MOOC LaTeX : Compte-rendu

Adrien Lafage

6 novembre 2016

Table des matières

I Enoncé :

L'objectif de ce projet est la création aléatoire d'un labyrinthe, ainsi que sa résolution. La méthode de génération du labyrinthe proposée permet de ne créer que des labyrinthes parfaits. Le code que je vais présenter permet la réalisation de labyrinthes aléatoires et fournit la meilleure solution pour le résoudre.

II Génération du Labyrinthe :

Cette partie se décompose en plusieurs étapes, tout d'abord la création d'une matrice de départ puis la génération d'un labyrinthe parfait.

La matrice initiale :

La création de la matrice initiale est gérée par la fonction création. Le principe est le suivant : la matrice est créée ligne par ligne. Si la ligne est paire alors celle-ci sera remplit de -1. Sinon on prend en compte la parité de la colonne : si celle-ci est paire on met -1, sinon on place des entiers dans un ordre croissant avec un pas de 1 en partant de 0. Ainsi les -1 et les nombres positifs s'alterne dans notre ligne.
Et pour finir, chaque liste contenant une ligne, se voit ajoutée dans la matrice finale (qui est une liste de listes).
Nous avons donc notre matrice. Maintenant il va falloir casser certains murs (ici représentés par les -1) afin d'obtenir notre labyrinthe.

L'algorythme de génération du labyrinthe :

C'est dans cette optique, que l'on va créer une liste de tous les murs cassables. En effet au cours de la génération aucun mur qui ne pouvait être cassé auparavant ne le devient. Aussi il suffit de répertorier dans une liste les murs cassables et d'ensuite en choisir au hasard dans

celle-ci. Cet objectif sera remplit par deux fonctions, une parcoura la matrice et retournera la liste finale ; l'autre vérifiera que le mur est cassable.
Nous avons notre liste, à présent nous allons prendre un mur, le casser et propager la plus petite valeur des deux cellules qui lui étaient adjacentes. C'est alors, que l'on utilise les fonctions "brokenWall" et "update".

brokenWall : On retient la valeur maximale et minimale des cellules adjacentes, et si elles ne sont pas égales, le mur prend la valeur minimale, et les coordonnées du murs sont effacées.

update : Cette fonction prend en arguments : la valeur maximale et minimale (énoncées plus tôt) et bien entendu la matrice. Elle va changer les valeurs de toutes les cases possédant la valeur maximale, en la valeur minimale.
Nous pouvons maintenant casser un mur et propager la valeur minimale, il reste à définir combien de fois il faut le faire pour obtenir un labyrinthe parfait. Le plus simple serait de d'analyser la matrice en vérifiant s'il n'y a que des 0 ou des -1. Cependant nous sommes en présece d'un labyrinthe parfait, ainsi il n'y a pour chaque dimension de matrice un nombre précis de mur à retirer pour l'obtenir.

Etude mathématique d'un labyrinthe parfait : Pour créer un labyrinthe parfait, il faut que chaque cellule de la matrice de départ soient reliées par un unique chemin. Et pour parvenir à ce phénomène il y a pour chaque dimension de matrice un nombre unique de murs à casser. Après quelques tests on arrive à la relation suivante :
murs à casser $= largeur * hauteur - 1$.

III Etude de la complexité :

Complexité de la fonction "création" :

Lignes	Affectation
1	1
2	2*hauteur+1
3	2*hauteur+1
4	0
5	4*hauteur*(largeur+1)+largeur+1
6	0
7	0
8	0
9	0
10	0
11	0
12	0
13	0
14	0

On a une complexité de O(hauteur*largeur).

Complexité de la fonction ”followTheWay” :

Lignes	Comparaison
1	0
2	compteur
3	0
4	0

On a une complexité de O(compteur)

IV Résolution du labyrinthe :

Nous avons un labyrinthe parfait avec une entrée et une sortie (préalablement demandées à l'utilisateur à la suite de la création du labyrinthe). Dans notre matrice l'entrée a pour valeur -2 et la sortie, -3.Notre résolution se base sur la méthode suivante : dans un premier temps nous allons remplir notre labyrinthe d'entiers positifs dans un ordre croissant en partant de l'entrée. Puis dans un second on part de la sortie et on remonte jusqu'à l'entrée en suivant les cases dans un ordre décroissant. Les cases par lesquelles nous passons au retour seront indiquée dans notre matrice par la valeur -5. Pour cette partie, ce référer aux commentaires au sein du programme concernant les fonctions suivantes :

— street
— detection
— starter
— contagion
— followTheWay
— begin
— way

V Etude du temps d'execution :

Selon les dimensions de la matrice le temps d'utilisateurs varie.

— 30x30 : 0,6s
— 50x50 : 4,0s
— 100x100 : 1m 30s

VI Bilan de projet :

Ma solution au problème posé se divise donc en trois parties : tout d'abord la création de la matrice représentant le labyrinthe. Puis on place l'entrée et la sortie du labyrinthe. Et finalement on résout le labyrinthe. Pour la réalisation de ce programme, il a été intéressant de discuter en groupe des différentes façons d'aborder le problème. Bien que j'ai fais l'ensemble de ce programme seul, il me semble important de préciser que certaines idées quand à la résolution du problème ont muris grâce à des échanges avec d'autres personnes travaillant sur ce projet.