Compte-rendu Projet Algorithmique et complexité

Perrin Matthieu - Gallay Jules

Décembre 2019

Sommaire

1 Introduction

Ce document est le rapport de notre projet d'algorithmique et complexité effectué lors du premier semestre de notre Master 1 Informatique. Pour ce projet, nous devions programmer en Ocaml une solution à un des problèmes proposés et nous avons choisi la résolution du taquin. Nous allons tout d'abord développer la problématique qui entoure ce projet, puis nous expliquerons l'approche choisie. Nous verrons ensuite les problèmes que nous avons pu rencontrer et les solutions que nous avons mis en œuvre pour les résoudre. On complétera le rapport avec des images de différentes situations afin de mieux comprendre notre raisonnement.

2 Problématique

Ce projet consiste à créer un algorithme de mélange puis de résolution d'un jeu de taquin. Il doit fonctionner pour plusieurs tailles de taquin. La première étape est donc de modéliser de taquin, puis de le mélanger aléatoirement et enfin nous devrons le résoudre en testant le temps d'exécution de l'algorithme pour différentes tailles de taquin. On doit déterminer à partir de quelle taille le temps d'exécution du programme est trop lent.

3 Approche

Nous avons choisi de représenter le taquin par une matrice d'entier. Le nombre 0 représente la case noire et les autres nombres sont les cases blanches. Notre programme crée deux taquins, le premier sera mélangé puis résolu et le deuxième servira de base au premier pour nous permettre de tester si les cases sont correctement placées.

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

taquin

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

taquin_init

Pour afficher le taquin nous utilisons la librairie graphics.cma et nous avons créé une fonction **remp_graph** qui prend en paramètre la taille du taquin et qui permet d'afficher l'état actuel du plateau dans une interface graphique

Nous avons ensuite dû mélanger le taquin, contrairement à ce qu'on pourrait croire dans un premier temps, il ne suffit pas de placer les entiers de la matrice dans un ordre aléatoire pour mélanger le plateau. Pour que l'on puisse résoudre un taquin mélangé, il faut partir d'un taquin résolu et faire des mouvements aléatoires. On souligne que l'on n'utilise pas de méthode où l'on fait les mouvements inverses pour revenir au taquin résolu.

Notre fonction **echange_case** s'occupe de cette étape, elle nous permet de choisir le nombre de mouvements que l'on veut faire et elle génère des entiers aléatoires pour décider des mouvements ce qui nous donne un plateau totalement aléatoire après le mélange. Les mouvements sont gérés avec la fonction **deplacer_case** qui prend en paramètre les coordonnées de la case à déplacer et un entier qui indique la direction. Bien entendu cette fonction ne doit être appelée qu'avec les coordonnées de la case noire du taquin.

Une fois ces quelques fonctions créées, nous avons pu nous attaquer au cœur du problème : la résolution du taquin. Très vite nous avons identifié quelques variables globales et quelques fonctions de base qui seraient utiles tout au long du projet. Nous avons donc des variables **icase** et **jcase** qui stockent les coordonnées de la case en cours de traitement, **ivide** et **jvide** qui stockent les coordonnées de la case noire, **ligneactuelle** et **colonneactuelle** qui nous permettent de connaître l'indice de la case à placer et quelques autres variables spécifiques à certaines situations sur lesquelles nous reviendrons plus tard.

Parmi les fonctions de base qui seront utilisées tout au long du projet, nous retrouvons **search case** qui permet de mettre à jour **icase** et **jcase** avec les

coordonnées de la case passée en paramètre. La fonction **getposblack** nous permet de mettre à jour **ivide** et **jvide** avec les coordonnées de la case noire, et la fonction **deplace** qui prend en paramètre un entier indiquant la direction nous permet de déplacer la case noire grâce à **deplacer_case**, de mettre à jour les variables contenant sa position grâce à getposblack puis de mettre à jour l'affichage grâce à **remp_graph**.

4 Méthode

Nous allons maintenant présenter la méthode de résolution du taquin que nous avons choisi d'implémenter. Notre méthode traite le taquin ligne par ligne, la première étape pour un taquin de taille n est de placer les n-2 cases de la première ligne. L'explication de la méthode sera complétée de plusieurs schémas d'un taquin de taille 4.

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

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

Quand il ne reste plus que 2 cases à placer sur la ligne, on place la case n-1 à l'emplacement final de la case n, et la case n doit se retrouver juste en dessous de la case n-1.

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

Il ne reste alors plus qu'a faire pivoter ces deux cases pour avoir une ligne complète.

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

Il faut ensuite répéter ces étapes jusqu'à ce qu'il ne reste plus que 2 lignes mélangées sur le taquin. A ce stade, la méthode de placement des cases change, on travaille cette fois -ci sur les colonnes. On utilise le même principe que pour le placement des 2 dernières cases d'une ligne, mais on l'applique à une colonne. En considérant le taquin comme un tableau d'élément de coordonnées (i,j), il faut placer la caser de coordonnées (i+1,j) à l'emplacement (i,j) et la case de coordonnées (i,j) doit se retrouver en (i,j+1), juste à droite de la case précédente.

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

On fait ensuite pivoter les deux cases pour obtenir la première colonne.

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

Il faut répéter ce processus jusqu'à ce qu'il ne reste que les 3 dernières cases non placées, il suffit ensuite de les faire pivoter pour résoudre le taquin.

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

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

5 Solution

La première étape a été de réussir à mettre les n-2 cases de la première ligne à leur place quelle que soit la configuration du taquin de départ. Durant tout notre raisonnement, le coin en haut à gauche du taquin représente les coordonnées (0,0) et celui en bas à droite les coordonnées (n-1,n-1) avec n la taille du taquin.

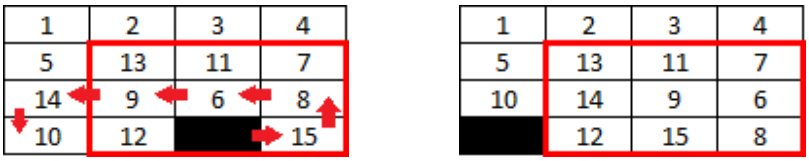
La marche à suivre que nous avons trouvé grâce à nos tests est plutôt simple. Dans un premier temps, il faut mettre la case à traiter sur le contour extérieur défini par la ligne et la colonne en cours de traitement ainsi que les côtés droits et bas du taquin. Une fois la case sur cette bordure, il suffit de la faire avancer en suivant le contour pour qu'elle finisse par arriver à sa place. On incrémente ensuite la variable **colonneactuelle** et on recommence jusqu'à ce qu'il ne reste que 2 cases non placées sur la ligne.

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

Le contour rouge correspond à la zone où l'on fait tourner la case 6 pour la placer correctement. Ici **colonneactuelle** et **ligneactuelle** sont égales à 2.

Notre code fonctionne de la manière suivante, si la case est déjà sur le contour (le contour rouge dans notre exemple ci-dessus) on la fait avancer directement pour la placer, sinon on appelle **get_on_side** qui place la case à traiter sur l'extérieur du taquin et non le contour. Ensuite on se place en dessous de la case grâce à la fonction **get_under**, elle prend en paramètre la case à traiter et permet de placer la case noire juste en dessous pour peu que la case à traiter ne soit pas déjà sur le contour.

Ensuite quand la case noire est en dessous de la case à traiter, on utilise la fonction **rotate** pour placer la case à traiter sur le contour. En considérant que la case à traiter se trouve en (i,j), cette fonction **rotate** fait tourner la case noire dans le sens inverse des aiguilles d'un montre sur les lignes i et i+1 jusqu'à ce que la case à traiter se retrouve sur le contour qui nous intéresse.



Chemin parcouru par la case vide après utilisation de la fonction **rotate**

Pour finir, on déplace la case à traiter le long du contour jusqu'à ce qu'elle soit placée. On répète cette opération sur les n-2 première cases de la ligne.

Après avoir mis dans le bon ordre les premières cases de la ligne, on traite les deux dernières cases différemment du reste. On va utiliser d'autres fonctions comme **place_prepa_fin**, **place_coin_haut**, **placeFin_debug** et **placeFin_debug2**.

Pour une meilleure compréhension on va continuer de prendre l'exemple sur un taquin de taille 4x4. Pour se situer sur le taquin, il faut que les deux dernières cases de chaque ligne (3 et 4 pour la première ligne) soient préalablement positionnées dans une position spécifique. Comme vu plus haut, si nous plaçons d'abord la case 3 sans s'occuper de la case 4, celle-ci sera déplacé lors de la mise en place du 4.

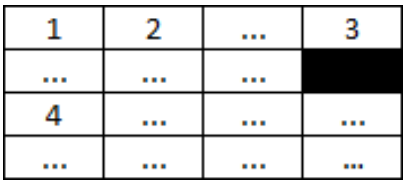
La fonction **place_prepa_fin** sert donc à préparer les deux cases pour ensuite les placer correctement. Pour cela, il est nécessaire que la case 3 se trouve à la position finale de la case 4 et que la case 4 se trouve juste en dessous, ensuite il suffit de faire une rotation pour bien les placer.

1	2	...	3
...	...	...	4
...	...	...	...
...	...	...	

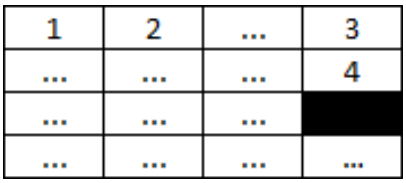
Dans cette fonction on va donc récupérer la position de la case que l'on traite et on va placer cette case sur une extrémité du taquin grâce à la fonction **get_on_side**. Ensuite on cherche à la placer sur la case qui se trouve en dessous de la case 4 finale. On utilise la fonction tourne jusqu'à que ce soit le cas. Pour finir on se place en dessous de cette case car nous repartirons de cette position ensuite.

1	2	...	...
...	...	...	3
4	...	...	
...	...	...	...

A partir du moment où cette case est placé comme nous le voulons, nous utilisons la fonction **place_coin_haut**. Cette fonction vérifie si la case est au bon endroit et fait un déplacement brut, c'est-à-dire que la case vide fera toujours ce déplacement : Gauche, Haut, Haut, Droite et Bas. Avec ce déplacement, la case 3 monte d'une case et se situe à l'endroit voulu pour la préparation.



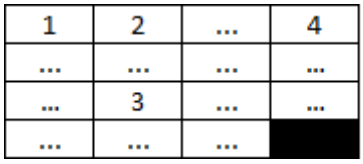
On recommence avec la case 4 pour que celle-ci se place en dessous du 3. La fonction **place_prepa_fin** est donc seulement nécessaire. On se retrouve au final dans la situation où la case 3 et 4 sont l'un au-dessus l'autre. La situation est respectée et il suffit de faire une rotation pour les placer définitivement.

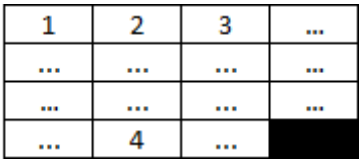


La fonction **place_coin_haut** est rappelée pour placer la case 4 dans sa position finale, en même temps la case 3 est décalé à gauche ce qui signifie que cette case est aussi à la bonne place.

Cette série de fonction est répétée jusqu'à ce qu'il ne reste plus que 2 lignes à résoudre sur le taquin. Mais concernant le placement des deux dernières cases, plusieurs situations produisent des erreurs avec cette méthode, il a donc fallu nous adapter.

Il y a 5 situations précisément où il faut utiliser la fonction **placeFin_debug** ou **placeFin_debug2** avant d'utiliser les autres fonctions. La première situation est lorsque la case 3 est déjà à la bonne place mais que le 4 n'y est pas. Il y a aussi la situation inverse, le 4 est placé mais pas le 3.





Pour régler ce problème, la fonction **placeFin_debug** est utilisé. Nous plaçons la case vide en dessous de la case rentrée en paramètre puis un déplacement brut est effectué s'il s'agit de la première situation ou de la deuxième situation. Les déplacements pour la situation 1 sont Haut, Droite et Gauche ; les déplacements pour la situation 2 sont Haut, Gauche et Bas.

A la fin de cette fonction on retrouve un **get_under** et un déplacement Haut. Cette combinaison permet de régler 2 autres des 5 situations qui posent

problèmes. Il s'agit de la situation où la case 3 est bien placé et que la case 4 est juste en dessous, et de la situation où la case 4 est bien placé et la case 3 en dessous de celle-ci.

1	2	...	4
...	...	...	3
...	...	...	...
...	...	...	

1	2	3	...
...	...	4	...
...	...	...	...
...	...	...	

La dernière situation que l'on veut gérer est quand l'avant dernière case se trouve à la place finale de la dernière case et inversement. Pour cela on utilise **placeFin_debug2** qui va placer la case vide en dessous de la dernière case (le 4 dans notre exemple) et qui va faire une rotation Haut, Droite, Bas et Gauche puis appeler **placeFin_debug** pour les 2 cases. Contrairement à l'autre fonction debug, celle-ci n'est utilisé seulement si cette situation est présente sur les lignes 1 à n – 3.

1	2	4	3
...	...		...
...	...	...	...
...	...	...	...

Une fois les n-2 première lignes placées, on doit placer les cases des dernières lignes, colonne par colonne. Pour commencer, on appelle la fonction **prepare_first** qui prend en paramètre la case à traiter et qui va placer la case se trouvant juste en dessous sur le taquin résolu à son emplacement. Pour être plus concret, sur un taquin de taille 4, cette fonction va placer la case 13 à l'emplacement final de la case 9.

1	2	3	4
5	6	7	8
13	9	...	...
...		...	...

Cette fonction consiste simplement à une rotation de la case noire sur les 2 dernières lignes en allant du bout du taquin jusqu'à la colonne définie par **colonneactuelle** tant que la case n'est pas à sa place.

Ensuite, on utilise la fonction **prepare_col** qui va venir placer la case à traiter jusqu'à la droite de son emplacement final. Dans notre exemple cela va mettre la case 9 à droite de la case 13 (voir le schéma ci-dessus).

Cette fonction ne peut opérer si la case 9 se trouve en dessous de la case 13, il y a donc une vérification dès le début de la fonction. Si c'est le cas, la

case noire se place à droite de la case 13 et effectue un mélange des cases en suivant les directions Gauche, Bas, Droite, Haut, Droite, Bas puis la fonction **prepare_first** est rappelée. Dans la plupart des situations où ce placement est présent, un seul mélange suffit à débloquer la situation.

Pour finir, la fonction **prepare_col** va venir effectuer une rotation pour placer les deux cases de la colonne en cours de traitement.

1	2	3	4
5	6	7	8
9		...	...
13	...	...	...

On répète cette opération jusqu'à ce qu'il ne reste que 3 cases non placées, à partir de là on utilise la fonction **last_piece** qui va placer la case d'indice (n-2,n-2) à son emplacement puis qui va mettre la case noire en (n-1,n-1) ce qui va finir de résoudre le taquin.

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

6 Conclusion

Au terme de ce projet, nous avons réussi à créer un algorithme qui résout n'importe quel taquin. En ce qui concerne le temps d'exécution, il augmente exponentiellement avec la taille du taquin à cause des nombreux parcours de la matrice qui sont effectués pendant la résolution. Bien entendu cela dépend aussi de la puissance de la machine utilisée. Sur nos ordinateurs personnels, nous pouvons aller jusqu'à des taquins de taille 18 avec un temps d'exécution égal à environ 20 minutes, et un taquin de taille 7 se résout en environ 30 secondes. Sur les machines de la fac par contre le taquin de taille 7 peut prendre 2 minutes à se résoudre et le taquin de taille 10 prend 15 minutes.

7 Bibliographie

https ://fr.wikipedia.org/wiki/Taquin
http ://maths.amatheurs.fr/index.php ?page=taquin