# Projet LSINF1121 - Algorithmique et structures de données

-

# Rapport intermédiaire Mission 3

Groupe 3.2


Boris Dehem (5586-12-00)

Sundeep Dhillon (6401-11-00)

Alexandre Hauet (5336-08-00)

Jonathan Powell (6133-12-00)

Mathieu Rosar (4718-12-00)

Tanguy Vaessen (0810-14-00)



Année académique 2014-2015

## Questions et réponses

### Question 1

Énoncé
Réponse

### Question 2

Énoncé
Réponse

### Question 3

Énoncé
Réponse

### Question 4

Énoncé
Réponse

### Question 5

Comment obtenir la liste des entrées mémorisées dans un Map implémenté par une table de hachage ? Cette implémentation conviendrait-elle pour un Map ordonné ? Quelles seraient les complexités temporelles des méthodes spécifiques au Map ordonné dans ce cas ? Si l'on dispose d'une liste triée de toutes les entrées d'un dictionnaire et que l'on suppose cette liste fixe (aucun ajout ou retrait n'est permis), est-il intéressant de mémoriser ces entrées dans une table de hachage ? Justifiez votre réponse. Quelle est la complexité spatiale d'une table de hachage ?

La méthode entrySet() permet d'obtenir une liste des entrées clé-valeur. Elle parcourt la table de hachage, et pour chaque élément non vide, elle crée un élément dans un tableau. Cette implémentation convient pour un Map ordonné. La complexité temporelle pour cette méthode spécifique au Map ordonné est 0(n). Si l'on dispose d'une liste triée fixe de toutes les entrées d'un dictionnaire, il n'est pas intéressant de la stocker dans une table de hachage, car on peut la stocker dans un tableau ordonné, ce qui prendrait moins d'espace car on ne gaspillerait pas d'espace pour éviter les collisions. De plus, une table de hachage mélangerait toutes les données du dictionnaire. La complexité spatiale d'une table de hachage est 0(1) car ajouter ou retirer un élément de la table ne change pas sa taille (sauf au cas exceptionnel où on décide d'augmenter la taille de la table car le « load factor » est trop élevé.

### Question 6

Qu'entend-on par la notion de collision dans une table de hachage ? Les collisions ont elles une influence sur la complexité des opérations ? Si oui, quelle(s) opération(s) avec quelle(s) complexité(s), sinon précisez pourquoi. Quelles sont les techniques utilisées pour gérer les collisions ? Peut-on, grâce à ces techniques, éviter les collisions ?

Une collision a lieu lorsque la fonction de hachage envoie une clé à une adresse déjà occupée. A cause des collisions, les foncions get(), put() et remove() pourraient théoriquement s'exécuter en 0(n) dans le pire des cas, car si tous emplacements qu'on essaie sont occupés, il faudra tous les parcourir avant d'en trouver un libre (dans le cas du « open adressing » ou bien il faudra parcourir tous les éléments qui ont déjà été stockés à cette adresse (dans le cas du « separate chaining ») En réalité, on minimise les collisions, de sorte à ce que ces fonctions s'exécutent en 0(1) la majorité du temps. La technique du « separate chaining » consiste à avoir la possibilité de stocker plusieurs éléments dans le même emplacement d'une table de hachage, par exemple en les liant à l'aide d'une liste chainée. La technique du « Open adressing » consiste à trouver un autre emplacement dans la table de hachage que celui renvoyé par la fonction de hachage. Ces techniques parmettent de gérer les collisions, mais pas de les éviter. On peut par d'autres moyens les minimiser, mais jamais complètement les éviter.

### Question 7

Énoncé

Réponse

### Question 8

Énoncé

Réponse

### Question 9

Énoncé

Réponse