Sorbonne Université

2I006 : Algorithmique appliquée et structures de données

Projet :

Sorting Robot

Kamil Rzeszutko
Angelo Ortiz

Licence d'Informatique
Année 2017/2018

Table des matières

Table des figures

Introduction

Présentation

Le projet a consisté en la résolution du problème du robot trieur au travers de différentes méthodes.

Le jeu se compose d'une grille de jeu de m lignes et n colonnes et d'un robot pouvant se déplacer sur la grille de jeu. Chaque case possède une couleur de fond et peut comporter une pièce colorée. Le nombre de couleurs utilisées dans la grille est $1 \leq c \leq m \cdot n$. Le but du jeu consiste à déplacer le robot de sorte que chaque pièce soit rangée sur une case de même couleur. À chaque fois qu'une pièce est amenée sur une case de même couleur, la case devient noire. Le but est donc d'obtenir un écran noir.

Objectif

Le but de ce travail a été de sensibiliser les étudiants sur l'impact des choix d'implémentations, notamment au niveau des structures de données et leurs algorithmes plus adaptés, sur le temps d'exécution et l'espace mémoire réquis.

Dans le cadre du projet, on a utilisé les listes doublement chaînées, les arbres AVL et les graphes orientés comme base des algorithmes mis en œuvre ; on a aussi eu besoin d'une implémentation de pile pour le dernier algorithme proposé.

Aperçu

Il est possible de jouer au jeu du robot trieur à travers une interface graphique. Les déplacements possibles sont associés aux touches flèches du clavier, et l'échange de pièces se fait avec l'appui de la touche espace. On vous en présente un exemple dans la figure 1.

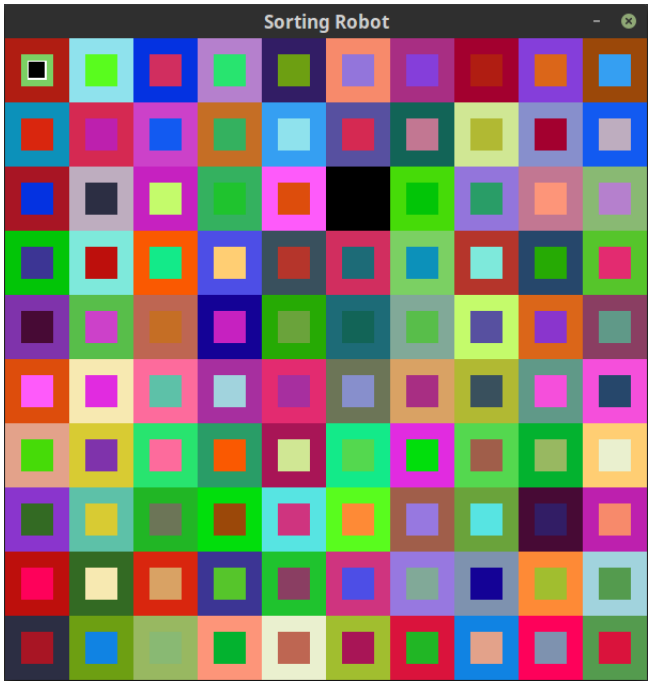


FIGURE 1 – L'interface graphique du jeu pour une grille de 10x10 avec le robot dans la case en haut à gauche repéré par un petit carré noir avec une ligne blanche l'entourant

Comme vous pouvez apercevoir dans le menu en console de la figure 2, on a regroupé nos différents algorithmes de résolution du problème donné de sorte que l'on puisse trouver en même temps des différentes solutions pour une configuration du jeu donnée.

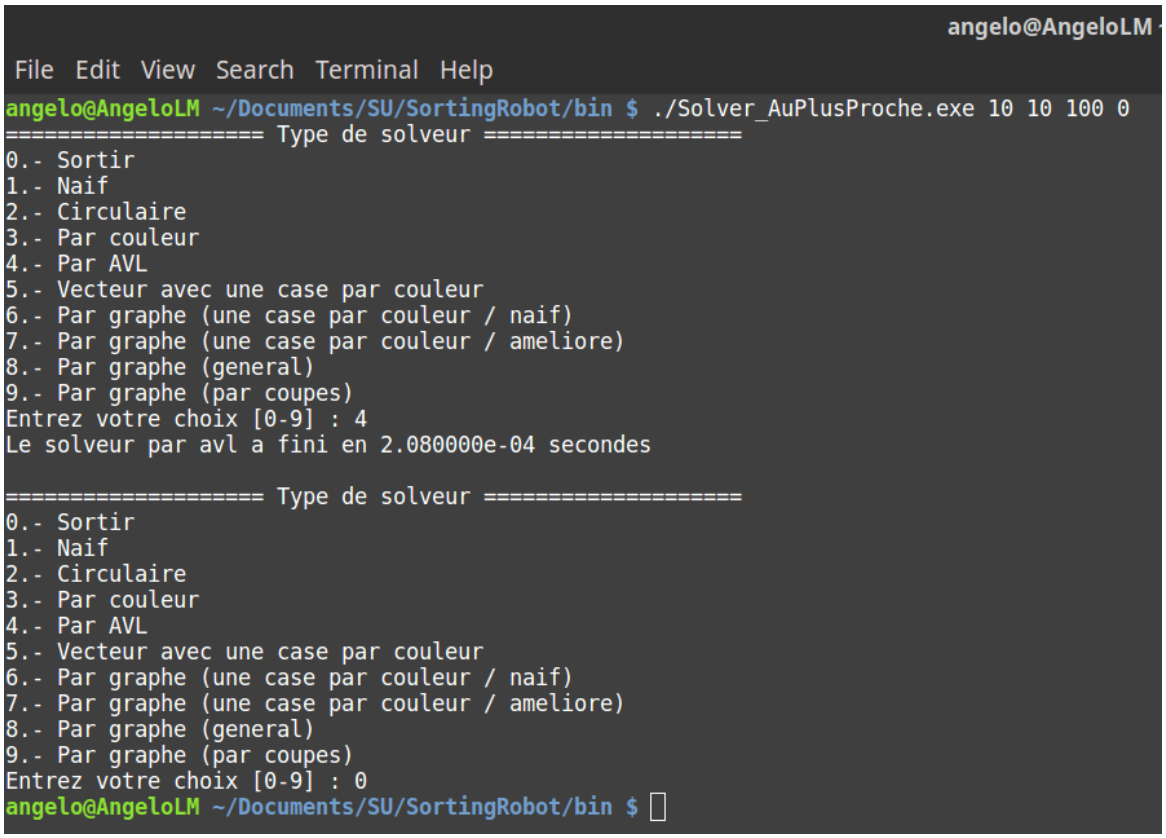


FIGURE 2 – Le menu en console permettant de résoudre le problème du robot à l'aide des 9 algorithmes implémentés

Structure

Le projet a été organisé de manière cohérente en plusieurs répertoires pour faciliter l'accès aux morceaux de code recherchés.

Dans un premier temps, on se focalise sur le cœur du projet, à savoir toute la partie liée au code. Le répertoire **lib** contient les fichiers sources fournis aussi bien de la première partie que de la deuxième partie. On a regroupé les fichiers sources des fonctions de base et des algorithmes de résolution dans le répertoire **src**. Les fichiers d'en-tête associés aux fichiers sources de ces deux répertoires se trouvent dans **include**. Les fichiers de test de fonctionnement et performance, et d'exploitation des données sont répertoriées dans **tests**. À l'issue de la compilation des fichiers `.c` du projet gérée par **Makefile**, on compte des fichiers objets et des exécutables : les premiers sont enregistrés dans **obj** et les deuxièmes, dans **bin**.

Dans un deuxième temps, on détaille la partie du projet dédiée au traitement des données obtenues. On compte notamment le répertoire **data** contenant les résultats des différents tests de performance et fonctionnement. Le répertoire **scripts** comprend des fichiers texte exploitant les données du répertoire précédent pour obtenir des images enregistrées dans **img**. Ce dernier comporte aussi les images utilisées dans ce document.

Finalement, il reste seulement deux répertoires à préciser : **report** contient tous les fichiers LaTeX nécessaires pour la bonne mise en forme de ce document, tandis que **solutions** comporte

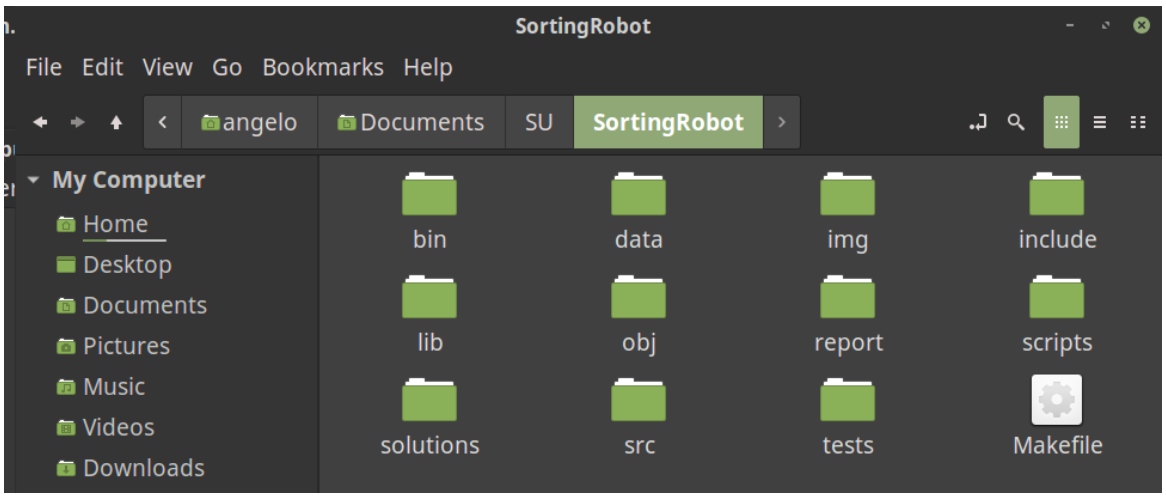


FIGURE 3 – La structure du projet en différents répertoires et le fichier d'aide à la compilation `Makefile`

des fichiers texte correspondant aux solutions obtenues par les algorithmes de résolution, i.e. les chaînes de caratères définissant les mouvements du robot trieur.

Première partie

Algorithme au plus proche

Dans cette première partie, on vous présente l'analyse faite pour chacune des versions implémentées de l'algorithme au plus proche, à savoir les versions naïve, circulaire, par couleur et par AVL.

1 Préliminaires

Avant de rentrer dans les détails des quatre implémentations, on veut vous présenter des résultats qui constituent les bases de l'algorithme au plus proche.

Plus court chemin

Tout d'abord, on montrera une propriété fondamentale utilisée tout au long du projet.

Soient les deux cases (i, j) et (k, l) dans une grille à m lignes et n colonnes. Soit la fonction $\texttt{dist}((i, j), (k, l)) = |k - i| + |l - j|$. On a donc la propriété suivante $P(r), r \geq 0$:

- le chemin VH qui consiste à se déplacer de $|k - i|$ cases verticalement vers (k, j), puis de $|l - j|$ cases horizontalement vers (k, l), et
- le chemin HV qui consiste à se déplacer de $|l - j|$ cases horizontalement vers (i, l), puis de $|k - i|$ cases verticalement vers (k, l),

sont des plus courts chemins, où $r = \texttt{dist}((i, j), (k, l)) \geq 0$.

Preuve

Montrons cette propriété par récurrence faible sur $r \geq 0$.

Base : Pour $r = 0$, $(k, l) = (i, j)$. On se déplace de 0 case verticalement et horizontalement. Ainsi, on reste dans la même case. Donc, la propriété est vérifiée pour $r = 0$.

Induction : Supposons que la propriété soit vérifiée pour un $0 \leq r \leq m+n-3$ fixé. Montrons que la propriété est aussi vérifiée pour $r + 1$.

Soient $p = |k - i|, q = |j - l| \in \mathbb{N}$, tels que $r + 1 = p + q$. Puisque $r + 1 \geq 1$, au moins l'une des variables est non nulle. Supposons sans perte de généralité que $p > 0$, i.e. $p - 1 \geq 0$.

Trois cas sont possibles :

1. $i = 0$:
   Tout d'abord, $p = |k - 0| = k$. Puis, pour aller de la case $(0, j)$ vers la case (k, l) avec un premier déplacement vertical d'une case, il faut passer par la case $(1, j)$. On a que $|k - 1| = p - 1$ et, en conséquence, $\texttt{dist}((1, j), (k, l)) = (p - 1) + q = r$. Par hypothèse de récurrence, $P(r)$ est vérifiée. De ces deux faits, le chemin se déplaçant de $|k|$ cases verticalement vers (k, j), puis de $|l-j|$ cases horizontalement vers (k, l) est un plus court chemin.
2. $i = m - 1$:
   Tout d'abord, $p = |k - (m - 1)| = m - k - 1$. Puis, pour aller de la case $(m - 1, j)$ vers la case (k, l) avec un premier déplacement vertical d'une case, il faut passer par la case $(m - 2, j)$. On a que $|k - (m - 2)| = (m - 2) - k = p - 1$ et, en conséquence, $\texttt{dist}((m - 2, j), (k, l)) = (p - 1) + q = r$. Par hypothèse de récurrence, $P(r)$ est vérifiée. De ces deux faits, le chemin se déplaçant de $|k - (m - 1)|$ cases verticalement vers (k, j), puis de $|l - j|$ cases horizontalement vers (k, l) est un plus court chemin.

3. $\underline{0 < i < m-1}$:
   Pour aller de la case (i, j) vers la case (k, l) avec un premier déplacement vertical d'une case, il faut passer par la case (h, j), où $h \in \{i-1, i+1\}$, telle que $0 \leq h < m$ et $|k-h| = p-1$. On a alors que $\texttt{dist}((h, j), (k, l)) = (p-1) + q = r$. Par hypothèse de récurrence, $P(r)$ est vérifiée. Donc, en particulier, le chemin se déplaçant de $|k-h|$ cases verticalement vers (k, j), puis de $|l-j|$ cases horizontalement vers (k, l) est un plus court chemin.

On fait de même pour $q > 0$ avec un traitement non plus sur les lignes, mais sur les colonnes de la grille, et on obtient que le chemin se déplaçant de $|l-j|$ cases horizontalement vers (i, l), puis de $|k-i|$ cases verticalement vers (k, l) est un plus court chemin

$\underline{\text{Conclusion}}$:

$$\left.\begin{array}{l} P(0) \text{ vraie} \\ \forall r \in \{0, \ldots, m+n-3\}, [P(r) \implies P(r+1)] \end{array}\right\} \begin{array}{l} \forall r \in \{0, \ldots, m+n-2\}, \\ \text{les chemins } VH \text{ et } HV \\ \text{sont des plus courts chemins.} \end{array}$$

Fonctions auxiliaires

Les quatres fonctions ci-dessous ont servi d'interface des fonctions fournies et améliorent énormément la lisibilité du code.

- `estCaseNoire` nous permet de savoir si une case est noire, i.e. si elle porte une pièce de même couleur.
- `estPieceNoire` nous permet de savoir si une pièce est non noire, i.e. si sa couleur est différente de -1.
- `robotPortePiece` nous permet de savoir si le robot porte une pièce, i.e. si la couleur de la "pièce" du robot est différente de -1.
- `couleurPieceRobot` nous permet de savoir dans le cas où le robot porte une pièce, sa couleur.

2 Principe

Le robot prend la pièce de la case la plus proche s'il en porte pas. Puis il cherche la case de même couleur la plus proche étant la plus collée à la bordure supérieure de la grille. S'il y en a plusieurs, il choisit celle qui est la plus à gauche. Il se déplace ensuite jusqu'à cette case-là et y dépose sa pièce. Il répète ce procédé jusqu'à ce que la grille soit complètement noire.

3 Analyse de la complexité

Même si l'analyse de la complexité de l'algorithme dépend de son implémentation, on peut commencer par les morceaux de code communs aux toutes. Dans le pire cas, il n'y a aucune case noire dans la grille, i.e. il y a $O(n^2)$ pièces à traiter, ce qui correspond à la boucle `while` dans le code fourni, en l'occurrence la boucle principale des fonctions. De plus, la recherche d'une pièce non noire aux alentours du robot est faite au tout début de l'algorithme et à chaque fois que l'on ferme un circuit [1] : cela dépend fortement du nombre de circuits dans la grille. Par conséquent, ceci n'est pas pris en compte pour l'analyse de la complexité et celle-ci se réduit à la recherche de la case cible pour le dépôt de la pièce du robot.

1. Un circuit est une suite d'*arcs* qui se referme (cf. section 8).

Par ailleurs, on a un paramètre caché correspondant au nombre maximal de pièces d'une même couleur dans le grille et ainsi des cases d'une même couleur. Ce paramètre est $\alpha = \left\lceil \frac{nm}{c} \right\rceil$, où m, n et c correspondent respectivement au nombre de lignes, de colonnes et de couleurs de la grille. On a $\alpha = \left\lceil \frac{n^2}{c} \right\rceil$ dans le cadre du projet.

4 Version naïve

Comme mentionné dans la section précédente, il suffit de calculer la complexité de la recherche de la case cible pour trouver la complexité totale d'une implémentation de l'algorithme au plus proche. La recheche de l'implémentation naïve de l'algorithme consiste en un parcours matriciel de la grille. Puisqu'on traite toutes les cases dans ce parcours, la complexité de cette recherche est de l'ordre de $\Theta(n^2)$. Ceci donne une complexité générale en $O(n^4)$ pour la version naïve.

5 Version circulaire

Pour cette version, la recherche est faite circulairement autour du robot. Autrement dit, on cherche d'abord dans les cases à une distance d'une case de la position du robot et si aucune d'entre elles ne remplit la condition sur la couleur, on continue la recherche pour une distance de deux cases et ainsi de suite jusqu'à trouver la bonne case. On voit très facilement que l'on ne traite toutes les cases que dans le pire des cas, à savoir lorsque la case se trouve dans le coin inférieur droit de la grille. La recherche ayant un ordre de grandeur de $O(n^2)$, celui de cette version au complet est en $O(n^4)$.

Même si l'ordre de grandeur de la complexité-temps pire-cas pour ces deux premières implémentations est le même, on remarque que la première croît plus vite que la deuxième. Ceci est dû à la différence en la recherche de la case cible : on a un ordre de grandeur en moyenne pour la version naïve, alors qu'il s'agit d'une borne supérieure pour la version circulaire. C'est pourquoi on a obtenu des meilleurs résultats pour la deuxième version.

6 Version par couleur

Dans un premier temps, on détermine un majorant de la complexité. On sait que le robot porte une pièce à presque tout moment du jeu et on cherche alors la case de même couleur la plus proche de lui. On a un accès en $\Theta(1)$ à la liste chaînée correspondant à la couleur de la pièce et on est obligé de parcourir toute la liste car l'ordre suivi lors de l'insertion n'est pas relevant. Soit l la longueur de la liste chaînée. On a donc que la complexité de la recherche de ladite case est en $\Theta(l)$. Or on a $l \leq n^2$. On en déduit que le traitement d'une pièce est de l'ordre de $O(n^2)$, ce qui donne une complexité en $O(n^4)$ pour cette version.

Dans un deuxième temps, on se rend compte qu'on a négligé un facteur clé dans cette version, à savoir le paramètre α mentionné précédemment. On en tient compte dans l'analyse et on remarque qu'il est une borne supérieure de la longueur des listes chaînées et il remplace en conséquence n^2 dans le paragraphe ci-dessus. De ce fait, on obtient que la complexité de la recherche dans la liste chaînée est en $O(\alpha)$, ce qui donne finalement une complexité en $O(\alpha n^2)$ pour la version par couleur de l'algorithme.

N.B. L'insertion de chacune des $O(n^2)$ cases non noires de la grille dans la liste chaînée correspondante est en $\Theta(1)$, ce qui donne une complexité de l'insertion en $\Theta(n^2)$. On remarque ainsi que les complexités données ci-dessus l'emportent à l'égard de celle de l'initialisation des listes.

7 Version par AVL

La dernière version de cette algorithme répose sur l'utilisation des arbres AVL. On a rangé les cases dans une matrice d'arbres : chaque ligne correspond à une couleur distincte et chaque colonne correspond à une ligne de la grille du jeu. Etant donné que l'on a implémenté la matrice sous la forme d'un tableau de tableaux, l'accès à un arbre quelconque est en $\Theta(1)$. Cependant, la recherche de la case la plus proche dans une ligne donnée est en $O(h(T))$, où $h(T)$ est la hauteur de l'arbre T correspondant. On sait [2] d'ailleurs que $h(T) \leq \lceil \log_2 n(T) \rceil$, où $n(T)$ est le nombre de nœuds dans l'arbre T. Or on a $n(T) \leq \min(n, \alpha)$. Par transitivité, ceci nous donne une complexité en $O(\log \min(n, \alpha))$. On en déduit que la recherche sur les n lignes de la grille est en $O(n \log \min(n, \alpha))$. On obtient finalement une complexité en $O(n^3 \log \min(n, \alpha))$ pour cette dernière implémentation de l'algorithme au plus proche.

N.B. L'insertion de chaque case non noire de la grille dans l'arbre AVL correspondant est en $O(\log \min(n, \alpha))$, ce qui donne une complexité de l'insertion en $O(n^2 \log \min(n, \alpha))$. On remarque ainsi que la complexité donnée ci-dessus l'emporte à l'égard de celle de l'initialisation des arbres.

2. En effet, l'une des propriétés des arbres AVL est l'équilibrage.

Deuxième partie

Résolution par graphes

Dans cette deuxième partie, on vous détaille le fonctionnement des procédures de résolution du problème du robot trieur à l'aide des graphes.

8 Méthode par circuits

Préliminaires

Étant donné une instance de grille G, on définit un graphe orienté $H = (V, A)$, nommé **graphe des déplacements** de la grille, de la manière suivante :

— l'ensemble des sommets V correspond aux cases (i, j) de la grille G,

— l'ensemble des arcs A correspond à des arcs allant du sommet-case non noir (i, j) au sommet-case non noir (i', j') tels que la case (i, j) contient une pièce ayant la même couleur que celle du fond de la case (i', j').

Il est à noter qu'un arc représente une succession de déplacements sur la grille. En effet, le robot doit traverser toutes les cases composant le trajet représenté par un arc.

La strucutre de données implémentant le graphe H répose sur une matrice de sommets pour avoir un accès en $O(1)$ à tous les sommets et des listes d'adjacence pour représenter les arcs adjacents à un sommet donné.

Principe

Cet algorithme de résolution par graphes se base sur la recherche de circuits. En effet, si on compte la liste de tous les circuits présents dans le graphe H associé à la grille G, résoudre le problème du robot trieur se réduit au parcours séquentiel de ces circuits. Cependant, le chemin suivi ne correspondra pas en général au plus court chemin, ce qui était le cas dans la partie précédente.

Implémentations

On a mis en œuvre cette méthode à l'aide de trois fonctions résolvant le problème du robot trieur.

Version naïve

Dans un premier temps, on a décidé de faire une recherche de circuits simple. Pour chaque sommet-case non noir, on recherche le sommet correspondant à la case en haut à gauche ayant la même couleur que celle de sa pièce. Compte tenu que l'ordre des sommets-case dans les listes d'adjacence est induit par le parcours matriciel [3] de la grille, cette première recherche des circuits est très rapide. Ce gain est nuancé par le nombre de pas utilisés par le robot pour rendre noire la grille du jeu dans le cas où il y a plusieurs pièces par couleurs, i.e. $\alpha > 1$. En effet, le robot essaie systématiquement de se déplacer vers le coin supérieur gauche de la grille aussi bien pour refermer un circuit que pour en commencer un nouveau.

3. Il s'agit d'un parcours séquentiel des lignes du haut vers le bas et des cellules à l'intérieur de gauche à droite.

Version améliorée

Dans un deuxième temps, on a modifié légèrement la méthode précédente : la recherche du circuit suivant n'est plus naïve. Elle devient ainsi la recherche dans la liste de circuits pas encore parcourus du circuit dont le premier sommet-case est le plus proche de la position courante du robot trieur. Ce changement comporte un compromis en lui-même dans la mesure où le robot réalise des trajets plus courts, mais que le temps d'exécution pour ce faire augmente.

Version générale

On a voulu finalement améliorer la recherche des circuits et l'ajouter à la version précédente. Puisque la grille contient autant de cases que des pièces, le graphe H a la particularité que tous les sommets appartiennent à exactement un circuit à la fin d'une recherche de circuits. On exploite donc ce fait et se permet de faire la modification suivante dans la recherche des circuits : pour chaque sommet, on fera une recherche du successeur le plus proche au lieu de prendre celui en haut à gauche. De la même manière que pour la version améliorée, cette modification entraîne un compromis : la recherche de circuits devient alors plus longue, mais le robot finit sont parcours de la grille en moins de pas qu'auparavant.

9 Vecteur avec une case par couleur

À la différence du cas général, il existe bien un algorithme polynômial résolvant le problème du robot trieur dans le cas particulier d'une grille réduite à une ligne, i.e. un vecteur, avec autant de couleurs qu'il y a des cases. Cet algorithme a été proposé par Daniel Graf et sa complexité est en $\Theta(n^2)$.

Préliminaires

L'algorithme répose sur la structure de liste de circuits trouvée à l'exercice précédent. Il faut néanmoins insérer les indices *jmin* et *jmax* des cases la plus à gauche et la plus à droite pour chaque circuit de la liste. De plus, le premier sommet-case de chaque circuit doit correspondre à sa case la plus à gauche, et la liste doit être rangée dans l'ordre croissant des *jmin*.

Heureusement pour nous, ces trois conditions sont déjà toutes remplies par notre méthode de recherche des circuits du graphe. En effet, notre fonction parcourt les arcs selon l'ordre défini lors de la création du graphe associé à la grille, à savoir celui du parcours matriciel.

Principe

Cet algorithme se base aussi sur le parcours des circuits. Cependant, il les coupe chemin faisant pour en commencer d'autres. Autrement dit, étant donné un sommet-case d'un circuit, si le prochain sommet-case par sa droite appartient à un autre circuit, il arrête le parcours courant. Puis il se dirige vers ce sommet-case en vue de commencer le parcours du circuit associé et il répète la démarche pour ce dernier sommet-case. Une fois ce circuit fermé, il revient au sommet-case de coupe et reprend le parcours initial.

L'idée de coupure de circuits réduit le nombre de pas utilisés pour se déplacer de la fin d'un circuit au début du suivant sans aucune pièce. Ce sont les déplacements que l'on qualifie de non essentiels ou encore optimisables.

Note d'implémentation

On a implémenté cet algorithme à l'aide de diverses fonctions de base pour simplifier le code de la fonction principale. On s'est rendu compte très rapidement qu'il y a un petit défaut dans l'algorithme. Effectivement, si le premier circuit ne commence pas dans la première case, à savoir $(0, 0)$, on fait des déplacements inutiles à la fin de l'algorithme pour revenir à cette case. Pour éviter ces pas superflus, on décale la position initiale du robot vers le premier sommet-case non noir avant de commencer le vrai algorithme.

Généralisation

On a beaucoup apprécié l'algorithme de Daniel Graf. C'est pourquoi on s'en est inspirés pour développer une nouvelle approche pour résoudre ce problème, ou plutôt une amélioration d'un algorithme précédent. Effectivement, on est partis sur la base de la version générale et on y a ajouté la notion de coupure de circuits.

Tout d'abord, on doit faire les remarques suivantes :

— il s'agit d'une méthode générale, c'est-a-dire que la grille de jeu comporte un nombre quelconque de lignes et de colonnes ; un tableau de pointeurs pour toutes les cases n'est donc pas envisageable ; et

— la notion de « case la plus à droite » est difficilement généralisable au cas d'une matrice.

Compte tenu de ces deux faits, on a choisi d'insérer la notion de coupure lors du traitement d'un circuit comme pratiqué dans la méthode générale : si à un moment du parcours on trouve un circuit à une distance plus petite que la *distance de référence*, on décide de commencer ce dernier circuit et celui qui a été interrompu est éventuellement repris. La distance de référence correspond à la distance entre la case courante du robot et celle qui la suit dans le circuit, multipliée par un *coefficient de coupure*. Une valeur petite de ce paramètre indique une faible prise de risques ; autrement dit, le robot ne peut couper un circuit que s'il en trouve un autre très proche de lui. Une valeur grande du paramètre implique par contre l'élargissement de l'éventail de circuits éligibles.

Cet algorithme s'est avéré au moins aussi efficace que la version générale de base : de manière générale, les résultats améliorent au fur et à mesure que le nombre de circuits augmente.

N.B. Pour que la signature de la fonction implémentée soit compatible avec celle des fonctions précédentes, le coefficient de coupure a été défini à travers une macro et pas comme argument de fonction.

Troisième partie

Comparaison et performances

Dans cette dernière partie, on vous présente les résultats des différents test de performance réalisés.

10 Taille de la grille variable

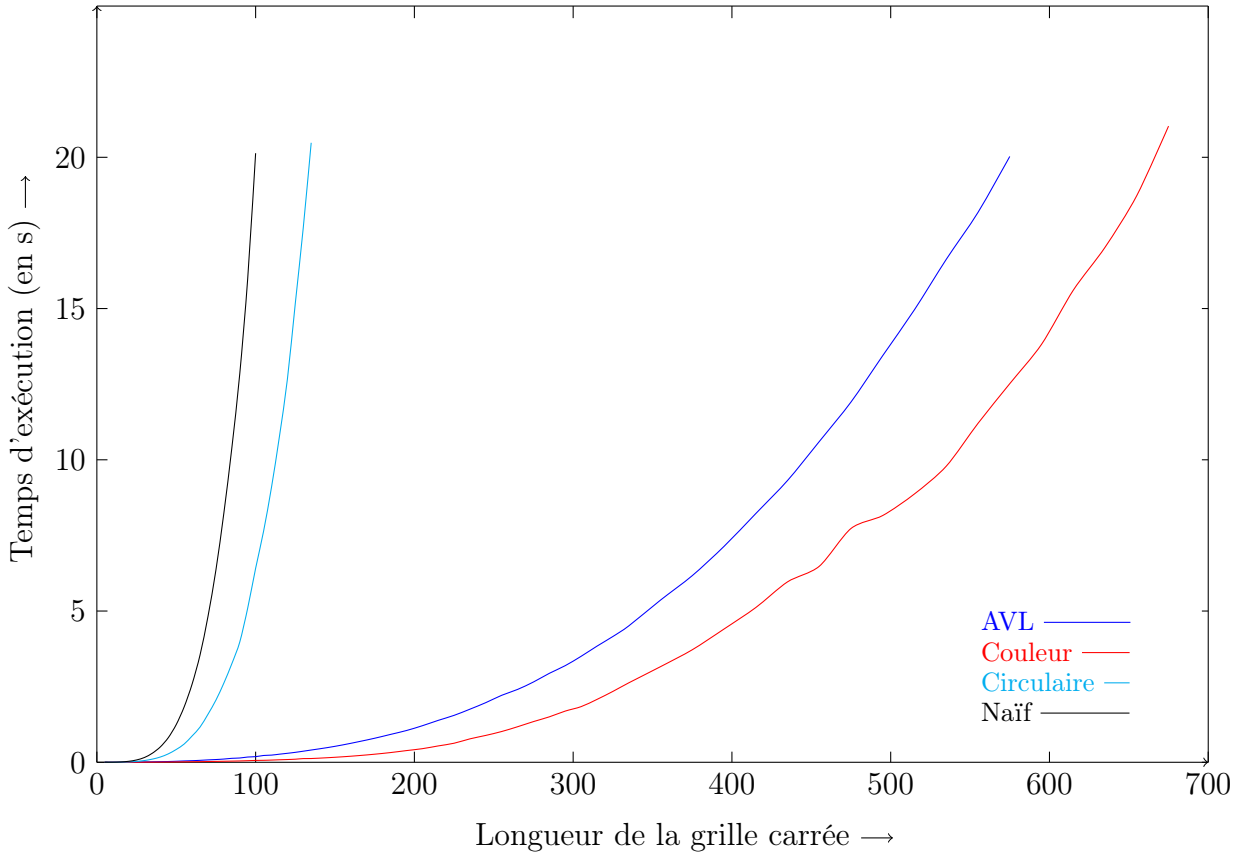
On a comparé les performances des toutes les fonctions implémentés résolvant le problème du robot trieur pour des grilles de taille variable.

Les expérimentations ont été faites avec $\alpha = n$, ce qui transforme les complexités données précédemment en $O(n^3)$ pour la version par couleur et $O(n^3 \log n)$ pour la version par AVL. On peut donc définir l'ordre suivant selon la vitesse d'exécution attendue des différentes versions de l'algorithme au plus proche :

$$\texttt{naïf} \prec \texttt{circulaire} \prec \texttt{AVL} \prec \texttt{couleur}$$

On a dressé dans le graphique ci-dessous les courbes correspondant aux temps d'exécutions de ces quatre versions. Ceci vérifie l'analyse faite auparavant.

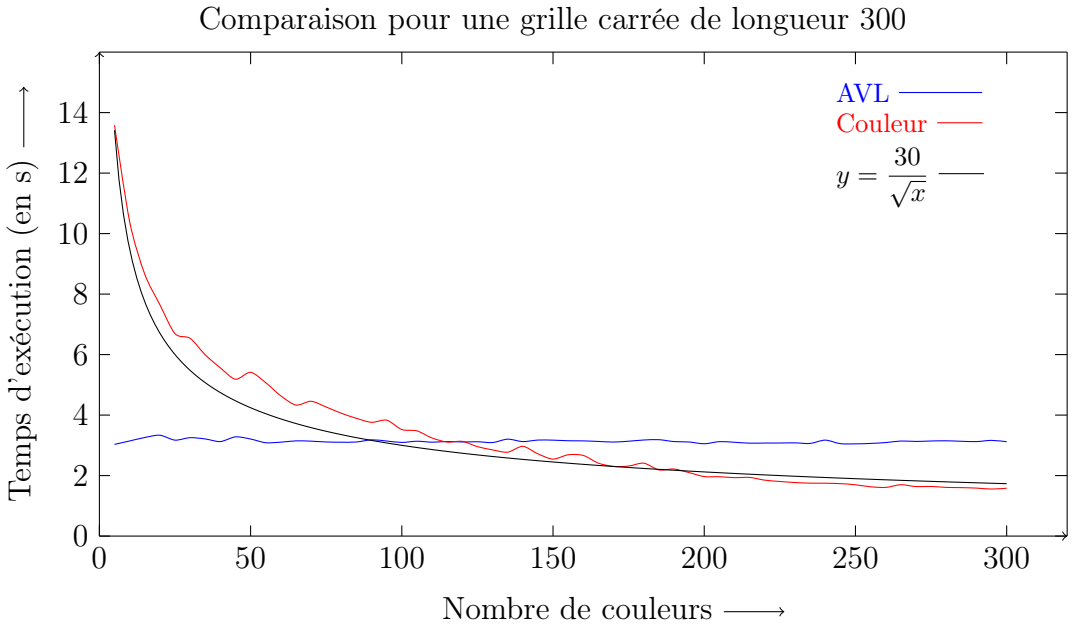
Comparaison des versions de l'algorithme au plus proche



11 Nombre de couleurs variable

On a ensuite analysé l'impact du nombre de couleurs dans la grille, i.e. du paramètre α, sur le temps d'exécution des algorithmes utilisant des structures de données additionnelles, à

savoir des listes doublement chaînées et des arbres AVL.



12 Vecteur avec une case par couleur

On a finalement testé nos algorithmes pour le cas particulier d'une grille réduite à un vecteur avec autant de cases que de couleurs.

Conclusion

...