# Indexation

# Sequence 1

1 . Supposons qu’on ait 10 blocs sur disque, que le buffer puisse contenir 3 blocs, et qu’on utilise la stratégie LRU, combien de blocs devra-t-on lire pour la séquence de lecture 2,3,4,5,2,3,4,5,6,7,2 ? (où les entiers représentent les identifiants de blocs.)

* 10 
* 11 
* 12 
* 13

2. Peut-on faire mieux avec une autre stratégie ?

* oui 
* non

# Séquence 2

1. Quelle mémoire est la plus rapide ?

*  RAM 
* disque optique 
* bande magnétique

2. Quelle mémoire est la plus chère ?

* RAM 
* disque optique 
* bande magnétique

3. Quelle mémoire est typiquement la plus massive (plus grande taille) ?

* RAM 
* disque optique 
* bande magnétique

4. Quelles mémoires ne sont pas en semi-conducteurs ?

* RAM 
* disque optique 
* disque magnétique 
* bande magnétique 
* flash

5. La vitesse d’accès à une information sur disque par rapport à l’accès à de l’information en RAM est de l’ordre de :

*  plusieurs ordres de grandeur plus rapide 
* même ordre de vitesse 
* plusieurs ordres de grandeur plus lente

# Séquence 3

1. Puisque les index accélèrent l’accès aux données, on pourrait imaginer que l’on indexe systématiquement tous les attributs. Quelles sont parmi ces raisons, celles pour lesquelles on ne le fait pas ?

* Les index coûtent en temps CPU en cas de mises à jour. 
* Ils utilisent de l’espace de stockage

2. Pourquoi ne peut-on avoir un index non dense que sur un seul attribut par relation ?

* Parce que la relation doit être triée suivant cet attribut 
* Parce qu’on ne peut pas avoir d’index non dense sur une relation avec plusieurs attributs.

3. En supposant qu'un index non dense contienne 99 clés et 100 pointeurs et que chaque bloc puisse contenir au plus 20 enregistrements. Combien d’enregistrements peut contenir au plus la relation ?

* 2000 
* 500 
* 100

4. Même question pour un index dense ?

* 2000 
* 500 
* 100

# Semaine 4

1. Index non dense. On suppose que l’on peut stocker 100 pointeurs et 99 clés par bloc, ou 20 enregistrements. Une relation indexée par un arbre-B de 1 niveau peut contenir 2000 enregistrements. Combien d’enregistrements peut contenir au plus une relation indexée par un arbre-B de 2 niveaux ?

* 10000 
* 100000 
* 200000

2. Index dense. On suppose toujours que l'on peut stocker au maximum 100 pointeurs et 99 clés par bloc. Le bloc racine est supposé toujours plein. Les autres blocs d'index contiennent au minimum 50 pointeurs (on rééquilibre en permanence l'arbre pour arriver à satisfaire cette contrainte). Quel est le nombre minimal de clés à partir duquel il peut être nécessaire de créer un troisième niveau ?

* 1000 
* 5051 
* 7265

3. En supposant des blocs de 16000 octets, des adresses de 20 octets, des clés de 8 octets et en supposant un remplissage de 75% des blocs de données, quel volume de données peut-on stocker environ avec 3 niveaux ?

* 2 mégaoctets 
* 2 gigaoctets 
* 2 téraoctets Bas du formulaire

# Semaine 5

1. Laquelle de ces phrases est vraie ?

* Le Hachage comme l’arbre B permet les requêtes d’intervalle 
* Seul l’arbre B les permet  Seul le hachage les permet

2. Supposons une table de hachage à 8 entrées et des blocs contenant 3 entrées. Si on n’a pas de chance, après combien d’éléments peut-t-on avoir un débordement ?

* 4 
* 9
* 25

# Semaine 6

*Supposons que nous démarrions avec une table de hachage vide utilisant 0 bit de la fonction de hachage, que les blocs de données contiennent 2 enregistrements et que les entrées, et leurs valeurs de hachage soient les suivantes :*

|  |  |
| --- | --- |
| A | 010 |
| B | 100 |
| C | 111 |
| D | 101 |
| E | 001 |
| F | 100 |

1. Combien de blocs de données y aura-t-il à l'arrivée ?

* 3 
* 4 
* 5

*Considérer l'état suivant d'une table de hachage. Observez que, comme expliqué en cours, des entrées distinctes de la table de hachage peuvent pointer vers le même bloc - par exemple, les 4 premières vers le bloc1. Les valeurs de hachage sont dans la colonne de gauche et les données dans celle de droite.*

|  |  |
| --- | --- |
| 000 | bloc1 = {A, E} |
| 001 | bloc1 = {A, E} |
| 010 | bloc1 = {A, E} |
| 011 | bloc1 = {A, E} |
| 100 | bloc2 = {B, F} |
| 101 | bloc3 = {D} |
| 110 | bloc4 = {C} |
| 111 | bloc4 = {C} |

2. On insère un nouvel élément, G. En supposant que l'on peut entrer deux éléments dans chaque bloc de données, parmi ces couples, lesquels pourraient conduire à doubler la table de hachage ?

* (G,010) 
* (G,101) 
* (G,111)

# Semaine 7

*On rentre 6 prénoms dans un filtre de Bloom avec deux fonctions de hachage retournant 4 bits (un nombre entre 0 et 15) :*

|  |  |  |
| --- | --- | --- |
| **Prénom** | **F1** | **F2** |
| Antoine | 0100 | 1000 |
| Béryl | 1001 | 0100 |
| Charlie | 1110 | 0001 |
| David | 1010 | 0100 |
| Elise | 0010 | 0111 |
| Fiora | 1001 | 0110 |

1. On veut vérifier si un prénom X est dans la liste en vérifiant que Bloom(F1(X)) et Bloom(F2(X)) sont tous les deux à 1. Quelle est la probabilité qu’un prénom X soit un faux positif ? :

*  3% 
* 10% 
* 32%

2. Si on utilisait un cinquième bit, la probabilité d’un faux positif serait ? :

* La même 
* Plus importante 
* Moins importante

# Exercice : faux positifs

Pour ce problème, on pourra utiliser une calculatrice, par exemple, [calc en ligne.](http://calc.name)

Nous allons considérer la commande spell de Unix qui n’est plus utile depuis longtemps. L’idée est de prendre un dictionnaire de *d* mots. On calcule son filtre de Bloom avec une table *T* de taille *n* et k fonctions de hachage, *h*1,…,*hk*. Plus précisemment, on commence par initialiser la table avec que des 0. Puis, pour chaque mot *w* du dictionnaire et chaque fonction *hi*, on met à 1 la position *hi*(*w*). (On peut alors jeter le dictionnaire.) On va considérer qu’un mot m est correct si ∀*i*∈[1..*k*],*T*[*hi*(*m*)]=1.

On s’intéresse à la probabilité qu’un mot soit considéré correct « par chance » (soit un faux positif). La probabilité du bit *j* à zéro (aucun mot d'un dictionnaire de taille *d* ne le mette à 1) est de : *p*=α*oO*!*e*−α=*e*−α avec α=*dk*/*n*. La probabilité pour que *m* arbitraire soit accepté est (1−*p*)*k*

Pour la commande spell de Unix, si on choisit *d*=25000, *n*=400000, *k*=11,   
quelle est la probabilité qu’un mot au hasard soit un faux positif ? :

*  0.000458711 
* 0.0458711 
* 0.458711