Projet:
Génération aléatoire de labyrinthe

Algorithmique des arbres

2019-2020

1 Introduction

Le but du projet est d'écrire un programme pour créer des labyrinthes aléatoires. Les sections 2 à 5 décrivent le projet minimal attendu, et la section 6 précise des améliorations à apporter au programme.

2 Nomenclature: labyrinthe, cellules, murs, etc.

Dans cette section, on fixe les termes nécessaires à la compréhension du sujet. Ils sont expliqués par un exemple qui suit les définitions.

Un labyrinthe $n \times m$ est un tableau de n lignes et m colonnes.

Chaque case du tableau est appelée *cellule* et chaque cellule a pour *coordonnées* (i, j) si elle appartient à la $i^{\text{ème}}$ ligne et la $j^{\text{ème}}$ colonne.

Chaque cellule a quatre cellules voisines, sauf les cellules des bords qui en ont trois et les cellules des coins qui n'en ont que deux.

Deux cellules voisines sont soit connectées, soit séparées par un mur.

L'entrée du labyrinthe se fait par la cellule en haut à gauche (c'est-à-dire $(0, 0)$) et la sortie par la cellule en bas à droite (c'est-à-dire $(n - 1, m - 1)$).

On dit qu'il existe un chemin qui relie deux cellules, si on peut atteindre l'une depuis l'autre en passant de proche en proche par des cellules connectées.

Un labyrinthe est *valide* s'il y a un chemin allant de l'entrée à la sortie.

(0,0)	(0,1)	(0,2)	(0,3)	(0,4)	(0,5)	(0,6)	(0,7)
(1,0)	(1,1)	(1,2)	(1,3)	(1,4)	(1,5)	(1,6)	(1,7)
(2,0)	(2,1)	(2,2)	(2,3)	(2,4)	(2,5)	(2,6)	(2,7)
(3,0)	(3,1)	(3,2)	(3,3)	(3,4)	(3,5)	(3,6)	(3,7)
(4,0)	(4,1)	(4,2)	(4,3)	(4,4)	(4,5)	(4,6)	(4,7)
(5,0)	(5,1)	(5,2)	(5,3)	(5,4)	(5,5)	(5,6)	(5,7)

Figure 1: Exemple de labyrinthe valide 6×8

La Figure 1 présente un labyrinthe 6×8.

- Dans chaque cellule sont indiquées ses coordonnées.
- En rouge sont mises en évidence les cellules voisines de la cellule (4,6), en bleu les voisines de (4,0).
- L'entrée est la cellule (0,0) et la sortie est la cellule (5,7).
- Il y a (entre autres) un mur entre les cellules (1,0) et (1,1), ces deux cellules sont déconnectées. Idem pour les cellules (4,2) et (4,3).
- Pour des raisons esthétiques, on a dessiné aussi les murs extérieurs du labyrinthe (sauf un pour l'entrée et la sortie).

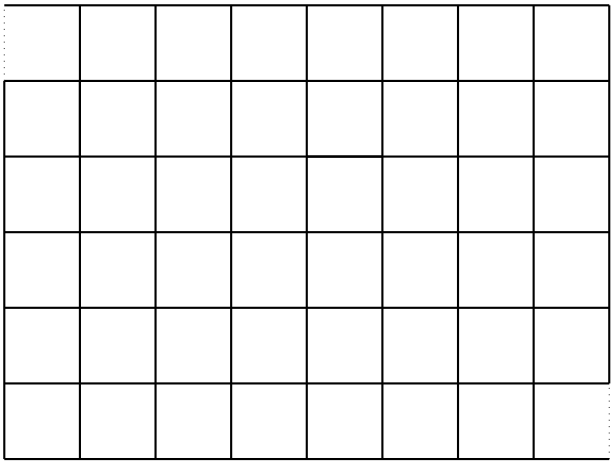
- Il y a un chemin reliant les cellules (0,0) et (5,7), celui allant verticalement jusqu'à la cellule (5,0) puis horizontalement jusqu'à la sortie. Ce labyrinthe est donc valide.
  En revanche, il n'y a pas de chemin reliant les cellules (5,0) et (1,7).

3 Processus de génération

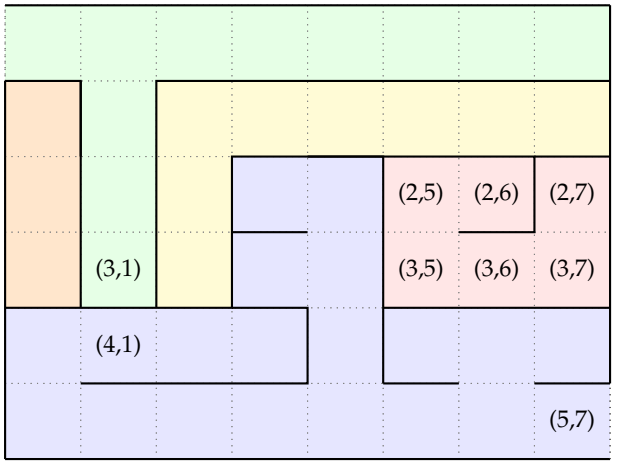
On utilisera le processus suivant pour créer aléatoirement un labyrinthe valide.

1) On part d'un labyrinthe où toutes les cellules voisines sont séparées par des murs.
2) On supprime des murs aléatoirement. On utilisera les fonctions `srand(unsigned int)` et `rand(void)` de `stdlib.h` pour tirer des nombres aléatoirement.
3) On s'arrête quand le labyrinthe est valide.

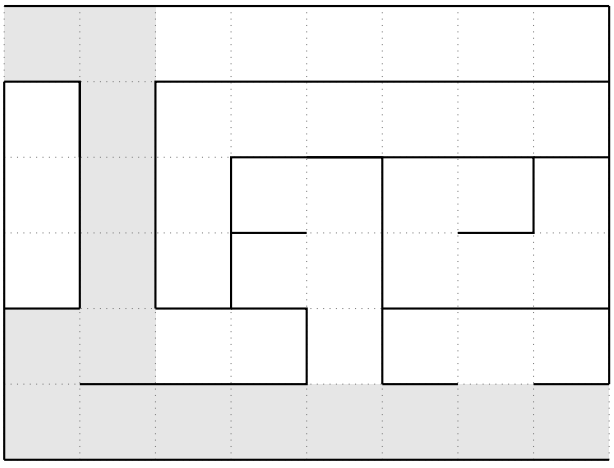
La figure 2 donne un exemple.



(a) Un labyrinthe 6×8 après initialisation



(b) Un labyrinthe possible après plusieurs suppressions de murs. Les cellules d'une même couleur sont reliées entre elles. Les coordonnées de quelques cellules sont indiquées pour aider la compréhension dans la suite.



(c) Après la suppression d'un mur supplémentaire : le labyrinthe est valide et l'algorithme s'arrête. Le chemin reliant l'entrée et la sortie est mis en évidence

Figure 2: Trois labyrinthes 6×8 à différentes étapes du processus de génération.

4 Structures de données

On demande d'utiliser l'algorithme *Unir et trouver* dans le processus de génération, avec les améliorations du rang et de la compression des chemins.

On maintiendra l'invariant suivant : deux cellules appartiennent à la même classe s'il y a un chemin qui les relie dans le labyrinthe.

Ainsi, l'algorithme de la section 3 s'arrête quand les cellules $(0, 0)$ et $(n-1, m-1)$ sont dans la même classe.

Sur la Figure 2b, les cellules de la même couleur appartiennent à la même classe pour *Unir et trouver*.

Quand le mur entre les cellules $(3, 1)$ et $(4, 1)$ est supprimé (pour obtenir le labyrinthe de la Figure 2c), il faudra fusionner les classes de ces deux cellules, et les cellules (0,0) et (5,7) appartiendront à la même classe pour *Unir et Trouver*: c'est la condition d'arrêt du processus de génération.

Dans le cours, l'algorithme *Unir et trouver* permet de manipuler une partition d'un ensemble d'entiers, mais nous avons besoin ici de partitionner un ensemble de couples d'entiers.

La structure utilisée par l'algorithme *Unir et trouver* est donc une forêt représentée par deux tableaux de tableaux `pere` et `rang`;

- chaque arbre représente une classe de cellules connexes;
- chaque noeud porte les coordonnées d'une cellule du labyrinthe.
- `pere[i][j]` contient les coordonnées du père de la cellule (i, j),
- si (i, j) est le représentant de sa classe, `rang[i][j]` contient l'estimation de la hauteur.

La Figure 3 donne un arbre possible pour la classe rouge du labyrinthe de la Figure 2b. (Les arbres des autres classes n'ont pas été représentés et l'algorithme du rang n'a pas été utilisé.)

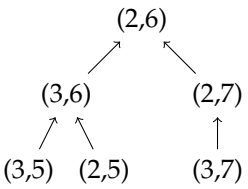


Figure 3: Possible arbre de la classe rouge (voir figure 2b)

5 Interface utilisateur

Le programme, de nom `labyrinthe` affiche les étapes de la création d'un labyrinthe (par défaut de taille 6×8). On effectuera un affichage après chaque suppression de mur, jusqu'à afficher un labyrinthe valide.
Il devra pouvoir se lancer dans deux modes différents.

1) **Mode graphique**
   Par défaut, le programme doit lancer une interface graphique écrite avec libMLV.
2) **Mode texte**
   En lançant le programme avec `./labyrinthe --mode=texte`, le programme se lancera en mode texte (sans interface graphique).
   Un exemple d'affichage de labyrinthe après toutes les étapes de création, est donné à la Figure 4.

```
+--+--+--+--+--+--+--+--+
                        |
+--+  +--+--+--+--+--+--+
|  |  |                 |
+  +  +  +--+--+--+--+--+
|  |  |  |     |     |  |
+  +  +  +--+  +  +--+  +
|  |  |  |     |        |
+--+  +--+--+  +--+--+--+
|           |  |        |
+  +--+--+--+  +--+  +--+
|
+--+--+--+--+--+--+--+--+
```

Figure 4: Exemple d'affichage en mode texte du labyrinthe de la Figure 2b

6 Améliorations

Le projet minimal décrit ci-dessus n'est pas suffisant pour obtenir u.ne note élevée. Il est attendu que les étudiants implémentent plusieurs améliorations.

6.1 Options

(1) **Taille du labyrinthe** Permettre à l'utilisateur de choisir la taille du labyrinthe souhaité (hauteur et largeur). Cela devra se faire à travers l'option : `--taille=6x8`.

(2) **Fixer la graine** Ajouter une option `--graine=X`, où *X* est un entier, qui permet de fixer la graine d'aléa (c'est-à-dire l'argument à donner à la fonction `srand(unsigned int)` au début du programme). Ainsi, deux appels à votre programme avec une même graine devront créer le même labyrinthe.

(3) **Gestion des étapes** Par défaut, le programme doit attendre entre chaque suppression de mur une entrée clavier de l'utilisateur (n'importe laquelle) avant de continuer.
Ajouter une option qui permet de gérer le tempo du programme.
- avec `--attente=0`, affiche sans attente le labyrinthe final;
- avec l'option `--attente=X` où `X` est une chaîne de caractères formant un nombre, le programme attend `X` millisecondes entre chaque suppression.

Avec ces trois options implémentées vous êtes noté sur 12.

6.2 Fonctionnalités

D'autres améliorations sont envisageables

(4) **Chemin unique** (option : `--unique`) Vous avez du remarquer que les labyrinthes obtenus ne ressemblent pas à celui de la Figure 2b. Pour obtenir des résultats plus esthétiques, empêcher la suppression d'un mur s'il sépare deux cellules appartenant déjà à la même classe (par exemple, le mur séparant les cellules bleues (2,6) et (3,6) du labyrinthe de la Figure 2b).

(5) **Accessibilité de toutes les cellules** (option `--acces`) Pour augmenter la difficulté du labyrinthe, faire en sorte que l'algorithme ne s'arrête que lorsque toutes les cellules appartiennent à la même classe. (Si l'implantation n'est pas adaptée, ceci a un impact important sur les performances. Trouvez une condition d'arrêt qui ne demande pas de tester systématiquement l'égalité des classes de toutes les cellules dans *Unir et trouver*..)

(6) **Meilleur affichage en mode texte** Améliorer l'affichage du labyrinthe en mode texte en utilisant des chaînes UTF8 pour dessiner les jointures des murs. Au lieu de '|', '-' et '+', on utilisera ' ', '╴', '╶', '└', '┴', '┼', etc. Ces chaînes sont regroupées dans le tableau `intersections` (voir le fichier `structures.h` pour plus de détails):

```
char* intersections[2][2][2][2] =
  { {{{" "," "},{"╴","╶"}},{{"╵","└"},{"┘","┴"}}},
    {{{"╷","┌"},{"┐","┬"}},{{"│","├"},{"┤","┼"}}} };
```

Le tableau 'intersections' contient les chaînes de caractères pour dessiner des boîtes en mode texte.
Bien qu'ils s'affichent comme des caractères dans le terminal et l'éditeur de texte, ce sont en réalité des chaînes de taille 2.

Chaque chaîne est une croix ┼ à laquelle il peut manquer des branches. Il faut l'utiliser comme suit:
'intersections[bas][haut][gauche][droite]', où 'bas', 'haut', 'gauche', 'droite' sont des booléens qui indiquent si la branche correspondante apparaît.

Par exemple:

┌ : intersections[1][0][0][1]

┤ : intersections[1][1][0][1]

╵ : intersections[0][1][[0][0]

Attention, à cause de la façon dont les variables globales sont gérées en C, les lignes ci-dessus devront être placées au début du fichier contenant vos fonctions d'affichage.

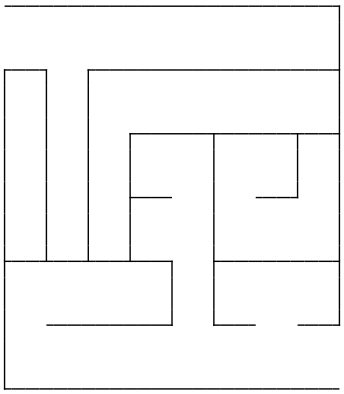


Figure 5: Exemple d'affichage en mode texte utf8 du labyrinthe de la Figure 2b

(7) **Chemin victorieux** (option : `--victor`) affiche le chemin de longueur minimale reliant l'entrée et la sortie. Penser au parcours en largeur ... (Difficile)

6.3 Performance

Établir et analyser les performances de votre programme.
Pour avoir une idée des performances,

- lancer le programme en mode texte ;
- tester votre projet sur le labyrinthe le plus grand possible (par exemple 400×300);
- utiliser la commande `time` pour mesurer précisément le temps d'exécution de votre programme.
- ajouter des options pour activer/désactiver les différentes optimisations que vous proposez.

(8) **Structure de données pour effacement**
Améliorer le tirage aléatoire des murs à supprimer.

- Ne tirer aléatoirement les murs que parmi ceux qui n'ont pas déjà été supprimés.
- On peut remarquer que quand un mur ne peut pas être supprimé à une étape donnée, il ne pourra pas non plus être supprimé dans une étape future. Faire en sorte que votre structure de donnée utilise ce fait.
- En bref, trouver une structure de donnée qui peut faire efficacement l'opération suivante : tirer un élément aléatoire dans la structure, l'enlever de la structure et renvoyer l'élément.

7 Implantation

On définit un type pour les couples d'entiers:

```
typedef struct coordonnees_t {
   int x;
   int y;
 } coordonnees_t;
```

Plusieurs solutions sont possibles pour représenter les cases et les murs du labyrinthe. Nous en proposons deux, d'autres sont envisageables.

7.1 Séparer cases et murs

Définir une structure particulière utilisée par *Unir et trouver*

```
typedef struct unionfind_t {
     coordonnees_t taille;
     coordonnees_t** pere;
     unsigned int** rang;
   } unionfind_t;
```

où `taille.x` est la hauteur du labyrinthe, et `taille.y` sa largeur; `pere` et `rang` sont des zones allouées dynamiquement utilisées comme des tableaux de `taille.x` tableaux de `taille.y` coordonnées ou entiers

. Les murs peuvent être représentés séparément : les murs horizontaux et les murs verticaux.
On utilise alors :

```
typedef struct labyrinthe_t {
  coordonnees_t taille;
  int** murs_hori;
  int** murs_vert;
} labyrinthe_t;
```

`mur_hori[i][j]` vaut 1 si un mur est présent. Prévoir de la place pour les murs extérieurs.

7.2 Utiliser un seul tableau

En remarquant que l'on peut décrire le labyrinthe en une juxtaposition de cases avec un mur est et un mur sud, (plus un grand mur fixe supérieur et un grand mur à gauche comprenant l'entrée), on définit les structures:

```
typedef struct{
  int murEst;
  int murSud;
  coordonnees_t pere;
  int rang;
}case_t;

typedef struct laby{
  coordonnees_t taille;
  case_t **cases; /* attention murs extremes a conserver*/
}laby_t;
```

7.3 Autres

D'autres représentations sont possibles.
Réfléchissez bien avant de programmer et expliquez vos choix dans votre rapport.