



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

Introduction	3
I Algorithme au plus proche	4
1 Préliminaires	4
2 Version naïve	5
3 Version circulaire	6
4 Version par couleur	6
5 Version par AVL	6
II Résolution par graphes	8
6 Méthode par circuits	8
7 Vecteur avec une case par couleur	9
III Comparaison et performances	10
8 Taille de la grille variable	10
9 Nombre de couleurs variable	10
10 Vecteur avec une case par couleur	11
Conclusion	12

Introduction

Présentation

Le projet mini projet consiste en la résolution du problème du robot trieur... L'objectif de ce travail est de sensibiliser les étudiants au choix de la structure de données la plus adéquate dans le cadre d'un projet défini, ici, la gestion efficace d'un conteneur de données, au travers des mesures de temps de calcul et de la mémoire utilisée pour une telle mise en place.

Aperçu

...

L'organisation du code

Le répertoire correspondant à ce projet comporte plusieurs sous-répertoires que l'on expliquera en détail ci-dessous.

bin

Ce dossier contient les fichiers exécutables des test des fonctions implémentées.

data

Ce dossier contient les résultats des différents tests de performance des algorithmes implémentés.

img

Ce dossier contient les courbes de comparaison obtenues à partir des données du dossier **data**.

include

Ce dossier contient les fichiers d'en-tête répartis en deux sous-dossiers **lib** et **src**, chacun contenant les fichiers associés aux fichiers source des répertoires du même nom.

lib

Ce dossier contient les fichiers source fournis

Makefile

Ce fichier automatise la compilation des sources

obj

Ce dossier contient les fichiers objet correspondant aux fichiers sources et aux fichiers exécutables

report

Ce dossier contient ce document et les fichiers \LaTeX nécessaires pour sa bonne mise en forme

scripts

Ce dossier contient les commandes à transmettre à *gnuplot* pour obtenir des courbes à partir des données brutes

solutions

Ce dossier contient les solutions obtenues, i.e. les chaînes de caractères définissant les mouvements du robot trieur, pour les différentes fonctions de résolution

src

Ce dossier contient les fichiers source des fonctions de base et des fonctions de résolution du problème du robot trieur que l'on a implémenté

tests

Ce dossier contient les fichiers source des tests de fonctionnement et de performance.

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 de cette première partie.

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

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, $\text{dist}((1, j), (k, l)) = (p - 1) + q = r$. 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, $\text{dist}((m - 2, j), (k, l)) = (p - 1) + q = r$. 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. $0 < i < m - 1$:

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 $\text{dist}((h, j), (k, l)) = (p - 1) + q = r$. 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

Conclusion :

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

Fonctions auxiliaires

Les quatre fonctions ci-dessous ont servi d'interface des fonctions fournies et améliorent grandement 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.

Analyse de la complexité

Pour l'analyse de la complexité des quatre implémentations, on commence 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 apparaît sous la forme d'une 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 : cela dépend fortement du nombre de cycles 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 des pièces. Ceci consiste pour chaque pièce à chercher la case la plus proche dont la couleur du fond coïncide avec celle de la pièce.

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

2 Version naïve

Comme mentionné dans le paragraphe précédent, il suffit de calculer la complexité de la recherche des pièces pour trouver la complexité totale d'une implémentation de l'algorithme au plus proche. La recherche 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 est de l'ordre de $\Theta(n^2)$.

3 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 complexité est donc en $O(n^2)$.

Même si l'ordre de grandeur de la complexité-temps pire-cas pour ces deux implémentations est le même, on remarque que la première croît plus vite que la deuxième. Ceci est dû à la différence en la recherche d'une case adéquate : 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.

4 Version par couleur

Dans un premier temps, on détermine une borne supérieure de la complexité. En effet, le robot porte une pièce à presque tout moment du jeu et 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 pour 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 n^2 cases 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.

5 Version par AVL

La dernière version de cette algorithme repose 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 travaille avec une matrice, l'accès à un certain arbre 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 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 chacune des n^2 cases 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.

α est-il important ?

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.

6 Méthode par circuits

Préliminaires

Étant donné une instance de grille G , on définit un graphe orienté $H = (V, A)$, dénoté **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 (i, j) au sommet-case (i', j') tels que la case (i, j) est non noire et qu'elle 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 structure de données implémentant le graphe H repose 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 décidé de faire une recherche de circuits simple. Pour chaque sommet dont la case associée est non noire, on recherche le sommet correspondant à la case en haut à gauche ayant la même couleur que celle de sa pièce. Compte tenu du fait que l'ordre des sommets-case dans les listes d'adjacence suit l'ordre de la recherche dans une matrice à deux dimensions, à savoir un parcours par lignes commençant par la première, 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 tant pour refermer un circuit que pour en commencer un nouveau.

Version améliorée

Dans un deuxième temps, on a modifié légèrement la méthode précédente : la recherche du circuit suivant n'est plus naïve. Cela se traduit par 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 son parcours de la grille en moins de pas qu'auparavant.

7 Vecteur avec une case par couleur

Fonctionnement

...

Troisième partie

Comparaison et performances

Dans cette troisième partie, on vous présente les résultats des différents tests de performance réalisés.

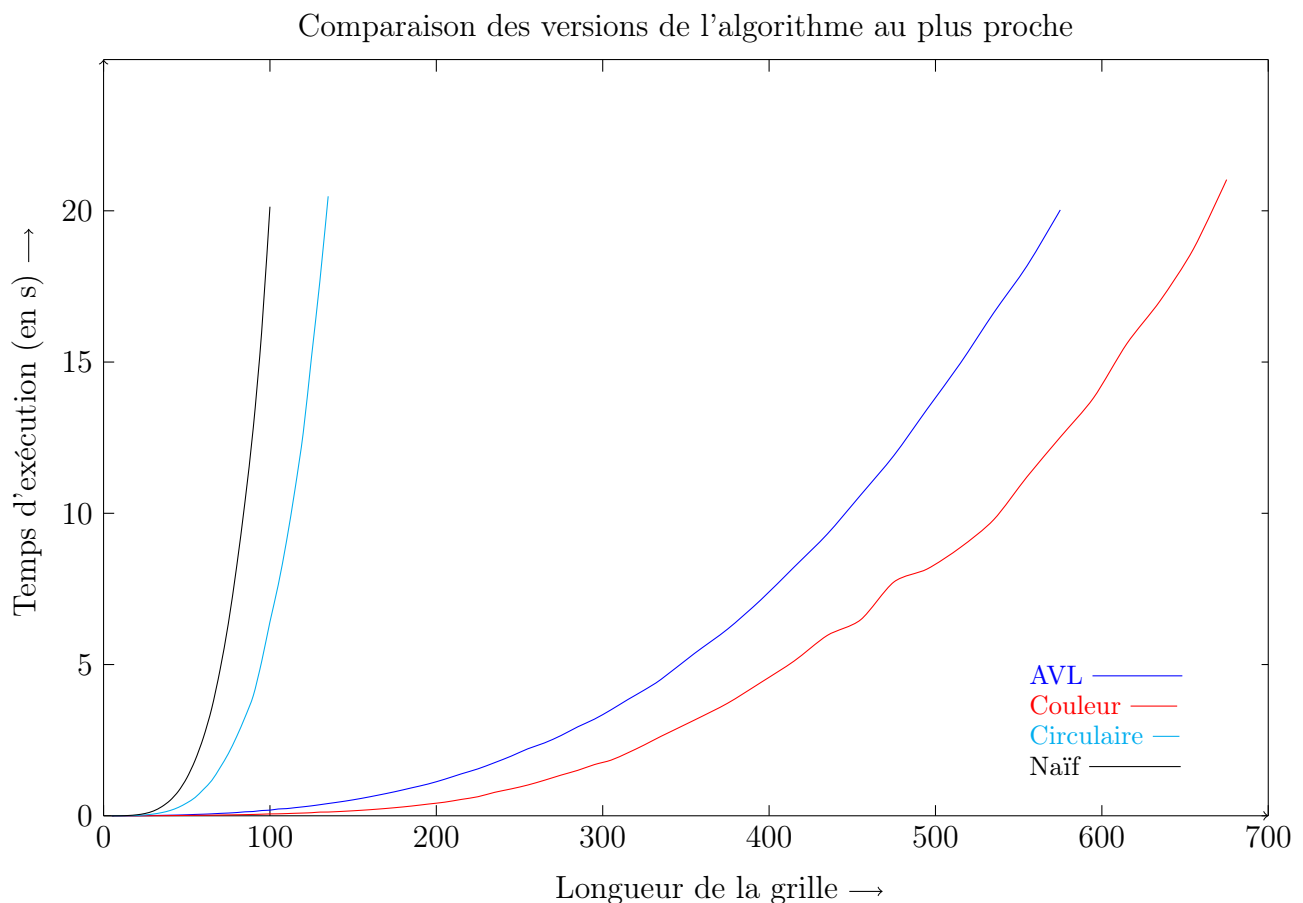
8 Taille de la grille variable

On a comparé les performances des toutes les fonctions implémentées 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 des différentes versions de l'algorithme au plus proche :

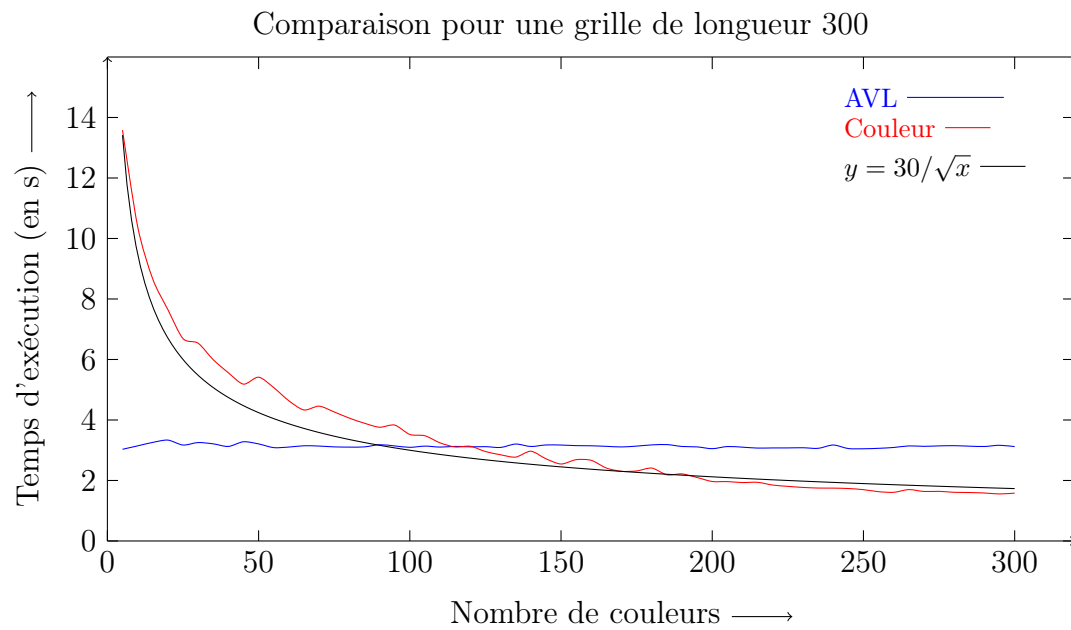
$$\text{naïf} \prec \text{circulaire} \prec \text{AVL} \prec \text{couleur}$$

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.



9 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.



10 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

...