**INF2010 - Structures de données et algorithmes**

**Automne 2018**

**Travail Pratique 2**

**Fonctions de hachage et tables de dispersement**

**Lynn Chararbsissy 1832027**

**Elie Rouphael 1829529**

# Exercice 1 : Trouver un sous-ensemble (1 point)

En utilisant le hashSet la complexité sera de O (m + n) ou m est la taille du premier tableau et n du deuxième tableau. Comme on utilise deux boucles for à la suite et non l’une dans l’autre la complexité est l’addition des tailles et non pas leur produit.

# Question 1: Linéaire ou pas?

1. Dans la classe QuadratiqueSpacePerfectHashing nous avons les conditions suivantes :

* Espace mémoire de taille m = n2, où n est le nombre d’éléments.
* m < p, 0 < a < p et 0 ≤ b < p.

Alors la taille maximale de la QuadratiqueSpacePerfectHashing est p et le nombre maximal d’élément est √p.

1. Pour LinearSpacePerfectHashing nous avons les conditions suivantes :

* La taille m = n < p, 0 < a < p, 0 ≤ b < p, où n est le nombre d’éléments.

Alors la taille maximale de la LinearSpacePerfectHashing est p et le nombre d’élément maximale est p.

De plus :

((a·x.hashCode() + b) mod p) si on prend x.hashCode() comme p-1 on et b = a = p-1 et m la taille maximale = p - 1

(((p-1) \* (p-1) + p-1) %p) %m = ((p^2) % p) % m = p.

Alors on obtient un bon dispersement des éléments entre 0 et p. Mais dans la fonction nous avons p^2 avec p = 46337. 46337^2 = 2147117569 < 2^31 -1(valeur maximale d’un entier de 32 bits). Si on prend le prochain nombre premier 46349, son carrée dépasse 2^31 -1. Ceci va causer un Overflow et des erreurs a l’exécution.

De plus on utilise un nombre premier pour avoir une bonne distribution de données.

Enfin on remarque que d’après le graphe suivant, que la taille nécessaire pour une LinearSpacePerfectHashing augmente linéairement en fonction des éléments présent, cependant la taille de la QuadratiqueSpacePerfectHashing augmente quadratiquement (n^2).

