Université de Liège

Programmation avancée

INFO2050

Projet 2 : Structures de données

Noémie Lecocq
Andrew Sassoye

2017-2018

1 Analyse théorique

a)

Pour implémenter la structure Union-Find à l'aide d'un arbre binaire, nous avons décidé de créer un vecteur dont chaque élément est initialement la racine d'un arbre binaire. Chaque arbre du vecteur représente donc un singleton. Lorsque nous procédons a une union, nous utilisons l'implémentation des arbres binaires vue au cours théorique.

b)

	Liste	Arbre
Fonction UfUnion Fonction UfFind		O(h) [1]

c)

d)

e) La structure de labyrinthe

La structure maze est composée de 4 éléments.

- size : contient la taille du labyrinthe. Pour une taille N, le labyrinthe sera un carré N x N.
- unionFind : contient l'ensemble disjoint qui a été utilisé pour créer le labyrinthe.
- neighbours : est un vecteur contenant les paires de cellules étant voisines.
- convert : est une matrice permettant de retrouver le numéro d'une case du labyrinthe à partir de ses coordonnées.

  En effet, nous avons choisi d'utiliser la plupart du temps des entiers pour identifier les cases afin de pouvoir utiliser la structure *UnionFind* préalablement définie (et qui contenait des entiers). Ainsi nous représentons chaque case par un chiffre de 0 à $N * N - 1$

f) Pseudocode

```
mzCreate(size)
1  Maze maze
2  maze.size = size
3  maze.unionFind = ufCreate(size)
4  while ufComponentsCount(maze.unionFind) > 1
5        (coord1, coord2) = a random pair of neighbours
6        if There is a wall
7              Remove it
8              ufUnion(maze.unionFind, coord1, coord2)
9  return maze
```

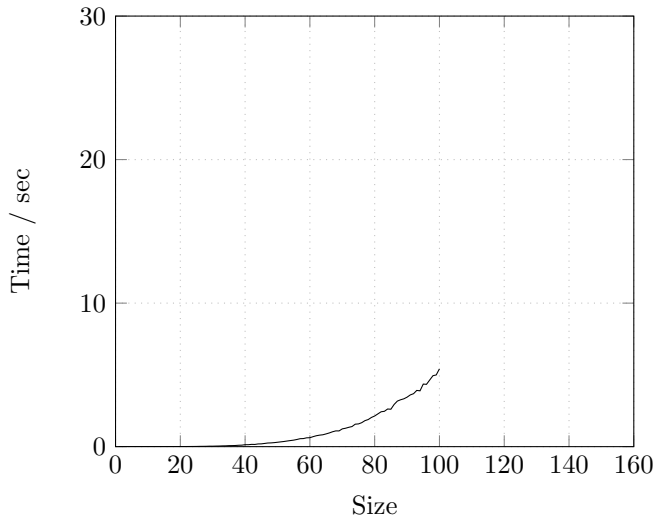
```
mzIsValid(maze)
1  if ufComponentsCount(maze.unionFind) > 1
2        return false
3  else
4        return true
```

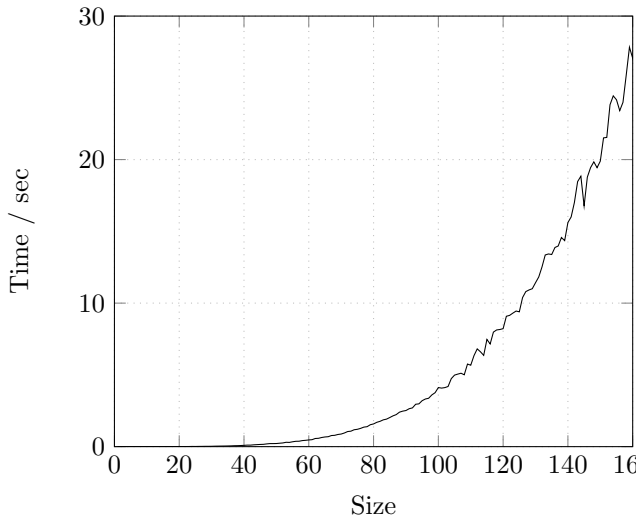
g) Complexité en temps avec UnionFindList

mzIsValid
mzIsValid est constant puisqu'il s'agit simplement d'aller lire la taille de l'*UnionFind*.

2 Analyse empirique

a)





b)