- Compléments sur les Listes
- Fonction de hachage
- Propriété
- Table de Hachage
- Collision
- Analyse

Fonction de hachage

•Une **fonction de hachage** (**hash**) est une fonction qui associe, à un grand ensemble de données, un ensemble beaucoup plus petit (de l'ordre de quelques centaines de bits) qui est caractéristique de l'ensemble de départ . Cette propriété fait qu'elles sont très utilisées en informatique, en particulier pour accéder rapidement à des données grâce aux Tables de hachage (ou **hash tables** en anglais).

•En effet, une fonction de hachage permet d'associer à une chaîne de caractères un entier particulier. Ainsi, si nous connaissons l'empreinte des chaînes de caractères stockées, nous pouvons rapidement vérifier si une chaîne se trouve ou non dans cette table (en **O(1)** si la fonction de hachage est suffisamment bonne). Les fonctions de hachage sont aussi extrêmement utiles en **cryptographie**.

- Compléments sur les Listes
- Fonction de hachage
- Propriété
- Table de Hachage
- Collision
- Analyse

Fonction de hachage

•On utilise une fonction H de l'ensemble des clés (souvent des chaînes de caractères) dans un intervalle d'entiers. Pour une clé x, $H(x)$ est l'endroit où l'on trouve x dans la table. Tout se passe parfaitement bien si H est une application injective. Pratiquement, on ne peut arriver à atteindre ce résultat. On tente alors de s'en approcher et on cherche aussi à minimiser le temps de calcul de $H(x)$. Ainsi un exemple de fonction de hachage est

$H(x)=(x[1]B^{l-1}+ x[2]B^{l-2} + \ldots + x[l]) \bmod N$

•On prend d'habitude $B=128$ ou 256 et on suppose que la taille de la table N est un nombre premier.

Pourquoi?

- Compléments sur les Listes
- Fonction de hachage
- Propriété
- Table de Hachage
- Collision
- Analyse

Fonction de hachage

• Les multiplications par des puissances de 2 peuvent se faire très facilement par des décalages, puisque les nombres sont représentés en base 2. Cette opération est nettement plus rapide que la multiplication par un nombre arbitraire.

•Quant à prendre N premier, c'est pour éviter toute interférence entre les multiplications par B et la division par N.

•En effet, si par exemple $N=B=256$, alors $H(x)=x[l]$ et la fonction H ne dépendrait que du dernier caractère de x. Le but est donc d'avoir une fonction H de hachage simple à calculer et ayant une bonne distribution sur l'intervalle $[0..N-1]$.

- Compléments sur les Listes
- Fonction de hachage
- Propriété
- Table de Hachage
- Collision
- Analyse

Propriétés mathématiques

Pour une fonction de hachage H, il faut que : H(x) ≠ H(y) implique x ≠ y et H(x) = H(y) implique probablement x = y.

Si l'ensemble des valeurs de x est plus grand que l'ensemble des valeurs prises par H, cette seconde propriété est difficile à obtenir.

En fait, la probabilité dépend grandement du domaine d'application de cette fonction de hachage. Pour les fonctions de hachage cryptographiques, utilisées pour stocker les mots de passe ou assurer l'authenticité des données (empreinte) , cette propriété s'écrit : pour tout x dont on connaît le hachage H(x), alors il est très difficile de calculer un y tel que H(x) = H(y).

- Compléments sur les Listes
- Fonction de hachage
- Propriété
- Table de Hachage
- Collision
- Analyse

Table de Hachage

En général, l'univers des clés est très grand alors que le nombre de clés présentes dans le conteneur est petit par rapport au nombre de clés possibles. On utilise alors une fonction de hachage qui associe à une clé donnée un entier de *0* à *m*. on range alors la clé au rang *h(cle)* dans la table.

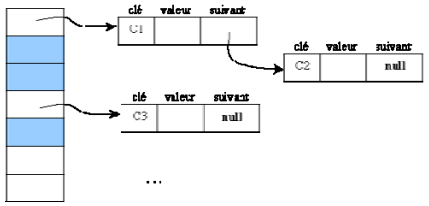
Le problème de cette technique est que plusieurs clés peuvent avoir le même indice par la fonction de hachage : on parle alors de collision

Résolution des collisions par chaînage

Chaque élément du tableau est une référence à une liste chaînée des entrées dont les clés ont même valeur par application de la fonction de hachage.

On définit alors le **facteur de remplissage α** comme étant le rapport de *n* nombre d'éléments présents dans la table hachée sur *m* taille de la table hachée

- Compléments sur les Listes
- Fonction de hachage
- Propriété
- Table de Hachage
- Collision
- Analyse



Analyse de la table hachée avec chaînage

Dans le pire des cas : toutes les clés se retrouvent dans le même élément du tableau, alors le comportement est le même que pour une liste chaînée.

Une recherche qui échoue prend un temps de l'ordre de 1+ **α**. Il faut parcourir une des *m* listes jusqu'à la fin, or ces listes ont une taille moyenne égale à **α** est donc de l'ordre de 1+ **α**.

Une recherche qui réussit prend un temps de l'ordre de 1+ **α**.

Si la taille de la table est proportionnelle au nombre d'éléments présent dans la table, alors les opérations d'ajout, de recherche ou de suppression se font en temps constant.

- Compléments sur les Listes
- Fonction de hachage
- Propriété
- Table de Hachage
- Collision
- Analyse

Retour sur le passage de paramètres par variable (Liste insertion en tête)

```c
#include <stdlib.h>
#include <stdio.h>

struct cellule{
  int valeur;
  struct cellule *suivant;
} cellule, *liste;

void inserer(int element, liste *Q){
  liste L;
  L = (liste)malloc(sizeof(cellule));
  L->valeur = element;
  L->suivant = *Q;
  *Q=L;
  return;
}
int main(){
  liste L=NULL;

  inserer(4,&L);
  inserer(7,&L);
  inserer(3,&L);
  inserer(8,&L);

/* ... */

return 0;}
```

```c
#include <stdlib.h>
#include <stdio.h>

struct cellule{
  int valeur;
  struct cellule *suivant;
} cellule, *liste;

void inserer(int element, liste *Q){
  liste L;
  L = (liste)malloc(sizeof(cellule));
  L->valeur = element;
  L->suivant = *Q;
  *Q=L;
  return;
}
int main(){
  liste *L;
  L=(liste *)malloc(sizeof(liste));
  *L=NULL;

  inserer(4, L);
  inserer(7, L);
  inserer(3, L);
  inserer(8, L);

/* ... */

return 0;}
```