ÉCOLE NATIONALE SUPÉRIEURE D'ARTS ET MÉTIERS
UNIVERSITÉ HASSAN II DE CASABLANCA

Algorithmique Avancée

Structures de données Probabilistes

Animé par : Dr. ibrahim GUELZIM

Email : ib.guelzim@gmail.com

Sommaire

- Rappels
  - Introduction et notions générales
  - Analyse et conception d'algorithmes
  - Complexité d'algorithmes classiques : 3 Tris de tableaux, 2 recherches dans un tableau, Schéma de Hörner
  - Preuves d'algorithmes
- Autres algorithmes de tri :
  - Tri par fusion
  - Tri par Tas
- Complexité moyenne :
  - Application au Tri rapide
  - Structures de Données Probabilistes :
    - Notions sur les Tables de Hachage et Fonctions de Hachage,
    - Bloom Filter,
    - Count Min Sketch
- Programmation dynamique
- Traitements de chaines de Caractères :
  - Recherche de chaine de caractères
  - Compression de données

Rappel : Structure de Données (SD)

- Une SD sert à **stocker** et **organiser** les données
- Objectif : **Faciliter** les opérations sur les données
  - Trouver un élément
  - Insérer un nouvel élément
  - Modifier des données
  - Supprimer un élément
  - Faire des traitements sur Ttes les données
  - Réorganiser : trier, …
- SD les plus utilisés :
  - Tableaux / Listes
  - Tables de Hashage
  - Arbres, Graphes, …

Rappel : Structure de Données (SD)

- Représentation des données : Structures contenant des données hétérogènes
- Expl : Etudiant :

  Id_Etu : Entier
  Nom_Etu : String
  Notes_Etu : Liste de Nombres

- Python → Class
- Expl python : List

```
Lst = [1 4 63 -9 10]
Lst.append(20)  # Lst devient [1 4 63 -9 10 20]
Lst.pop()
X = Lst[2]
for elt in Lst:
    print(elt)
```

- https://docs.python.org/fr/3/tutorial/datastructures.html

Tableaux dynamiques

- Tableaux en python : liste qui ne contient qu'un seul type de données
- Exemple utilisation :

```
import array as arr
a = arr.array('i', [1, 6, 7, 12] ) # 'i' pour les entiers
del a[3]
print(a) # affiche : array('i', [1, 6, 7])
```

- Plus simple de manipuler des listes que des tableaux
- **Pour faire simple,** nous allons confondre dans ce cours Tableau et Liste

- Tableau dynamique :
  - Réservation dans la RAM d'une taille T initiale
  - Après remplissage du tableau du taux a.T : extension automatiquement T = 2T
  - T et a sont des caractéristiques du langage

Tableaux dynamiques

- Insérer d'un élément à une position i d'un tableau
  - Exemple :

1	2	5	12	9	8				

  - Nombre (maximal) d'opérations ~ n-i opérations de décalage
  - Complexité (Pire des cas : insérer à la tête / début) : O(n)

- Supprimer un élément : de même Complexité O(n)

Dictionnaire (Python)

- Dictionnaire en Python :
  - type de données
  - permet d'associer une valeur (simple, ou composée) à une **<u>clé</u>**. (clé , val)
  - Analogie avec un dictionnaire linguistique : accéder à une définition / mot

- Opérations sur dictionnaires (Exemples):
  - Trouver une Val / clé
  - Aj / Supp / Rempl (clé , val)

Table de Hashage (TH)

- SD qui permet de :
  - Éviter les inconvénients des SD linéaires
    - Exemple : recherche, insertion ou suppression à un coût O (n)
  - Associer une clé / valeur(s) ; tableau associatif
  - Groupement d'éléments sans ordre éventuel / clés

- Objectif :
  - Faire la recherche d'un élément de la TH dans un <u>temps moyen</u> proche de O(1)
  - Comment ?
    - Réaliser l'association clé/valeur(s) via des fonctions de Hachage (FH) (cf plus loin)

Table de Hashage (TH)

- Tâches souhaitées : Trouver, Insérer, Supprimer, màj une clé ou/et val
- Idée de base :
  - Dans une TH vide de m cases
  - insérer des éléments simple et/ou composés [clé ; valeur] :
    - ["Adil" ; 27]
    - ["Hiba" ; 23]
    - …
  - Remarque :
    - Faire une insertion séquentielle (1er élément à la position 0, 2ème à la position 1, …)
      → SD Linéaire
      → O(n)

Table de Hashage (TH)

- Question : Comment faire pour éviter insertion séquentielle ?
- Réponse :
  - Table de m éléments : Position dans [0, m-1]
  - Loi pour trouver la position de l'élément (clé , val) à insérer
  - Proposition :
    - Associer la clé à une position de 0 à m-1
    - Via une fonction qui retourne une valeur modulo m : appelée Fonction de Hachage (FH)
- Exemple :
  - Associer à la clé (chaine de caractères), la somme modulo m des positions de chaque lettre dans l'alphabet
  - A = 0, B = 1, C = 2, … , Z = 25
  - Taille de la table m = 7 , clé = "Adil"
  - Hash("Adil") $\equiv (0 + 3 + 8 + 11) \bmod(7)$

    $\equiv 22 \bmod(7)$

    $= 1$

Table de Hashage (TH)

- Exemple (suite) :: insérer [clé ; valeur] :
  - ["Adil" ; 27]
  - ["Hiba" ; 23]
  - ["Ali" ; 25]
  - …
- Association :
  - Hash("HIBA") $\equiv (7 + 8 + 1 + 0) \bmod(7)$

    $\equiv 16 \bmod(7)$

    $= 2$
  - Hash("ALI") $\equiv (0 + 11 + 8)\ [7]$

    $\equiv 19\ [7]$

    $= 5$

	Clé	Valeur(s)
0		
1	"Adil"	27
2	"Hiba"	23
3		
4		
5	"Ali"	25
6		

Table 1. Exemple de Table de Hashage

Table de Hashage (TH)

- Opérations à faire :
  - Trouver si une clé est dans la TH
    - Si oui, retourner la valeur correspondante
  - Insérer une paire (clé , valeur) dans la TH
  - Supprimer une paire (clé , valeur) de la TH
  - ...
- Rôle de la Fonction de Hashage (FH) est de connecter une clé (dans l'exemple précédant une chaine de caractères) à un slot (N° clé)
- Cas particulier : insérer "OMAR" dans la table 1
  - Hash("OMAR") = 1 [7]
  - Or dans TH 1, le slot N° 1 est déjà rempli par "Adil" !!

  → **Collision**

Table de Hashage (TH)

- Q : comment gérer la collision
- Solution 1 : chainage

0	**["clé 1" ; valeur 1]**	**["clé 2" ; valeur 2]**	**...**
1			
2			
3			
4			
5	["Joseph" ; (17 , 5.2 , "HR" , 7.1)]	["Simon" ; (23 , 4.2 , "CS" , 2.1)]	
6			

- Càd : slot → liste de paires [clé ; valeur]
  - Remarque : le champ valeur peut être composé
  - Pour une clé K, Hash(K) = i

  →Insérer [K ; val_K] à la fin de la liste / Slot i

Table de Hashage (TH)

- Gestion de Collision par chainage
  - Supprimer ("Simon") :
    - Hash ("Simon") = 5

0	["clé 1" ; valeur 1]	["clé 2" ; valeur 2]	...
1			
2			
3			
4			
5	["Joseph" ; (17 , 5.2 , "HR" , 7.1)]	["Simon" ; (23 , [illegible]2 , "CS" , 2.1)]	
6			

Table de Hashage (TH)

- Gestion de Collision par chainage
  - Complexité = coût (t) pour une TH de <u>n éléments</u> répartis sur <u>m slots</u>:
    - Supprimer TH avec m slots
    - Insérer n éléments à la TH
  - Worst case :
    - Tous éléments en collision dans le même slot
    - Insertion, recherche, suppression : O(n)

Table de Hashage (TH)

- Gestion de Collision par chainage
  - Cas moyen:
    - $P[\, h(k_i) = j \,] = 1 / m$
  - Taille moyenne liste de chaque slot : n / m

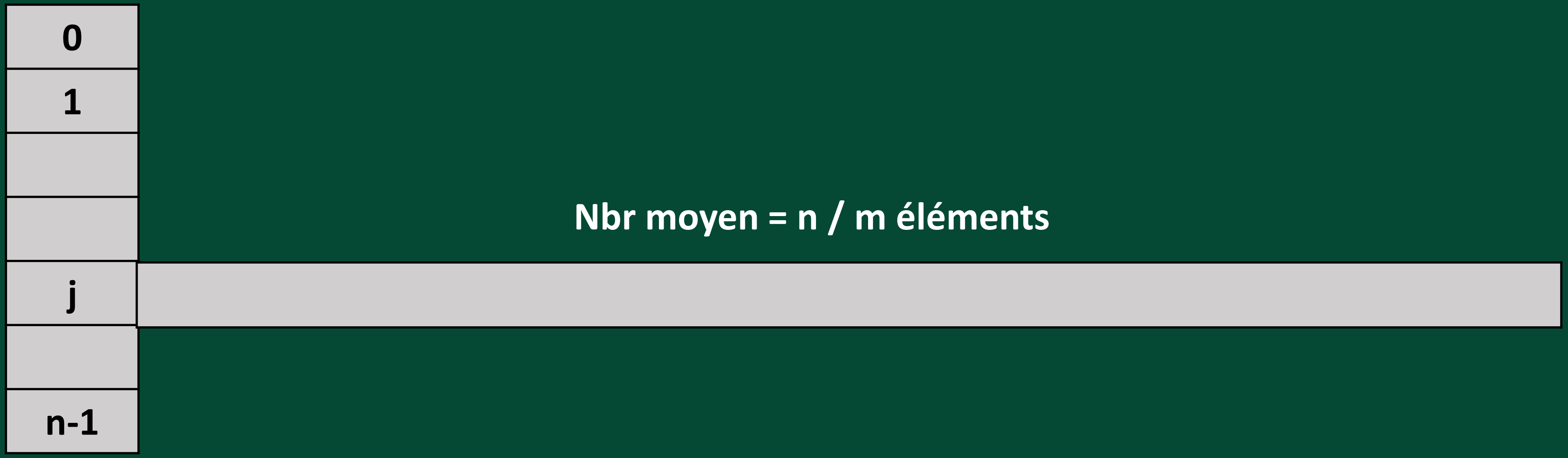


Table de Hashage (TH)

- Gestion de Collision par chainage
  - Exemple :
    - Si m = 100 et n = 200; nbr moyen / slot = 200 / 100 = 2
  - Taille moyenne liste de chaque slot : n / m
  - Nommée : Load Factor α
  - $\alpha = \dfrac{\text{\# elemts ds TH}}{\text{\# slots ds TH}}$
  - Remarque :
    - Par convention : O(append) dans liste (dynamique) ~ O(1)
    - Pour plus de précisions, voir analyse amortie.

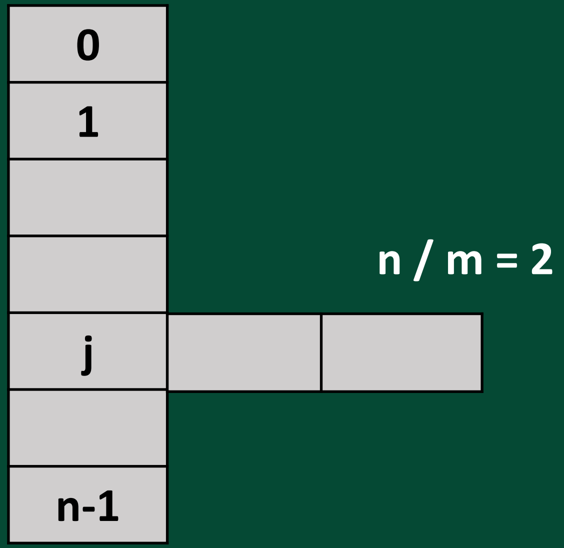


Table de Hashage (TH)

- Gestion de Collision par chainage
  - Paramètre α_{lim}, définit par les concepteurs
  - Problème : si α_{lim} est dépassé ? (càd $\alpha > \alpha_{lim}$)
    - La TH devient trop pleine,
    - Nombre de collisions augmente,
    - Ralentissement des opérations d'insertion, la recherche et la suppression.

    → Changer les conditions de la TH pour ne pas ressembler à une SD Linéaire
  - ReHashing :
    - Redimensionnement de la TH lorsqu'elle atteint le seuil de capacité
      - Création d'une nouvelle table de taille 2m (par convention)
      - La taille est une puissance de 2 ou un nombre premier (cf analyse plus loin)
      - Recalcule des Codes de Hachage :
        - Les nouveaux slots adéquats sont recalculés en fonction de la nouvelle taille de la table.
        - Réinsérer les éléments existants de TH originale dans la nouvelle table.
    - Libérer l'ancienne table

Table de Hashage (TH)

- Gestion de Collision par chainage
  - Temps de Rehashing :
    - T (ReH) = O (m + n), en prenant en considération O(append) = O(1)
  - Cependant, ce coût est amorti, car les opérations ultérieures sont plus rapides grâce à la table agrandie.
  - Espace Mémoire : Le rehashing peut également entraîner une augmentation temporaire de l'utilisation de la mémoire.
  - En résumé du ReHashing :
    - Redimensionnement dynamique pour gérer des quantités croissantes d'éléments
    - Maintenir l'efficacité des tables de hachage
    - Maintenir les performances optimales.

Fonction de Hashage (FH)

- Conception des Fonctions de Hashage (FH)
  - Utilisées dans :
    - cryptographie,
    - BD : MD5,
    - Blockchain (bitcoin) : SHA256 , Etherum : SHA3 , …
  - Bonne FH évite au plus les collisions:
    - $x1 \neq x2$, $h(x1) = h(x2)$

- Changement de 1 bit en input → changement de > 50% bits en output

- Python implémente FH pour types:
  - int , string et float

Fonction de Hashage (FH)

- Challenge 1:
  - Besoin de hasher # types de clés au-delà des nombres:
    - Listes / tableaux
    - Tuples
    - Strings
    - Autres objets définis

- Challenge 2:
  - S'assurer que la FH ne souffre pas de trop de collisions

Fonction de Hashage (FH)

- Exemple 1: hasher une liste de nombres (n éléments , m slots)
  - Idée faire somme de tous ai modulo(m)

  - Hash([a1, a2, … , an] , m) = (a1 + a2 + … + an)[m]
    = (a2 + a1 + … + an)[m] = Hash([a2, a1, … , an] , m)
    …
    = (an + … + a2 + a1)[m] = Hash([an, … , a2, a1] , m)

En quête d'un Hashage Universel

- Proposer une famille de FH : FMH = { h_1, … , h_N } telle que pour tout h_i , le nombre de collision de h_i( ) appliqué à la TH soit raisonnable.
- Si la taille de la TH = m, alors FMH est universelle si pour $h_i \in$ FMH

$\forall\ k_1, k_2$ tq $k_1 \neq k_2$, $Pr(h_i (k_1) = h_i (k_2)) \leq 1/m$

- Exemple de coneption d'une famille de FH :
  - Soient clés $\in$ { 1, … , n }
  - Choisir p nombre premier tp p > n
  - Soit la famille { h_1, … , h_{p-1} } de FH tq :

    $h_a(j) = ((a*j) [p]) [m] \ / \ a \in \{1, \dots , p-1 \}$
  - Ex : n = 10, p = 13, m = 7, H = { h_1, … , h_{12} }
    - $h_5 (j) = ((5j) [13]) [7]$
    - $h_5 (10) = (50 [13]) [7] = 11 [7] = 4$
    - $h_8 (10) = (80 [13]) [7] = 2 [7] = 2$

En quête d'un Hashage Universel

- Question : est ce que $\{ h_1, \ldots , h_{p-1} \}$ est universelle ?
- Réponse :
  - Soient k_1, k_2 tq $k_1 \neq k_2$, en collision :

    $$((a * k_1)[p])[m] = ((a * k_2)[p])[m]$$

  - Càd : $((a * k_1 - a * k_2)[p])[m] = 0$
  - Soit $\beta = a(k_1 - k_2)[p]$, alors $\beta[m] = 0$ sachant que $\beta < p$

alors $\beta \in \{ 0, m, 2m, 3m, \ldots , \gamma m \}$

tq $\gamma = \lfloor \frac{p}{m} \rfloor$ partie entière de $\frac{p}{m}$, et $(\gamma + 1) m > p$

Exemple (suite) : $m = 7$, $p = 61$, $\gamma = 8$

En quête d'un Hashage Universel

- D'où : $a (k_1 - k_2) [p] \in \{ 0, m, 2m, 3m, \dots , \gamma m \}$
- D'après le théorème de Bezout / Euclide étendu, et puisque p est premier :

$a \in \{ (k_1 - k_2)^{-1}_p 0, (k_1 - k_2)^{-1}_p m, (k_1 - k_2)^{-1}_p 2m, \dots , (k_1 - k_2)^{-1}_p \gamma m \}$

Soit $s = (k_1 - k_2)^{-1}_p$, $(s.s^{-1} = 1 [p])$

Donc : $a \in \{ s.0, s.m, 2s.m, \dots , \gamma s.m \}$

Càd γ possibilités de la valeur de <u>a</u> pour avoir collision (rappel $a \in \{1, \dots , p-1 \}$)

→ γ possibilités sur (p-1), causent une collision

$Pr(h_i (k_1) = h_i (k_2)) = \frac{\gamma}{p-1} = \frac{p}{m} . \frac{1}{p-1} = \frac{1}{m} . \frac{p}{p-1} \sim \frac{1}{m}$ (puisque $\frac{p}{p-1} \sim 1$)

D'où

$Pr(h_i (k_1) = h_i (k_2)) >\sim \frac{1}{m} \leq \frac{2}{m}$

D'où cette famille de FH est <u>presque universelle</u>,

En quête d'un Hashage Universel

- D'où Pr(collision (x, y)) $\leq \frac{2}{m}$

$$\sum_{y \in TH} \Pr\big(\, Collision\ (\, x, y\,)\big) \leq \sum_{y \in TH} \frac{2}{m} \leq \frac{2.n}{m}$$

$$\sum_{y \in TH} \Pr\big(\, Collision\ (\, x, y\,)\big) \leq 2.\alpha$$

D'où si la famille H est presque universelle, la longueur moyenne d'une liste L(x) $\leq 2.\alpha$

→ Coût moyen : insertion, suppression , recherche ~ $2.\alpha$

En quête d'un Hashage Universel

- Gestion de Collisions - Solution 2 ;
  - Hachage à Adressage ouvert :
    - Idée élément = (clé , val) / slot dans la TH
    - Si place libre : insérer (clé , val) dans le slot approprié
    - Sinon, trouver une place alternative,

- Sondage Linéaire :
  - pour insérer (k , v) dans TH
    - Calculer Hach(k) = j
    - Tenter d'insérer / slot j
    - Si ok, terminer
    - Sinon :
      - Tenter slot j + 1
      - Sinon tenter slot j + 2
      - …

En quête d'un Hashage Universel

- Sondage Linéaire :
  - pour chercher une clé k dans TH : de même
    - Calculer Hach(k) = j
    - Chercher dans le slot j
    - Si ok, terminer
    - Sinon :
      - Chercher dans le slot j + 1
      - Chercher dans le slot j + 2
      - …

En quête d'un Hashage Universel

- Sondage Linéaire : Pseudo code Insertion | Recherche k dans TH

```
Insertion ( k , v ):
i ← 0
Répéter :
        j ← h ( k, i )
        si TH[ j ] = NULL
                TH[ j ] = ( k , v )
                retourner j
        sinon
                i ← i + 1
jusqu'à i = m
si i = m
        Err ( " Débordement TH " )
```

```
Recherche ( k , v ) :
        i ← 0
        Répéter :
                j ← h ( k, i )
                si TH[ j ] = ( k , v ):
                        retourner j
                sinon
                        i ← i + 1
        jusqu'à (TH[ j ] = NULL ou i = m )
        retourner NULL
```

- Tq $h (k , i) = (h'(k) + a.i) [m]$, (a : pas, h' : est une FH)

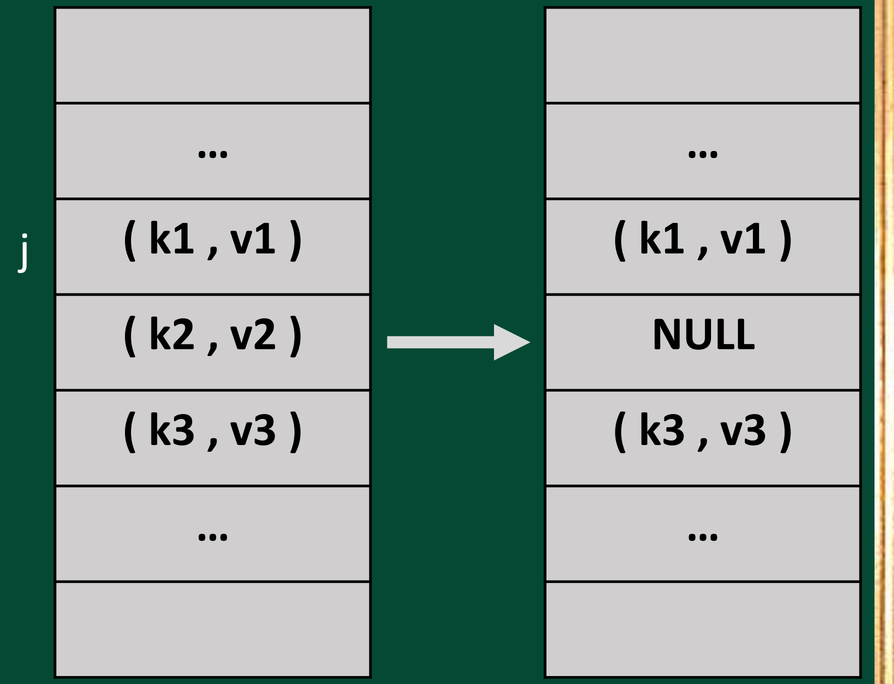
En quête d'un Hashage Universel

- Sondage Linéaire :
  - Suppression de (k , v) de la table TH :
    - Semble de même que insertion et recherche, mais
    - Exemple : soient k1, k2, k3 ≠ tq :

      h (k1) = h (k2) = h (k3) = j , tq h est une FH
      - Scénario :

        Suppression de (k2 , v2)
        - j = h (k2)
        - TH[j] = (k1 , v1) ≠ (k2 , v2)
        - j = j + 1, TH [j] = (k2 , v2)
        - Suppression en mettant TH[j] = NULL

        Recherche (k3 , v3)
          - h (k3) = j ,
          - TH [j] = (k1 , v1) ≠ (k3 , v3)
          - j = j + 1, TH [j] = NULL
          - Arret → (k3 , v3) ∉ TH :
          - FAUX



En quête d'un Hashage Universel

- Sondage Linéaire :
  - Conclusion / suppression :
    - Après suppression d'une clé dans la TH, remplacer par une valeur spéciale et non pas par NULL
  - Inconvénients :
    - Formation de clusters : perte de temps pour l'insertion ou recherche
  - En conclusion :
    - Sondage linéaire efficace pour la gestion de collision si la TH n'est pas trop pleine.

En quête d'un Hashage Universel

- Sondage Quadratique:
  - $h(k, i) = (h'(k) + a.i + b.i^2)[m]$, tq $b \neq 0$, $i = 0, 1, \ldots, m-1$; a, b = constantes et $h'(k)$ est une FH
  - Exemple :
    - Soit $h(k, i = 0) = (h'(k) + i^2)[m]$, tq $b = 1$, $a = 0$ et choisir $h'(k) = 3k + 4\,[m]$
    - Prendre $m = 10$, puis
    - Insérer dans l'ordre, les éléments : 32, 70, 17, 14, on aura :

0	1	2	3	4	5	6	7	8	9
32				**70**	**17**	**14**			

- $h(32, 0) = h(32) = h'(32) + 0^2\,[10] = 96 + 4\,[10] = 0$
- $h(70, 0) = h(70) = h'(70) + 0^2\,[10] = 210 + 4\,[10] = 4$
- $h(17, 0) = h(17) = h'(17) + 0^2\,[10] = 51 + 4\,[10] = 5$
- $h(14, 0) = h(14) = h'(14) + 0^2\,[10] = 42 + 4\,[10] = 6$

En quête d'un Hashage Universel

- Sondage Quadratique:
  - Exemple (suite) :
    - Puis, insérer dans TH l'élément : 22,
      - $h(22, 0) = h(22) = h'(22) + 0^2 [10] = 66 + 4 [10] = 0$ ✗ (rempli)
      - $h(22, 1) = h'(22) + 1^2 [10] = 67 + 4 [10] = 1$ ✓ (libre)
    - Puis, insérer dans TH l'élément : 30
      - $h(30, 0) = h(30) = h'(30) + 0^2 [10] = 90 + 4 [10] = 4$ ✗ (rempli)
      - $h(30, 1) = h'(30) + 1^2 [10] = 90 + 5 [10] = 5$ ✗ (rempli)
      - $h(30, 2) = h'(30) + 2^2 [10] = 90 + 8 [10] = 8$ ✓ (libre)
    - Puis, insérer dans TH l'élément : 60
      - $h(60, 0) = h(60) = h'(60) + 0^2 [10] = 180 + 4 [10] = 4$ ✗ (rempli)
      - $h(60, 1) = h'(60) + 1^2 [10] = 180 + 5 [10] = 5$ ✗ (rempli)
      - $h(60, 2) = h'(60) + 2^2 [10] = 180 + 8 [10] = 8$ ✗ (rempli)
      - $h(60, 3) = h'(60) + 3^2 [10] = 180 + 13 [10] = 3$ ✓ (libre)
    - Remarque : cluster (secondaire) lors de l'insertion de 60 car $h'(60) = h'(30)$

- Sondage quadratique mieux que le sondage linéaire, mais le problème de clustering <u>persiste</u>.

0	1	2	3	4	5	6	7	8	9
32	**22**		**60**	**70**	**17**	**14**		**30**	

En quête d'un Hashage Universel

- Double hachage
  - $h(k, i) = (h_1(k) + i.h_2(k))\ [m]$, tq h_1, h_2 : FH
  - Exemple :
    - $m = 10$
    - $h_1(k) = 3k + 4$
    - $h_2(k) = 2k + 5$
  - Reprenons l'exemple précédant :
    - insertion de 32, 70, 17, 34 de même puisque $i = 0$:
    - Insertion de 22
      - $h(22, 0) = h_1(22)$ $= 66 + 4\ [10] = 0$ ✗ (rempli)
      - $h(22, 1) = h_1(22) + 1.h_2(22)$ $= 70 + 49\ [10] = 9$ ✓ (libre)

0	1	2	3	4	5	6	7	8	9
32				70	17	14			22

En quête d'un Hashage Universel

- Double hachage
  - Exemple (suite) :
    - insérer dans TH l'élément : 30
      - h (30 , 0) = h_1(30) = 90 + 4 [10] = 4 ✗ (rempli)
      - h (30 , 1) = h_1(30) + 1.h_2(30) = 94 + 65 [10] = 9 ✗ (rempli)
      - h (30 , 2) = h_1(30) + 2.h_2(30) = 94 + 130 [10] = 4 ✗ (rempli)
      - h (30 , 3) = h_1(30) + 3.h_2(30) = 94 + 195 [10] = 9 ✗ (rempli)
      - ...
      - Tourne indéfiniment (dégénérescence) :
        - Ici (m = 10, h1(k) = 3k + 4 , h2(k) = 2k + 5)

        car 5 et 10 ne sont pas premiers entre eux
      - Risque : ne pas visiter tous les slots de la TH

0	1	2	3	4	5	6	7	8	9
32				**70**	**17**	**14**			**22**

En quête d'un Hashage Universel

- Double hachage
  - En général $h_2(k) \wedge m = 1$, pour éviter la dégénérescence
  - Ex :
    - $m = 2^a$ et h_2 produit toujours un nombre impair
    - $m = p$ (premier) et $h_2 < p$

En quête d'un Hashage Universel

- Cuckoo Hashing (Cuckoo : espèce d'oiseau)
  - 2 TH, 2 FH : h1 ( ) et h2 ( )

	TH1 , FH1	TH2 , FH2
0		
1		
2		
3	Nabil	
4		
5	Salma	
6		
7		
8		
9		

- X = { Nabil, Salma, Salim, Imad, Noura, Ikbal, Omar }
- h1 ('Nabil') = 3 ✓
- h1 ('Salma') = 5 ✓

En quête d'un Hashage Universel

- Cuckoo Hashing (Cuckoo : espèce d'oiseau)
  - 2 TH, 2 FH : h1 ( ) et h2 ( )

	TH1 , FH1	TH2 , FH2
0		
1		Nabil
2		
3	Salim	
4		
5	Salma	
6		
7		
8		
9		

- X = { Nabil, Salma, Salim, Imad, Noura, Ikbal, Omar }
- h1 ('Nabil') = 3
- h1 ('Salma') = 5
- h1 ('Salim') = 3 ✗
  - h2('Nabil') = 1 ✓

En quête d'un Hashage Universel

- Cuckoo Hashing (Cuckoo : espèce d'oiseau)
  - 2 TH, 2 FH : h1 ( ) et h2 ( )

	TH1 , FH1	TH2 , FH2
0		
1		Nabil
2		
3	Salim	
4		
5	Salma	
6		
7		
8	Imad	
9		

- X = { Nabil, Salma, Salim, Imad, Noura, Ikbal, Omar }
- h1 ('Nabil') = 3
- h1 ('Salma') = 5
- h1 ('Salim') = 3
  - h2('Nabil') = 1
- h1 ('Imad') = 8 ✓

En quête d'un Hashage Universel

- Cuckoo Hashing (Cuckoo : espèce d'oiseau)
  - 2 TH, 2 FH : h1 ( ) et h2 ( )

	TH1 , FH1	TH2 , FH2
0		
1		Salma
2		
3	Nabil	
4		
5	Noura	
6		Salim
7		
8	Imad	
9		

- X = { Nabil, Salma, Salim, Imad, Noura, Ikbal, Omar }
- h1 ('Nabil') = 3
- h1 ('Salma') = 5
- h1 ('Salim') = 3
  - h2('Nabil') = 1
- h1 ('Imad') = 8
- h1 ('Noura') = 5 ✗
  - h2('Salma') = 1 ✗
  - h1 ('Nabil') = 3 ✗
  - h2('Salim') = 6 ✓

- **En résumé :**
  - Si trop de déplacement > Log (n) :
    → Choisir 2 autres FH aléatoirement
  - Si taille TH trop petite/n
    → Rehashing

Hashage Parfait

- Hachage Parfait:
  - Idée : ne pas avoir de collision
  - Utilisation de TH très volumineuse :
    - Soit H une famille de FH universelle et n clés ≠ $k_1, \ldots, k_n$:
      - Étape 1 : choisir aléatoirement une FH, $h \in H$
      - Étape 2 : créer TH à $k.n^2$ ($k.n^2 = m$) slots / k à déterminer
      - Étape 3 : insérer chaque clé dans TH
      - Étape 4 : Si une collision se produit : abandonner et amler à Étape 1
    - Remarque : Probabilité : Pr (boucle ∞) ≠ 0 , mais très improbable
    - Probabilité collision $k_i \neq k_j$:
      - $Pr (h(k_i) = h (k_j)) \leq \frac{C}{m} = \frac{C}{k.n^2}$
    - Probabilité d'avoir au moins une collision :

$$= Pr_{Coll} ((K_1 , K_2) \mid (K_1 , K_3) \mid \ldots \mid (K_1 , K_n) \mid (K_2 , K_3) \mid \ldots \mid (K_2 , K_n) \mid \ldots \mid (K_{n-1} , K_n))$$

$$\leq \frac{C}{k.n^2} \cdot \frac{n(n-1)}{2} \leq \frac{C}{2k}$$

Hashage Parfait

- TH très volumineuse :
  - Probabilité de collision p_{coll} :
    - si k = 2C : p_{coll} = Pr (au moins une collision) $\leq \frac{1}{4}$
  - Probabilité de ne pas avoir de collision = 1 - p_{coll}
    - Càd Pr (aucune collision) $\geq \frac{3}{4}$ (75 %)
  - Q : Probabilité Pr (boucle ∞) = Pr (toutes itérations échouent) = ?
    - Analogie :
      - Soit une pièce de monnaie tq : Pr ("Pile") = $\frac{1}{4}$, Pr ("Face") = $\frac{3}{4}$
      - On jette la pièce , jusqu'à avoir la "Face"
      - Probabilité de ne pas réussir à avoir "Face" dans les 100 premiers essais :

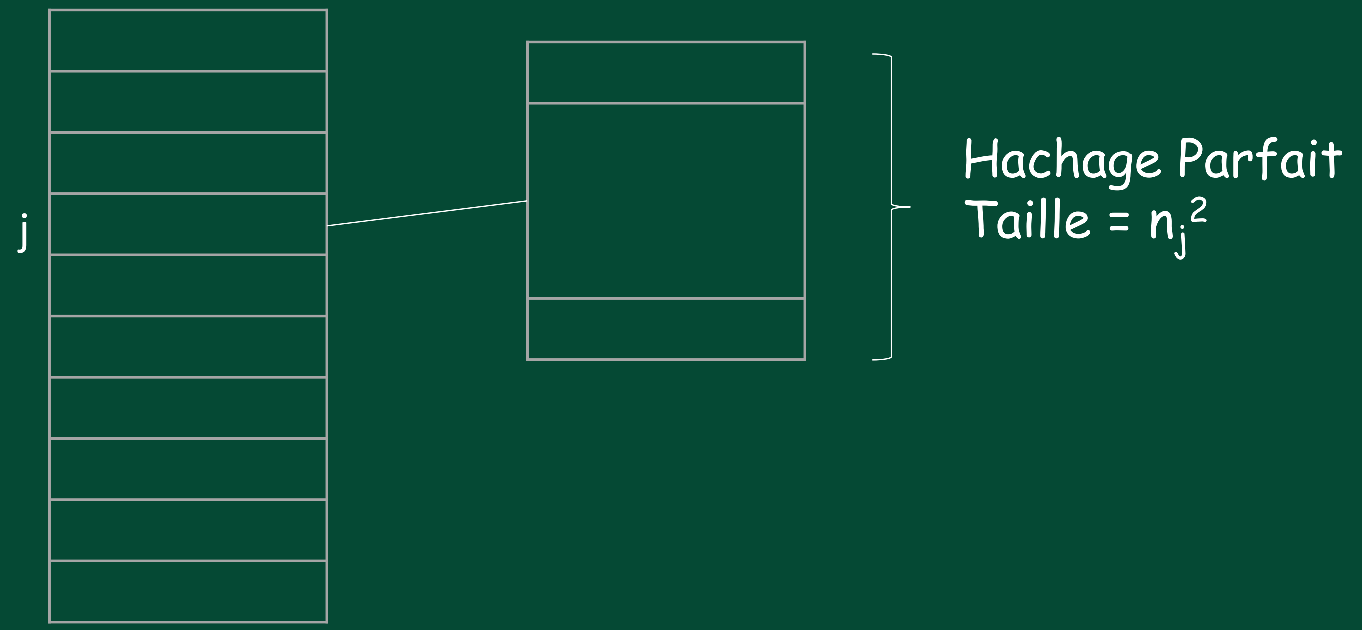
        = Pr ("Pile") dans 100 premiers essais = $(\frac{1}{4})^{100} = \frac{1}{4^{100}}$: probabilité négligeable

Hashage Parfait

- TH très volumineuse :
  - Analogie :
    - Q : Nombre moyen d'essais pour avoir "Face" ?
    - Rép : cf livre Intro à l'Algorithmique CLRS:
      - E = $\frac{1}{1-\frac{1}{4}}$ = $\frac{4}{3}$ (interprétation : il faut en moyenne $\frac{4}{3}$ essais pour avoir "Face")
    - Dém :
      - E = $\sum_1^\infty (n\, p_n)$, or p_n = $(\frac{1}{4})^{n-1} . \frac{3}{4}$ / tq p_n : proba avoir "Face" à la $n^{ème}$ itération
      - Si q ∈] -1 , 1 [, alors $\sum_1^\infty (nq^n) = \frac{q}{(1-q)2}$
      - Or q = $\frac{1}{4}$ et $\Sigma\ n.q^{n-1} = \frac{\Sigma\ n.qn}{q} = \frac{1}{(1-q)2}$ □ (cqfd)
    - D'où, par analogie, le nombre moyen de réitérer l'essai d'insertion dans la TH pour ne pas avoir de collision est $\frac{4}{3}$ ($Proba_{Coll}$ = Proba Réitérer ⟲ ≤ $\frac{1}{4}$)

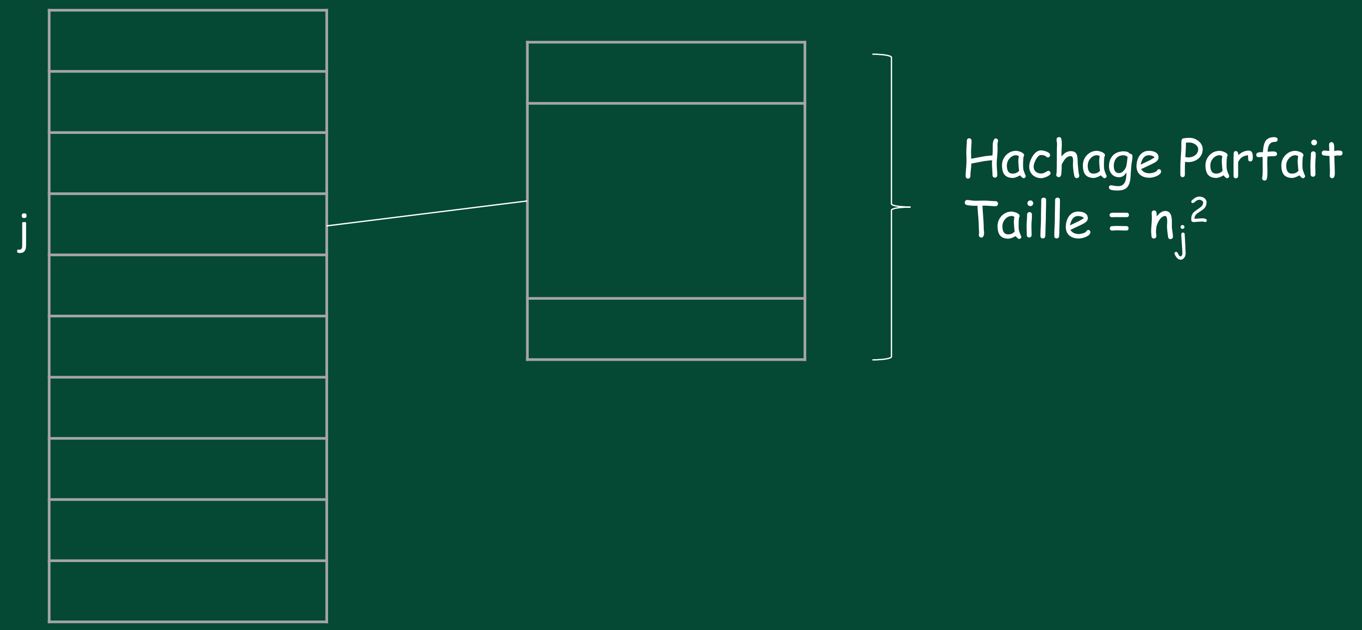
Hashage Parfait

- Table de Hachage de 2 niveaux:
  - Supposons que :
    - On connait toutes les clés à l'avance
    - On connait n_j = nbr élément qui hachent $j^{ème}$ slot, avec $\Sigma\, n_j = n$
    - On ne peut pas insérer de nouvelles clés
    - SD : TH où les slots sont des TH
    - Taille TH = n = m (n : est le nombre d'éléments à insérer)



Hashage Parfait

- Table de Hachage de 2 niveaux:
  - Supposons qu'on veut insérer n éléments au TH de 2ème niveau (parfaite) :
  - # total slots $\leq \Sigma\, nj^2$
  - Théorème 11.10 (CLRS) : E [$\Sigma\, nj^2$] < 2n (Rappel : $\Sigma\, n_j = n$)
  - Interprétation :
    - Quantité moyenne de mémoire / TH secondaire de hachage parfait est < 2n



Application Th, FH : Bloom Filter

- Exemple :
  - Soit T la table de bits de taille 9 et soient les 2 FH :
    - h1 (x) = 2x + 1 [9]
    - h2 (x) = 4x + 1 [9] , et
    - X = { 7, 11, 26}
  - T est initialisée à des 0
  - Pour signaler qu'un x ∈ X , il suffit de mettre :
    - T [h1(x)] = 1 , T [h2(x)] = 1
    - Insérer 7 : h1(7) = 6 , h2(7) = 2
    - Insérer 11 : h1(11) = 5 , h2(11) = 0
    - Insérer 26 : h1(26) = 8 , h2(26) = 6
  - Requete : 12, 28, 8 ∈ X ?

0	1	2	3	4	5	6	7	8
0	0	0	0	0	0	0	0	0
0	0	1	0	0	0	1	0	0
1	0	1	0	0	1	1	0	0
1	0	1	0	0	1	1	0	1

Application Th, FH : Bloom Filter

- Exemple :
  - Requete : 12, 28, 8 $\in$ X ?
    - Vérifier 12 :
      - $h_1(12) = 7$, $h_2(12) = 4$, or T [7] = 0 et T [4] = 0
        → 12 $\notin$ X
    - Vérifier 28 :
      - $h_1(28) = 3$, $h_2(28) = 5$, or T [3] = 0 et T [5] = 1
        →28 $\notin$ X (sinon il faut que T [$h_1(28)$] = T [$h_2(28)$] = 1)
    - Vérifier 8 :
      - $h_1(8) = 8$, $h_2(8) = 6$, or T [8] = 1 et T [6] = 1
        → 8 $\in$ X
        FAUX
        → Jugement faux ou FAUX POSITIF
- EN RÉSUMÉ :
  - Si y $\in$ X $\Rightarrow h_1(y) = h_2(y) = 1$
  - (pas d'équivalence, uniquement $\Rightarrow$)

0	1	2	3	4	5	6	7	8
0	0	0	0	0	0	0	0	0
0	0	1	0	0	0	1	0	0
1	0	1	0	0	1	1	0	0
1	0	1	0	0	1	1	0	1

Application Th, FH : Bloom Filter

- D'où le Bloom filter est une Structure de Données (SD) <u>probabiliste</u>
  - Jugement qu'un élément $x \notin X$ est correcte à 100 %
  - Jugement qu'un élément $x \in X$ est correcte à p % ($p \leq 100$)

- Idée de base d'un Bloom Filter:
  - k fonctions de hachage : $h_1, \ldots, h_k$
  - TH de m bits nommé T :

0	1		m -1
0	0		0

  - Pour insérer x à l'ensemble X:
    - Il faut mettre à 1 :

    $T[\, h_1(x) \,]$, $T[\, h_2(x) \,]$, … , $T[\, h_k(x) \,]$

- Si jugement d'appartenance est correcte à un pourcentage p%

→Possibilité d'avoir des faux positifs de recherche est à (100 – p) %

- Temps d'insertion θ (k)

Application Th, FH : Bloom Filter

- Calcul de la proba d'avoir des faux positifs :
  - k fonctions de hachage : $h_1, \dots, h_k$
  - TH de m bits nommé T :
  - Pour insérer n éléments à l'ensemble X
- Lors de l'insertion d'un élément x via une FH $h_a(x)$
  - La proba qu'un bit i soit choisi pour basculer vers 1 est $\frac{1}{m}$

  → La proba que le bit i ne soit pas choisi par $h_a(x)$ est $1 - \frac{1}{m}$
  - Pour a = 1, 2, … , k ; la proba que le bit ne soit pas basculé vers 1 est $(1 - \frac{1}{m})^k$
  - Généralisation, $x \in \{ x_1, \dots, x_n \}$
    - Proba que le bit i ne soit pas basculé vers 1 est $[(1 - \frac{1}{m})^k]^n = (1 - \frac{1}{m})^{kn}$
    - Si kn et m >> , alors $(1 - \frac{1}{m})^{kn} \sim e^{-\frac{kn}{m}}$, (car $\lim (1 - \frac{1}{t})^t \sim e^{-1}$, pout t très grand)
    - D'où proba qu'un bit i soit basculé vers 1 est : $1 - e^{-\frac{kn}{m}}$

Application Th, FH : Bloom Filter

- Calcul de la proba d'avoir des faux positifs :
  - Pour avoir un faux positif / élément y, il faut que k bits $i_1, \ldots, i_k$ soient mis à 1 aux endroits :
    - T [h_1(y)] , T [h_2(y)] , … ,T [h_k(y)]
    - Càd Proba (Faux Positifs) ~ $(1 - e^{-\frac{kn}{m}})^k$
    - A.N : (n, m, k) = (5.10^3, 25.10^3, 3)
      - Proba (Faux Positifs) = 0.09

Application Th, FH : Count Min Sketches (CMS)

- Appliquée en général aux flux de données :
  $x_1, x_2, \dots, x_W \in \{x_1, x_2, \dots, x_N\}$ / $N \leq W$ = taille

- Problème :
  - Estimer la fréquence (approximative) d'apparition d'un élément,
  - Pas nécessairement exacte, mais avec une tolérance
- Exemple : traitement d'URLs par un web server
  - W : nbr d'URL à traiter, N : nbr d'URL uniques
  - N est non connu à l'avance
  - Objectif :
    - Calculer ApproxCount (x_j) à une tolérance $\varepsilon.w$ pour $\delta.N$ éléments
    - A.N : $\delta = 0{,}99$ (99%) , $w = 10^9$, $\varepsilon = 10^{-6}$, $\varepsilon.W = 1000$
  - Interprétation :
    - Pour 99% des éléments, donner une estimation du NbrOcc à une erreur ne dépassant pas 1000.

Application Th, FH : Count Min Sketches (CMS)

- Idée de base :
  - Utiliser m compteurs c(1), … , c(m) tq m << W
  - Choisir famille FH, H = { h_1 , h_2 , … , h_N }
  - Incrémenter le compteur de l'élément du flux x_j : c[h(x_j)] ++
  - Pour connaitre l'approximation du compteur de k = c[h(k)]
- Analyse de l'erreur du CMS:
  - On sait que ApproxCount (j) ≥ count(j) , (count : Réel)
  - Pour une famille de FH universelle :
    - $Pr_{i \neq j}$ (h(i) = h (j)) ≤ $\frac{C}{m}$
  - Alors :
    - E[ApproxCount(j)] $= \text{count}(j) + \sum_{i \neq j} \Pr(\text{Collision}(i,j)).\text{Count}(i)$

$$\leq \text{count}(j) + \frac{C}{m}\sum_{i \neq j} \text{Count}(i)$$

$$\leq \text{count}(j) + \frac{C}{m}W$$

E[ApproxCount(j) - count(j)] ≤ $\frac{C}{m}W$

E[Erreur(j)] ≤ $\frac{C}{m}W$

Application Th, FH : Count Min Sketches (CMS)

- Analyse de l'erreur du CMS:
  - Inégalité de Markov, pour E(X) , Pr(X)

$$Pr\left(X \geq t \right) \leq \frac{E(X)}{t}$$

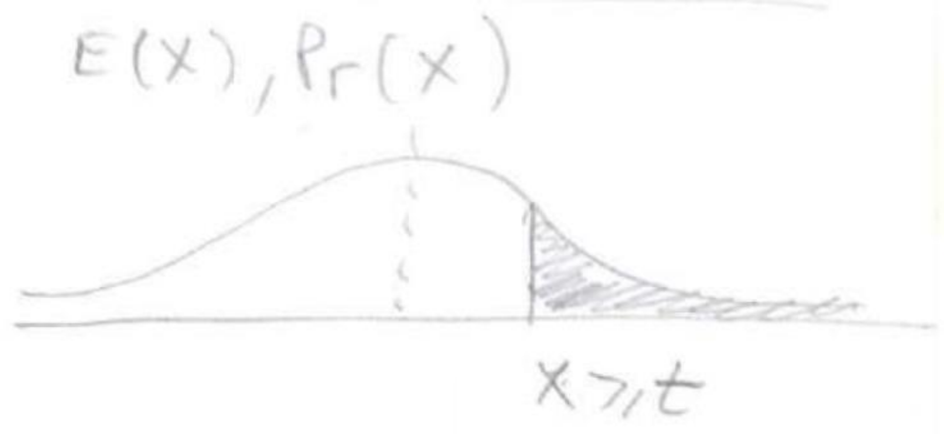
  - Application (Err ← X) :
    - $Pr\left(Err(j) \geq \varepsilon.W \right) \leq \frac{1}{\varepsilon.W} \frac{c}{m} W \leq \frac{c}{m\varepsilon}$
    - Si on choisit : $m = e.\frac{c}{\varepsilon}$

→ $Pr\left(Err(j) \geq \varepsilon.W \right) \leq \frac{1}{e}$

→ Question : $\frac{1}{e} \leq 1 - \delta$? ,

A.N : $c = 1$, $\varepsilon = 10^{-6}$, $e = 2.73$ → $Pr\left(Err(j) \geq \varepsilon.W \right) \leq \frac{1}{e} \sim 0{,}37 = 37\%$: **Insuffisant !**

- **Solution : utiliser k tables de comptage au lieu d'une seule,** (i.e une table / FH)



Application Th, FH : Count Min Sketches (CMS)

- Analyse de l'erreur du CMS:
  - **Solution : utiliser k tables de comptage au lieu d'une seule,** (i.e une table / FH)
  - Exemple :
    - Soient les FH $h_i()$, et les tables T_i , i = 1, 2, 3, 4, toutes de taille 16, initialisées à 0
    - Soient x, y, z, t : éléments représentant les adr ip :
      - input 1 : x = 192.170.10.1
        - $h_1(x) = 1$, $h_2(x) = 6$, $h_3(x) = 3$, $h_4(x) = 1$,
        - Il faut incrémenter $T_i[h_i(x)]$,

        $T_1[1] \leftarrow 0+1$, $T_2[6] \leftarrow 0+1$, $T_3[3] \leftarrow 0+1$, $T_4[1] \leftarrow 0+1$,

	0	1	2	3	4	5	6	…	15
T1	0	1	0	0	0	0	0	…	0
T2	0	0	0	0	0	0	1	…	0
T3	0	0	0	1	0	0	0	…	0
T4	0	1	0	0	0	0	0	…	0

Application Th, FH : Count Min Sketches (CMS)

- Analyse de l'erreur du CMS:
  - **Solution : utiliser k tables de comptage au lieu d'une seule, (i.e une table / FH)**
  - Exemple :
    - Soient les FH $h_i()$, et les tables T_i , i = 1, 2, 3, 4, toutes de taille 16, initialisées à 0
    - Soient x, y, z, t : éléments représentant les adr ip :
      - input 2 : y = 75.245.0.1
        - $h_1(y) = 1$, $h_2(y) = 2$, $h_3(y) = 4$, $h_4(y) = 6$,
        - Il faut incrémenter $T_i[h_i(y)]$,

        $T_1[1] \leftarrow 1+1$, $T_2[2] \leftarrow 0+1$, $T_3[4] \leftarrow 0+1$, $T_4[6] \leftarrow 0+1$,

	0	1	2	3	4	5	6	…	15
T1	0	2	0	0	0	0	0	…	0
T2	0	0	1	0	0	0	1	…	0
T3	0	0	0	1	1	0	0	…	0
T4	0	1	0	0	0	0	1	…	0

Application Th, FH : Count Min Sketches (CMS)

- Analyse de l'erreur du CMS:
  - **Solution : utiliser k tables de comptage au lieu d'une seule,** (i.e une table / FH)
  - Exemple :
    - Soient les FH $h_i()$, et les tables T_i , i = 1, 2, 3, 4, toutes de taille 16, initialisées à 0
    - Soient x, y, z, t : éléments représentant les adr ip :
      - input 3 : Z = 10.125.22.10
        - $h_1(z) = 3$, $h_2(z) = 4$, $h_3(z) = 1$, $h_4(z) = 6$,
        - Il faut incrémenter $T_i[h_i(z)]$,

$T_1[3] \leftarrow 0+1$, $T_2[4] \leftarrow 0+1$,
$T_3[1] \leftarrow 0+1$, $T_4[6] \leftarrow 1+1$,

	0	1	2	3	4	5	6	…	15
T1	0	2	0	1	0	0	0	…	0
T2	0	0	1	0	1	0	1	…	0
T3	0	1	0	1	1	0	0	…	0
T4	0	1	0	0	0	0	2	…	0

Application Th, FH : Count Min Sketches (CMS)

- Analyse de l'erreur du CMS:
  - **Solution : utiliser k tables de comptage au lieu d'une seule, (i.e une table / FH)**
  - Exemple :
    - Soient les FH $h_i()$, et les tables T_i , i = 1, 2, 3, 4, toutes de taille 16, initialisées à 0
    - Soient x, y, z, t : éléments représentant les adr ip :
      - input 4 : t = 192.170.10.1
        - $h_1(x) = 1$, $h_2(x) = 6$, $h_3(x) = 3$, $h_4(x) = 1$,
        - Il faut incrémenter $T_i[h_i(x)]$,

        $T_1[1] \leftarrow 2+1$, $T_2[6] \leftarrow 1+1$, $T_3[3] \leftarrow 1+1$, $T_4[1] \leftarrow 1+1$,

	0	1	2	3	4	5	6	…	15
T1	0	3	0	1	0	0	0	…	0
T2	0	0	1	0	1	0	2	…	0
T3	0	1	0	2	1	0	0	…	0
T4	0	2	0	0	0	0	2	…	0

Application Th, FH : Count Min Sketches (CMS)

- Analyse de l'erreur du CMS:
  - **Solution : utiliser k tables de comptage au lieu d'une seule,** (i.e une table / FH)
  - Exemple :
    - Calcul # occurrence :
      - Ex v = 192.170.10.1 (v = x = t)
      - $h_1(v) = 1$, $h_2(v) = 6$, $h_3(v) = 3$, $h_4(v) = 1$,
      - $T_1[1] = 3$, $T_2[6] = 2$, $T_3[3] = 2$, $T_4[1] = 2$,
      - Les occurrences de v dans les tables : (3, 2, 2, 2)
      - ApproxCount(v) = min (3, 2, 2, 2) = 2
    - En conclusion : On prend le min pour minimiser l'Erreur

	0	1	2	3	4	5	6	…	15
T1	0	3	0	1	0	0	0	…	0
T2	0	0	1	0	1	0	2	…	0
T3	0	1	0	2	1	0	0	…	0
T4	0	2	0	0	0	0	2	…	0

Application Th, FH : Count Min Sketches (CMS)

- Analyse de l'erreur du CMS:
  - On sait que pour chaque TH:

    $$Pr\ (\ Err(j) \geq \varepsilon.W\ ) \leq \frac{1}{e}$$

  - Pour que toutes les k FH font l'erreur :

    $$Pr\ (\ Err(j) \geq \varepsilon.W\ ) \leq \left(\frac{1}{e}\right)^k$$

    $$\leq 1 - \delta\ \ ?$$

  - Pour que $\left(\frac{1}{e}\right)^k \leq 1 - \delta$, il suffit que :

    $$k \geq -\ln(1 - \delta)$$

  - Ex : si $\delta = 0.99 \rightarrow k = 5$
- En résumé : cas général CMS
  - k compteurs ; k FH : $h_1, \ldots, h_k$; k Tables de comptage
    - $ApproxCount(v) = Min\ \{T_1[\ h_1(\ v\ )\ ]\ ,\ T_2[\ h_2(\ v\ )\ ]\ ,\ \ldots\ ,\ T_k[\ h_k(\ v\ )\ ]\ \}$
- EX : $\varepsilon = 10^{-6}$, $W = 10^9$, $\delta = 0.9$, $m = \frac{e}{\varepsilon} \sim 3.10^6 \rightarrow k = -\ln(1 - \delta) \sim 3$
- Interprétation : utiliser 3 tables de compteurs / résultats correctes à une tolérance 1000 pour 90% des éléments.

FIN