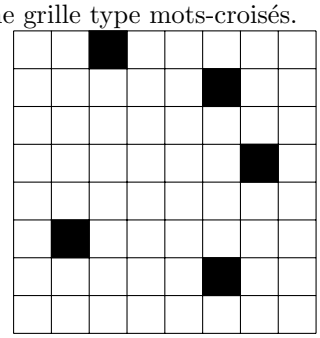
Université Savoie Mont Blanc,
UFR SceM
INFO504, L3 INFO

September 23, 2019

Problème 2, Plus grand rectangle

(Tableaux bi-dimensionnels, boucles, structures, optimisation)

On cherche un algorithme pour trouver le plus grand rectangle “blanc” que l'on peut trouver dans une grille type mots-croisés.



Question 1. Structure de données

Les tableaux à deux dimensions (ou plus) sont possibles, avec une syntaxe `T[i][j]` pour accéder à la i-ème ligne et j-ème colonne. Attention cependant, en mémoire les cases mémoires sont placées de manière contigüe, comme un tableau mono-dimensionnel. Simplement, comme le nombre de colonnes est connu, les indices de lignes sautent en mémoire toutes les cases d'une ligne d'un coup. On peut vérifier ainsi:

```
// 16 lignes et 16 colonnes
#define M 16
#define N 16
typedef char Grille[M][N]; // chaque ligne fait 16 octets
Grille T;                  // 16*16 variables de type char créées.
printf("%p\n", &T[0][0] ); // une adresse A
printf("%p\n", &T[0][1] ); // l'adresse A+1
printf("%p\n", &T[0][2] ); // l'adresse A+2
printf("%p\n", &T[1][0] ); // l'adresse A+16
printf("%p\n", &T[1][1] ); // l'adresse A+17
printf("%p\n", &T[1][2] ); // l'adresse A+18
printf("%p\n", &T[2][0] ); // l'adresse A+32
printf("%p\n", &T[1][N] ); // l'adresse A+32 aussi !!
```

Par convention, on peut mettre `' '` pour vide et `'*'` pour plein.

Question 2. Rectangle vide

Ecrire une fonction qui retourne vrai si le rectangle spécifié est vide:

```
// 0 <= i1 <= i2 < M,  0 <= j1 <= j2 < N,
int est_vide( Grille T, int i1, int j1, int i2, int j2 );
```

Question 3. Type Rectangle

C'est pénible de toujours manipuler 4 paramètres. On propose donc de faire une structure `Rectangle` ainsi:

```
typedef struct {
  int i1, j1, i2, j2;
} Rectangle;
```

Réécrire la fonction précédente avec maintenant le prototype:

```
int est_vide( Grille T, Rectangle* R );
```

Faut-il passer le rectangle par valeur ou par adresse ?

Ecrivez aussi une fonction qui calcule l'aire d'un rectangle.

```
int aire( Rectangle* R );
```

Question 4. Plus grand rectangle à une position donnée

Proposez maintenant une fonction qui retourne un rectangle de coin inférieur (i, j) de plus grande aire et qui ne contient aucune case noire.

```
Rectangle plus_grand_rectangle_ij( Grille T, int i, int j );
```

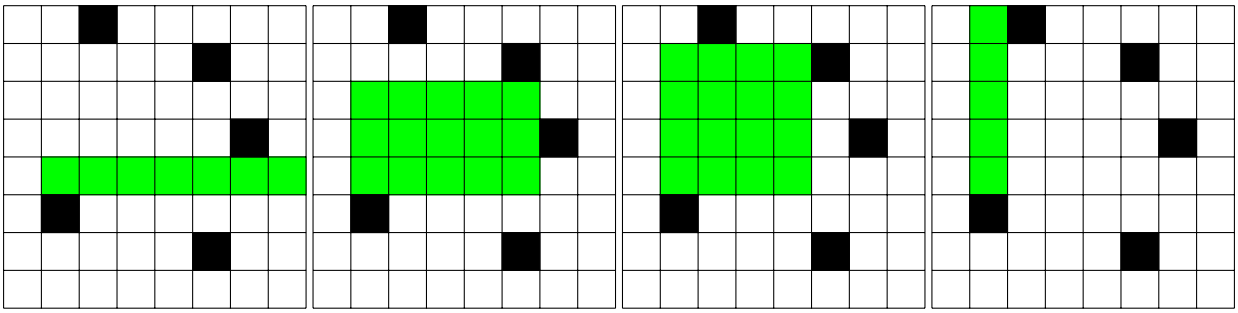
Question 5. Plus grand rectangle

Ecrire enfin une fonction qui retourne un rectangle de plus grande aire et qui ne contient aucune case noire.

```
Rectangle plus_grand_rectangle( Grille T );
```

Remarques : Est-ce une fonction efficace ? Si on double M et N, est-ce que le programme s'exécutera 1x, 2x, 4x, 8x, 16x, ..., plus lentement ?

Question 6. Plus grand rectangle à une position donnée, avec décroissance



On constate que, à (i, j) fixé, plus il y a de lignes dans le rectangle, moins il peut y avoir de colonnes. On peut donc réécrire la fonction `plus_grand_rectangle_ij` en utilisant la colonne la plus lointaine trouvée à la ligne précédente.

```
Rectangle plus_grand_rectangle_ij_v2( Grille T, int i, int j );
```

Question 7. Test d'un rectangle vide

Chaque test de rectangle vide est assez cher (temps proportionnel à la taille du rectangle testé, dans le pire cas). On peut faire un précalcul pour éviter de calculer complètement l'aire. Il y a différentes façons de faire. On propose ici d'utiliser la formule de Green de calcul de l'aire.

Cette formule induit le précalcul d'un tableau A de taille `M * (N+1)`. Ensuite chaque appel à `est_vide_green` prendra un temps proportionnel au nombre de lignes du rectangle seulement.

Question 8. Meilleur algorithme possible

Réfléchissez (ou cherchez sur le net) le meilleur algorithme possible pour calculer le plus grand rectangle.