Projet Algo 5 - Partie 2

Licence Informatique - 2nde année

Année 2022-2023

La seconde partie de ce projet va consister à créer un labyrinthe et à l'afficher sous forme d'une image au format *SVG*. Vous compléterez le fichier source fourni avec cet énoncé et vous rendrez une archive contenant cette partie du projet sous la forme exo2_XXX, où XXX sera remplacé par votre nom.

Structure de données

La représentation d'un labyrinthe va se faire ici par l'intermédiaire d'une grille, dont chaque case représentera une pièce. Chaque pièce sera initialement entourée de 4 murs, 1 ou 2 d'entre-eux étant ensuite supprimés lors de la construction du labyrinthe. La structure de données qui vous est fournie est la suivante :

```
struct labyrinthe {
  int largeur, hauteur;// nombre de colonnes et de lignes du labyrinthe
  int *cell; // cellules du labyrinthe
  bool *mursV;// état des murs verticaux des cellules
  bool *mursH;// état des murs horizontaux des cellules
};
```

avec :

- `largeur` et `hauteur`, les dimensions de la grille support du labyrinthe ;
- `cell` un pointeur vers la zone mémoire permettant de représenter toutes les cases de la grille (donc de taille `largeur x hauteur`). Ces cases seront stockées de manière adjacente pour une ligne et les lignes seront stockées les unes après les autres (voir schéma de la figure 1) ;
- `mursV` une zone mémoire permettant de préciser si un mur vertical est présent ou pas dans la grille. Pour une grille de dimensions `largeur x hauteur`, il y aura donc `largeur+1` murs verticaux pour chaque ligne (voir schéma de la figure 1) ;
- `mursH` une zone mémoire permettant de préciser si un mur horizontal est présent ou pas dans la grille. Pour une grille de dimensions `largeur x hauteur`, il y aura donc `hauteur+1` murs horizontaux pour chaque colonne (voir schéma de la figure 1).

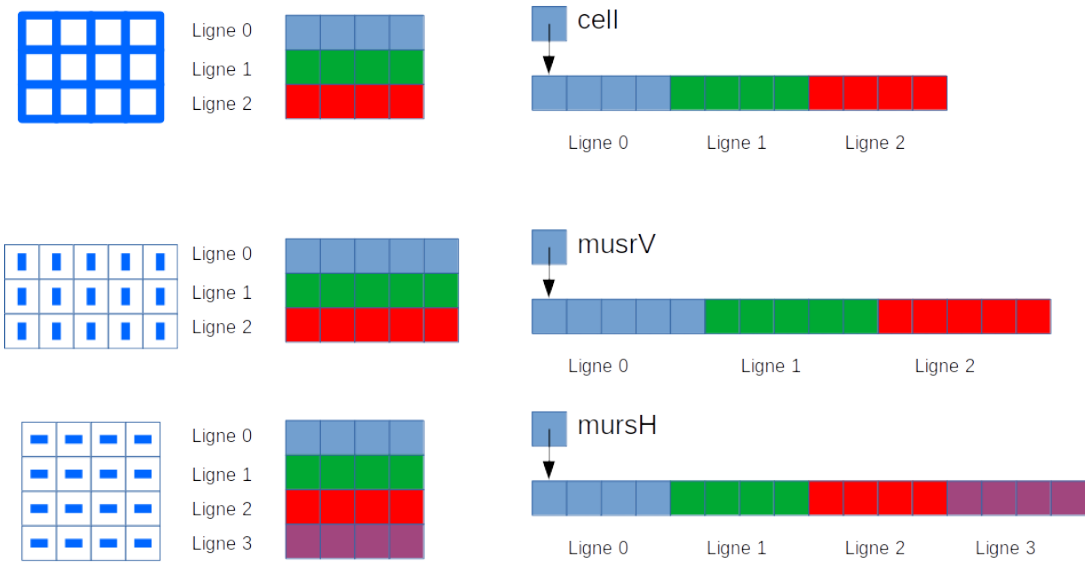

FIGURE 1 – Aperçu graphique des différents champs dynamiques de la structure `labyrinthe`

Initialisations et dessin

Dans cette première partie, vous allez mettre en place les fonctions permettant d'initialiser la grille support au labyrinthe, de dessiner cette grille au format *SVG* et enfin d'effacer la mémoire utilisée par la grille.

Initialisation de la grille

Ecrire le code de la fonction suivante :

```
void initialiserLabyrinthe(labyrinthe &laby, int largeur, int hauteur);
```

dont l'objectif est d'initialiser les différentes grilles qui permettront de créer un labyrinthe. Ses paramètres sont :

- `laby` une variable de type `labyrinthe` qui sera initialisée par la fonction ;
- `largeur` le nombre de colonnes de la grille support ;
- `hauteur` le nombre de lignes de la grille support.

On précise les points suivants :

- la zone mémoire `cell`, de taille `largeur x hauteur` devra être créée et chacune de ses cases initialisée avec le numéro de la case, variant de 0 à $(largeur \times hauteur) - 1$ (voir figure 2) ;
- la zone mémoire `mursV`, de taille `(largeur+1) x hauteur` devra être créée et chacune de ses cases initialisée avec la valeur `true`, signifiant qu'un mur vertical est présent à cet endroit ;
- la zone mémoire `mursH`, de taille `largeur x (hauteur+1)` devra être créée et chacune de ses cases initialisée avec la valeur `true`, signifiant qu'un mur horizontal est présent à cet endroit.

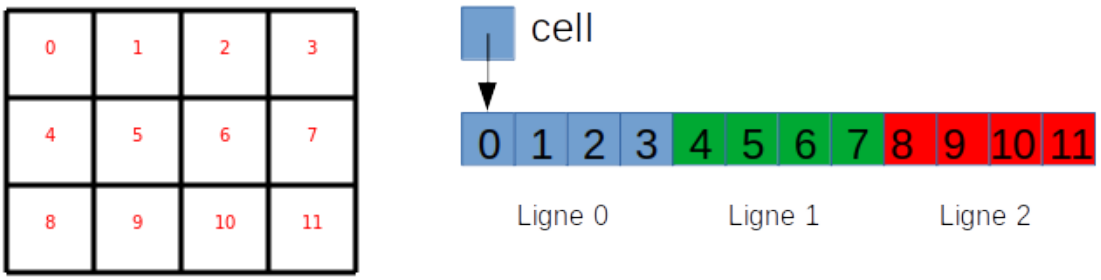

FIGURE 2 – Aperçu de la numérotation des cellules de la structure `labyrinthe`

Suppression de la grille

Ecrire le code de la fonction suivante :

```
void effacerLabyrinthe(labyrinthe &laby);
```

dont l'objectif est d'effacer les zones mémoire utilisées pour représenter le labyrinthe passé en paramètre.

Dessin de la grille

1. Ecrire le code de la fonction suivante :

   ```
   void dessinerLabyrinthe(const labyrinthe &laby, const string &nomFichier);
   ```

   dont l'objectif est de dessiner le labyrinthe passé en premier paramètre dans le fichier, au format *SVG*, dont le nom est passé en second paramètre. Vous vous appuyerez sur le travail réalisé dans la première partie du projet, en tenant compte des remarques suivantes :
   - un mur vertical ne doit être dessiné que si la valeur correspondante dans le champ `mursV` est à `true` ;
   - un mur horizontal ne doit être dessiné que si la valeur correspondante dans le champ `mursH` est à `true` ;

   Dans cette première version, tous les murs étant présents, vous devez obtenir une grille ...

2. Complétez le code de cette fonction, de manière à ce qu'elle affiche le numéro de la case au centre de chaque pièce. Vous utiliserez la fonction `texte` fournie. On précise que pour convertir un nombre entier (le numéro d'une case) en une chaîne de caractères, vous pourrez utiliser la fonction `intToString` qui vous est fournie.

Génération du labyrinthe

Dans cette partie, vous allez implanter un algorithme de génération de labyrinthes dits *parfaits*. Dans ce type de labyrinthe, chaque cellule est reliée à toutes les autres de manière unique. Ils ne contiendront donc par exemple pas de boucles, ni de cellules inaccessibles.

On supposera ici que l'on génère un labyrinthe à partir d'une grille rectangulaire, de taille `largeur x hauteur` et que chaque case est initialement entourée d'un mur sur chacun de ses côtés. Pour pouvoir créer des cheminements dans la grille, il faut ôter des murs. Pour obtenir un labyrinthe parfait, il faut ôter exactement `(largeur x hauteur) - 1` murs [1]. La manière d'ôter ces murs dépend de l'approche de construction choisie.

Algorithme

Vous allez construire le labyrinthe en suivant une méthode dite de *fusion aléatoire de chemins*, qui suppose que chaque case de la grille de départ est bien identifiée par un numéro unique. Elle fonctionne alors de manière itérative, en créant des chemins aléatoires dans la grille et en les fusionnant lorsqu'un mur qui les sépare est supprimé :

- à chaque itération, on va choisir une cellule aléatoirement dans la grille ;
- on va ensuite choisir au hasard l'un des quatres murs de cette cellule ;
- si ce mur existe, on va regarder si ce mur peut être supprimé :
  - si les identifiants des deux cellules de chaque côté du mur sont différents, cela signifie que les deux cellules sont sur des chemins différents et on peut donc supprimer le mur. On va ensuite renommer les identifiants du second chemin avec l'identifiant du premier chemin, puisque la fusion vient de créer un chemin plus grand ;
  - si les identifiants sont les mêmes, cela signifie que les cellules appartiennent au même chemin et qu'on ne peut pas supprimer le mur, au risque de créer un trou dans la grille.
- on poursuit ce processus jusqu'à ce que `(largeur x hauteur) - 1` murs aient été ôtés.

Les différentes étapes de cet algorithme sont illustrées sur la figure 3, sur une grille de taille 4×3.

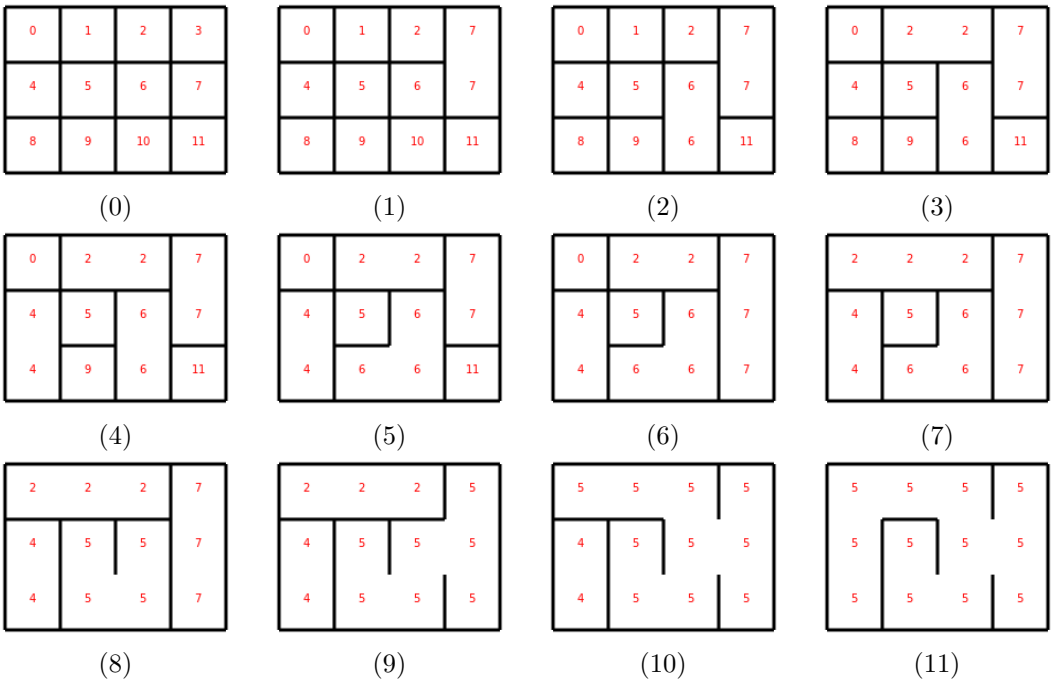

FIGURE 3 – Vue des 11 étapes successives de construction d'un labyrinthe de taille 4×3, à partir de la grille initiale figurant en (0).

1. On ne détaillera pas ici la raison de cette propriété.

0. l'état initial des grilles : tous les murs sont présents et chaque case a un identifiant différent ;
1. la case 7 a été choisie, ainsi que son mur du haut. La case figurant de l'autre côté de ce mur porte le numéro 3. On peut donc ôter le mur et fusionner les 2 cellules, qui porteront toutes les deux le numéro 7 ;
2. la case 6 a été ensuite choisie, ainsi que son mur du bas. La case figurant de l'autre côté de ce mur porte le numéro 10. On peut donc ôter le mur et fusionner les 2 cellules, qui porteront toutes les deux le numéro 6 ;
3. la case 2 a été choisie, ainsi que son mur de gauche. La case figurant de l'autre côté de ce mur porte le numéro 1. On peut donc ôter le mur et fusionner les 2 cellules, qui porteront toutes les deux le numéro 2 ;
4. la case 4 a été choisie, ainsi que son mur du bas. La case figurant de l'autre côté de ce mur porte le numéro 8. On peut donc ôter le mur et fusionner les 2 cellules, qui porteront toutes les deux le numéro 4 ;
5. la case numérotée initialement 10 (son identifiant est désormais 6) a été choisie, ainsi que son mur de gauche. La case figurant de l'autre côté de ce mur porte le numéro 9. On peut donc ôter le mur et fusionner les 2 cellules, qui porteront désormais l'identifiant 6 ;
6. etc.

Application

Codez cet algorithme dans la fonction suivante :

```
void genererLabyrinthe(labyrinthe &laby);
```

On précise que le tirage d'un nombre aléatoire peut être effectué par la fonction `rand()` du langage C, qui retourne un nombre entier compris entre 0 et une valeur maximale nommée RAND_MAX. Obtenir un nombre entier v compris dans un intervalle $[0, N-1]$ peut alors être obtenu avec l'opérateur % (modulo) de la manière suivante : `v = rand() % N`.

Lorsque le labyrinthe aura été généré, vous le sauvegarderez sous la forme d'un dessin au format *SVG*.