourne 1 si la case est assez    loingne    de toutes les bougies
int isZero(struct coord candles[], int n, struct coord c, int light)
{
    for (int i=0; i<n; i++)
        if (dist(candles[i], c) < light)
            return 0;
    return 1;
}

int main()
{
    int N;
    scanf("%d", &N);
    int Light;
    scanf("%d", &Light);
    // Il y a au maximum une bougie pas cases
    struct coord *candles = (struct coord *)malloc(N * N * sizeof(struct coord));
    int n = 0;
    for (int i = 0; i < N; i++) {
        for (int j = 0; j < N; j++) {
            char cell[4];
            scanf("%s", cell);
            if (cell[0] == 'C')
            {
                candles[n].l = i;
                candles[n].c = j;
                n++;
            }
        }
    }

    // Recherche le nombre de case non    claires    .
    int res = 0;
    for (int i=0; i<N; i++)
        for (int j=0; j<N; j++)
        {
            struct coord c = {i, j};
            res += isZero(candles, n, c, Light);
        }
    printf("%d\n", res);

    free(candles);

    return 0;
}
```