

Programmation en langage C – EI3

Travaux Pratiques

Objectif(s)

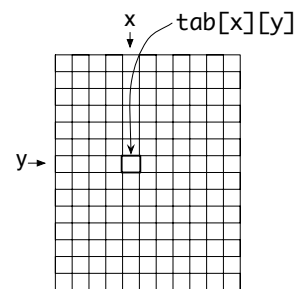
- ★ Pointeur, passage par adresse
- ★ Arithmétique des pointeurs
- ★ Lien pointeurs/tableaux

Le but du TP est de rechercher le plus court chemin, s'il existe, allant d'une case de départ à une case d'arrivée d'un labyrinthe. Il s'appuie sur quelques fonctions d'affichage créées pour le TP, déclarées et définies dans `affichage.h` et `affichage.o` situé dans le répertoire .

1 Représentation du labyrinthe

Le labyrinthe sera représenté par un tableau rectangulaire à deux dimensions `lab[TAILLE_X][TAILLE_Y]` (où `TAILLE_X` et `TAILLE_Y` sont des constantes définies dans `affichage.h`).

Chaque case du tableau est un entier et contient l'entier 0 si la case est libre, ou -1 si la case est bloquée par un obstacle.



Question 1

Dans un fichier `labyrinthe.c`, écrire une fonction qui construit le labyrinthe à partir d'un fichier texte dont le nom est passé en paramètre. Les fichiers textes contenant des labyrinthes ont la structure suivante : les 4 premiers entiers donnent les coordonnées des cases de départ et d'arrivée et les entiers suivants donnent le labyrinthe *ligne par ligne* selon le codage ci-dessus. On fournit pour exemple les trois labyrinthes suivants : `laby1.txt`, `laby2.txt`, `laby3.txt` (il peut être intéressant d'y jeter un coup d'œil).

Cette fonction devra renvoyer 1 si tout c'est bien passé, et 0 si la lecture n'a pu se faire.

Question 2

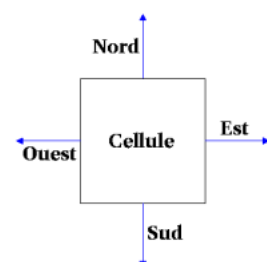
Dans le même fichier, créer la fonction principale `main` permettant de visualiser les labyrinthes donnés à l'aide des fichiers `affichage.h` et `affichage.o`. Vous prendrez soin de bien lire les commentaires dans le fichier `affichage.h` décrivant l'utilisation des différentes fonctions proposées.

Question 3

Transférer la fonction de construction du labyrinthe depuis un fichier texte dans un fichier `lecture.c` et créer le fichier `lecture.h` correspondant. Indiquez en commentaire de `labyrinthe.c` et `lecture.c` les commandes que vous utilisez pour la compilation et à l'édition de lien.

2 Déplacement dans le labyrinthe

On visualise le labyrinthe comme une carte géographique. Les seuls déplacements autorisés seront ceux en direction du Nord, de l'Est, de l'Ouest et du Sud. À partir d'une case (x, y) du labyrinthe, les seules cases accessibles pour se déplacer sont les quatre cases voisines (voir la figure ci-contre).



Question 4

Écrire une fonction qui, à partir d'une case du labyrinthe (x,y) et d'une direction d (modélisée par un nombre de 0 à 3, correspondant aux directions *Nord*, *Est*, *Sud* et *Ouest* respectivement), donne les coordonnées de la case voisine de (x,y) par rapport à la direction d .

Question 5

Indiquer en commentaire dans le code ce qu'il se passe si on cherche à utiliser les coordonnées des quatre cases voisines d'une case se trouvant sur le bord du tableau.

Pour se prémunir de ce problème sans devoir compliquer l'algorithme, on considèrera que toutes les cases en périphérie du labyrinthe sont bloquées par un obstacle, et qu'ainsi on n'appellera pas la fonction précédente depuis une case se trouvant à la périphérie du tableau.

3 1^{er} Algorithme : recherche du plus court chemin

L'algorithme de recherche du plus court chemin dans un labyrinthe se décompose en deux étapes. La première, la phase d'expansion, consiste à parcourir le tableau à la recherche des différents chemins allant de la case de départ à la case d'arrivée. La seconde, la phase de remontée, permet de retrouver le plus court chemin et de le tracer.

Pour cela, il nous faut enrichir la représentation du labyrinthe par un tableau. Désormais, chaque case du tableau contient un entier selon le codage suivant :

$$T(x,y) = \begin{cases} -2 & \text{case libre appartenant au chemin recherché} \\ -1 & \text{case bloquée par un obstacle} \\ 0 & \text{case libre non visitée} \\ r > 0 & \text{case libre visitée à une distance } r \end{cases}$$

3.1 Phase d'expansion

- On marque la case de départ avec la distance $r = 1$.
- On parcourt tout le tableau dans un ordre quelconque et on marque toutes les cases libres d'obstacle, non déjà traversée et voisine d'une case marquée r avec la distance $r + 1$ (on trouve les cases voisines en utilisant la fonction de la question précédente). Puis on incrémente le compteur de distance.
- On procède ainsi jusqu'à ce que la case d'arrivée soit atteinte.

Question 6

Écrire une procédure qui à partir d'un labyrinthe, d'une case de départ et d'une case d'arrivée effectue la phase d'expansion décrite ci-dessus. On prendra soin d'effectuer des tests pas à pas (à l'aide des fonctions d'affichage de labyrinthe fournies) pour vérifier le bon déroulement de l'algorithme.

Question 7 – Sans issue

Que se passe-t-il avec le labyrinthe `laby3.txt` ? Modifier la procédure précédente pour que l'itération s'arrête lorsqu'un parcours entier du labyrinthe n'entraîne aucune nouvelle case de marquée (cas où le chemin n'existe pas. Les cases d'arrivée et de départ sont séparées par des obstacles incontournables). Transformer la procédure en fonction pour qu'elle renvoie un booléen indiquant si un chemin existe entre les cases d'arrivée et de départ.

3.2 Phase de remontée

Quand un chemin existe, on veut tracer le chemin en remontant de la case d'arrivée jusqu'à la case de départ.

- On mémorise la longueur r du plus court chemin (distance marquée à la case d'arrivée).
- On marque la case d'arrivée comme appartenant au chemin (-2).
- On cherche la case voisine de la case courante qui est marquée à la distance $r - 1$ (on sait qu'elle existe sinon la case courante ne serait pas marquée à la distance r), on la marque appartenant au chemin et finalement on décrémente r .
- On procède ainsi tant que la case marquée n'est pas la case de départ.

Question 8

Écrire une fonction qui effectue la phase de remontée. La fonction pourra retourner un booléen indiquant si la phase de remontée s'est bien déroulée ou non (si on a bien trouvé une case voisine de la case courante qui est à la distance $r - 1$). Tester pas à pas (et graphiquement) la phase de remontée.

Question 9

Écrire une procédure qui nettoie le labyrinthe, c'est-à-dire marque les cases du labyrinthe n'appartenant pas au chemin tracé et non bloquées par un obstacle comme libre.

4 Algorithme récursif (Bonus)

Question 10

Il est possible d'améliorer la complexité de la phase d'expansion en utilisant des fonctions récursives. Écrire une nouvelle fonction réalisant la phase d'expansion (avec la même spécification que la précédente) utilisant une fonction auxiliaire récursive. De la même façon qu'à la question 4, on prendra soin d'effectuer des tests pas à pas.

