Programmation en C et structures de données

guillaume.revy@univ-perp.fr

CC1, Jeudi 26/10/2023 de 13h45 à 17h15

Ce programme est à réaliser seul, sous environnement GNU/Linux. Il doit être rendu sur Moodle, sous forme d'un unique fichier `<nom_prenom.c>`. Seuls les documents disponibles sur Moodle sont autorisés. Tout autre document (cours, exercices, ...), ou autre matériel (téléphone portable, ...) est interdit.

Conseils :
- Il est à noter que la qualité primera sur la quantité : un programme court, clair, bien conçu et qui compile sera mieux noté qu'un gros programme mal conçu, incompréhensible, voire qui ne compile pas. Utilisez des noms de variables explicites, indentez votre programme, et n'hésitez pas à le commenter.
- Lisez tout le sujet avant de commencer.
- Veuillez respecter les noms de fonctions et structures proposées dans le sujet.

Flood It est un jeu qui consiste en une grille de taille $N \times N$, où chaque cellule à une couleur choisie aléatoirement parmi C couleurs différentes. Dans le jeu original, $N \in \{14, 21, 28\}$ et $C = 6$. À chaque tour de jeu, le joueur modifie la couleur de la cellule en haut à gauche, de la couleur C_1 vers la couleur C_2. Cette modification entraîne également la modification de la couleur de toutes les cellules de la région adjacente, vers la couleur C_2. La région adjacente est la région composée des cellules de couleur C_1 (couleur de la case en haut à gauche), toutes adjacentes à la cellule en haut à gauche ou à une cellule elle-même adjacente à la cellule en haut à gauche. Ceci est illustré sur la Figure 1, où la région adjacente de chaque grille est hachurée. Le but est de faire en sorte que toutes les cellules soient de la même couleur en un nombre minimum de coups.

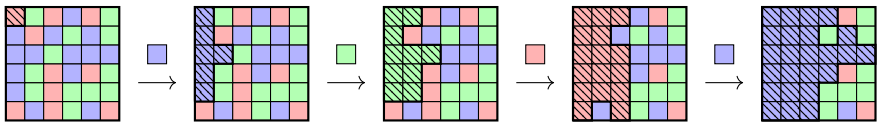

FIGURE 1 – Exemple de partie de Flood It pour $(N, C) = (6, 3)$.

L'objectif du TP est d'écrire un programme qui résoud une grille de Flood It.

1. Création et affichage de la grille

Dans ce TP, une grille est vue comme un tableau 2D de cellules. Elle est caractérisée par une taille N et un nombre de couleurs $C \leq 6$. *Pour le tableau 2D, vous utiliserez :*
- *soit des pointeurs et de l'allocation dynamique,*
- *soit un tableau 2D statique de taille supérieure à $N \times N$ (ie. 30×30), dans lequel vous stockerez la grille.*

Dans la suite de l'énoncé, nous considérerons la version SANS allocation dynamique.
Ensuite dans cette grille, une cellule est caractérisée par une couleur dans $\{0, \cdots, C-1\}$ et un entier dans $\{0, 1\}$ indiquant si cette cellule a déjà été manipulée (utile lors du parcours de la grille).

► **1.** Définir une structure `cellule` représentant une cellule de la grille et (au moins) composée des champs `couleur` et `traitee`, ainsi qu'une structure `grille` représentant la grille elle-même.

► **2.** Écrire une fonction `remplir_grille` qui, étant données la taille N et le nombre de couleur C, initialise une grille de manière aléatoire.

► **3.** Écrire une fonction `afficher_grille` qui affiche la grille.

Pour afficher des caractères colorés dans le terminal, vous pouvez utiliser les lignes de commandes suivantes.

```
printf("\x1b[30mNOIR\x1b[0m");
printf("\x1b[31mROUGE\x1b[0m");
printf("\x1b[32mVERT\x1b[0m");
printf("\x1b[33mJAUNE\x1b[0m");
```

```
printf("\x1b[34mBLEU\x1b[0m");
printf("\x1b[35mMAGENTA\x1b[0m");
printf("\x1b[36mCYAN\x1b[0m");
printf("\x1b[37mBLANC\x1b[0m");
```

2. Flood It en mode interactif

Maintenant, nous allons compléter le programme pour permettre à l'utilisateur de modifier la couleur de la case en haut à gauche, et de tenter de remplir la grille d'une seule couleur en un nombre limité K de coups. Pour cela, nous allons manipuler la position dans la grille de différentes cellules, sous forme de couples de coordonnées (ℓ, c), la cellule en haut à gauche étant de coordonnées $(0, 0)$. Une position est invalide si au moins une des coordonnées est égale à -1, et elle est valide sinon.

► **1.** Définir une structure `position` pour représenter une position sur la grille, c'est-à-dire, les coordonnées entières d'une cellule sur la grille.

Étant donné que nous travaillons sur un tableau statique de taille 30×30, nous aurons besoin d'un tableau de taille $30 \times 30 = 900$ pour stocker toutes les positions possibles. Soit T un tableau de positions de taille 900. Ce tableau contient P positions valides, avec $P \in \{0, \cdots, 300\}$, puis $900 - P$ positions invalides, comme le tableau ci-dessous pour $P = 5$:

$$T = \{\underbrace{(0,0),(0,1),(0,2),(0,3),(1,3)}_{\text{positions valides}}, \underbrace{(-1,-1),\cdots,(-1,-1)}_{\text{positions invalies}}\}.$$

► **2.** Écrire une fonction `initialiser` qui initialise le tableau T avec des positions invalides.

► **3.** Écrire une fonction `ajouter` qui ajoute une nouvelle position valide au tableau T, après les P premières positions valides, et uniquement si cette position n'existe pas déjà dans le tableau. Par exemple :

$$\texttt{ajouter}(T,(2,3)) \quad \to \quad \{(0,0),(0,1),(0,2),(0,3),(1,3),(2,3),(-1,-1),\cdots,(-1,-1)\}.$$

► **4.** Écrire une fonction `retirer` qui retire et retourne la position d'indice 0 du tableau T, et décale toutes les autres d'un indice vers la gauche (en ajoutant une position invalide en fin). Par exemple :

$$\texttt{retirer}(T) \quad \to \quad (0,0), \quad \{(0,1),(0,2),(0,3),(1,3),(2,3),(-1,-1),\cdots,(-1,-1)\}.$$

► **5.** Écrire une fonction `afficher` qui affiche les positions valides présentes dans le tableau T.

Nous allons enfin compléter le programme pour modifier la couleur de la cellule en haut à gauche, ainsi que celle des cellules de la région adjacente. Pour cela, nous allons écrire une fonction itérative qui utilise un tableau comme le tableau T définit précédemment. Ensuite :

— elle ajoute la position de la cellule en haut à gauche dans ce tableau,
— puis tant que ce tableau n'est pas vide, elle retire le premier élément, modifie la couleur de la cellule à cette position, puis ajoute la position de ses 4 cellules adjacentes.

Si vous n'arrivez vraiment pas à écrire une version itérative de la fonction, vous pourrez utiliser une verison récursive.

► **6.** Écrire une fonction `modifier_couleur` qui modifie la couleur de la cellule en haut à gauche et celle de la région adjacente avec une couleur passée en paramètre.

► **7.** Écrire une fonction `taille_region_adjacente` qui détermine la taille (en nombre de cellules) de la région adjacente.

► **8.** Écrire enfin une fonction `jouer_floodit` qui prend en paramètre une grille et un nombre de coups maximum, et qui permet de jouer au Flood It.

► **9.** Illustrer l'utilisation de ces fonctions dans un programme principal.

Vous pouvez tester votre programme en utilisant la grille de la Figure 1. Pour l'utiliser, au lieu d'utiliser la fonction `remplir_grille`, vous pourrez utiliser la fonction suivante.

```
struct grille
grille_figure1(void)
{
  struct grille G;
  G.N = 6; G.C = 3;

  G.tab[0][0].couleur = 0; G.tab[0][0].traitee = 0;
  G.tab[0][1].couleur = 2; G.tab[0][1].traitee = 0;
  G.tab[0][2].couleur = 0; G.tab[0][2].traitee = 0;
  G.tab[0][3].couleur = 1; G.tab[0][3].traitee = 0;
  G.tab[0][4].couleur = 0; G.tab[0][4].traitee = 0;
  G.tab[0][5].couleur = 2; G.tab[0][5].traitee = 0;

  G.tab[1][0].couleur = 1; G.tab[1][0].traitee = 0;
  G.tab[1][1].couleur = 0; G.tab[1][1].traitee = 0;
```

```
    G.tab[1][2].couleur = 1; G.tab[1][2].traitee = 0;
    G.tab[1][3].couleur = 2; G.tab[1][3].traitee = 0;
    G.tab[1][4].couleur = 1; G.tab[1][4].traitee = 0;
    G.tab[1][5].couleur = 2; G.tab[1][5].traitee = 0;

    G.tab[2][0].couleur = 1; G.tab[2][0].traitee = 0;
    G.tab[2][1].couleur = 1; G.tab[2][1].traitee = 0;
    G.tab[2][2].couleur = 2; G.tab[2][2].traitee = 0;
    G.tab[2][3].couleur = 1; G.tab[2][3].traitee = 0;
    G.tab[2][4].couleur = 1; G.tab[2][4].traitee = 0;
    G.tab[2][5].couleur = 1; G.tab[2][5].traitee = 0;

    G.tab[3][0].couleur = 1; G.tab[3][0].traitee = 0;
    G.tab[3][1].couleur = 2; G.tab[3][1].traitee = 0;
    G.tab[3][2].couleur = 0; G.tab[3][2].traitee = 0;
    G.tab[3][3].couleur = 1; G.tab[3][3].traitee = 0;
    G.tab[3][4].couleur = 0; G.tab[3][4].traitee = 0;
    G.tab[3][5].couleur = 2; G.tab[3][5].traitee = 0;

    G.tab[4][0].couleur = 1; G.tab[4][0].traitee = 0;
    G.tab[4][1].couleur = 2; G.tab[4][1].traitee = 0;
    G.tab[4][2].couleur = 0; G.tab[4][2].traitee = 0;
    G.tab[4][3].couleur = 2; G.tab[4][3].traitee = 0;
    G.tab[4][4].couleur = 2; G.tab[4][4].traitee = 0;
    G.tab[4][5].couleur = 2; G.tab[4][5].traitee = 0;

    G.tab[5][0].couleur = 0; G.tab[5][0].traitee = 0;
    G.tab[5][1].couleur = 1; G.tab[5][1].traitee = 0;
    G.tab[5][2].couleur = 0; G.tab[5][2].traitee = 0;
    G.tab[5][3].couleur = 2; G.tab[5][3].traitee = 0;
    G.tab[5][4].couleur = 1; G.tab[5][4].traitee = 0;
    G.tab[5][5].couleur = 0; G.tab[5][5].traitee = 0;

    return G;
}
```

3. Résolution d'une grille de Flood It

Nous voulons maintenant résoudre automatiquement une grille donnée de Flood It. Pour cela, nous allons utiliser un algorithme glouton qui va de manière itérative choisir une couleur pour la cellule en haut à gauche, jusqu'à remplir la grille avec la même couleur. Pour choisir la couleur à appliquer à la cellule en haut à gauche, deux approches peuvent être utilisées :

1. choisir la couleur qui maximise la taille de la région adjacente (après modification de la couleur),
2. ou choisir la couleur qui apparait le plus sur la frontière de cette région.

Par exemple, si nous considérons la deuxième grille de la Figure 1, la frontière de la région adjacente est composée de 4 cellules vertes et 2 cellules rouges. Les positions (ℓ, c) de ces cellules sont données ci-dessous (la cellule en haut à gauche étant la cellule $(0, 0)$).

$$\{\underbrace{(0,1),(1,1),(2,2),(3,1),(4,1),(5,0)}_{\text{frontière}},(-1,-1),\cdots,(-1,-1)\}.$$

Ensuite si l'algorithme réussit à résoudre la grille, il pourra également renvoyer la liste des couleurs successivement appliquées. Pour simplifier, cette liste sera représentée par un tableau M de taille $30 \times 30 = 900$, où la valeur $M[i]$ indique la i-ème couleur choisie.

► **1.** Écrire une fonction `algo_glouton_approche1` qui étant donnée une grille de Flood It et un nombre de coups maximum, tente de résoudre la grille en utilisant la première approche, et retourne un entier indiquant si la grille a été résolue ($= 1$) ou non ($= 0$). En cas de succès, elle retourne également la suite de couleurs choisies.

► **2.** Écrire une fonction `calculer_frontiere` qui détermine et retourne la frontière de la région adjacente d'une grille passée en paramètre. Cette frontière pourra être représentée sous forme d'un tableau de positions, comme le tableau T présenté précédemment.

► **3.** Écrire une fonction `algo_glouton_approche2` dont la spécification est identique à celle de la fonction `algo_glouton_approche1` et qui implante la seconde approche.

► **4.** Modifier le programme principal pour illustrer l'utilisation de ces nouvelles fonctions de résolution.