UNIVERSITE D'OTTAWA

Dans le cadre du cours de CSI 2510

Problème du Traversier

Dans le cadre du cours de
Numéro d'étudiant :
300063565

Amirath SOUHOUIN
Robert Laganière

Introduction

Le problème du traversier consiste à trouver la meilleure combinaison pour placer des véhicules alignés dans deux files différentes. Il correspond à un problème de knap-sack aussi mais en deux files.
Il nous as été demandé dans ce projet de résoudre le problème du traversier à l'aide d'un tableau à deux dimensions et d'une table de hachage. Ainsi, l'objectif était la comparaison des deux structures de données pour la résolution de ce problème avec la méthode du backtracking. Nos résultats sont listés ci-dessous.

Partie A

La solution que nous avons proposée passe les tests basiques proposées(voir fichier output dans le package Bigtable) mais ne passe pas les tests plus complexes proposées par OnlineJudge. Celà est sûrement dû au fait que il y a des aspects de la logique du problème que notre programme n'a pas pu couvrir.
En outre, le temps d'execution que nous avons pu estimé(relevé sur la machine) avant l'appel de la méthode faisant le backtracking et juste après est d'environ : 0.5 secondes pour les petites données et jusqu'à 2secondes pour les plus grandes. Les dernières lignes de chacun de nos fichiers d'exécutions affichent le temps en nanosecondes. La méthode utilisée pour enrégister le temps en(figure 2).

Nous avons aussi la figure 3 qui présente juste après le résultat sur OnlineJudge.

```
long startTime = System.nanoTime();
backtrackSolve(0,L);
long endTime = System.nanoTime();
long duration = (endTime-startTime);
```

Figure 1

Wrong answer	JAVA	0.520
Wrong answer	JAVA	0.460
Wrong answer	JAVA	0.490
Wrong answer	JAVA	0.490

Figure 2

Partie B

Pour cette partie nous avons toujours comme réponse le WA. Mais l'estimation du temps que nous avons relevé est de : 10^-9 s pour les petites données à 1 secondes pour les plus grandes. Le temps d'exécution est à la dernière ligne de chacun de nos fichiers output. On constate que l'exécution est vraiment rapide par rapport à la Bigtable. Aussi, l'espace mémoire est moindre par 2. Nous avons obtenu les résultats attendus pour les données proposées par le professeur.

Verdict	Language	Run Time	Submission Date
Wrong answer	JAVA	0.940	2020-12-08 04:16:44

n

Partie C

Notre table de Hachage est la Hashtable de la librairie Java: https://docs.oracle.com/javase/8/docs/api/java/util/Hashtable.html. Notre table de hachage prend juste des éléments au départ. Nous ne fixons pas une table initiale. Donc notre table de hachage a juste pour objectif d'enregistrer toutes les possibilités que nous avons visiter dans notre implémentation. En outre, la façon dont nous avons fait notre logique est la suivante :
Notre méthode backStrack prend en paramètre la voiture courante et l'espace actuel disponible et retourne un entier. Les conditions pour pouvoir arrêter notre récursivité sont soit il n'y a plus d'espace disponible à gauche, soit il n'y a plus de voitures à faire passer soit la taille de la première voiture est largement supérieur à la taille du bateau et dans ces cas la méthode backStrack(currK,currS) = 0 ;
Nous avons aussi, une autre structure de donnée un tableau à 2 dimensions dp[nombre de voiture + 1][L+1] mais elle sert simplement à savoir quelles sont les endroits nous avons déjà visité. Lorsqu'un endroit n'est pas visité il est mis à -1.
Pour changer les états on vérifie seulement si il y a de l'espace à gauche avec la variable int v1 = int v1 = backtrackSolve(currK+1,currS) . L'avantage de notre méthode est que nous avons juste à mettre pour le côté gauche et dans la méthode main on vérifie les voitures qui n'ont pas pu être placé et on les places au droit. Donc on fait notre travail en un temps.