MTA SZTAKI Közlemények 35/1986 p. 79-87

INDEX TREATMENT OF A DBMS FOR A MINICOMPUTER

Lic. Maria Elena Bragado BRETANA
Lic. Miguel Fonfria ATAN

Institute Central de Investigacion Digital
Habana, Cuba

Introduction

In this paper it is shown the method implemented for index treatment in the Data Base Management System dBASE-300, for the cuban minicomputer CID-300/10 (like the PDP-11/05).

The technique employed to treat indexes in dBASE-300 is the organisation called B+ tree. With the use of this technique it is possible to accomplish all transactions defined in dBASE-300 in a suitable way as it is efficient for both random and sequential access to records and modification operations.

In order to have a better understanding of the implementation of B+ tree in dBASE-300 File Control System, the analysis is divided into the following sections:

- B-trees and their data structures.
- Find, Insertion and Deletion.
- Operation costs.
- B+ trees.
- Other variants of B-trees.

The first two sections are referred to B-trees because the characteristics explained are also present in B+ trees.

B-trees and their data structures.

The structure implemented for index treatment in dBASE-300 corresponds to the B-tree definition given in [Bay], i.e., it has the following properties:

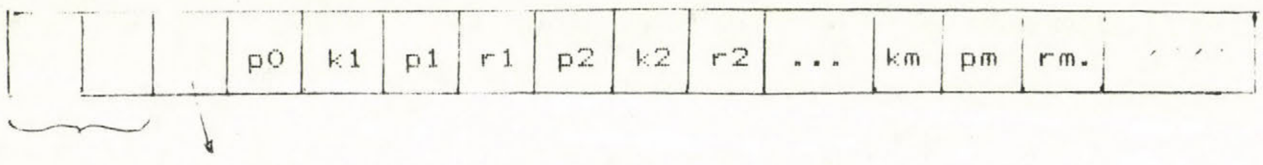
a) Each path from the root to any leaf has the same length.

b) Each node, except the root and leaves, has at least k+1 sons, where k is a natural number. The root is a leaf and has at at least two sons.

c) Each node has at most 2k+1 sons.

A node of a B-tree is the page in which the index is stored.

A page of the B-tree in dBASE-300 looks like this:



```
next &    number                  pi - pointer
previous  of keys in              ki - key
page      the node                ri - record number
```

This data structure of a page has the following properties [Bay]:

a) Each page contains between k and 2k keys except the root page which may contain between 1 and 2k keys.

b) Let m be the number of keys on a page (not a leaf). P has m+1 sons.

c) The keys in a page are sequential in increasing order and each page contains at most 2k keys and 2k+1 pointers.

d) Let P(pi) be the page to which pi points, let K(pi) be the set of keys on the pages of that maximal subtree of which P(pi) is the root. Then the following conditions always hold:

$\forall\ y \in K(p0) \Rightarrow y<k1$

$\forall\ y \in K(pi) \Rightarrow ki<y<ki+1 \quad ; \quad 1\leq i<m$

$\forall\ y \in K(pm) \Rightarrow km<y$

The record number behind each key in the page enables to access directly the requested record in the relative file.

This information in the page is a peculiar characteristic of this implementation, taking advantage of the relative organisation of the data file.

Find, Insertion and Deletion

Find.

A find operation in a B-tree of order k never visits more than $1+\log_k n$ pages, where n is the number of records in the file.

This is possible because record operations in a B-tree always leave the tree balanced.

The B-tree balancing scheme restricts changes in the tree to a single path from a leaf to the root, so it can not introduce "runaway" overhead [Com].

The find algorithm is simple logically.

Insertion

The insertion operation requires a previous find operation with which it is possible to get the page where the new key is going to be inserted.

Three cases may be found (assuming the new key is not present):

i) empty tree

ii) page not full

iii) page full

In the first case it is necessary to create a root page with the new key.
In the second case the new key may be inserted in the correct position.

In the third case it is necessary to split the page, that is, the smallest k keys are placed in one page, the largest k keys are placed in another page, and the remaining value is promoted to the parent page where it serves as a separator. In the worst case splitting propagates all the way to the root and the tree increases in height by one level.

Deletion

Like insertion, the deletion operation needs a previous find operation to locate the key to be deleted.
Assuming the key is present, it is possible to find two cases:
a) the key is on leaf page
b) the key is not on a leaf page.
In the first case the key can be deleted from leaf.
In the second case it is necessary to find the adjacent key and place it in the position of the deleted key. To find the adjacent key in key-sequence order it is necessary to search for the leftmost leaf in the right subtree of the deleted key.

In both cases it is necessary to check whether an underflow condition is present, that is, if the leaf has less than k keys. In this case it is possible to redistribute the remaining keys between the two neighboring pages but only if there are at least 2k keys to distribute. If there are less than 2k keys it is necessary to perform a concatenation process, where keys are combined into one page and the other is discarded. Then the separating key in the ancestor is no longer necessary and is also added to the single remaining leaf.

As can be seen, the process of concatenation may force concatenating at the next higher level and so on, to the root of the level, and it is possible that B-tree decreases in height by 1.

Operation costs

The cost of a find operation is increased with the growth of the file size logarithm. This is:

$$h \leqslant \log_k \frac{n+1}{2} \qquad \text{[Bay] [Com]}$$

where : h - height of the page tree

n - number of keys

k - minimum number of keys per page

Insertion and deletion take time proportional to $\log_k n$ in the worst case because these operations need aditional access beyond the cost of a find operation as it progresses back up the tree. The costs are at most double, so that the height of the tree still is the main element for these costs.

As can be seen, the number of keys in a page is the parameter on which the performance of all operations depends.

Bayer and McCreight [Bay] show a way to determine the optimal number of keys in a page to achieve minimum time per transaction, in terms of fixed time spent per page, transfer time, key size, and a factor for average page accupancy.

In the case of dBASE-300 there are practical limits to page size. The disks on which the system is supported are divided into fixed blocks of 512 bytes length.

Thus, each page in dBASE-300 is 512 bytes in order to avoid extra overhead in the process.

In practice, considering standard key size the behaviour of the algorithm with this size page was quite suitable.

B+ trees

The organisation chosen was B+ tree, a variant that avoids the problems with sequential processing in B-trees.

These problems are the requirement of extra space for at least $\log_k(n+1)$ pages [Knu] in main memory to avoid reading them twice in the sequential processing.

Also, the next operation may require to access several pages before finding the desired key.

In a B+ tree all keys reside in the leaves.

The relative organisation of a file in the dBASE-300 enables implementing B+ trees in a very comfortable way, because this file can be accessed directly once we have the number of correspondent record in the sequence order.

With B+ tree the logarithmic cost properties for operations by key are retained and the next operation for a sequential processing requires at most 1 access. Besides, no page will be accessed more than once.

On the other hand, the feature of B+ trees that all keys reside in the leaves allows to simplify the deletion operation because the removal of the key to be deleted is simple, and the tree need not be changed while the leaf remains al least half-full. It means that a copy of a deleted key will be present in the tree and can direct searches to the correct leaf.

Redistribution or concatenation process will be necessary only if an underflow condition arises in the leaf and this process is similar to that in B-tree.

Other variants of B-trees

Several variants of B-trees were analyzed in order to choose a technique for the implementation of index treatment in dBASE-300. These variants were: B* tree [Knu], Virtual B-trees [Com], Compression technique [Wagn], and Binary B-trees [Bay1].

B* tree was discarded because its implementation seems to be a little more complicated and then it requires more main memory with the resulting increase of overhead.

In the case of Virtual B-trees there is a practical inconvenience because our system computer does not have facilities of virtual memory.

With the compression technique pointers can be compressed using a displacement from a page address instead of the absolute address value. This technique is particulary useful for virtual B-trees where pointers take on large address values.

Finally, Binary B-trees are appropriated for a one-level store, bacause a Binary B-tree is a B-tree of order 1. This means that each page has 1 or 2 keys and 2 or 3 pointers.

Conclusions

B+ trees retains the logarithmic cost properties for operations by key and has no problems with the next operation for a sequential processing.

In practice, B+ tree proved a very suitable organisation for index ,treatment in dBASE-300.

References

[Bay] Bayer, R. and Mc Creight. "Organization and Maintenance of Large Ordered Indexes". Acta Informatica 1,3(1972) 173-189.

[Bay1] Bayer, R. "Binary B-trees for Virtual Memory". Proc. 1971 ACM SIGFIDET Workshop, ACM New York, (219-235).

[Bel] Bell, D.A. and S.M. Deen. "Hash trees versus B-trees". The Computer Journal Vol.27, No. 3 1984.

[Bla] Black, J.P. et al. "A robust B-tree implementation". Proceedings of the Fifth International Conference on Software Engineering. pp 63-70. (9-12 March 1981).

[Com] Commer, D. "The ubiquitous B-tree". Computer Surveys, Vol. 11, No. 2, June 1979.

[Knu] Knuth, D. "The Art of Computer Programming", Vol. 3.

[Wag] Wagner, R. "Indexing designs considerations". IBM Syst. J.4 (1973) 351-367

Miniszámitógépek számára irt DBMS index-kezelése

M.E. Bragado Bretana, M.F. Atan

Összefoglaló

A cikkben a kubai CID-300/10 /∿ PDP-11/05/ mini-számitógép számára irt dBASE-300 adat-kezelő rendszer index-kezelését ismertetik. Az index-kezelési technika "B+tree"-nek nevezett szervezésen alapszik.

Трактовка индексов в DBMS для мини-компьютеров

М.Е. Брагадо Бретана, М.Ф. Атан

Р е з ю м е

В статье речь идет о трактовке индексов в имплементации пакета обработки данных dBSE-300 для кубинского мини-компьютера CID-300/10 /~ PDP-11/05/. Техника трактовки индексов основана на организации названной "B+tree".