2I006 : ALGORITHMIQUE APPLIQUÉE ET STRUCTURES DE DONNÉES

Projet : Sorting Robot

17 Mai 2018

Par :
Celina HANOUTI
Farouk AIT ALI BRAHAM

Table des matières

Introduction

Dans ce projet on s'intéresse au jeu du Robot trieur (Sorting Robot) qui consiste à déplacer un robot dans une grille à m lignes et n colonnes dont chaque case possède une couleur de fond et peut comporter une pièce colorée, dans le cas contraire, elle comporte un rectangle noir. Chaque fois qu'une pièce est amenée par le robot sur une case de même couleur, celle-ci devient noire. Le but de ce projet est de déterminer des solutions telles que toutes les pièces soient ranger dans des cases de même couleur (et ainsi obtenir un écran noir) avec un nombre minimum de déplacement élémentaire sur la grille.

Description du code

Le projet a été implémenté en C et réparti en plusieurs fichiers *.c ,*.h et un Makefile pour chaque partie afin de faciliter la lisibilité et la compréhension.

Le projet a été organisé en deux répertoires, Partie1 et Partie2. Chaque répertoire contient les fichiers (fichiers sources et les fichiers d'en-tête) fournis ainsi que les fichiers contenant les fonctions que nous avons implémentées.

Dans le répertoire Partie1, le fichier auPlusProche.c contient les implémentations des différentes versions de l'algorithme «au plus proche», les fichiers AVL.c, AVL.h (resp. LDC.c, LDC.h) contiennent la structure d'AVL (resp. la structure de listes doublement chaînées) et la bibliothèque de fonctions qui manipulent cette structure. Afin de lancer l'exécution, il suffit d'exécuter le main ./main_auPlusProche suivi des dimensions de la grille, du nombre de couleurs et de la graine.

Dans le répertoire Partie2, toutes les fonctions implémentées se trouvent dans les fichiers Graphe.c, LCircuit.c et exo7.c (qui contient un main). Le main des exercices 5 et 6 se trouve dans le fichier main_Graphe.c.

Première partie :

Algorithme «Au plus proche»

Introduction

L'objectif de cette première partie est de donner plusieurs implémentations de l'algorithme « Au plus proche » en utilisant des structures de données différentes et ainsi comparer leurs complexités et leurs temps d'exécution.

L'algorithme « Au plus proche » consiste au préalable à vérifier si le robot porte une pièce, si c'est le cas, il se déplace à la case la plus proche ayant la même couleur de fond que cette pièce et la dépose, dans le cas contraire, il cherche d'abord la case la plus proche non-noire, portant une pièce et se déplace pour prendre cette dernière.

Version Naïve

Dans cette version dite « naïve », nous avons implémenter plusieurs fonctions utiles afin d'organiser et d'améliorer la lisibilité du code.

- **PlusCourtChemin :** qui ajoute une séquence de caractère (L,R,U,D,S) à la solution S qui correspond au plus court chemin entre deux cases.

- **CaseNoire :** qui nous permet de savoir si une case donnée est noire, on teste si la couleur du fond est la même que celle de la pièce
- **PieceNoire :** qui nous permet de savoir si une pièce est noire, on teste si sa couleur est différente de -1.
- **robotPortePiece :** qui nous permet de savoir si le robot porte une pièce, on teste si la couleur de la pièce portée par le robot est différente de -1.

Plus court chemin entre deux cases

Soient les deux cases (i, j) et (k, l) dans une grille.
Soit la propriété suivante $P\ (n),\ n \geq 0$:
— Le chemin se déplaçant de $|k - i|$ cases verticalement vers (k, j), puis de $|l - j|$ cases horizontalement vers (k, l),
— le chemin se déplaçant de $|l - j|$ cases horizontalement vers (i, l), puis de $|k - i|$ cases verticalement vers (k, l), sont des plus courts chemins, soit $n = |k - i| \geq 0$.

Montrons cette propriété par récurrence faible sur n *et* $m \geq 0$.
Base : pour $n = |k - i| = 0$ (respectivement $m = |l - j| = 0$), $k{=}i$ (respectivement $l = j$). Les deux cases sont sur la même ligne (respectivement sur la même colonne) donc en se déplaçant uniquement de $|l - j| = 0$ cases horizontalement (respectivement $|k - i|{=}0$ cases verticalement) est un plus court chemin. $P(0,m)$ *et* $P(n,0)$ sont donc vraies pour tout n et m.
Induction : Supposons la propriété vraie au rang $n \geq 0$, $m \geq 0$.
Montrons que la propriété est vraie au rang $n{+}1$ et $m{+}1$.

P(n + 1,m) et *P(n,m + 1)* : Soit *(i',j)* tels que */k− i'/ = n + 1*, donc *i'= i + 1* ou *i'= i – 1*, on sait que */j -l/ = m*, et le plus court chemin entre *(i' , j)* et *(k , l)* est donc la somme des plus courts chemins entre les cases *(i',j)*, *(i , j)* et les cases *(i,j)*, *(k,l)*. Comme */i' - i/ < n+1* et */k – i/ < n+1* donc par hypothèse de récurrence, le plus court chemin est le déplacement de n+1 cases verticalement et m case horizontalement. Donc *P(n + 1,m)* est vraie.
Un même raisonnement s'applique pour *P(m,m+1)*.
P(n+1,m+1) : soit *(i', j')* tel que */i' - k/ = n+1* et */j' - l/ = m+1*, donc soit */i' -i/ = 1* et */j' - j/ = 1* , donc le plus court chemin entre *(i' , j')* et *(k,l)* est donc la somme des plus courts chemins entre les cases *(i',j')*, *(i,j)* et les cases *(i,j)*, *(k,l)*. Comme */i' -i/ < n* et */j'-j/ < m* et par hypothèse de récurrence le plus court chemin est le déplacement de n+1 cases verticalement et m+1 cases horizontalement. Donc *P(n+1,m+1)* est vraie.
Conclusion :

la propriété *P(n ,m)* est vraie pour tout n et m.

La recherche d'une case dans une grille $n \times m$ s'effectue en $\Theta(n \times m)$, en effet, on parcourt toute la grille afin de trouver celle qui soit la plus proche de la case (i,j), pour des grilles carrées la complexité de la recherche est donc en $\Theta(n^2)$. Étant donné qu'on effectue la recherche pour chaque pièce de la grille (n^2 pièces lorsqu'il n'y a aucune case noire dans la grille), la complexité de l'algorithme naïf est donc en $\Theta(n^4)$.

Version Circulaire

La méthode circulaire consiste à rechercher circulairement autour d'une case (i,j) donnée (les coordonnées du robot) la case la plus proche en augmentant le rayon de recherche d'une case jusqu'à ce qu'on trouve la case recherchée.

Pour cette méthode, le pire cas correspond à lorsque la case recherchée est la plus éloignée possible de la case courante, dans ce cas, il faudra donc parcourir toute la grille (n^2 cases), cependant, contrairement à la version naïve, on peut arrêter la recherche dès que le robot trouve la case recherchée, dans le cas où la case recherchée est voisine de la case courante la recherche se fera en $\Theta(1)$. On a donc une complexité en $O(n^4)$.

Version par Couleur

Dans un premier temps, on a implémenté des fonctions de manipulation de listes linéaires chaînées : initialisation, insertion en fin en $\Theta(1)$, suppression en $\Theta(1)$ .. etc.

Étudions la complexité de la méthode par couleur en fonction de n et du nombre maximal de pièce d'une même couleur α. L'accès à la liste chaînée correspondante à la couleur de la pièce portée par le robot se fait en $\Theta(1)$. L'insertion de chacune des cases non-noire dans la liste chaînée correspondante est en $\Theta(1)$, ce qui donne une complexité de l'insertion en $\Theta(n^2)$, pour rechercher la case la plus proche du robot ayant la même couleur, il faudra parcourir toute la

liste chaînée, et donc la complexité est en $\Theta(l)$, avec l la longueur de la liste chaînée, comme on a $l \leq \alpha$, et qu'il faut répéter le traitement pour les n^2 pièces, on obtient une complexité en $O(n^2\alpha)$. Cependant, si on ne tient pas compte de ce paramètre α, la longueur des listes chaînées aura une borne supérieure de n^2 ($l \leq n^2$) et donc, dans ce cas, la complexité serait en $O(n^4)$.

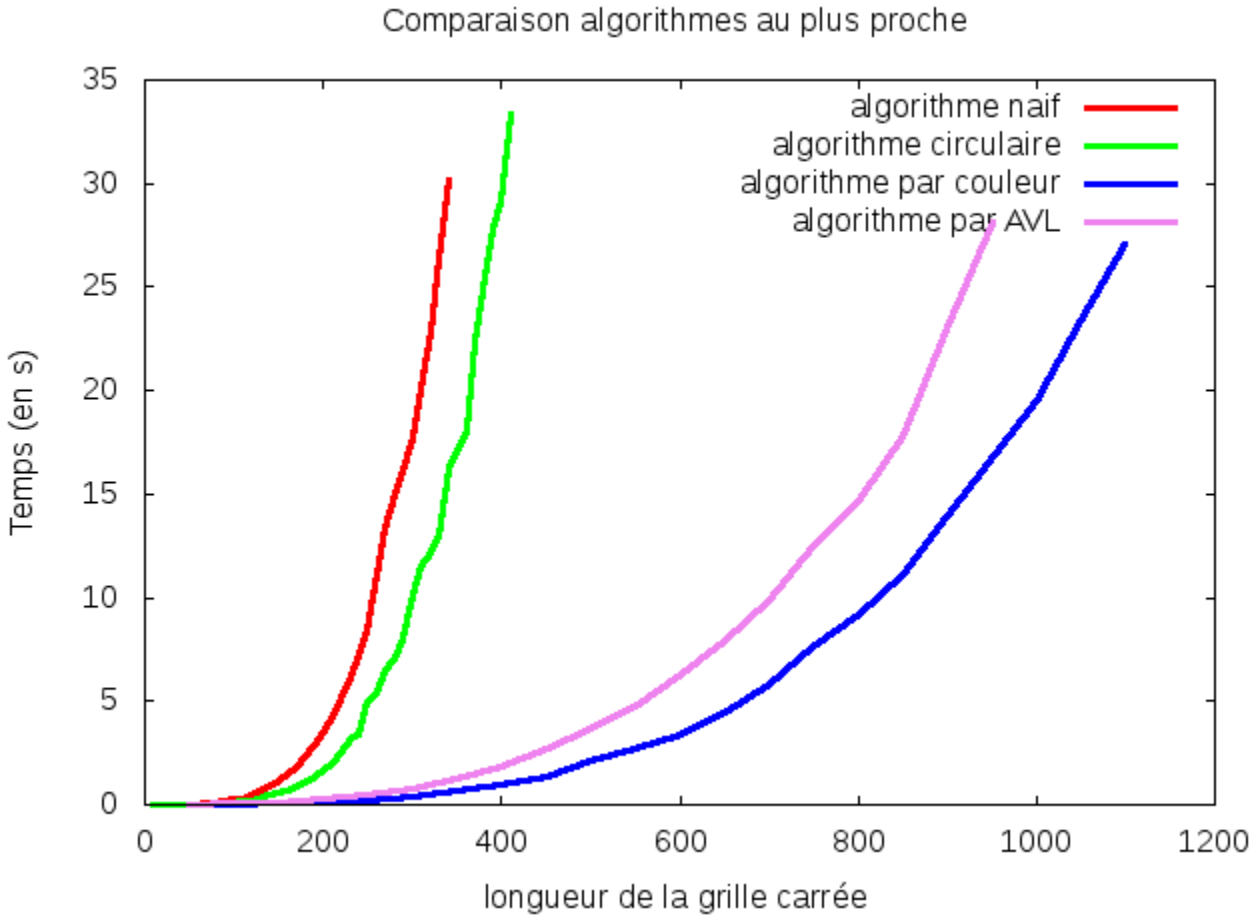
Version par AVL

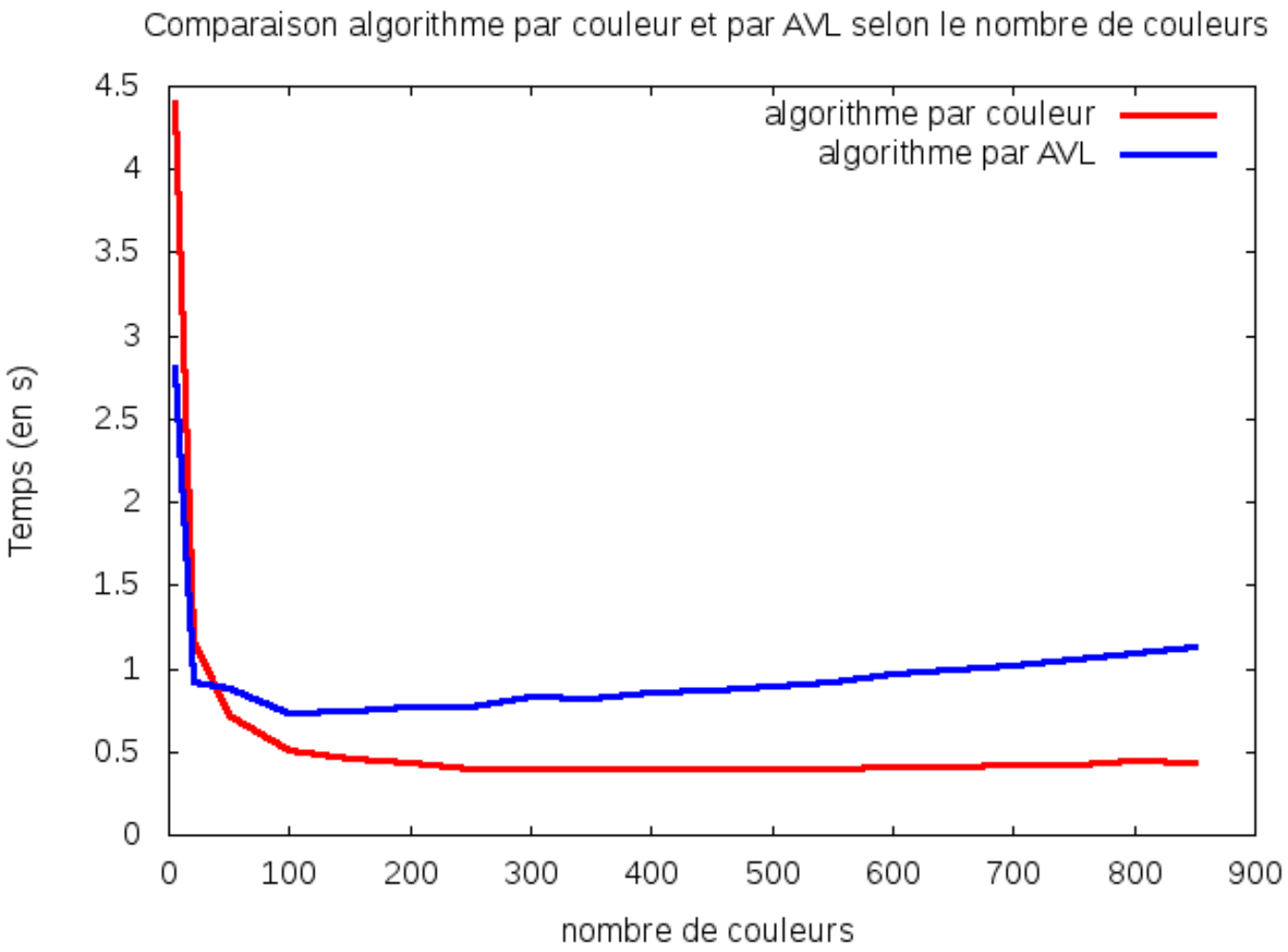
Dans cette version, on utilise des arbres binaires de recherche de type AVL, l'implémentation consiste à insérer les cases de la grille dans une matrice d'AVL selon la couleur de fond de chaque case. Chaque ligne de cette matrice correspond à une couleur distincte et chaque colonne correspond à chaque ligne de la grille du jeu. l'accès à une case (arbre AVL) de cette matrice se fait en $\Theta(1)$, la recherche de la case la plus proche d'une case donnée est en $O(\log(\beta))$, avec β le nombre de nœuds de l'AVL correspondant, car en effet, la recherche d'une case la plus proche sur une ligne donnée est équivaut à faire une recherche dans l'AVL correspondant, et donc on en déduit qu'une recherche sur les n lignes se fait en $O(n\log(\beta))$. Étant donné, qu'on répète cette recherche pour les n^2 pièces, et que $\beta \leq n$ (le nombre de nœuds dans un AVL est borné par le nombre de pièces n sur une ligne), on obtient donc une complexité en $O(n^3\log(n))$. Cependant si on prend en considération «le paramètre caché», $\beta \leq min(n,\alpha)$, et dans ce cas, la complexité de cette version serait en $O(n^3\log(min(n,\alpha)))$.

Analyse expérimentale

La figure ci-dessous donne les temps d'exécution des différentes implémentations de l'algorithme «Au plus proche» pour des grilles carrées. On rappelle que l'analyse théorique nous donne une complexité en $\Theta(n^4)$ pour la méthode naïve, en $O(n^4)$ pour la méthode circulaire et par couleur (sans prendre le paramètre caché en considération) et en $O(n^3\log(n))$ pour la version par AVL.

Expérimentalement, on remarque que les courbes vérifient bien l'analyse de la complexité effectuée précédemment.





Pour une grille carrée de taille 300, on s'aperçoit que le nombre de couleurs (en l'occurrence le paramètre α) a un impact sur le temps d'exécution de l'algorithme par couleur et de l'algorithme par AVL.

Algorithme	Dimensions de la grille	Temps d'exécution (en seconde)
Algorithme naïf	340 x 340	30.12 s
Algorithme circulaire	410 x 410	33.3 s
Algorithme par couleur	1200 x 1200	37,2 s
Algorithme par AVL	1050 x 1050	37.6 s

Table 1 - Ordre de grandeur (taille de la grille) pour lequel le temps d'exécution des algorithmes est au moins de 30 secondes.

Deuxième partie :

Grilles de jeu à une case par couleur

Introduction

Dans cette deuxième partie, on s'intéresse à la résolution du problème du robot trieur à l'aide des graphes. Nous allons considérer deux cas particuliers, le cas où le nombre de couleurs est égal au nombre de cases dans la grille, autrement dit, une couleur n'apparaît qu'une seule fois en tant que couleur de fond et une seule fois en tant que couleur de pièce, et un deuxième cas qui consiste à considérer des vecteurs, c'est-à-dire, des grilles limitées à une seule ligne.

Graphe et Circuit

Étant donné une grille G, on considère un graphe orienté $H=(V,A)$ qu'on appellera **graphe de déplacements.** On l'obtient de la façon suivante :

- L'ensemble des sommets V correspond à l'ensemble des cases (i,j) de la grille G.
- L'ensemble des arcs A correspond à des arcs allant d'un sommet-case (i,j) au sommet-case (i',j') tels que la case (i,j) non-noire contienne une pièce de la même couleur que celle du fond de la case (i',j').

Les fonctions fournies nous permettent de créer un graphe H à partir d'une grille G avec une implémentation par liste d'adjacence.

Chaque sommet est représenté par un struct Sommet qui contient ses coordonnées, sa liste de successeurs et un champ visit que l'on a utilisé pour la recherche de circuits.

En utilisant cette structure de graphe, nous nous sommes intéressés à la recherche de circuits couvrants les sommets-cases non-noirs du graphe H. Chaque circuit est représenté par un struct Cell_circuit contenant une liste linéaire chaînée de paire (*i,j*) appartenant à ce circuit, *jmin* et *jmax*. Dans un premier temps nous avons implémenté deux fonctions (version itérative et récursive) qui, à partir d'un sommet S, recherchent un circuit partant de ce sommet et insèrent tous les sommets-cases appartenant à ce circuit dans la structure mentionnée précédemment. Dans le cas d'une grille à une case par couleur, l'algorithme de recherche de circuits est relativement simple, car chaque sommet n'a qu'un seul successeur, on insère d'abord le sommet S dans la liste chaînée, on insère ensuite son successeur, le successeur de son successeur.. etc jusqu'à ce qu'on ferme le circuit en bouclant sur le sommet initial. À l'aide de la structure Lcircuit (liste chaînée contenant des Cell_circuit), nous avons ensuite implémenté une fonction qui recherche tous les circuits du graphe et qui les insère dans une liste linéaire chaînée de circuits rangés par *jmin* croissant.

L'algorithme de résolution par graphes se réduit au parcours de ces circuits, le robot effectue les déplacements nécessaires pour un circuit et ensuite il passe au suivant. Cependant, un parcours séquentiel ne nous garantit pas forcément un nombre de pas minimal, particulièrement dans le cas d'une grille réduite a une ligne i.e vecteur.

Vecteur avec «une case par couleur»

Dans le cas particulier d'un grille limitée à une ligne (vecteur) avec une case par couleur il existe un algorithme polynomial qui résout le problème du robot trieur. Cet algorithme a été proposé par Daniel Graf et sa complexité est en $\theta(n^2)$.

Principe :

Cet algorithme se base sur la structure de liste de circuits et du parcours de ces derniers, il s'effectue en plusieurs étapes:

- Détecter les circuits.
- Trier les circuits selon les indices *Jmin* croissant (afin de pouvoir parcourir d'abord les circuits les plus à gauche)
- Pour chaque circuit, on se positionne sur la case la plus à gauche.
- On effectue les circuits en les parcourant, cependant, si deux circuits sont emmêlés, on interrompt le premier pour pouvoir effectuer le deuxième, lorsque ce dernier est fermé, on se positionne au sommet-case où la coupe a été effectuée et on reprend le parcours du premier circuit.
- Une fois tous les circuits traités, on obtient une solution optimale permettant de résoudre le problème du robot trieur pour ce cas de grille i.e vecteur.

Implémentation :

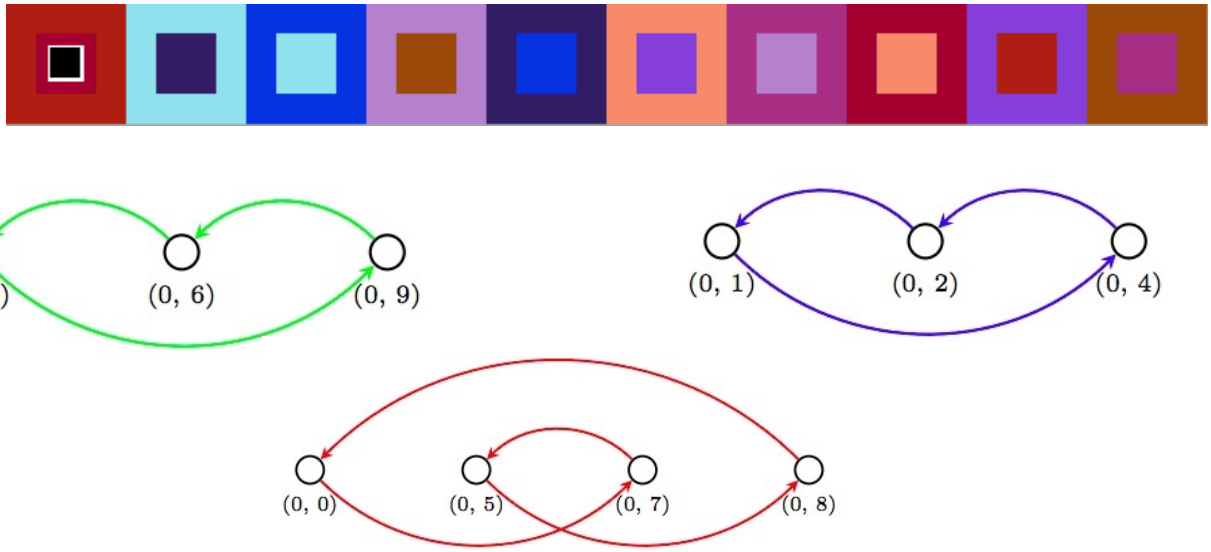
Afin de bien implémenter l'algorithme de Daniel Graf, on s'est appuyé sur plusieurs fonctions qu'on a implémenté, permettant de détecter des circuits, les trier, les traiter et d'ajouter des séquences à la solution.

Exemple: pour un cas d'une grille [1 10 10 5], on aura trois circuits emmêlés:

Circuit 1 : (0,0) (0,7) (0,5) (0,8)

Circuit 2 : (0,1) (0,4) (0,2)

Circuit 3 : (0,3) (0,9) (0,6)



A l'issue de l'exécution de l'algorithme de Daniel Graf, on obtient alors la solution finale optimale :

S R R R R R R R S L L S R R R S L L L L L L L S R R R S L S R R R R R R S L L L S L L L S L S L S L S

Solution pour le cas d'une grille à une case par couleur

Nous nous sommes inspirés de l'algorithme de Daniel Graf pour élaborer une nouvelle méthode pour résoudre ce problème dans le cas d'une grille à une case par couleur. En effet, l'algorithme s'appuie sur la notion de coupure de circuits, lors d'un parcours d'un circuit, si le robot passe par une case (non-visitée) appartenant à un autre circuit ayant un diamètre inférieur au circuit initial, on interrompt celui-ci pour le traiter. Pour cela, il faut d'abord modifier la structure de liste de circuits en ajoutant *imin* et *imax* afin de pouvoir calculer le diamètre d'un circuit et il faut également modifier l'implémentation de la liste chaînée d'un circuit en la rendant circulaire pour pouvoir le traiter à partir de n'importe quel sommet-case appartenant à ce circuit, ce qui permettra de diminuer considérablement le nombre de pas à effectuer pour aller d'un circuit à un autre. Cependant, par manque de temps, on a opté pour un algorithme assez simple, qui consiste à parcourir la liste des circuits et de traiter à chaque fois le circuit le plus proche de la position du robot.

Conclusion

Dans ce projet, nous nous sommes intéressés au jeu du robot trieur qui est un jeu combinatoire considéré comme étant NP-difficile, nous avons ainsi appris qu'il était tout de même possible de trouver de très bonnes solutions en temps raisonnable mais qu'il était également possible pour certains cas particuliers (Vecteur à une case par couleur) de résoudre le problème en temps polynomial. Ce projet nous a également permis de mettre en pratique les notions théoriques abordées en cours, notamment la manipulation de structures de données abstraites ainsi que de très bonnes pratiques de programmation.