Optimizing Jan Jannink's Implementation of B^+-tree Deletion

R. Maelbrancke and H. Olivié
Katholieke Universiteit Leuven, Department of Computer Science
Celestijnenlaan 200A, B-3001 Heverlee, Belgium
E-mail:{rudim,olivie}@cs.kuleuven.ac.be

Abstract

In this note we propose optimization strategies for the B^+-tree deletion algorithm. The optimizations are focused on even order B^+-trees and on the reduction of the number of block accesses.

Keywords: B-trees; design of algorithms; data structures.

1 Introduction

In [Jan95] Jan Jannink proposes a B^+−tree deletion algorithm implementation. This implementation is a reaction to the lack of real implementation schemes in literature. Jannink's deletion algorithm is based on Wirth's deletion algorithm for B-trees [Wir76].

However, the algorithm described in [Jan95] does not always preserve all B^+−tree properties proposed by Knuth [Knu73] and causes an excess of block accesses, so we propose optimization strategies for the algorithm.

2 About B^+−tree Properties

In the last two decades a large number of publications covering data structures have appeared [Knu73, AHU87, Wir76, Com79, SB87, Mil87, EN94, Woo93]. Although most of these cover B^+−trees, only a few of them contain a complete definition.

A B^+−tree is a leaf tree version of a B-tree and the internal nodes satisfy the B-tree properties. In all definitions, the root should contain at least one key. For the B-tree two slightly different definitions are used frequently. These two definitions differ in the way the maximum number of children of a node, also denoted by *order p of a B-tree*, is defined. Wirth [Wir76], Comer [Com79] and Aho et al. [AHU87] restrict p to odd values. Although others do not restrict p, the even case is rarely discussed. As we point out further on, an odd p guarantees a 50% filling degree, while an even p does not. We define filling degree as

$$\frac{\text{number of keys in a node}}{\text{maximum number of keys in a node}}$$

For the external nodes, which are specific to the B^+−tree, the definitions differ more. They all require the location of the leaf nodes to be at equal distance from the root. Aho et al. [AHU87], Wood [Woo93], Comer [Com79] and Smith & Barnes [SB87] do not put a lower bound on the number of data items in the leaf nodes of a B^+−tree. This lack of lower bound on the number of keys in the leaf nodes may result in an underpopulation of the nodes. Miller [Mil87] even gives different upper bounds for the leaf nodes than the internal nodes, but again lower bounds are not discussed. Jannink [Jan95] has chosen to put the same lower bound on the leaf nodes as defined for the internal nodes. We use a stricter lower bound based on Knuth [Knu73], which guarantees at least a 50% filling degree of the leaf nodes for all values of p.

Knuth was the first to mention this variation of B-trees. Although he does not give a definition of B^+−trees, he writes [Knu73](p. 477), "*... Under this interpretation the leaf nodes grow and split just as the branch nodes do, except that a record is never passed up from a leaf to the next level. Thus the leaves are always* **at least half filled to capacity**". By these sentences he means that the lower bound on the number of keys in a leaf node with maximum $p-1$ keys is $\lceil \frac{p-1}{2} \rceil$ while for an internal node this is $\lfloor \frac{p-1}{2} \rfloor$. If p is odd, internal nodes have the same lower bound as leaf nodes. However, if p is even, the lower bound on the number of keys in the leaf nodes is one higher than for the internal nodes.

We use the following definition of a B^+−tree which is based on Elmasri and Navathe [EN94]:

Definition 1 Let p denote the maximum number of children of an internal node, P_i a tree pointer, K_i a key and Pr_i a data pointer.

1. The structure of an internal node of a B^+−tree of order p is as follows:

(a) Each internal node is of the form

$$< P_1, K_1, P_2, K_2, ..., P_{q-1}, K_{q-1}, P_q >$$

where $q \leq p$.

(b) An internal node with q tree pointers, $q \leq p$, has $q-1$ keys.

(c) Within each internal node, $K_1 < K_2 < ... < K_{q-1}$.

(d) For all keys X in P_i's subtree, we have $K_{i-1} < X \leq K_i$ for $1 < i < q$, $X \leq K_1$ for $i = 1$ and $K_{q-1} < X$ for $i = q$.

(e) Each internal node, except the root, has at least $\lfloor \frac{p-1}{2} \rfloor$ keys. The root has at least 1 key.

2. The structure of a leaf node of a B$^+$−tree of order p is as follows:

(a) Each leaf node is of the form

$$<< K_1, Pr_1 >, < K_2, Pr_2 >, ...$$

$$, < K_{q-1}, Pr_{q-1} >, P_{next} >$$

where $q \leq p$ and P_{next} points to the next leaf node of the B$^+$−tree.

(b) A leaf node with $q-1$ data pointers, $q \leq p$, has $q-1$ keys.

(c) Within each leaf node, $K_1 < K_2 < ... < K_{q-1}$.

(d) Each leaf node in a tree with at least 1 internal node, has at least $\lceil \frac{p-1}{2} \rceil$ keys.

3. All leaf nodes are at the same level and together they contain all the stored keys.

Observe the two parts in a B$^+$−tree, (1) the set of internal nodes which represents an *ordinary* B−tree (Def. 1, part 1) and (2) a linked list of leaf nodes (Def. 1, part 2), and that the minimal number of keys in the leaf nodes differs from the minimal number in the internal nodes (Def. 1, part 1e & part 2d) when p is even.

If p is odd all nodes are at least half filled. However, if p is even, the lower bound on the number of keys in the internal nodes is $\frac{p-2}{2}$ and $\frac{p}{2}$ in the leaf nodes. This means that the filling degree fd_i of the internal nodes is:

$$fd_i \geq \frac{\frac{p-2}{2}}{p-1} = \frac{1}{2} - \frac{1}{2(p-1)}$$

The filling degree fd_l of the leaf nodes is:

$$fd_l \geq \frac{\frac{p}{2}}{p-1} = \frac{1}{2} + \frac{1}{2(p-1)}$$

Noteworthily, in a B$^+$−tree of order 4, a 2-4$^+$-tree, the leaf nodes have a guaranteed $\frac{2}{3}$ filling degree.

Since the number of leaf nodes at least equals the number of internal nodes, the average filling degree of the whole tree fd_t in the worst case is:

$$fd_t \geq fd_i + fd_l = \frac{1}{2}$$

If p is even the leaves in Jannink's approach may contain one item less than allowed in definition 1. Thus $fd_i = fd_l$, which may result in a filling degree that is less than 50% .

3 Deletion Policies

Jannink's deletion algorithm [Jan95] is intended to be as general as possible. Because deletion policies for B$^+$−trees mostly depend on the environment the implementation is used in, we propose optimizations for the algorithm. These optimizations are mainly focused on the way underflowing nodes are handled.

While we typically consider siblings, that is, neighbour nodes with the same parent, Jannink considers both neighbours as actors in the restructuring process. If one of the neighbours has more keys than the lower bound requires, keys are shifted, otherwise the underflowing node is merged with a sibling. This way of underflow handling reflects the generality of Jannink's algorithm.

If the neighbours of an underflowing node are siblings, they can be reached easily. However, in some cases a neighbour can be reached only by traversing a different search path. The worst case occurs when the discriminator between the two neighbours is the root (figure 1 (a)). In this case an extra traversal of a path from the root to a leaf is needed. To accelerate the access to a neighbour, Jannink uses a top-down strategy. For every node he traverses towards a leaf, the neighbours can be accessed immediately. This means that during each deletion action, extra nodes should be visited to determine the neighbours. In the worst case Jannink's algorithm traverses almost 3 times a path from the root to a leaf (figure 1 (b)). Since the maximum height $h_{\max}$ of a B$^+$−tree with n keys is

$$h_{\max} \leq 1 + \log_{\frac{p}{2}} \left(\left\lceil \frac{n}{\frac{p}{2}} \right\rceil \right) \approx \log_{\frac{p}{2}}(n)$$

this results in $1 + 3\,(h_{\max} - 1)$ node visits. The window in figure 1 (b) shows the nodes visited.

Depending on the environment, we propose following optimization strategies.

A general optimization strategy considers local actions ahead of non-local actions. It adapts the restructuring using a non-sibling only if multiple node deletions can be avoided. This optimization was introduced by Jannink as a result of the tuning of his implementation.

Additionally, in a disk based environment, the top down calculation of neighbour node addresses should be exchanged for a calculation on request. In this way extra block accesses are only caused by an underflow.

We propose that in a disk based concurrent environment, restructuring should be done as locally as possible. If we change Jannink's algorithm in such a way that one sibling is visited if a node underflows, then node locking in the search path back to the root concurrently affects a parent and two children. If the visited sibling contains the minimal number of keys, the underflowing node is merged with it in such a way that the right node is deleted, otherwise keys are shifted towards the node that underflows.

In the worst case $1 + 2\,(h_{\max} - 1)$ node visits are needed to accomplish the task. The window in figure 1 (c) highlights the visited nodes. This means that even then the number of node visits is reduced by 30%. In the best case the restructuring causes one node visit extra against two in the general implementation.

Observe that this optimization may cause more node deletions, but that the search path is shortened.

In general this minimal-reads optimization should be weighed against the benefit of a reduced number of node deletions.

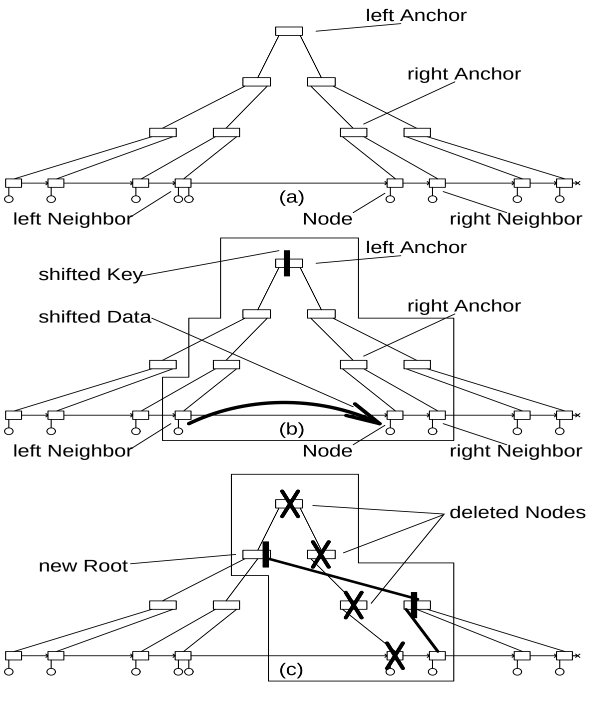

Figure 1: A worst case deletion in a $2-3^{+}-tree$ (a), Jannink's algorithm (b), our algorithm (c)

4 Conclusion

Jannink's deletion algorithm can be optimized easily to handle even order $\mathrm{B}^{+}-$trees and to reduce the block accesses significantly by

- setting the lower bound on the number of keys in leaf nodes to $\lceil \frac{p-1}{2} \rceil$ for order-p $\mathrm{B}^{+}-$trees, and
- selecting only **one sibling** for shifting or merging in case of an underflow.

$\mathrm{B}^{+}-$tree algorithms including the first optimization are available by anonymous ftp from db.stanford.edu in directory ftp/pub/jannink/btree/, or over the web at http://www-db.stanford.edu:80/pub/jannink/btree/.

Acknowledgements The authors are very grateful to Jan Jannink for his helpful comments and the diagram (fig. 1).

References

[AHU87] A. V. Aho, J. E. Hopcroft, and J. D. Ullman. *Data Structures and Algorithms.* Addison-Wesley Publishing Company, 1987.

[Com79] D. Comer. The ubiquitous B-tree. *ACM Computing Surveys*, 11(2):121–137, 1979.

[EN94] R. Elmasri and S. B. Navathe. *Fundamentals of Database Systems.* Benjamin/Cummings Publishing Company, Inc., 1994.

[Jan95] J. Jannink. Implementing deletion in $\mathrm{B}^{+}-$trees. *ACM SIGMOD Record*, 24(1):33–38, 1995.

[Knu73] D. E. Knuth. *The Art of Computer Programming 3: Sorting and Searching.* Addison-Wesley Publishing Company, 1973.

[Mil87] N. E. Miller. *File Structures Using Pascal.* Benjamin/Cummings Publishing Company, Inc., 1987.

[SB87] P. D. Smith and G. M. Barnes. *Files & Databases: an introduction.* Addison-Wesley Publishing Company, 1987.

[Wir76] N. Wirth. *Algorithms + Data Structures = Programs.* Prentice-Hall, Englewood Cliffs, New York, 1976.

[Woo93] D. Wood. *Data Structures, Algorithms and Performance.* Addison-Wesley Publishing Company, 1993.