Partie 4

Agustín Cartaya

Le chemin le plus court

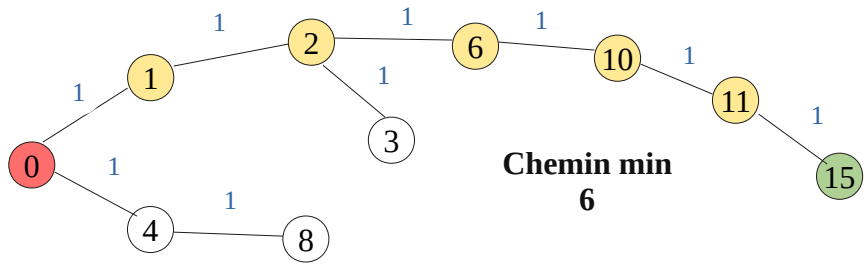
Pour trouver la distance minimale entre un point "a" et un point "b", l'algorithme dikstra a été implémenté. Pour ce faire, la grille qui contient le labyrinthe a été représentée en forme de graphe en prenant les carrés libres comme sommets et ses connexions comme arrêts avec une distance de 1. Par exemple:

Grille = [0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 1, 1, 1, 0]

a

0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15

b



Représenter la grille en forme de graphe

Methode en charge

int** lab2Graphe (int *caseLibreInitial, int *caseLibreFinal)

Paramètres:

- caseLibreInitial: pointeur vers l'id de la case initiale.
- caseLibreFinal: pointeur vers l'id de la case finale.

Renvoi:

- Tableau contenant le graphe résultant de la grille sous forme de liste de contiguïté.

modifie:

- caseLibreInitial: id (cases libres) de la case initiale.
- caseLibreFinal: id (cases libres) de la case finale.

PAS 1

Un tableau appelé "id2LibreID" a été créé, celui-ci va contenir une nouvelle grille avec des ids par rapport au nombre des cases libres de la grille0(si l'id anterieur n'est par une case libre on place -1)

Nota.

Dans ce même passage, il est vérifié que caseLibreInitial et caseLibreFinal sont valides (s'ils ne le sont pas on établi des valeur par défaut).

En prenant la grille précédente comme exemple

Grille = [0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 1, 1, 1, 0]
id20ID = [0, 1, 2, 3, 4, -1, 5, -1, 6, -1, 7, 8, -1, -1, -1, 9]

0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15

ID (Original)

0	1	2	3
4	-1	5	-1
6	-1	7	8
-1	-1	-1	9

ID (Cases libres)

PAS 2

Convertir la grille en graphe sous la forme de liste de contiguïté.
Le graphe résultant de la grille est représenté comme une table de contiguïté, qui a 5 colonnes, où la cinquième colonne représente l'id des cellules libres.
Les 4 premières représentent les ids des voisins libres qui y sont connexes aux ids de la 5eme colonne. Si le voisin n'existe pas ou n'est pas libre, cela est représenté par un -1.

Nota.
Les premières cases connectées sont représentées dans l'ordre de haut en bas et de gauche à droite

En prenant la grille précédente, nous obtenons

0	1	2	3
4	-1	5	-1
6	-1	7	8
-1	-1	-1	9

Up	left	right	down	id
-1	-1	1	4	0
-1	0	2	-1	1
-1	1	3	5	2
-1	2	-1	-1	3
0	-1	-1	6	4
2	-1	-1	7	5
4	-1	-1	-1	6
5	-1	8	-1	7
-1	7	-1	9	8
8	-1	-1	-1	9

Implémentation de l'algorithme dikstra

int* dijkstra(int** tabNoeud, int n, int init, int fin)
Méthode responsable de faire l'algorithme dijkstra
Paramètres:

- tabNoeud: Tableau contenant le graphe résultant de la grille sous forme de liste de contiguïté
- n: nombre de carrés libres
- init: id (case libre) de la case initial
- fin: id (case libre) de la case final

Renvoi:
tableau d'entiers de n + 1 positions où n représente la distance minimale entre la case initial et la case final (ces cases d’extrémité sont incluses) constituée comme suit.
Première position: Distance minimale

Autres éléments: id des cases à parcourir dans la grille pour obtenir le chemin minimum

PAS 1

- Création du tableau dijkstra avec des 0 à l'intérieur.
  Le tableau dijkstra est un tableau de taille n où les colonnes représentent les étapes effectuées lors de l'algorithme et les lignes représentent les nœuds du graphe. Chaque cellule du tableau contient 2 valeurs où la première est le nœud précédent et la seconde est la distance entre le nœud initial et celui-ci.

PAS 2

- Definition de la position actuelle égale à init (init passé par paramètre)
- Fermer le premier nœud et avancer un pas (fermer un nœud signifie placer -1 à la fin du tableau)

PAS 3

Nous appliquons la méthode dikstra de la même manière que si elle était faite à la main jusqu'à ce que d'obtenir un tableau comme le suivant (les données de ce tableau sont par rapport au labyrinthe obtenu au début)

Pas / Noeud	0	1	2	3	4	5	6	7	8	9
0	0, 0	0, 0	0, 0	0, 0	0, 0	0, 0	0, 0	0, 0	0, 0	0, -1
1	0, 1	0, 0	0, 0	0, 0	0, 0	0, 0	0, 0	0, 0	0, 0	0, -1
2	0, 0	1, 2	1, 2	0, 0	0, 0	0, 0	0, 0	0, 0	0, 0	0, -1
3	0, 0	0, 0	0, 0	2, 3	2, 3	0, 0	0, 0	0, 0	0, 0	0, -1
4	0, 1	0, 1	0, 0	0, 0	0, 0	0, 0	0, 0	0, 0	0, 0	0, -1
5	0, 0	0, 0	0, 0	2, 3	2, 3	2, 3	0, 0	0, 0	0, 0	0, -1
6	0, 0	0, 0	4, 2	4, 2	0, 0	0, 0	0, 0	0, 0	0, 0	0, -1
7	0, 0	0, 0	0, 0	0, 0	0, 0	0, 0	5, 4	0, 0	0, 0	0, -1
8	0, 0	0, 0	0, 0	0, 0	0, 0	0, 0	0, 0	7, 5	0, 0	0, -1
9	0, 0	0, 0	0, 0	0, 0	0, 0	0, 0	0, 0	0, 0	8, 6	0, 0

PAS 4

Nous parcourons le tableau à l'envers, en partant du nœud final et en sautant au nœud suivant (indiqué par la première position de la cellule) jusqu'à ce que nous atteignions le nœud initial.

Le resultat obtenu par la méthode dijkstra est le suivant (par rapport au tableau présenté ci-dessus)
[7, 15, 11, 10, 6, 2, 1, 0]

IDs de nœuds à parcourir

le nombre minimum de nœuds à parcourir de l'init à la fin (init et end inclus)

Résumé des méthodes utilisées

Privées

- int** lab2Graphe(int *caseLibreInitial, int *caseLibreFinal)
  Renvoie le labyrinthe sous forme d'un graphe (liste de contiguïté)

- int* dijkstra(int** tabNoeud, int n, int init, int fin)
  renvoie un tableau qui contient dans sa première position distance minimale entre la case init et la la case fin, et dans les autres positions l'id des cases à parcourir.

- void desingChemin(int* cases, char mark)
  change la velaur des cases qui conforment le lemin le plus court dans le labyrinthe.

Publiques

- int* getCheminMin(int caseInitial, int caseFinal)
  Fait appel aux méthodes lab2Graphe et dijkstra et renvoie le résultat envoyé par la méthode dijkstra

- int distMin(int caseInitial, int caseFinal)
  Fait appel a la méthode getCheminMin et retourne la première position de ce tableau moins 1

- void chemin(int caseInitial, int caseFinal)
  Designe le chemin le plus court de caseInitial à caseFinal en appelant les méthodes getCheminMin et desingChemin

Résultats

```
--------- Creation du labyrinthe 15 x 15 --------------
labyrinthe cree apres 232649 essais
  Cases         Attendu         Reel
Noires:         145             69
Blanches:       80              156
Total:          225             225
-----------------------------------------------------
35, 224, 223, 222, 221, 220, 219, 218, 217, 216, 215, 200, 185, 186, 171, 156, 141, 126, 127, 112, 97
, 82, 67, 52, 37, 22, 21, 20, 35, 34, 33, 32, 17, 16, 1, 0,
  ..-XXXX-X------
  X..X-...XX-X---
  XX....X.-XX--XX
  -X-XX-X.XX--XXX
  ---X-XX.-X-X---
  --X----.X----X-
  -X----X.-X-X-X-
  XX--X-X.X---X--
  ---X-X..-X-XX--
  -X---X.X-X-----
  -X--XX.-XXX-X--
  ------.X------X
  ---X-..X-------
  --XX-.X--------
  ----X..........
```