Université Pierre et Marie Curie - Master d'Informatique

Saison février - juin 2017

M1

4I803

Bases de Données Réparties

TD

Partie 1

Stéphane GANCARSKI, Hubert NAACKE

URL : www-bd.lip6.fr/wiki/doku.php/site/enseignement/master/bdr/start

TABLE DES MATIERES

TD 1 : Arbre B+

Notations et conventions

Nombre de valeurs dans un nœuds. Pour un arbre B+ d'ordre d, le nombre de valeurs qu'un nœud peut contenir est :

dans l'intervalle [1, 2.d] pour la racine,

dans l'intervalle [d, 2.d] pour les nœuds intermédiaires et les feuilles.

Certains exercices n'indiquent pas l'ordre de l'arbre mais directement le nombre de valeurs dans un nœud.

Dessin d'un arbre. Pour repérer facilement les nœuds d'un arbre quand on le dessine, on peut attribuer un nom à chaque nœud : R pour la racine, Ni pour les nœuds intermédiaires et Fi pour les feuilles. On peut utiliser la syntaxe suivante pour représenter le contenu d'un nœud : N(v_1, v_2, …) où N est le nom du nœud et les v_i sont les valeurs.

Insertion d'une valeur en cas de débordement : quand la feuille *F* déborde, on garde les *d+1* plus petites valeurs dans *F*, les *d* autres valeurs vont dans une nouvelle feuille. Quand le nœud intermédiaire *N* déborde on garde les *d* plus petites valeurs dans *N*, les *d* plus grandes valeurs vont dans un nouveau nœud. La valeur restante est insérée dans le nœud père.

Suppression d'une valeur. Si le nœud ne contient que *d* valeurs avant la suppression, alors on considère d'abord la redistribution avec le nœud voisin (de même père) situé à gauche, puis avec celui situé à droite. Si aucune redistribution n'est possible, on considère la fusion avec le voisin (de même père) situé à gauche, puis avec celui situé à droite. Une suppression avec redistribution nécessite d'ajuster le contenu du nœud père. Une suppression avec fusion nécessite de supprimer une valeur dans le nœud père.

Décompte du nombre de **nœuds** lus et écrits. Une opération d'insertion ou de suppression peut nécessiter de lire, modifier ou créer des nœuds. On appelle *L* le nombre de nœuds lus pendant une opération, et respectivement *E* le nombre de nœuds écrits ou créés.

Exercice 1 : Effet d'une séquence insertion-suppression

On étudie l'effet de l'insertion d'une valeur *v* suivie de la suppression immédiate de *v*. Cela ne modifie pas l'ensemble des valeurs indexées. Mais est-ce que cela modifie la structure de l'arbre ? Pour répondre à cette question, on considère un arbre B+ d'ordre 2, nommé A1. La racine de A1 contient les clés 13, 17, 24, 30. Ses feuilles contiennent (toutes feuilles confondues) les valeurs 2, 3, 5, 7, 14, 16, 19, 20, 22, 24, 27, 29, 33, 34, 38, 39.

1) Dessiner A1.
2) Donner 4 valeurs de clé telles que leur insertion successive puis leur suppression dans l'ordre inverse résulte dans un état identique à l'état initial
3) Donner une valeur de clé dont l'insertion suivie de la suppression résulte dans un état différent de l'état initial.
4) Montrer l'état de l'arbre résultant, à partir de l'arbre A1, de l'insertion de la clé 30.
5) On veut déterminer le nombre minimal de clés à insérer consécutivement dans l'arbre **A1 initial** pour qu'il gagne deux niveaux en hauteur (c'est-à-dire pour passer de 2 à 4 niveaux) ?
   a) Sachant qu'on veut insérer le moins de clés possible, quelle est, parmi F1 à F5, la première feuille dans laquelle on insère une valeur? Par la suite dans quelles feuilles est-il préférable d'insérer les autres valeurs ?
   b) Quels nœuds de l'arbre doivent être pleins pour que l'insertion d'une clé dans une feuille provoque (en cascade) un éclatement de la racine ?
   c) Représenter l'arbre obtenu **après** la dernière insertion qui fait passé l'arbre de 3 à 4 niveaux.

Exercice 2 (partiel 2016) Index composé et index couvrant une requête

Soit la relation **Joueur** (<u>nujoueur</u>, nom, prénom, âge, ville, sport)

Tous les index sont non-plaçants. Il y a 3 index :

I1 sur Joueur(ville) **I2** sur Joueur(âge, sport) **I3** sur Joueur(sport, âge)

Soit les requêtes :
R1 : select sport from Joueur where âge >20

R2 : select max(age) from Joueur where sport = 'vélo'

R3 : select distinct ville from Joueur

R4 : select avg(age) from Joueur where ville='Paris' and sport='vélo'

R5 : select prénom from Joueur where prénom like 'Ted%' age = 18 and sport = 'judo' order by prénom

a) Est-ce que I2 est utilisable pour R1 ? Entourer `oui` `non.` Justifier.

b) Est-ce que I3 est utilisable pour R1 ? Entourer `oui` `non.` Justifier.

c) Est-ce que I2 est utilisable pour R2 ? Entourer `oui` `non.` Justifier.

d) Est-ce que I3 est utilisable pour R2 ? Entourer `oui` `non.` Justifier.

e) Est-ce que I1 couvre R3 ? Entourer `oui` `non.` Justifier.

f) Peut-on couvrir R4 avec un (ou plusieurs) index parmi ceux existants ? Si oui lesquels ?

g) On sait que R5 peut être évaluée sans accéder à la table Joueur. Expliquer comment évaluer R5 en utilisant I2(âge, sport) et le nouvel index I4(prénom) sans lire aucun nuplet de Joueur.

Exercice 3 (partiel 2016): Arbres B+

Sauf indication contraire, tous les arbres sont de type arbreB+ d'ordre **1** (i.e., il y a 1 ou 2 valeurs par nœud). On utilise la syntaxe suivante pour représenter un nœud de l'arbre : $N\ (\ v_1,\ v_2,\ \ldots)$ où N est le nom du nœud et les v_i sont les valeurs. Quand la feuille F déborde, on garde les **2** plus petites valeurs dans F, la plus grande valeur sera dans la nouvelle feuille. S'il faut choisir une valeur pour un nœud intermédiaire, la choisir, autant que possible, identique à une valeur existant dans une feuille. Toutes les valeurs sont des nombres entiers.

Question 1)

a) Au moment d'insérer une nouvelle valeur, quel est l'inconvénient d'avoir un arbre où tous les nœuds sont déjà remplis?

b) Les trois premières feuilles d'un arbre sont **F1**(9) **F2**(10, 15) **F3**(16). Lorsqu'on insère la nouvelle valeur 13, pourquoi décide-t-on de ne **pas** redistribuer avec F1 ou F3 ?

c) Un arbre a 20 feuilles. Les feuilles et les autres nœuds sont le **plus** remplis possible. Combien de niveaux a l'arbre ? Tenir compte du niveau de la racine et de celui des feuilles. Par exemple, un arbre avec une racine et 3 feuilles a 2 niveaux.

d) Un arbre a 17 feuilles. Les feuilles et les autres nœuds sont le **moins** remplis possible. Combien de niveaux a l'arbre ?

e) Un arbre contient dans ses feuilles les valeurs consécutives {10,11, ... ,26}. Les feuilles et les autres nœuds sont le plus remplis possible. Que contient la racine ?

Question 2) Soit l'arbre A0 composé d'une racine $\mathbf{N_1}(8, 21)$, et des feuilles $\mathbf{F_1}(4)$, $\mathbf{F_2}(10, 13)$ et $\mathbf{F_3}(21)$.

On insère 8. On obtient A1. Dessiner A1.

Question 3) Lors d'une suppression, on considère si possible la redistribution à gauche puis à droite, seulement entre des nœuds ayant le même père. L'arbre initial S_0 est composé :

d'une racine R(27)
des valeurs {21, 25, 100} dans les nœuds intermédiaire nommés N_i
et des valeurs {1, 2, 21, 24, 25, 26, 91, 100} dans les feuilles nommées F_j

a) Dessiner S_0.

b) On supprime 91 dans S_0. Représenter l'arbre S_1 obtenu (dessiner seulement les nœuds modifiés).

c) On supprime successivement les valeurs 24 puis 21 dans l'**arbre initial S_0**. Représenter l'arbre S_2 obtenu.

d) On supprime successivement dans l'**arbre initial S_0** les valeurs, dans l'ordre croissant, 1, 2, 21,... jusqu'à ce que l'arbre perdre un niveau. Représenter l'arbre S_3 obtenu.

Exercice 4 : (partiel 2002) Profondeur des arbres B+

Soit la relation **R** (a1, a2, a3) contenant 10 millions de tuples. L'attribut a1 est la clé primaire, il est indexé par un arbre B+. L'index est dense : chaque valeur de a1 correspond à une clé d'une feuille de l'arbre.

Question 1) Le nombre de clé par nœud est *nc* tel que : $nc \in [4,8]$ pour tout nœud sauf la racine, $nc \in [1,8]$ pour la racine
La profondeur *p* d'un arbre correspond au nombre de niveaux, racine incluse, soit:

$p = 1$ pour un arbre réduit à sa seule racine,
$p = 2$ pour un arbre ayant seulement une racine et des feuilles,
etc…

Quelle est la profondeur minimale de l'index sur a1? Expliquez brièvement votre réponse.

Exercice 5 (partiel 2002): Arbres B+ : insertion et suppression de clés pts

Soit un arbre B+ dont le nombre de clé par nœud est *nc* tel que :
$nc \in [2,4]$ pour tout nœud sauf la racine, et $nc \in [1,4]$ pour la racine.
a) Dessiner l'arbre **A1** ayant 2 niveaux tels que :
- la racine a les clés 10, 20, 23, 40.
- les feuilles ont les clés 1, 2, 4, 6, 10, 12, 14, 20, 22, 24, 30, 32, *c*, 42 avec $(32 < c < 42)$.

b) Donner toutes les valeurs possibles pour la clé *c*.
c) Représenter l'arbre **A2** après insertion de la clé 8 dans A1, sans redistribuer les clés avec les voisins. En cas d'ajout d'un nœud, le nœud créé doit contenir le nombre minimal de clés.

d) Représenter l'arbre **A3** après suppression de la clé 8 dans A2. On considère la redistribution éventuelle avec le voisin de gauche d'abord.
e) Combien de clés au maximum peut on supprimer dans l'arbre **A3** sans qu'il perde un niveau ? Donner un exemple de clés que l'on peut supprimer.

Exercice 6 (partiel 2003) : Arbre B+

Soit un arbre B+ avec 3 niveaux. Le nombre de clés par nœud est tel que :
la racine et les nœuds intermédiaires contiennent de **1** à **2** clés,
les feuilles contiennent de **2** à **3** clés,

Question 1 :

1.1) Dessiner l'arbre **A1** ayant 3 niveaux tels que :
Les feuilles ont les clés 1, 4, 9,16, 25, 36, 49, 54, 61, 70, 81, 84, 87, 88, 95, 99.
Le niveau intermédiaire a les clés 9, 54, 70, 88
La racine contient 2 clés (pour les clés de la racine: choisir les **plus petites** valeurs possibles **parmi les clés des feuilles**).

1.2) Représenter l'arbre A2 après insertion de la clé 32 dans A1, sans jamais redistribuer de clé avec les voisins. Créer un nouveau nœud en cas de débordement d'une feuille ou d'un nœud intermédiaire.

1.3) Soit l'arbre A3 après insertion de la clé 32 dans **A1**. L'arbre A3 est obtenu en redistribuant si possible les clés des niveaux intermédiaires. Quel est le nombre de nœuds de l'arbre A3 (racine, nœuds intermédiaires et feuilles inclus) ?

1.4) Représenter l'arbre A4 après suppression de la clé 16 dans **A1**. L'arbre A4 est obtenu en redistribuant si possible les clés des feuilles avec les voisins.

1.5) Quel est le nombre minimum de clé à supprimer dans A1 pour qu'il perde un niveau ? Justifier votre réponse. Donner un exemple de clés à supprimer (en choisissant des valeurs de clé les plus petites possibles).

Question 2 : Index plaçant et non dense
Soit la relation **Produit** (<u>numéro</u>, prix). Les produits sont stockés dans l'ordre croissant du numéro. L'attribut *numéro* est une clé, il est indexé par un arbre B+. Les attributs sont indépendants et leur distribution est uniforme.
La relation *Produit* a 100 000 n-uplets, une page de données contient 100 n-uplets.
L'index est non dense (i.e., une seule clé par page de données).

2.1) Quel est le nombre minimum de clés au niveau des feuilles ? Justifier votre réponse.

2.2) Quel est le nombre minimal de niveaux de l'index ? Justifier votre réponse.

Exercice 7 : (partiel 2004) Arbre B+

On considère des arbres B+ contenant des données de type entier, tel que les nœuds et les feuilles contiennent au plus quatre valeurs. Les noeuds (sauf la racine) et les feuilles doivent être au moins à moitié pleins (2 valeurs au moins). La hauteur d'un arbre est égale à son nombre de niveaux. Un arbre de hauteur 1 est réduit à sa seule racine.

Question 1. Donnez un exemple d'arbre B+ dont la hauteur passe de 2 à 3 lorsqu'on y insère la valeur 25. Donnez les deux arbres, avant et après l'insertion. Utiliser la trame quadrillée pour dessiner les nœuds et laisser un espace entre deux nœuds. Répondre page suivante.

Question 2. Donnez un exemple d'arbre B+ dans lequel la suppression de la valeur 25 conduit à une redistribution. Donnez les deux arbres, avant et après la suppression.

Question 3. Donnez un exemple d'arbre B+ dans lequel la suppression de la valeur 25 conduit à une fusion de deux nœuds, mais ne modifie pas la hauteur de l'arbre. Donnez les deux arbres, avant et après la suppression.

Exercice 8 : (partiel 2005) Arbre B+

On considère un arbre B+ tel qu'un nœud quelconque peut contenir de **1** à **3** clés.

Question 1

a) Dessiner l'arbre résultant de l'insertion successive des 10 clés 1, 2, ..., 10, dans l'ordre croissant, dans un arbre initialement vide. En cas de débordement d'une feuille, l'éclater en 2 feuilles ayant chacune le même nombre de clés. Utiliser la trame quadrillée pour dessiner les nœuds et laisser un seul espace entre deux nœuds.

b) On considère l'insertion successive de N clés 1, 2, ..., N, dans l'ordre croissant, dans un arbre initialement vide. Donner la valeur maximale de N lorsque l'arbre obtenu est de profondeur 3 (un seul niveau intermédiaire).

c) On considère l'arbre de profondeur 3 qui a le nombre maximum de clés (sans se préoccuper de la façon d'obtenir cet arbre), et M le nombre total de clés dans ses feuilles. N_{max} est il égal à M ? Justifier.

Question 2

Donner une suite de 10 clés à insérer successivement, dans un arbre vide afin d'obtenir un arbre de profondeur 2 (pas de niveau intermédiaire), en choisissant la valeur des clés parmi les entiers dans [1, 10]. Puis représenter l'arbre obtenu.

Indication : Répondre de manière à insérer autant que possible les plus petites valeurs en premier. Parmi toutes les réponses possibles, donner celle qui est la plus "proche" d'une suite croissante. C'est-à-dire, donner la suite qui est composée d'un nombre minimum de sous-suites croissantes.

EXERCICES DE RAPPEL (cf 3I009)

Ex. 9 (rappel) : Insertions successives

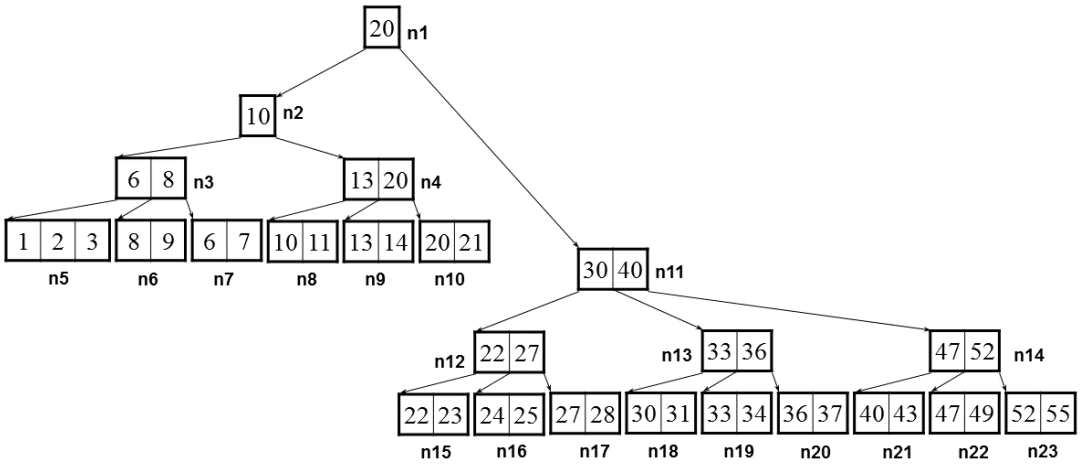
Dans cet exercice, les arbres sont d'ordre 1 (i.e., il y a 1 ou 2 valeurs par nœud).

1) Soit l'arbre A0 composé d'une racine N1(10), et de deux feuilles F1(5) et F2(20). A0 a 2 niveaux. On insère successivement dans A0 les nombres entiers 8 puis 30 puis 15. On obtient A1. Dessiner A1.

2) On insère dans A1 le nombre 10. On obtient A2. Dessiner A2.

Ex.10 (rappel) : Chercher l'erreur

On considère l'arbre B+ d'ordre 2 suivant :



1) Cet arbre est-il équilibré ? Pourquoi ?

2) Trouver les erreurs dans cet arbre. Indiquer quel nœud n_i est erroné et expliquer brièvement l'erreur. S'il est possible de corriger l'erreur sans restructurer l'arbre, mais en modifiant seulement des valeurs de clés, alors suggérer une correction.

Ex. 11 (rappel) : Insertion, suppression, perte d'un niveau

On considère un arbre B+ d'ordre d=2. La racine de l'arbre **A1** contient la valeur 50. Un seul niveau intermédiaire contient (tous nœuds confondus) les valeurs 8, 18, 32, 40 73, 85. Les feuilles contiennent (toutes feuilles confondues) les valeurs 1, 2, 5, 6, 8, 10, 18, 27, 32, 39, 41, 45, 52, 58, 73, 80, 91, 99.

1) Dessinez l'arbre A1.
2) Dessiner l'arbre A2 après l'insertion de la valeur 9 dans A1. Combien valent L et E ?
3) Dessiner l'arbre A3 après l'insertion de la valeur 3 dans A1. Combien valent L et E ?
4) a) Dessiner l'arbre A4 après suppression de la valeur 8 dans A1. Si nécessaire, on envisage une redistribution à gauche. Combien valent L et E ?

   b) Même question mais en considérant uniquement la redistribution à droite si possible, sinon fusionner 2 feuilles.
5) Montrer l'état de l'arbre résultant, à partir de l'arbre A1, de l'insertion de la clé 46 suivie de la suppression de la clé 52
6) Montrer l'état de l'arbre résultant, à partir de l'arbre A1, de la suppression successive des clés 32, 39, 41, 45 et 73.

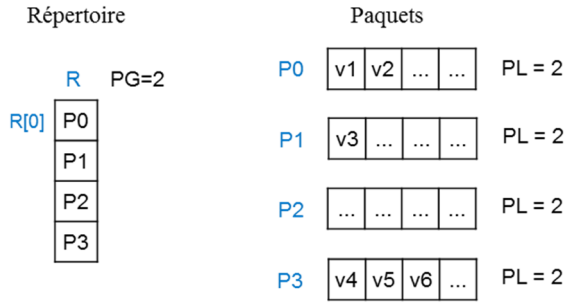
TD 2 : Hachage extensible

Notations et conventions

Pour toutes les tables de hachage étudiées en TD, on limite le nombre de valeurs dans un paquet à **4** au maximum. Ceci permet d'étudier plus facilement sur des petites structures les cas d'insertion avec d'éclatement et de suppression avec fusion. On utilise la notation suivante :

Le répertoire est noté **R** [P_0, P_1, P_2,..., P_k] **PG=***pg* avec
P_i ... les noms d'un paquet,
pg la profondeur globale.
Rmq : le répertoire contient k cases avec k = 2^{pg} -1
Un paquet est noté $P_i(v_j, ..., ...)$ **PL=***pl* avec
P_i le nom du paquet, par exemple P0, P1, A ou B
V_j. les valeurs que contient le paquet,
pl la profondeur locale du paquet P_i.

On peut aussi préciser le contenu d'une case particulière du répertoire avec R[i]= P_j (avec R[0] étant la 1ère case).



Suppression : Lors d'une suppression, si un paquet devient vide, on tente de le fusionner seulement si sa profondeur locale est égale à la profondeur globale, sinon il reste vide.

Exercice 1 (partiel 2016) : Table de hachage extensible

Question 1. Dans cette question. Un paquet peut contenir au maximum **2** valeurs (rmq : seulement 2, pas 4).

1) On considère une structure de hachage extensible de profondeur globale PG=3. Le répertoire contient 8 paquets ayant tous une profondeur locale PL=3 : R [P0, P1, P2, P3, P4, P5, P6, P7].

On insère les valeurs 1, 6, 12, 15, 19, 32, 42, 81, 808.

Que contiennent les paquets P0 à P7 ?

2) On considère une table de hachage extensible **T1** de profondeur globale PG=2. Le répertoire contient 4 paquets ayant tous une profondeur locale PL=2 : R [P0, P1, P2, P3].

P0(4, 44) P1(13, 17) P2(6) P3(3)

On supprime 6 de la table **T1** en tentant de fusionner les paquets vides. Quel répertoire obtient-on?

Préciser les paquets supprimés et/ou modifiés :

Question 2. Dans cette question un paquet peut contenir jusqu'à **4 valeurs** au maximum.

On considère la table **T2** avec les paquets A(9) et B(10).

a) Le répertoire est-il R[A, B] ou R[B, A] ? On insère 1, 2, 5, 7, 8, 12 dans T2. Représenter la table **T2'** obtenue.

b) On insère 13 dans **T2'** obtenue à la question précédente. Représenter la table **T3** obtenue.

c) On insère 41 dans **T3** obtenue à la question précédente. Représenter la table **T4** obtenue.

Question 3. On a deux tables de hachage H0 et H1 ayant une profondeur PG=1. Chaque table a 2 paquets avec **4 valeurs par paquet**. On veut indexer les nombres pairs dans H0 et impairs dans H1. H0 doit indexer les valeurs {2, 4, 8, 10,12} et H1 les valeurs {1, 5, 9, 11, 19}. Expliquer quelle fonction de hachage utiliser pour qu'on puisse indexer ces valeurs en remplissant les paquets existants, sans créer aucun nouveau paquet.

Exercice 2 (partiel 2015)

1) La table de hachage T0 contient seulement les valeurs 1, 4, 5, 6, 9, 14, 25. De combien de paquets a-t-on besoin au **minimum**? Préciser leur contenu, leur PL, le contenu du répertoire.

2) On insère 7 dans T0. On obtient T1. Représenter T1. Préciser ce que contient chaque case du répertoire R

3) On insère 13 dans **T1** obtenu à la question précédente. On obtient T2. Représenter T2.

4) On considère la table T3 avec les paquets A(10), B(16), C(17,31), D(30), E(36). Représenter T3. Rmq : vous devez préciser le contenu de chaque case du répertoire (R[0] est différent de A, la taille du répertoire est une puissance de 2).

5) On supprime 10 dans T3. On obtient T4. Représenter T4

6) On supprime 30 dans T4 obtenu à la question précédente. On obtient T5. Représenter T5.

Exercice 3 (partiel 2014) : Table de hachage extensible pts

Dans toutes les tables de hachage considérées, un paquet ne peut pas contenir plus de 4 valeurs.
Une table de hachage T1 contient *seulement* les 5 paquets suivants:

A (8)
B (9, 29)
C (10, 22, 42)
D (11, 27)
E (23, 39)

1) Structure de T1.

a) Est-ce que le répertoire est tel que R[0] = A et R[4] = E ?

b) Pour chaque paquet, quelle est sa profondeur locale ?

c) Que contient le répertoire? Préciser le nom du paquet référencé dans chaque case.

2) Dans T1, on insère successivement 6 puis 18. Quels sont les paquets créés et/ou modifiés et leur contenu ? Préciser aussi les cases modifiées du répertoire.

3) Dans T1, on insère successivement 3 valeurs dans le paquet E. Est-ce que cela a pour effet de doubler la taille du répertoire ?

4) Dans T1, on supprime 9 puis 29. Quels sont les paquets supprimés et/ou modifiés et leur contenu ?

5) Dans T1, on supprime 8. Quels sont les paquets supprimés et/ou modifiés et leur contenu ?

6) On a un million (10^6) de valeurs à indexer. Quelle sera la taille minimale du répertoire ?

7) Question bonus : index bi-attribut. On considère la relation Restau(a, b, c) contenant 200 triplets. Les valeurs a, b, c sont des entiers. On crée un index avec la commande :

create index IAB on Restau (a, b);

L'index utilise des techniques de hachage avec des paquets contenant au plus 4 valeurs. On veut utiliser un répertoire en forme de matrice pour pouvoir accéder à une case du répertoire en fonction de la valeur de *a* et de celle de *b*. Préciser la (ou les) fonction(s) de hachage à utiliser et expliquer comment accéder au triplet dont les valeurs (a, b) valent (12, 130).

Exercice 4 (partiel 2013) : Indexation avec table de hachage 4 pts

La fonction x modulo y est notée : *x mod y*. Chaque paquet **contient au plus 2 valeurs.**

Question 1. On considère un répertoire de profondeur globale PG=1. Avec 2 paquets P0 et P1 tels que R=[P0,P1]. Initialement les deux paquets contiennent :

P0(4,8) **P1**(1,3)

On insère la valeur 12.

a) Est-ce que cette insertion provoque plus d'un éclatement ? Quelle sera la profondeur globale de la table obtenue après l'insertion ?
b) Détailler la table obtenue après insertion. Préciser le contenu des paquets modifiés ou créés, et leur profondeur locale (PL).

Question 2. On a un répertoire de profondeur PG=3. Les paquets contiennent les valeurs : 1, 7, 8, 10, 11, 16, 20.
a) De combien de paquets a-t-on besoin au minimum? Préciser leur contenu, leur PL, les liens entre les cases et les paquets

b) On insère 19. Détailler la table obtenue en précisant *seulement* les paquets, les liens et les profondeurs modifiés.

Question 3. Répartition d'une table de hachage. Pour indexer un attribut dont le domaine a de nombreuses valeurs, on veut construire une « grande » table de hachage dont le répertoire a 32 cases et autant de paquets. Or, on suppose que la plus grande table tenant dans la mémoire d'une machine a un répertoire de 8 cases et 8 paquets de 2 valeurs. Ainsi, on propose de répartir l'index sur 4 machines M0 à M3 gérant chacune la table de hachage T0 à T3 respectivement. Pour chaque Ti, la profondeur est PG=3.
Initialement, chaque Ti a déjà 8 paquets (PL=3) notés $P_{i,0}$ à $P_{i,7}$. Il y a une valeur et une place libre dans chaque paquet.

On insère la valeur v = 49. Décrire les étapes de l'insertion, et préciser dans quelle Ti et quel paquet est insérée la valeur 49.

EXERCICES DE RAPPEL (*cf.* 3I009)

Exercice 4 (rappel): Hachage extensible

On considère une base de données contenant la relation Personne (nom, prénom, ville). On veut indexer les personnes sur l'attribut ville. La relation contient 11 villes différentes dont les codes sont 1, 4, 5, 7, 10, 12, 15, 16, 19, 21, 32. Pour cela, on construit une structure de hachage extensible contenant une entrée pour chacune des 11 valeurs.

1) Initialement, on veut construire une table de hachage avec suffisamment de place pour contenir les 11 valeurs. Avec au plus 4 valeurs par paquets, combien faut-il de paquets au minimum? Quelle doit être la taille minimale du répertoire pour atteindre ces paquets ?
2) Dessiner la table de hachage, nommée T1, après avoir inséré les 11 valeurs. Représenter le répertoire avec le nom des paquets qu'il contient et sa profondeur globale. Représenter chaque paquet avec les valeurs qu'il contient et sa profondeur locale.
3) Expliquer pas à pas comment retrouver le paquet contenant la valeur 7.
4) Insérer 13
5) Insérer 20
6) Insérer 29
7) On considère la table obtenue. Donner un exemple de plus petit ensemble de valeurs à supprimer pour obtenir une division par 2 du répertoire.

Exercice 5 (rappel) : Hachage extensible

On considère une structure de hachage extensible de profondeur globale 3. Dans l'état initial, nommé **T0**, les paquets du répertoire sont les suivants : R[A, B, C, D, A2, B, C, D] (*i.e.*, R[0]=A, ..., R[7]=D), et PG=3

Les valeurs (tous paquets confondus) sont les suivantes : 1, 4, 5, 7, 10, 12, 15, 16, 20, 21, 36, 51, 64.

1) Représenter T0.
2) Insérer 68 dans T0.
3) On considère la table T0 initiale. Insérer 17 et 69 dans T0.
4) Supprimer 21 de T0.
5) Supprimer 10, puis 64 et 16 deT0.

TD Optimisation de requêtes

Notations et conventions

Dans tous les exercices, on suppose que la distribution des attributs est uniforme. Les attributs sont tous indépendants les uns des autres.

On s'intéresse au coût des opérations en termes de quantité d'accès aux données. L'unité de coût est soit la page (exercice 1), soit le nuplet (ex. 2 et 3).

Une opération (sélection, projection, jointure) qui accède à une *table* stockée dans le SGBD a un coût.

Une opération de sélection (ou projection) qui accède au *résultat* d'une requête R ne rajoute pas de coût supplémentaire car on peut évaluer la sélection (ou la projection) sur chaque nuplet résultant de R, sans accéder à aucune autre donnée de la base.

Le coût de la jointure notée $A \bowtie_{A.a=B.a} B$ représente la quantité d'accès à la relation B nécessaire pour évaluer la jointure. Evaluer la jointure consiste à parcourir les nuplets résultant de A afin d'accéder itérativement aux nuplets de B qui satisfont le prédicat de jointure (cf. cours : algorithmes de jointure). Cela suppose que A soit le *résultat* d'une requête et que B soit une *table* stockée dans le SGBD. Le coût de A est calculé de manière séparée et vient s'ajouter à celui de la jointure. Si B n'est pas une table de la base mais une requête, alors on stocke B comme une *table temporaire* dans le SGBD avant d'effectuer la jointure.

Exercice 1 : Jointure entre 2 relations

Soit les relations :

R (a, b) et **S** (c),

Le domaine des attributs est : $a \in [1,1000]$, $b \in [1,10]$, $c \in [1,100]$.
Les cardinalités des relations sont : card(R) = 1000, card(S)= 100
La taille en nb de pages des relations est : P(R) = 500, P(S) = 10

Le modèle de coût simplifié est :
Le coût d'une équi-jointure entre les relations A et B, avec le prédicat A.a = B.a est :

$coût(A \bowtie_{A.a=B.a} B) = card(A) * coût(\sigma_{a=v} B)$ où v est une valeur quelconque.

Le coût d'une sélection sur la relation A avec le prédicat a=v est :

si *a* est indexé : $coût(\sigma_{a=v} A) = card((\sigma_{a=v} A)$
sinon : $coût(\sigma_{a=v} A) = P(A)$.

Le coût d'une lecture séquentielle est le coût d'une sélection sans index.

Soit la requête :

```
select *
from R, S
where a = c and b =1
```

Pour chaque question, énumérer tous les plans équivalents (notés P1 à P4) et calculer leur coût. Pour cela, représenter un plan d'exécution sous la forme d'un arbre d'opérateurs. Donner le coût et la cardinalité de chaque opérateur de l'arbre, donner le coût total de l'arbre.

1) Il y a seulement un index sur R.a et un index sur R.b,
2) Il y a seulement un index sur S.c
3) Il y a seulement un index sur R.b et un index sur S.c.

Exercice 2 : Jointure entre 3 relations

L'objectif de cet exercice est de comparer deux méthodes pour le choix du plan d'exécution d'une requête. La première méthode est la transformation de plan basée sur des heuristiques. La 2ème méthode est la génération exhaustive de l'espace de recherche associée au choix du plan candidat de moindre coût.

Soit le schéma **R1**(A,B, …), **R2**(A,B, …) et **R3**(A, B, …)

Soit la requête :

```
select R1.A, R3.B
from R1, R2, R3
where R1.A=R2.A and R2.A=R3.A and R1.B=1 and R2.B=2
```

1) Donner l'arbre algébrique, nommé P1, correspondant en respectant l'ordre des prédicats donnés dans la clause *where*.

2) Donner un arbre équivalent, nommé P2, en appliquant les opérations les plus réductrices (sélection puis projection) d'abord.

3) Soit le modèle de coût simplifié suivant, où l'unité de coût est l'accès à un nuplet.

- Pour toute relation R, si card(R) est le nombre de tuples de R, le coût d'une *sélection* sur égalité est :

  coût ($\sigma_{a.=valeur}(R)$) = card($\sigma_{a.=valeur}(R)$) s'il y a un index sur l'attribut *a*

  = card(R) sinon.

- Le coût d'une lecture séquentielle est le coût d'une sélection sans index.
- Pour toutes relations R ayant card(R) nuplets et S ayant card(S) nuplets, le coût d'une *equi-jointure* est :

  coût ($R \bowtie_{a.} S$) = card(R) s'il y a un index sur l'attribut *a* de *S*

  = card(R) * card(S) sinon.

- On suppose que R1, R2, R3 ont les caractéristiques suivantes :
  - il y a un **index** sur l'attribut B de R1, A est clé primaire de R1, R2 et R3 (il existe un index pour chaque clé primaire)
  - card(R1) = card(R2) = card(R3) = 1000, $\pi_A(R1) = \pi_A(R2) = \pi_A(R3)$
  - il y a 10 valeurs possibles pour B, uniformément réparties dans R1 et aussi dans R2 et dans R3.

Questions :

a) Quelle est la cardinalité du résultat de la requête ?

b) Pour simplifier, on ignore les projections (car leur coût est nul). Donner l'expression algébrique de P1' (resp P2') correspondant à P1 (resp. P2) sans aucune projection.

c) Donner le coût de P1' et P2'. Préciser votre réponse en détaillant le coût et la cardinalité des résultats intermédiaires.

d) Quel est l'arbre de coût minimal pour évaluer la requête ? Quel est son coût ?

Exercice 3: Club de joueurs - A faire ou à continuer en TME

Soit le schéma relationnel :

Joueur (licence: integer, cnum : integer, salaire: integer, sport: char(20))

Club (cnum: integer, nom: char(20), division: integer, ville : char(10))

Finance (cnum: integer, budget: real, dépense: real, recette: real)

On considère la requête :

```
SELECT C.nom, F.budget
FROM Joueur J, Club C, Finance F
WHERE J.cnum = C.cnum AND C.cnum = F.cnum
AND C.division = 1 AND J.salaire > 59000 AND J.sport = 'aviron'
```

1) Déterminer un arbre d'opérateurs de l'algèbre relationnel qui reflète l'ordre des opérations qu'un optimiseur de requête peut choisir. Combien y a-t-il de possibilités équivalentes pour ordonner les jointures ?

2) Pour réduire l'espace de recherche exploré pendant l'optimisation, on considère seulement les arbres de jointure qui n'ont pas de produit cartésien et qui sont linéaires à gauche. Donner la liste de tous les arbres de jointure construits. Expliquer comment vous obtenez cette liste.

Les informations suivantes sont extraites du catalogue du SGBD.
Les attributs Joueur.cnum, Joueur.salaire, Club.division, Club.cnum et Finance.cnum sont indexé par un arbre B+.
Le salaire d'un joueur est compris entre 10.000 et 60.000 EUR.
Les joueurs peuvent pratiquer 200 sports différents. Un club est en division 1 ou 2.
La BD contient au total 50000 joueurs et 5000 clubs. Il y a un nuplet d'information financière par club.

3) Pour chaque relation (Joueur, Club et Finance) estimer le nombre de nuplets qui sont sélectionnés après avoir traité les prédicats de sélection et avant de traiter les jointures.

4) D'après la réponse à la question précédente, quel est l'arbre de jointure de coût minimum que l'optimiseur construit ?

EXTRAIT DU PARTIEL BDA (AVRIL 2002)

Exercice 5 : Dictionnaire pour les statistiques pts

Un SGBD maintient des statistiques sur les données de la BD. Périodiquement et si les données ont été mises à jour, le SGBD actualise les statistiques au moyen de requêtes SQL.
Soit la relation R(a1, a2, a3). La distribution des valeurs des attributs de R est uniforme.
a) Donner les requêtes SQL pour calculer les statistiques suivantes :
Les bornes du domaine de a1
Le nombre de valeurs distinctes de a2
b) Donner une requête SQL pour vérifier que a2 et a3 sont indépendants. Expliquer comment interpréter le résultat de la requête.
c) Le dictionnaire du SGBD est modifié pour gérer la distribution **non** uniforme des attributs
Relation(nomRel, cardinalité), Attribut(nomRel, nomAtt, val, freq)
Un attribut est identifié par son nom *nomAtt* et le nom de sa relation *nomRel*. ; *val* représente une valeur de l'attribut ; *freq* représente le nombre de tuples dont l'attribut *nomAtt* vaut *val*.
Ecrire, en pseudo-code, un algorithme pour collecter ces statistiques en utilisant des ordres SQL pour manipuler les données du dictionnaire.
Donner en fonction de *R*, *att* et *v*, la requête SQL (sur les relations du dictionnaire) pour déterminer le **facteur de sélectivité** (SF) de l'opérateur de sélection sur la relation *R* avec le prédicat *att*<*v* lorsque la distribution de l'attribut *att* est non uniforme. (*ie*. SF($\sigma_{att<v}(R)$))

Exercice 6 : Optimisation de requêtes pts

Soit les relations R(a,b), S(c), la distribution des attributs est uniforme,
Le domaine des attributs est : a∈[1,1000], b∈[1,10], c∈[1,100].
La cardinalités des relations sont : card(R) = 1000, card(S)= 100
La taille en nb de pages des relations est : P(R) = 500, P(S) = 10
Le modèle de coût simplifié est :
Le coût d'une jointure '∞' entre A et B, avec le prédicat A.att = B.att est :

coût($A \infty_{A.att=B.att} B$) = card(A) * coût ($\sigma_{att=x}(B)$) où x est une valeur quelconque.

Le coût d'une sélection sur A avec le prédicat att=val est :

si att est indexé : coût($\sigma_{att=val}(A)$) = card(($\sigma_{att=val}(A)$)

sinon : coût($\sigma_{att=val}(A)$) = P(A).

Le coût d'une lecture séquentielle est le coût d'une sélection sans index.
Le coût d'un opérateur traité en pipeline est nul.

Soit la requête : select * from R,S where a=c and b =1
Pour chaque cas a), b) et c), représenter le plan d'exécution de coût minimal de la requête, sous la forme d'un arbre d'opérateurs. Donner le coût et la cardinalité de chaque opérateur de l'arbre, donner le coût total de l'arbre.

a) il y a seulement un index sur R.a et un index sur R.b,

b) il y a seulement un index sur S.c,

c) il y a seulement un index sur R.b et un index sur S.c.

Extrait du partiel BDR (avril 2003)

Exercice 1 : Optimisation — 6 pts

On considère la jointure naturelle des relations R(a,b), S(c,b), T(c,d) et U(a,d).
Pour traiter un arbre de jointures, le SGBD exécute les jointures les unes après les autres, en matérialisant le résultat d'une jointure avant de traiter la jointure suivante.
Le coût d'un arbre de jointures est le nombre de n-uplets matérialisés dans les relations intermédiaires.
On souhaite trouver l'arbre de jointures de coût minimum.

Le modèle de coût d'une jointure est donné par la règle :

coût (A ⋈ B) = card (A ⋈ B) pour un nœud interne (car la relation intermédiaire est matérialisée)
= 0 pour la racine (car le résultat est directement envoyé au processus client)

Les relations R et U ont 1000 n-uplets alors que S et T ont 100 n-uplets.Il y a 100 valeurs différentes pour tous les attributs de toutes les relations excepté pour S.c et T.c qui ont seulement 10 valeurs différentes. On note ces informations de la manière suivante :

card(R) = card(U) = 1000
card(S) = card(T) = 100
nb_val(R.a) = nb_val(R.b) = nb_val(S.b) = nb_val(T.d) = nb_val(U.a) = nb_val(U.d) = 100
nb_val(S.c) = nb_val(T.c) = 10

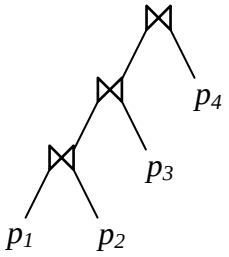
La cardinalité du résultat d'une jointure est donnée par la formule :

card(A ⋈ B) = card(A) * card(B) / max(nb_val(A.*x*), nb_val(B.*x*)) (*x* est l'attribut de jointure)

On rappelle que la jointure naturelle de deux relations est une équi-jointure sur les attributs de mêmes noms.

Question 1 :

On considère uniquement les arbres linéaires gauches de la forme :



1.1) Parmi R, S, T, U, quelles relations peuvent être à la place p_1 ?

1.2) Si R est à la place p_1, quelles relations peuvent être à la place p_2 ?

1.3) Combien existent-ils d'arbres linéaires gauches équivalents pour la jointure naturelle de R, S, T, U ?

1.4) Donner un des arbres linéaires gauches de plus faible coût ? Donner son coût.

Question 2 :

Si on considère maintenant tous les arbres de jointures : les arbres linéaires gauches, les arbres linéaires droits et les arbres équilibrés (dits touffus).

Quel est un des arbres de plus faible coût ? Donner son coût.

Extrait du partiel BDR (avril 2004)

Exercice 3 : Optimisation de requêtes 4 pts

Soit le schéma relationnel décrivant l'organisation d'un laboratoire en départements contenant des employés et des projets.

Emp (Enum, salaire, age, Dnum)

Dept (Dnum, Pnum, budget, statut)

Proj (Pnum, code, description)

Tous les attributs sont des nombres entiers sauf *statut* et *description* qui sont des chaînes de caractères. Les attributs soulignés forment une clé. La clé de Dept est composée de deux attributs, (l'attribut Dnum seul n'est pas une clé primaire de Dept).

La taille d'un n-uplet est respectivement de 20 octets pour Emp, 40 octets pour un Dept et 2000 octets pour Proj. Le nombre de n-uplets par relation et respectivement de 20 000 pour, Emp, 5000 pour Dept et 1000 pour Proj. La taille d'une page, sur le disque ou en mémoire, vaut 4000 octets. La fonction page(R) retourne le nombre de pages contenant les n-uplets de R.

La distribution des valeurs des attributs est uniforme. Les attributs sont indépendants. Le domaine de l'âge des employés est l'ensemble des nombres entiers de 20 à 69 inclus : {20, 21, …, 69}. Le domaine des budgets est l'ensemble des multiples de 1000 inclus dans]100 000, 600 000].

On suppose que tous les index sont des arbres B+. Un index sur l'attribut A est dit *plaçant* si les données sont **triées** sur le disque dans l'ordre des valeurs de A. Un index sur l'attribut A est dit *non plaçant* si les données ne sont **pas triées** sur le disque dans l'ordre de A.

L'estimation du coût des opérations repose sur le modèle suivant:

- Le coût d'une lecture séquentielle de la relation R est égal au nombre de pages de R.
- Le coût d'une sélection avec un prédicat *pred* de la forme *a op v*

  où *a* est un attribut de type entier et *op* est l'opérateur = (égal) , < (inférieur à) ou < (supérieur à)

  coût(σ_{pred} (R)) = card(σ_{pred} (R)) si l'attribut *a* est indexé par un index non plaçant,

  coût(σ_{pred} (R)) = page(R) * card(σ_{pred} (R)) / card(R) si l'attribut *a* est indexé par un index plaçant,

  coût(σ_{pred} (R)) = **page**(R) si l'attribut *a* n'est pas indexé.

Précisez clairement toute hypothèse supplémentaire que vous jugez nécessaire pour répondre aux questions posées.

Question 1. Soit la requête R1: `select * from Emp where age=30`

a) Quel est le coût pour traiter R1 s'il existe un index non plaçant sur l'attribut *age* ?

b) Quel est le coût pour traiter R1 s'il existe un index plaçant sur l'attribut *age* ?

Question 2. Soit la requête R2 `select * from Proj where code=20`

On suppose que la cardinalité de R2 est identique à celle de R1.

a) Quel est le coût pour traiter R2 s'il existe un index non plaçant sur l'attribut *code* ?

b) Quel est le coût pour traiter R2 s'il existe un index plaçant sur l'attribut *code* ?

Question 3. Soient *n* un nombre entier et la requête R3 dépendant de *n* :

```
select * from Dept where budget > n
```

a) On suppose qu'il existe un index **non plaçant** sur l'attribut *budget*. Pour quelles valeurs de *n*, la lecture séquentielle de Dept est moins coûteuse que l'accès par index, pour traiter R3 ? Répondre en donnant toutes les valeurs de *n*.

b) Même question mais avec un index **plaçant** sur l'attribut *budget*. On suppose, pour cette question, que la traversée de l'index coûte 3 lectures.

BDR
EXTRAIT Partiel du 5 avril 2005

Exercice 2 : Optimisation de requêtes 10 pts

Soit le schéma suivant :

Emp (ne, salaire, age, ns) -- un employé est identifié par son numéro *ne*

Service (ns, np, budget, statut) -- *ns* fait référence à un numéro de service.

Projet (np, code, libellé) -- un projet est identifié par son numéro *np*

La taille d'un n-uplet est de 20 octets pour Emp, 40 octets pour Service, 2000 octets pour Projet. Les attributs *ne, ns* et *np* ont chacun 4 octets. La cardinalité des relations est de 20 000 pour Employés, 5000 pour Service et 1000 pour Projet. La relation Service représente l'**association N-M** d'un service avec un projet. La clé de Service est composée des attributs *ns* et *np* (ainsi, l'attribut *ns* n'est pas unique dans Service).

Chaque Service, identifié par *ns* a en moyenne 10 Projets. Les données sont stockées sur disque dans des pages de 4000 octets. Les attributs sont indépendants et leur distribution est uniforme.

Soient les fonctions auxiliaires :

ntp(R) le nombre de tuples par pages pour la relation *R*,

D(R, c) le nombre de valeurs distinctes du domaine de l'attribut *R.c*,

largeur(R) la taille d'un tuple de R,

et *arr(x)* l'arrondi de *x* par excès à une valeur entière,

Un index sur l'attribut *c* est dit *plaçant* si les données sont **triées** sur le disque dans l'ordre des valeurs de *c*. Un index sur l'attribut *c* est dit *non plaçant* si les données ne sont **pas triées** sur le disque dans l'ordre de *c*.

L'estimation du coût des opérations repose sur le modèle suivant. Le modèle de coût estime le nombre de pages à lire et écrire, sans prendre en compte l'écriture du résultat final.

- Le coût d'une lecture séquentielle de la relation R est égal au nombre de pages de R (noté page(R)).

Le coût d'une sélection avec un prédicat *pred* de la forme *a op v* où *a* est un attribut numérique et *op* est l'opérateur = (égal), < (inférieur à) ou < (supérieur à), est :

coût(σ_{pred} (R)) = card(σ_{pred} (R)) si l'attribut *a* est indexé par un index non plaçant,

coût(σ_{pred} (R)) = arr(page(R) * card(σ_{pred} (R)) / card(R)) si l'attribut *a* est indexé par un index plaçant,

coût(σ_{pred} (R)) = page(R) si l'attribut *a* n'est pas indexé.

- Le coût d'une jointure naturelle avec un prédicat de jointure *pred* de la forme *R.c = S.c* dépend de l'algorithme utilisé et de la présence d'index sur les attributs de jointure. Les algorithmes de jointure sont numérotés J1 à J9 :

  - J1: jointure par boucles imbriquées, l'itération sur R imbrique l'itération sur S, sans utiliser aucun index.
  - J2: jointure avec itération sur *R* et accès par index plaçant sur *S.c*
  - J3: jointure avec itération sur *R* et accès par index **non** plaçant sur *S.c*
  - J4 : jointure par tri de *R* et *S* selon *c*, puis fusion sans utiliser aucun index
  - J5 : jointure par fusion, avec des index plaçants sur *R.c* et sur *S.c*
  - J6 : jointure par fusion, avec des index **non** plaçants sur *R.c* et sur *S.c*

- J7 : jointure par tri de *R* selon *c*, puis fusion en utilisant un index plaçant sur *S.c*
- J8 : jointure par tri de *R* selon *c*, puis fusion en utilisant un index **non** plaçant sur *S.c*
- J9 : création d'une table de hachage (non plaçante) de *R* sur *c*, puis jointure avec itération sur *S* et accès à *R* par la table de hachage.

On propose 9 formules de coût :

C1: coût($R \bowtie_{pred} (S)$) = page(R) + card(R) x card(S) / D(S, c)

C2: coût($R \bowtie_{pred} (S)$) = 3 x page(R) + card(S)

C3 : coût($R \bowtie_{pred} (S)$) = 3 x (page(R) + page(S))

C4 : coût($R \bowtie_{pred} (S)$) = page(R) + page(S)

C5: coût($R \bowtie_{pred} (S)$) = page(R) + card(R) x page(S)

C6: coût($R \bowtie_{pred} (S)$) = page(R) + card(R) x arr(page(S) / D(S, c))

C7: coût($R \bowtie_{pred} (S)$) = page(R) + page(S) + card(S) x card(R) / D(R,c)

C8: coût($R \bowtie_{pred} (S)$) = 3 x page(R) + page(S)

C9: coût($R \bowtie_{pred} (S)$) = card(R) + card(S)

Précisez clairement toute hypothèse supplémentaire que vous jugez nécessaire.

Question 1

1) Donner le nombre de services distincts et le nombre moyen de services par projet

2) Donner le nombre de tuples par page des 3 relations.

3) Donner la taille des relations en nombre de pages.

4) Combien de pages sont nécessaires pour stocker conjointement tous les tuples de R qui ont une même valeur pour l'attribut *c* ? Donner une formule en fonction de *R, c* et des notations définies ci-dessus.

Question 2

1) Parmi C1 à C9, quelles sont les formules symétriques (*i.e.,* pour lesquelles R et S ont le même rôle)
2) Parmi J1 à J9, quels sont les algorithmes symétriques (*i.e.,* pour lesquels R et S ont le même rôle)
3) Associer chaque algorithme de jointure avec la formule la plus appropriée. Expliquer vos choix en une phrase. Répondre en complétant le tableau. Indication :

Coût	Explication
J1: C5	page(R) : lire R une seule fois card(R) * page(S): lire S autant de fois qu'il y a de tuples dans R

Question 3.

Soit la requête R1:

```
Select *
From Emp e, Service s
Where e.ns = s.ns
```

1) On suppose qu'il existe seulement un index plaçant sur Emp.ns. Dire, pour chaque algorithme J1 à J9, s'il est utilisable pour traiter R1? Si oui, donner le coût du plan d'exécution de R1. Sinon expliquer brièvement pourquoi.

2) On suppose maintenant qu'il existe un index plaçant sur Emp.ns et un index plaçant sur Service.ns.

Quel est l'algorithme de coût minimal ? Donner son coût.

3) On suppose maintenant qu'il existe seulement un index plaçant sur Service.ns. Le SGBD possédant seulement 7 pages disponibles en mémoire, on choisit de traiter R1 avec l'algorithme suivant :

Etape 1: Lire Emp par blocs de 6 pages (1 page mémoire étant réservée pour écrire le résultat du tri), et créer des sous-listes triées.

Etape 2: Tant que le nombre de sous-listes est supérieur ou égal à 6 :

fusionner 6 sous-listes en 1 seule liste, écrire le résultat sur disque.

Etape 3: Fusionner les listes restantes avec la relation Service

a) Donner le coût de R1, en utilisant cet algorithme, en nombre de pages (lues ou écrites).

b) Combien faut-il de pages mémoire au minimum pour pouvoir utiliser J7 ?

Question 4.

Rappel des formules évaluant la cardinalité de la sélection et de la jointure :

$card(\sigma_F(R)) = SF(\sigma_{(F)}) * card(R)$

où $SF(\sigma_{A = valeur}) = 1 / D(R, A)$

$SF(\sigma_{A > valeur}) = (max(A) - valeur) / (max(A) - min(A))$

$SF(\sigma_{A < valeur}) = (valeur - min(A)) / (max(A) - min(A))$

$card(R \bowtie_{A=B} S) = card(S)$ si A est clé de R, et B est clé étrangère de S.

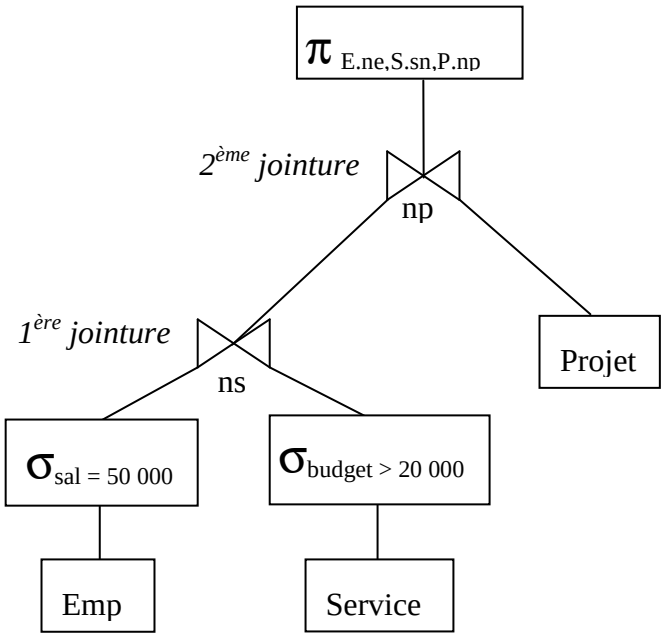
sinon $card(R \bowtie_{A=B} S) = SFj * card(S) * card(R)$ où SFj = 1 / max(D(R, A), D(S, B))

On donne SFj = 1/500 pour la jointure entre Emp et Serv.

Soit la requête **R2** :

```
Select E.ne, S.ns, P.np
From Emp E, Service S, Projet P
Where E.Salaire = 50000
    and S.budget > 20000
    and E.ns=S.ns and S.np=P.np
```

On suppose une répartition uniforme des salaires (par tranche de 5000 dans l'intervalle [10000, 105000]) et des budgets (dans l'intervalle [10000, 30000]). Il y a un index plaçant sur l'attribut salaire de Emp, sur l'attribut ns de Service et sur l'attribut np de Projet.
On considère le plan d'exécution **P1** suivant :



1) Estimez le nombre de n-uplets résultant de chacune des opérations.

card ($\sigma_{sal = 50\,000}$ (Emp)) = ...

card ($\sigma_{budget > 20\,000}$ (Service)) = ...

card (($\sigma_{sal = 50\,000}$ (Emp)) $\bowtie_{ns}$ ($\sigma_{budget > 20\,000}$ (Service))) = ...

card (résultat final) =

2) Calculez le coût du plan d'exécution P1, en précisant les algorithmes de jointure utilisés.

coût ($\sigma_{sal = 50\,000}$ (Emp)) = ...

coût ($\sigma_{budget > 20\,000}$ (Service)) = ...

coût (1ère jointure) = ...

coût (2ème jointure) = ...

coût total = ...

3) Donnez un plan d'exécution de coût optimal pour la requête R2 et calculez son coût.

4) On suppose maintenant que l'index sur l'attribut *np* de Projet est non plaçant. Calculez le coût du plan optimal sous cette hypothèse.
5) Même question en supposant que les index sur *salaire* et sur l'attribut *ns* de *Service* sont non-plaçants. Justifiez votre réponse.