TD 5 : Indexation (corrigé)

Partie 1 : Organisations arborescentes

Arborescences équilibrées et puissance des arbres-B

Question 1

La hauteur d'un arbre-B évolue de façon logarithmique par rapport au nombre de clés stockées dans l'arbre. En effet, dans un arbre-B d'ordre m, chaque nœud possède entre (m+1) et (2m+1) fils. On peut donc stocker, au niveau i+1 de l'arbre-B, entre (m+1) et (2m+1) fois plus de clés qu'au niveau i, suivant que l'arbre est faiblement ou fortement occupé. Les formules de calcul de la hauteur minimale et maximale d'un arbre-B, en fonction de son ordre et du nombre de clés qu'il contient, sont donc:

$$h_{min} \approx \log_{2m+1} n \qquad h_{max} \approx \log_{m+1} n$$

Par exemple, un arbre-B d'ordre 100 peut contenir plus de 8 millions de clés sur seulement trois niveaux.

Question 2

Lors d'une recherche (ou d'une écriture) dans un index, l'accès à chaque nœud traversé dans l'arborescence génère une E/S. Les hauteurs minimale et maximale de l'arborescence bornent donc le coût en E/S de recherche d'une clé dans l'index. Or, il a été montré dans la question précédente que plus l'ordre de l'arbre-B est grand, plus sa hauteur est faible. D'autre part, on a montré que le nombre de comparaisons engendrées par la recherche d'une clé dans un arbre-B est indépendant de l'ordre de cet arbre-B. Il est donc intéressant de choisir un ordre aussi grand que possible afin de stocker un maximum de clés par nœud et minimiser la hauteur de l'index. Ce choix doit cependant tenir compte de la taille maximale des nœuds d'arbre-B. Cette taille est bornée par la taille d'une page disque afin d'assurer que la lecture d'un nœud nécessite une seule E/S.

Question 3

Une arborescence équilibrée garantit un coût d'accès constant à toute valeur de clé indépendamment de la distribution des clés. Il est donc possible de borner le nombre d'E/S nécessaire à la recherche ou à l'écriture d'un article. Cette propriété est particulièrement importante en bases de données. En effet, lorsque plusieurs chemins d'accès peuvent être empruntés pour accéder à un article, un SGBD doit être capable de faire automatiquement le choix optimal.

Question 4

Un nœud d'arbre-B contient un ensemble de couples (clé, pointeur fils) et un pointeur additionnel référençant le premier nœud fils (un nœud contenant n clés possède n+1 fils), comme illustré Figure 3. Si la taille des clés est fixe, cette taille est stockée dans le descripteur d'index. Si la taille des clés est variable, deux solutions sont envisageables:

– les clés sont encadrées par un caractère spécial. Ce caractère, appelé séparateur, doit avoir un codage binaire différent de tous les octets susceptibles d'être contenus dans une clé. De plus, la lecture de la $i^{ème}$ clé d'un nœud nécessite la lecture octet par octet de toutes les clés la précédant dans ce nœud.

– les clés sont précédées d'un champ longueur. Cette technique est la seule applicable si le codage binaire des clés couvre l'ensemble des codes binaires possibles. La $i^{ème}$ clé d'un nœud peut ainsi être accédée en sautant de champ longueur en champ longueur.

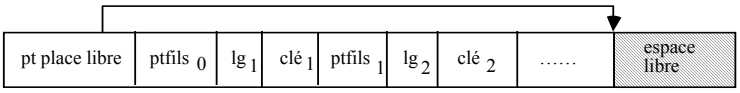

Figure 1. *Format de stockage d'un noeud d'arbre-B*

Les bornes m et 2m assurent que le taux de remplissage d'un nœud varie entre 50% et 100% (et est donc en moyenne de 75%). Dans la définition des arbres-B, ce taux de remplissage k est exprimé en nombre de clés par nœud. Lorsque l'on utilise des arbres-B contenant des clés de taille variable, le taux de remplissage d'un nœud doit alors être exprimé en nombre d'octets plutôt qu'en nombre de clés.

Question 5

Les insertions des clés 37 puis 36 sont illustrées sur la Figure 4. L'insertion de la clé 36 provoque l'éclatement de la feuille où elle doit être insérée. La remontée de la clé médiane 37 provoque à son tour l'éclatement du nœud père de cette feuille. L'arbre-B croît donc par le haut et reste équilibré.

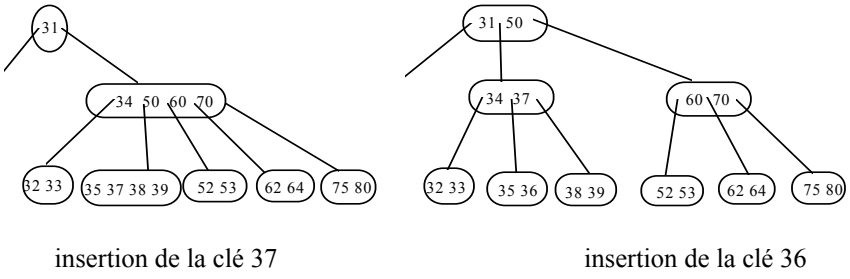

Figure 2. *Insertion des clés 37 puis 36*

Question 6

La suppression de la clé 17 entraîne une sous-occupation de la feuille contenant la clé 17. On choisit ici de fusionner cette feuille avec sa sœur droite (feuille appartenant au même nœud père). La clé médiane 24 redescend dans la feuille résultat de la fusion. La fusion de deux feuilles peut être suivie d'un ré-éclatement si la feuille résultat de la fusion se trouve sur-occupée. Lors de la fusion de 2 nœuds (resp. feuilles), le choix du frère (resp. sœur) avec lequel effectuer la fusion peut être systématique (frère droit ou frère gauche à chaque fois) ou bien dynamique. Un choix dynamique permet de fusionner avec le nœud (resp. feuille) qui produira un résultat de fusion le plus stable possible, c'est-à-dire ni sur-occupé ni sous-occupé. Cette optimisation nécessite cependant la lecture des deux frères (resp. sœurs) pour un gain hypothétique.

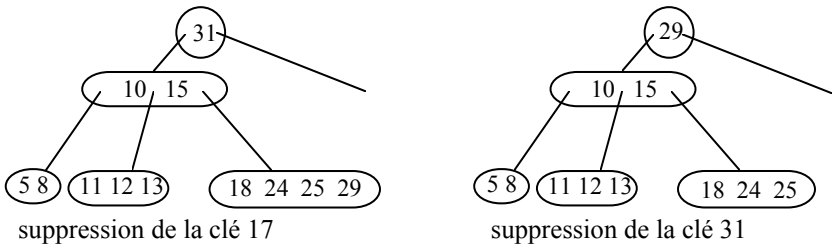

Figure 3. *Suppression des clés 17 puis 31*

La suppression de la clé 31 introduit un problème particulier du fait qu'elle n'appartient pas à une feuille. Pour supprimer une clé dans un nœud non feuille, on la remplace par la plus grande clé du sous-arbre gauche (clé 29 dans l'exemple précédent) ou par la plus petite clé du sous-arbre droit (clé 32 dans l'exemple) dont la clé à supprimer est racine. La clé choisie doit à son tour être supprimée de la feuille en utilisant la procédure de suppression classique.

Question 7

La structure d'arbre-B permet les accès séquentiels triés puisque les clés sont triées dans l'arborescence. Cependant, un parcours séquentiel trié de l'arbre-B nécessite, après lecture de toutes les clés d'une feuille (resp. d'un nœud), de remonter au nœud père pour accéder à la feuille sœur droite (nœud frère droit) afin de continuer la lecture des clés dans l'ordre trié. Ainsi, un nœud père sera accédé sur disque à chaque lecture de l'un de ses fils, afin de récupérer l'adresse de ce fils. Il en résulte qu'une lecture séquentielle triée des 12 nœuds et feuilles de l'arbre-B présenté figure 4.2 nécessite au total 20 E/S.

Une optimisation de la structure d'arbre-B pour les parcours séquentiels triés consiste à dupliquer toutes les clés stockées dans les nœuds non feuille au niveau des feuilles puis à chaîner ces feuilles entre elles dans l'ordre du tri. L'adresse de la première feuille peut de plus être stockée dans le descripteur de fichier. Ainsi, un parcours séquentiel trié ne nécessite que la lecture des feuilles de l'arbre. Ce type d'arbre est appelé arbre-B^+ [Comer79]. L'arbre-B^+ correspondant à l'arbre-B de la figure 4.2 est présenté Figure 6.

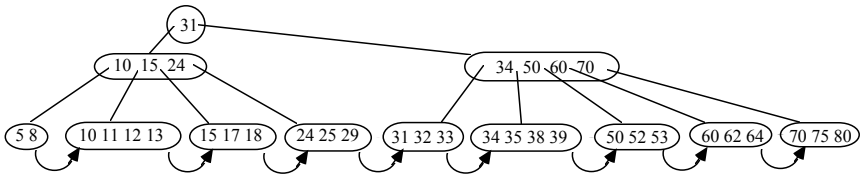

Figure 4. *Arbre-B^+ correspondant à l'arbre-B de la figure 2*

Partie 2 : Organisations aléatoires

Organisations aléatoires statiques

Question 1

La taille d'un paquet de hachage doit être la plus grande possible afin de limiter les débordements en cas de collision. En revanche, un paquet doit pouvoir être lu en une seule E/S. La taille d'un paquet de hachage est donc limitée par la taille d'une page disque (considérée comme l'unité d'E/S). Le format de stockage d'un paquet contenant des articles de taille variable est illustré figure 5.1. Les informations de débordement dépendent de la stratégie employée (bit indicateur de débordement ou numéro du premier paquet de débordement (cf. Question 3).

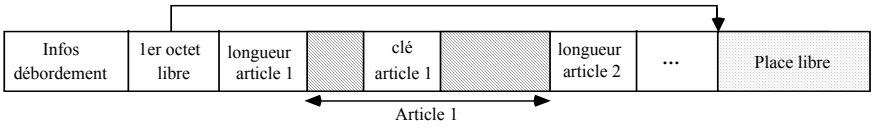

Figure 5. *Format de stockage d'un paquet de hachage*

Question 2

De nombreuses fonctions de hachage sont envisageables pour déterminer un numéro de paquet compris entre 0 et N-1 à partir d'une valeur de clé. Les plus courantes sont :

– *Division entière:* c'est la fonction la plus simple et la plus fréquemment employée. Le résultat de cette fonction, couramment appelée *modulo*, est le reste de la division entière de la clé par le nombre de paquets de hachage N. Il est recommandé de s'assurer que N est un nombre premier afin de limiter les risques de collisions. A titre d'exemple, soit un fichier haché dans 11 paquets, l'article de clé 35 sera placé dans le paquet numéro 2 (2 = 35 modulo 11). Si la clé d'accès n'est pas numérique, la fonction modulo est appliquée à la représentation binaire de cette clé, considérée comme un entier.

– *Extraction du carré:* la clé est tout d'abord élevée au carré puis un ensemble de bits (généralement les bits de poids faibles ou de poids moyens) est extrait du nombre obtenu pour former un numéro de paquet. Etant donné que k bits permettent d'adresser 2^k paquets, on extraira $\log_2 N$ bits pour former un numéro de paquet compris entre 0 et N-1.

– *Pliage généralisé:* si l'on considère qu'un numéro de paquet est exprimé par p bits (avec $p = \log_2 N$), on divise la clé par tranches de p bits et on applique une opération binaire "ou exclusif" entre toutes ces tranches afin d'obtenir une nouvelle chaîne de p bits identifiant un paquet. L'opération "ou exclusif" est préférée à l'opération "et" qui privilégie l'apparition de nombreux 0 dans la chaîne de bits résultat et à l'opération "ou" qui privilégie l'apparition de nombreux 1.

Question 3

Si le paquet désigné par la fonction de hachage est saturé, il faut stocker l'article en insertion dans un des N-1 paquets restants et être capable de le retrouver ultérieurement. Plusieurs techniques de débordement sont envisageables :

– *Adressage ouvert:* supposons que le paquet saturé soit le paquet i, cette technique consiste à rechercher séquentiellement une place libre pour l'article en insertion dans les paquets i+1, i+2, …, i+k, jusqu'à trouver un emplacement de taille suffisante. On mémorise dans le paquet i le numéro du premier paquet dans lequel il a débordé afin d'accélérer les recherches ultérieures des articles en débordement. La recherche de tous les articles appartenant au paquet i s'effectue de la façon suivante. Les articles stockés dans le paquet i sont tout d'abord lus séquentiellement en vérifiant pour chacun d'eux qu'il appartient bien au paquet i (une comparaison de clé permet d'éliminer les articles d'autres paquets stockés éventuellement en débordement dans le paquet i). Les articles du paquet i stockés en débordement sont ensuite recherchés séquentiellement dans les paquets suivants à partir du premier paquet de débordement.

– *Chaînage en débordement:* cette technique est identique à l'adressage ouvert mais tous les articles appartenant à un même paquet sont ici chaînés entre eux. Ce chaînage évite le coût d'une recherche séquentielle pour retrouver tous les articles d'un même paquet stockés en débordement. En revanche, outre le coût de stockage supplémentaire lié au chaînage, cette technique peut engendrer de nombreuses E/S si la chaîne des articles est longue et si le chaînage passe fréquemment d'un paquet à l'autre (on peut alors être amené à lire plusieurs fois le même paquet). Dans le pire des cas, le parcours du chaînage peut générer une E/S pour récupérer chaque article (cf. Chapitre 2, question 4).

– *Table d'adresse des articles en débordement:* cette technique, très similaire à la technique du chaînage en débordement, stocke les adresses de tous les articles en débordement dans une table associée à chaque paquet. Les adresses stockées dans cette table peuvent être maintenues triées afin d'éviter plusieurs lectures d'un même paquet. Cette technique pose cependant le problème de la place occupée et de la taille statique de chacune de ces tables.

– *Re-hachage:* si le paquet i déborde, on mémorise dans le paquet le fait qu'il a débordé et on applique une nouvelle fonction de hachage aux articles devant être insérés dans ce paquet. Cette seconde fonction génère un numéro de paquet différent de i et compris entre 0 et N-1. La recherche de tous les articles appartenant au paquet i s'effectue de la façon suivante. Les articles stockés dans le paquet i sont tout d'abord lus séquentiellement en vérifiant pour chacun d'eux qu'il appartient bien au paquet i (afin d'éliminer les articles d'autres paquets stockés éventuellement en débordement dans le paquet i). les articles du paquet i stockés en débordement sont ensuite recherchés dans le paquet identifié par la seconde fonction de hachage appliquée à la clé d'accès.

Ces techniques offrent une réponse au problème de gestion des articles en débordement mais aucune n'est vraiment satisfaisante si le taux de collisions est élevé. Les performances se dégradent très rapidement lors des recherches d'articles en débordement. Ce phénomène est dû au fait que les articles en débordement d'un paquet i sont stockés en lieu et place des articles d'un paquet j, provoquant rapidement un débordement du paquet j par une réaction en chaîne.

Question 4

Les techniques de débordement présentées question 3 restent applicables, à quelques modifications près, à un fichier possédant une zone de débordement indépendante. Dans le cas de l'*adressage ouvert*, le premier emplacement libre permettant de stocker un article en débordement n'est plus cherché dans les paquets de hachage primaires suivants mais séquentiellement en zone de débordement. Les techniques de *chaînage en débordement* et de *table d'adresses des articles en débordement* restent inchangées mais les liens de chaînage

correspondent à des adresses en zone de débordement. Enfin, la technique de *re-hachage* nécessite de découper la zone de débordement en P paquets de hachage. Ainsi, lors de l'insertion d'un article dans le fichier, on applique tout d'abord une division entière par N à sa clé afin de déterminer un numéro de paquet primaire. Si ce paquet primaire est saturé, on applique une division entière par P à cette même clé afin de déterminer un numéro de paquet secondaire. Si ce paquet secondaire est saturé à son tour, alors on utilise une des techniques présentées question 3 dans la zone de débordement.

L'avantage de séparer la zone primaire de la zone de débordement est d'éviter le phénomène de réaction en chaîne dû à la saturation de certains paquets générée par le débordement d'autres paquets (cf. question 3). Ici, le débordement d'un paquet primaire n'influe pas sur le taux d'occupation des autres paquets primaires. Cette gestion des débordements est donc plus robuste face à un taux élevé de collisions. Cependant, la recherche des articles en zone de débordement reste une opération coûteuse.

Conclusion : l'avantage majeur des organisations aléatoires statiques est leur simplicité de mise en oeuvre et les excellentes performances qu'elles offrent tant que les débordements de paquets sont peu nombreux. En effet, si aucun paquet n'a débordé, les organisations aléatoires statiques garantissent la recherche de tous les articles ayant une valeur de clé donnée en une seule E/S. En revanche, les débordements provoquent rapidement un effondrement des performances quelle que soit la technique de gestion des débordements employée. Ce problème nécessite une réorganisation périodique des fichiers dès que le taux de débordement dépasse un seuil de tolérance donné.

Organisations aléatoires dynamiques

Question 5

L'arbre d'éclatement correspondant au fichier décrit est représenté sur la figure 5.2. Les paquets grisés correspondent aux paquets occupés par le fichier à l'instant t. On peut noter que la chaîne de bits de la signature est développée de droite à gauche. Ceci permet d'exploiter en priorité les bits de poids faibles (dont la distribution est généralement plus aléatoire) des clés de hachage.

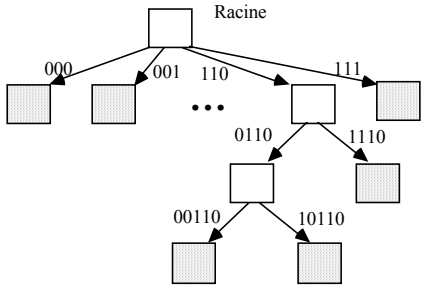

Figure 6. *Arbre d'éclatement valué par les signatures de paquets*

Question 6

Le catalogue est matérialisé par une table assurant la correspondance entre signature et adresse de paquet. Cette table contient une entrée pour chaque paquet occupé par le fichier à un instant t, comme illustré figure 5.3.

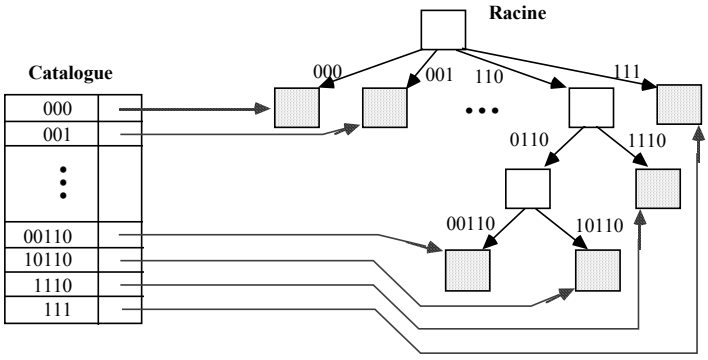

Figure 7. *Catalogue correspondant à l'arbre d'éclatement de la question 6*

Pour effectuer la recherche de tous les articles ayant une clé donnée, il suffit de trouver dans le catalogue l'entrée dont la signature correspond à un préfixe de la chaîne de bits issue du hachage de la clé recherchée. Par exemple, la recherche des articles dont la clé est 54 s'effectue de la façon suivante. Cinq bits au plus étant actuellement exploités dans les signatures de paquet, on applique la fonction de hachage modulo 2^5 à la clé 54. Le résultat du hachage est donc 22 (54 mod 32). En binaire, 22 s'écrit 10110. On recherche donc la signature 10110 dans le catalogue afin de trouver l'adresse du paquet correspondant. Il en va de même pour retrouver les articles de clé 65. 65 modulo 2^5 donne 1, soit en binaire sur 5 bits 00001. L'entrée 001 du catalogue sera sélectionnée car c'est la seule entrée correspondant à un préfixe de 00001.

Question 7

La structure proposée dans la question 7 impose une lecture séquentielle du catalogue. Pour pallier cet inconvénient, on modifie le principe de mise à jour du catalogue. Tant qu'il n'y a pas eu d'éclatement de paquet, le catalogue contient des signatures de p bits adressant les 2^p paquets initiaux. Lors du premier éclatement d'un de ces paquets, la taille du catalogue est doublée et on développe un bit supplémentaire de chaque signature de sorte que toutes les entrées du catalogue contiennent des signatures de p+1 bits. Les deux paquets issus de l'éclatement sont alors référencés chacun par une signature distincte de p+1 bits. Un nouvel éclatement de l'un de ces deux paquets entraînera à nouveau un doublement du catalogue. En revanche, les paquets n'ayant pas éclaté se trouvent référencés par deux signatures de p+1 bits dont les p premiers bits sont identiques. Si l'un de ces paquets éclate à son tour, le doublement du catalogue n'est pas utile puisque l'on dispose déjà de deux signatures de p+1 bits pour adresser les deux paquets issus de l'éclatement. Dans ce cas, seule l'adresse de paquet associée aux deux signatures doit être mise à jour. La figure 5.4 montre l'évolution du catalogue après l'éclatement du paquet 110 puis du paquet 0110. Sur cette figure, *ai* représente l'adresse du paquet *i*.

L'avantage de ce principe est double. D'une part, les doublements du catalogue étant rares, l'éclatement d'un paquet nécessite le plus souvent une simple mise à jour des adresses associées aux signatures. D'autre part, toutes les signatures ont un nombre identique de bits développés. Ainsi, si le catalogue est grand et occupe plusieurs pages, il est possible de déterminer par calcul dans quelle page se trouve la (ou les) signature(s) recherchée(s). Une seule E/S est donc nécessaire pour effectuer la recherche dans le catalogue, quelle que soit sa taille.

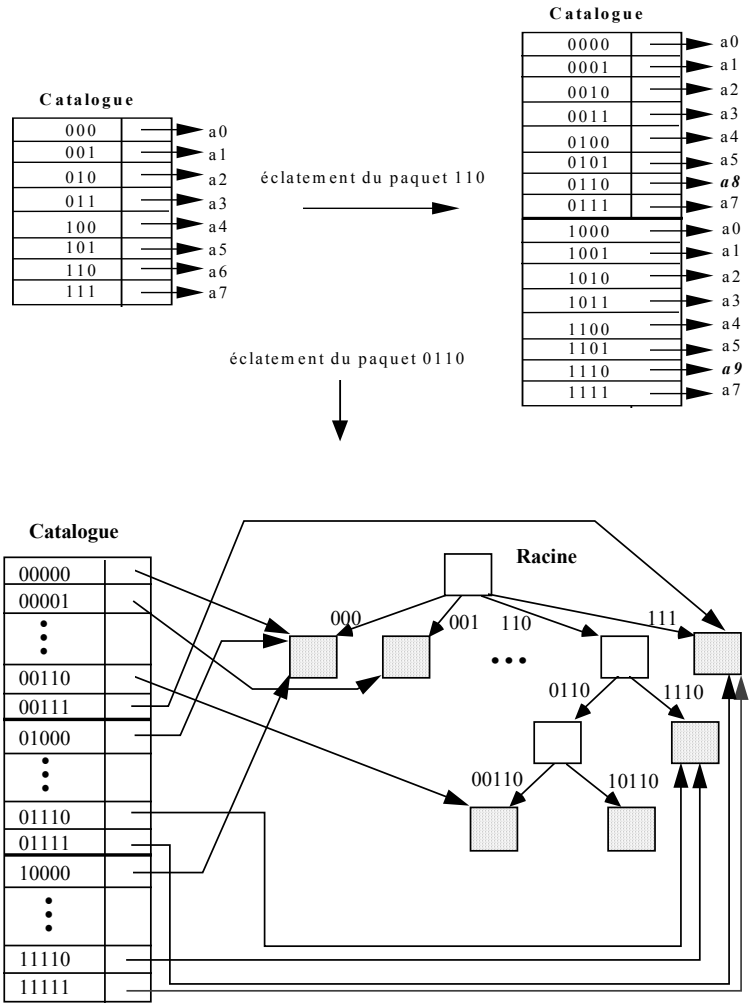

Figure 8. *Etapes de l'évolution du catalogue*

Le paquet initial 000 est désormais référencé par quatre entrées contenant la même adresse physique mais dont les signatures respectives sont: 00000, 01000, 10000, 11000. On remarque que ces quatre signatures ont le même profil xx000.

La recherche des articles de clé 54 s'effectue comme dans la question 7, à savoir que l'on sélectionne l'entrée du catalogue de signature 10110. La recherche des articles de clé 65 revient à sélectionner l'entrée de signature 00001qui, dans le cas présent, référence le même paquet que les entrées de signature : 01001, 10001 et 11001. En effet, le paquet initial 001 n'ayant pas éclaté, il est référencé par toutes les entrées dont le profil de signature est 001.

Remarque: l'entrée du catalogue contenant la signature 10110 est l'entrée 22 (10110 en binaire). Par conséquent, il n'est pas utile de stocker les signatures dans ce type de catalogue car cette information est redondante avec le numéro de chaque entrée.

Question 8

Lorsque la signature d'un paquet atteint le nombre de bits total de la chaîne de bits issue du hachage de la clé, ce paquet ne peut plus être éclaté puisque tous les bits des clés des articles qu'il contient ont été exploités. Le fichier est dans ce cas saturé. Il est cependant possible de chaîner un paquet de débordement à chaque paquet saturé afin de permettre leur extension. Dans ce cas, le fichier n'a plus de limite théorique de taille.

Question 9

Afin de déterminer le numéro du paquet dans lequel doit être recherché ou inséré un article, il faut connaître le niveau k du fichier et savoir si le paquet en question a déjà été éclaté ou non, ceci afin de déterminer si l'on doit utiliser la fonction de hachage h_k ou h_{k+1}. Pour cela, on mémorise dans le descripteur de fichier : le nombre de fois où le fichier a doublé de taille (niveau k du fichier) et la position du pointeur courant d'éclatement référençant le prochain paquet à éclater. Par connaissance du niveau k du fichier, on applique la fonction de hachage h_k à la clé de l'article à rechercher ou à insérer. Si le numéro de paquet déterminé par cette fonction de hachage est inférieur au pointeur courant d'éclatement, alors le paquet correspondant a déjà été éclaté et l'on re-hache la clé de l'article avec la fonction h_{k+1}. Sinon, le paquet désigné par la fonction h_k peut être directement exploité.

Question 10

L'algorithme déterminant la fonction de hachage à utiliser, lors de la recherche ou de l'insertion d'un article dans le fichier, prend en entrée le descripteur du fichier, la clé de l'article et rend en sortie le numéro de paquet dans lequel devrait être recherché ou inséré cet article. La fonction de hachage utilisée dans cet algorithme est h (k, clé) = clé modulo (2^k N), où le paramètre k correspond au niveau du fichier.

```
fonction Calcul_n°paquet (descrip_fich: descripteur, clé:type_clé): entier;
var   numpaquet, k: entier;
début
      k := descrip_fich.k;          // niveau du fichier
      numpaquet := h (k, clé);
      si (numpaquet < descrip_fich.ptéclat) alors Calcul_n°paquet := h (k+1, clé);
            sinon Calcul_n°paquet := numpaquet;
fin
```

L'algorithme d'insertion d'un nouvel article dans un fichier organisé selon la méthode du hachage linéaire est présenté ci-dessous. La fonction Calcul_n°paquet utilisée dans cet algorithme est celle présentée question 10. La fonction Ecrire(article, numpaquet) écrit l'article "article" dans le paquet numpaquet ou le chaîne en débordement si ce paquet est saturé.

```
procédure Insérer(descrip_fich: descripteur, clé: type_clé, article: type_art)
var
      numpaquet, nvnumpaquet: entier;
      k: entier;           // niveau du fichier
      ptéclat: entier;     // pointeur courant d'éclatement

début
      k := descrip_fich.k;
      ptéclat := descrip_fich.ptéclat; numpaquet := Calcul_n°paquet (descrip_fich, clé);
      si (numpaquet est saturé) alors
            // éclater le paquet de numéro ptéclat dans les
            // paquets ptéclat et (ptéclat + 2^k N)
            pour chaque article_i ∈ ptéclat faire
                  nvnumpaquet := h (k+1, article_i.clé);
                  Ecrire (article, nvnumpaquet);
            fpour
            si (numpaquet = ptéclat) alors
                  // déterminer si le nouvel article doit être inséré
                  // dans le paquet ptéclat ou (ptéclat + 2^k N)
                  numpaquet := h (k+1, article.clé);
            fsi
            // incrémenter le pointeur courant d'éclatement
            si (ptéclat = (2^k N) - 1)
                  alors // m-à-j du  niveau du fichier
                        k := k + 1;
                        ptéclat := 0;
```

```
                    sinon ptéclat := ptéclat + 1;
                            // m-à-j du descripteur de fichier
                            descrip_fich.k := k;
                            descrip_fich.ptéclat := ptéclat;
                fsi
        fsi
        // écrire l'article dans le paquet numpaquet
        Ecrire (article, numpaquet);
Fin
```

Partie 3 : Accès multi-critère

Question 1

Les algorithmes de gestion des arbre-B^+ introduits dans le chapitre 2 font l'hypothèse d'un tri mono-critère (c-à-d, prenant en compte un seul attribut). Afin de prendre en compte plusieurs attributs dans la méthode de placement, il suffit d'adapter la fonction *Rech_dichoto* (cf. chapitre 2, question 11) à un tri multi-critères. La clé à utiliser dans le tri est la concaténation des attributs A1, A2, …, An. La comparaison de 2 clés s'effectuent d'abord sur l'attribut A1. En cas d'égalité, on prend en compte la valeur de l'attribut A2 et ainsi de suite. Une clé c1 est donc considérée comme supérieure à une clé c2 si ($\exists$i / c1.Ai > c2.Ai) et (i=1 ou $\forall$j<i c1.Ai = c2.Ai).

Cette méthode ne traite pas les différents attributs de façon équitable. Ainsi, si le critère de tri choisi est A1, A2, …, An, il est possible d'accélérer les recherches en fonction d'une clé complète ou en fonction d'une clé partielle à condition que cette clé partielle préserve les premiers attributs de la clé complète. Ainsi, une recherche de tous les articles tels que (A1=valeur1 et A2= valeur2) est possible. Par contre, une recherche de tous les articles tels que (A2=valeur1 et A3= valeur2) sans préciser de valeur pour A1 ne peut pas être accélérée par l'arbre-B^+.

Question 2

a) Les N paquets d'un fichier haché selon une méthode statique multi-attributs sont identifiés par des chaînes de p bits (où $p=\log_2 N$) construites par concaténation de b1, b2, …, bn bits (où b1+b2+ …+bn = p) associés respectivement aux attributs A1, A2, …, An. La recherche de tous les articles du fichier satisfaisant la condition Ai = <valeur> nécessite la lecture de tous les paquets dont les numéros sont identifiés par une chaîne de p bits dans laquelle les valeurs des bi bits associés à Ai correspondent aux valeurs des bi bits issus du hachage de la valeur recherchée par la fonction hi. On sélectionne donc tous les paquets dont les numéros ont la forme présentée en figure 6.1. Ces paquets sont ensuite balayés séquentiellement pour récupérer tous les articles satisfaisant le critère de recherche.

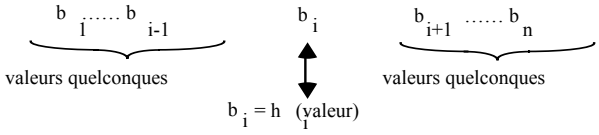

Figure 9. *Signature des paquets sélectionnés*

b) S'il n'y a pas eu de débordement, le nombre d'E/S engendrées par une telle recherche correspond au nombre de paquets dont les numéros ont la forme citée précédemment. Puisque le fichier contient 2^p paquets (2^p = nombre de combinaisons possibles de p bits), le nombre de paquets sélectionnés (nombre d'E/S) est égal à 2^{p-bi}, c'est-à-dire au nombre de combinaisons possibles de p-bi bits puisque bi bits sont fixés par le critère de recherche.

c) En suivant le même raisonnement, le nombre d'E/S générées par une recherche avec la condition (Ai = <valeur1> et Aj= <valeur2>) est égal à $2^{p-bi-bj}$. Ce coût est donc inférieur au coût calculé en b). Le nombre d'E/S générées par une recherche avec la condition (Ai = <valeur1> ou Aj= <valeur2>) est quant à lui égal à $2^{p-bi} + 2^{p-bj}$.

Question 3

De façon identique au hachage statique multi-attributs, la recherche de tous les articles du fichier satisfaisant la condition Ai = <valeur> nécessite la lecture de tous les paquets dont les numéros sont identifiés par une chaîne de bits dans laquelle les valeurs des bi bits associés à Ai correspondent aux valeurs des bi bits issus du hachage de la valeur recherchée par la fonction hi. La différence est que les paquets sont ici créés dynamiquement et que les signatures de paquets sont développées au fur et à mesure des éclatements. On commence donc par construire une chaîne de bits, appelée profil de signature, dans laquelle seules sont renseignées les valeurs des bi bits associés à Ai. Ce profil de signature est ensuite utilisé pour filtrer le catalogue afin de sélectionner toutes les signatures dont les valeurs des bi bits de Ai correspondent à celles du profil de signature, indépendamment des valeurs des autres bits de la signature. Si certains bits du profil de signature correspondent à des bits non encore développés dans une signature, on applique la comparaison uniquement sur les bits développés.

Le hachage de la valeur recherchée 12 par hi donne 3, soit 11 en binaire. Le filtrage du catalogue va donc s'effectuer avec le profil ??1?1 (où ? symbolise une valeur de bit inconnue). Ce filtrage va permettre de sélectionner les 2^{5-2} entrées du catalogue contenant les signatures 00101, 00111, 10101, 10111, 01101, 01111, 11101 et 11111. Le nombre d'E/S résultant est simplement 2 car l'ensemble de ces signatures référencent deux paquets initiaux 101 et 111 qui n'ont jamais éclaté.

Question 4

Dans une méthode de placement statique, l'ordonnancement des bits dans la signature d'un paquet n'a pas d'importance. En revanche, dans une méthode de placement dynamique, l'éclatement des paquets se fait en exploitant les bits de la signature l'un après l'autre. Seuls les bits exploités de la signature participent au placement, les autres bits restant non significatifs lors des recherches. Privilégier l'accès à un attribut revient donc à placer les bits qui lui sont associés en début de signature (c'est à dire à droite) de telle sorte qu'ils deviennent significatifs dès les premiers éclatements.

Soient les fonctions de hachage h1(A1), h2(A2), h3(A3) suivantes:

$h1(A1) = i^1\ i^2\ i^3 \ldots i^n$ où i^q correspond au $q^{ème}$ bit de h1(A1)

$h2(A2) = j^1\ j^2\ j^3 \ldots j^n$ où j^q correspond au $q^{ème}$ bit de h2(A2)

$h3(A3) = k^1\ k^2 k^3 \ldots k^n$ où k^q correspond au $q^{ème}$ bit de h3(A3)

Privilégier les accès à l'attribut A2, puis à l'attribut A3 puis enfin à l'attribut A1 dans cet ordre de priorité revient à placer les bits issus des fonctions de hachage h1(A1), h2(A2) et h3(A3) dans l'ordre suivant: $i^1\ i^2\ i^3 \ldots i^n\ k^1\ k^2 k^3 \ldots k^n\ j^1\ j^2\ j^3 \ldots j^n$. Dans cet exemple, le placement sur l'attribut A1 ne deviendra effectif que lorsque le fichier aura atteint une grande taille, après un nombre important d'éclatements de paquet.

Question 6

Afin de traiter de façon équitable chacun des attributs de placement, la méthode du grid–file mélange les bits issus de chaque fonction de hachage dans la chaîne de bits afin d'exploiter un bit d'un attribut différent à chaque éclatement d'un paquet. Dans l'exemple du fichier haché sur les attributs A1, A2 et A3, la méthode du grid–file va exploiter un bit de h1(A1) lors du premier éclatement d'un paquet, puis un bit de h2(A2) au deuxième éclatement de ce paquet, puis un bit de h3(A3) au troisième éclatement puis à nouveau un bit de h1(A1) au quatrième éclatement et ainsi de suite en tournant cycliquement sur chacun des attributs.

Cette équité est donc obtenue en plaçant les bits issus des fonctions de hachage h1(A1), h2(A2) et h3(A3) dans l'ordre suivant: $k^1 j^1 i^1\ k^2\ j^2\ i^2 \cdots k^n\ j^n\ i^n$, où i^q, j^q, k^q ont même signification que dans la question 5.