Theoretical Analysis of Matching Times in A Balanced Tree

1 Introduction

In our previous work, we use the B+ tree as in-memory index structure to manage massive secret keys in Hadoop Distributed File System (HDFS). Within a system of more querying than modifying, the efficiency largely depends on the query efficiency of B+ tree, i.e. the matching times between the target key name and names in nodes. In the previous paper, we propose the theoretical analysis of matching times during query keys, but the verification process was omitted due to page limitation. Here we present the complete analysis.

The remaining chapters are arranged as follows. The definitions and notations are introduced in Section 2. In Section 3, the matching times of the most dominant part of searching, i.e. the binary search, is discussed. Section 4 gives the theorems matching times calculation, and the corresponding detailed proofs are attached. In Section 5, the correctness of the theorems and corollary are demonstrated by the experiments on a simulator.

2 Preliminaries

This section will introduce some basic properties of B+ tree, further, some lemmas needed in the later proof are included.

2.1 Definition of B+ Tree

A B+ tree[1] is an *n*-ary tree with a variable but often large number of children per node. A B+ tree[1] consists of a root, internal nodes and leaves. The root may be either a leaf or a node with two or more children. A B+ tree can be viewed as a variant of B-tree in which each node contains only keys (not key-value pairs), and to which an additional level is added at the bottom with linked leaves. The detailed limitation of B+ tree is shown as follow.

Definition 1 [2]. The nodes in B+ tree contain several names, the number of names have upper and lower bound. Supposing one node has at most *n* names, where it needs to satisfy following condition.

(i) Key names in root node ranges from 1 to *n*.

(ii) Key names in leaf node ranges from $\left\lfloor \frac{n+1}{2} \right\rfloor$ to n.

(iii) Key names in internal node ranges from $\left\lfloor \frac{n}{2} \right\rfloor$ to n.

For a *k*-keys node, it has *k*+1 children nodes. During the insertion and removal of keys, root node and internal nodes will merge or split to satisfy the balance of tree.

2.2 Notations

To clearly describe the subsequent analysis, we give the notations as shown in Table 1.

Table 1 Notations

Symbols	Description
n	The maximum key number in every nodes
h	Height of the B+ tree
N	Total number of key in B+ tree

[1] https://en.wikipedia.org/wiki/B%2B_tree

m	The least key number of leaf nodes, $m = \left\lfloor \frac{n+1}{2} \right\rfloor$
l	The least key number of internal nodes, $l = \left\lfloor \frac{n}{2} \right\rfloor$
r	The least children number of internal nodes, $r = \left\lfloor \frac{n}{2} \right\rfloor + 1 = l + 1$
q	The most children number of internal nodes, $q = n + 1$
$s(n)$ or $t_1(n)$	In a *n*-node binary tree, the searching number of a target key.
$S(n)$	The total comparing time of searching *n* nodes in binary tree
$t(n)$	In a *n*-node binary tree, the searching number of interval between leaf nodes, i.e. the target keys does not exist in the tree.
$t_1(n)$	In a *n*-node binary tree, the searching number of first interval between leaf nodes, i.e. the target key is less than all keys in the tree.
$T(n)$	The total comparing time of searching *n*+1 intervals in binary tree

2.3 Total Key Number

Total key number in B+ tree is associated with the fullness of keys in nodes. With same height *h* and the maximum key number of every node *n*, when every node in B+ tree contains *n* keys, i.e. the number reaches the upper bound, we call the B+ tree is full, and it can hold maximum keys in all. Similarly, when every node occurs to have minimum number of keys, we say the tree is half-full, and the total key number is minimized. Thus, we can calculate the theoretical range of total key number *N*.

Lemma 1. The total number of key in B+ tree N can be calculated by height *h* and the maximum key number *n*, which can be denoted as formula (1).

$$2r^{h-2}m \le N \le q^{h-1}n \tag{1}$$

Proof of Lemma 1..

As analyzed above, the maximum of total key number occurs when the B+ tree is full, that is, every internal node has maximum keys and children *q*. The relation between maximum number of keys in B+ Tree N_{max} and height *h* is shown in Table 2.

Table 2 Relation Between N_{max} and *h*

Height (*h*)	**Number of Leaf Nodes**	**Maximum number of keys in B+ Tree $N_{\max}$**
1	1	n
2	q	qn
3	q^2	q^2n
…	…	…
h	q^{h-1}	$q^{h-1}n$

From recursive calculation, we can get $N_{\max} = q^{h-1}n$.

Meanwhile, the minimum of total key number occurs when the B+ tree is half full, i.e., every internal node has minimum keys *m* and children *r*. The relation between minimum number of keys in B+ Tree N_{min} and height *h* is shown in Table 3.

Table 3 Relation Between N_{min} and h

Height (h)	**Number of Leaf Nodes**	**Minimum number of keys in B+ Tree N_{min}**
1	1	1
2	2	$2m$
3	$2r$	$2rm$
4	$2r^2$	$2r^2m$
…	…	…
h	$2r^{h-2}$	$2r^{h-2}m$

From recursive calculation, we can get $N_{\min} = \begin{cases} 1, h = 1 \\ 2r^{h-2}m, h \geq 2 \end{cases}$.

Therefore, the range of total key number is $2r^{h-2}m \leq N \leq q^{h-1}n$.

The proof of Lemma 1 is completed. □

2.4 Height of B+ Tree

The range of height *h* only depends on the total number of keys *N* and the maximum key number *n* in a node. With same total key number *N* and the maximum key number *n*, the height *h* is largest when the B+ tree is half full, and *h* is smallest when B+ tree is full. Thus, we can calculate the theoretical range of height *h*.

But within range, the exact height *h* is associated with the inserting and deleting sequence of keys, as nodes will merge or split to keep balance.

Lemma 2. The range of height *h* only depends on the total number of keys *N* and the maximum key number *n* in a node, and it can be calculated with formula (2).

$$1 + \log_q \frac{N}{n} \leq h \leq 2 + \log_r \frac{N}{2m} \quad (2)$$

Proof of Lemma 2..

(1) Maximum height

According to the definition of B+ tree, the maximum of height is directly associated with total key number *N*, the least key number of leaf nodes *m*, and the least children number of internal nodes *r*. From Lemma 1, we can get $N_{\min} \geq 2r^{h-2}m$.

Thus, we can get the maximum of height $h \leq 2 + \log_r \frac{N}{2m}$.

(2) Minimum height

The minimum of height is directly associated with total key number *N*, the most children number of internal nodes *q*, and the maximum key number *n*. From Lemma 1, we can get $N_{\max} \leq q^{h-1}n$.

Thus, we can get the maximum of height $h \geq 1 + \log_q \frac{N}{n}$.

Therefore, the range of height is $1 + \log_q \frac{N}{n} \leq h \leq 2 + \log_r \frac{N}{2m}$.

The proof of Lemma 2 is completed. □

From Lemma 2, we can see that the height *h* of B+ tree is logarithm to the total key number *N*.

The larger the maximum key number *n* in a node has, the lower the height *h* becomes.

3 Matching Times in Binary Search

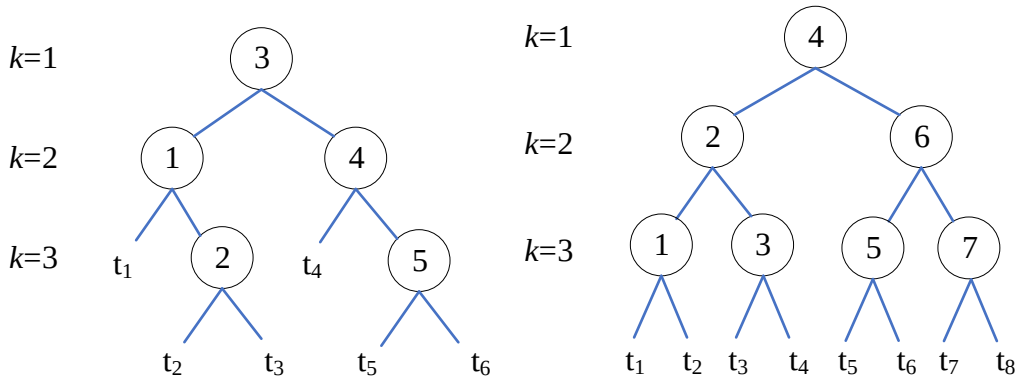
Matching time can be divided into 2 parts: the matching times of B+ tree nodes, and the binary matching times within a node, i.e. the *n*-node binary tree.

3.1 Matching Times within Once Binary Search

A binary search within an ordinal sequence, can be seen as a search within a *n*-keys ordinal balanced binary tree, where the height of tree is denoted as *k*.

Definition 2.[3] as a variant of binary tree, a tree such as Figure 1 is called the *ordinal balanced binary tree* with *n* internal nodes.

The tree has an the useful property that from a left-order traversal of the ordinal balanced binary tree, we can get an ordinal sequence, same as the ordinal keys in a B+ tree node.



(a) Half-full Binary Tree (b) Full Binary Tree

Figure 1. Binary Trees with Height of 3

1) Height of a Binary Tree ***k***

Lemma 3. The maximum height of tree is $k = \lceil \log_2(n+1) \rceil$.

2) Matching Times of target Key ***s(n)***

As shown in the Figure 1, matching times is equal with the height *k* of the key. For example, in Figure 1(a), the matching times of key 3 is 1, the matching times of key 1 and key 4 are 2, and the matching times of key 2 and key 5 are 3.

Lemma 4. Let s_i be the matching times of key *i*. Then in a binary tree, $s_i=k$, and $1 \le s_i \le \lceil \log_2(n+1) \rceil, 1 \le i \le n$.

3) Matching Times of Missing Key ***t(n)***

When the target key is not in the tree, the search will hit an interval, like t_i in Figure 1. The matching times of a missing keys is equal with the matching times of last node. For example, when searching the missing key 0 in Figure 1(a), the searching ends after comparing with key 1, and matching times is 2, same as searching key 1. Since the binary tree is balanced, the interval only occurs in the *k*-1 tier or the *k* tier.

Lemma 5. Let t_j be the matching times of interval *j*. Then in a balanced binary tree, we can get:

(1) $t_{\max} = k = \lceil \log_2(n+1) \rceil$;

(2) $t_{\min} = \begin{cases} k = \lceil \log_2(n+1) \rceil, n = 2^k - 1 \\ k - 1 = \lfloor \log_2(n+1) \rfloor, n \neq 2^k - 1 \end{cases}$.

When the tree is full, every interval has the same matching times, i.e. the height of tree *k*.

From Lemma 4 and Lemma 5, we can see that $t_{\max} = s_{\max}$.

4) Matching Times of the smallest Missing Key ***$t_1(n)$***

In binary tree, $t_1(n)$ lays in the left side of the tree, i.e. searching for a key that smaller than every key in the tree.

Lemma 6. The matching times of first interval is computed as follows.

$$t_1(n) = \lfloor log_2(n+1) \rfloor$$

3.2 Total Matching Times of a Binary Tree

1) Total Matching Times of n Keys ***S(n)***

Lemma 7. For an ordinal balanced binary tree, let *S(n)* be the total matching times of *n* nodes. *S(n)* can be calculated as follows.

$$S(n) = (n+1)\lceil \log_2(n+1) \rceil - 2^{\lceil \log_2(n+1) \rceil} + 1 \quad (3)$$

Proof of Lemma 7..

From Lemma 4, total matching times of *n* nodes is total nodes number in every tier. Binary tree has at most 2^{i-1} nodes in tier *i*, and in last tier, it has $n - \sum_{i=1}^{k-1} 2^{i-1}$ nodes.

Add up number of nodes in every tier, we can get expression of *S(n)*.

$$S(n) = \sum_{i=1}^{k-1} 2^{i-1} \cdot i + (n - \sum_{i=1}^{k-1} 2^{i-1})k \quad (4)$$

To simplify the formula after the plus, from summation formula of geometric series, we can have $S(n) = \sum_{i=1}^{k-1} 2^{i-1} \cdot i + (n+1)k - 2^{h-1} \cdot k$. (5)

For $\sum_{i=1}^{k-1} 2^{i-1} \cdot i$, we simplify it with dislocation subtraction.

Let $\mathrm{A} = \sum_{i=1}^{k-1} 2^{i-1} \cdot i$, and let $j = i + 1$.

Then,

$$\begin{aligned} 2\mathrm{A} &= \sum_{i=1}^{k-1} 2^i \cdot i \\ &= \sum_{j=2}^{k} 2^{j-1} \cdot (j-1) \\ &= \sum_{j=2}^{k} 2^{j-1} \cdot j - \sum_{j=2}^{k} 2^{j-1} \\ &= (\sum_{j=1}^{k-1} 2^{j-1} \cdot j - 1 + 2^{k-1} \cdot k) - \sum_{j=2}^{k} 2^{j-1} \\ &= (A - 1 + 2^{k-1} \cdot k) - (2^k - 2) \end{aligned}$$

$$\therefore A = 2^{k-1} \cdot k - 2^k + 1 \quad (6)$$

From formula (5) and (6), we have $S(n) = (n+1)k - 2^k + 1$.

Considering $k = \lceil \log_2(n+1) \rceil$, we can get $S(n) = (n+1)\lceil \log_2(n+1) \rceil - 2^{\lceil \log_2(n+1) \rceil} + 1$.

The proof of Lemma 7 is completed. □

*2) Total Matching Times of Missing Keys **T(n)***

Lemma 8. For a binary search in an ordinal sequence, let *T(n)* be The total matching times of *n*+1 intervals. *T(n)* can be calculated as follows.

$$T(n) = (n+1)(\lceil log_2(n+1) \rceil + 1) - 2^{\lceil log_2(n+1) \rceil} \quad (7)$$

Proof of Lemma 8..

In an ordinal sequence, *n* keys have *n*+1 interval. In other words, in an ordinal balanced binary tree, all leaf nodes have 2 kids as interval, and all vacancy in leaf tier are interval, like t_1 and t_4 in Figure 1(a). As we analyze before, the matching times of an interval, is same as the matching times of its parent node, that is, the height of father node.

Let H be the number of leaf nodes, then leaf nodes have 2H children with matching times of every interval is *k*. Since total number of interval is $n+1$,leaf node tier has $n+1-\mathrm{H}$ vacancy with matching times of every interval is $k-1$.

$$\mathrm{H} = n - \sum_{i=1}^{k-1} 2^{i-1} = n + 1 - 2^{k-1} \quad (8)$$

Then we have,

$$\mathrm{T}(n) = \sum_{i=1}^{n+1} t_i = 2\mathrm{H} \cdot k + (n+1-2\mathrm{H}) \cdot (k-1) = (n+1)(k-1) + 2\mathrm{H} \quad (9)$$

From formula (8) and (9), we can simplify T(n) as $\mathrm{T}(n) = (n+1)(k+1) - 2^k$.

Considering $k = \lceil \log_2(n+1) \rceil$, we can get $T(n) = (n+1)(\lceil \log_2(n+1) \rceil + 1) - 2^{\lceil \log_2(n+1) \rceil}$.

The proof of Lemma 8 is completed. □

4 Matching Times in B+ Tree

4.1 Matching Times within Once Searching in B+ Tree

In this subsection, we calculate the maximum and minimum of the total matching times of once key search in B+ tree, where the B+ tree is full and half respectively. When search a key in B+ tree, total matching times of a certain key, is to add up the matching times within every nodes in the path, i.e. the total matching times of every binary search.

Property. A key in B+ tree at most occurs twice: (1) within internal nodes, a key occurs once or not; (2) within leaf nodes, a key occurs once.

1) Matching Times of Target Keys in Full B+ Tree

Figure 2 shows an example of a 3-tier full B+ tree ($h = 3$), where *n* is set as 2 and *q* is 3. From Lemma 1, $N_{\max} = q^{h-1}n = 18$, i.e. the B+ tree can contain at most 18 keys.

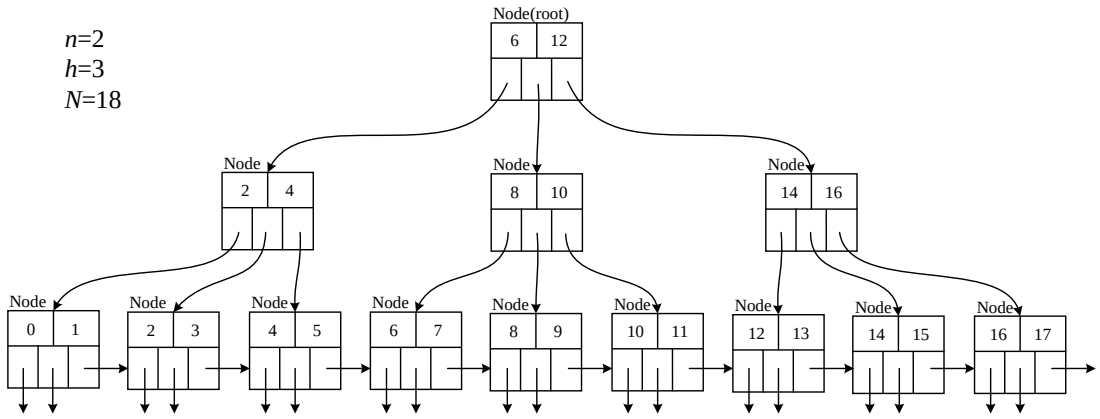


Figure 2. A 3-tier full B+ tree with *n* is 2

Lemma 9. Let *u* be the total matching time of a contained key in B+ tree. When the tree is full, we have the range of *u*.

$$1+(h-1)\lfloor \log_2(n+1) \rfloor \le u \le h\lceil \log_2(n+1) \rceil \qquad (10)$$

Proof of Lemma 9..

① Maximum Matching Times

The maximum matching times occurs when we are querying the last key in the tree, such as the 18 in Figure 2. Let u_i be the total searching time of the first *i* tiers. In this case, searching misses in all internal nodes, thus the matching times needs $u_{n-1}=(h-1)t_{\max}$; searching hits after maximum matching in leaf node, the thus the matching times needs $s_{\max}$. With Lemma 4 and Lemma 5, adding up and we can have $u_{\max}$.

$$u_{\max}=(h-1)t_{\max}+s_{\max}$$
$$\because s_{\max}=t_{\max}$$
$$\therefore u_{\max}=h\lceil \log_2(n+1) \rceil$$

② Minimum Matching Times

The minimum matching times occurs when searching minimum times in every nodes: matching times in internal takes up $t_{\min}$, and hits after one match in leaf node. Then we can get $u_{\min}$.

$$u_{\min}=(h-1)t_{\min}+1$$
$$\therefore u_{\min}=(h-1)\lfloor \log_2(n+1) \rfloor+1$$

To sum up condition ① and ②, total matching times of searching a target key in full B+ tree have

$$1+(h-1)\lfloor \log_2(n+1) \rfloor \le u \le h\lceil \log_2(n+1) \rceil \qquad (11)$$

The proof of Lemma 9 is completed. □

2) Matching Times of Target Keys in Half-full B+ Tree

Figure 3 shows an example of a 3-tier half-full B+ tree ($h=3$), where *m* is 1 and *r* is 2. From Lemma 1, $N_{\min}=2r^{h-2}m=4$, i.e. the B+ tree can at least contain 4 keys.

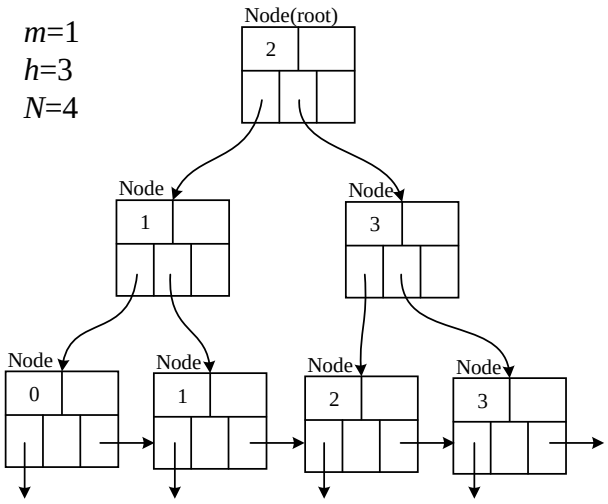


Figure 3. A 3-tier half-full B+ tree with *m* is 1 and *r* is 2

Lemma 10. Let *u* be the total matching time of a contained key in B+ tree. When the tree is half-full, we have the range of *u*.

$$2+(h-2)\lfloor \log_2(l+1) \rfloor \le u \le 1+(h-2)\lceil \log_2(l+1) \rceil + \lceil \log_2(m+1) \rceil \quad (12)$$

① Maximum Matching Times

Same as the full tree, the maximum matching times occurs in half-full tree when we are querying the last key in the tree, such as the key 3 in Figure 3.

Let u_i be the total searching time of the first *i* tiers. In this case, the root node has only one key but searching misses, and searching misses in all $h-2$ internal nodes, thus the matching times needs $u_{n-1}=1+(h-2)t_{\max}(r)$; searching hits after maximum matching in leaf node, the thus the matching times needs $s_{\max}(m)$. Adding up and we can have $u_{\max}$.

$$\begin{aligned} u_{\max} &= 1+(h-2)t_{\max}(r)+s_{\max}(m) \\ \because t_{\max} &= \lceil \log_2(l+1) \rceil \\ s_{\max} &= \lfloor \log_2(m+1) \rfloor \\ \therefore u_{\max} &= 1+(h-2)\lceil \log_2(l+1) \rceil + \lfloor \log_2(m+1) \rfloor \end{aligned}$$

② Minimum Matching Times

The minimum matching times occurs when searching minimum times in every nodes: (1) matching once in root node, (2) matching times in internal takes up $t_{\min}$, and (3) hits after one match in leaf node. Then we can get $u_{\min}$.

$$\begin{aligned} u_{\min} &= 1+(h-2)t_{\min}+1 \\ \therefore u_{\min} &= 2+(h-2)\lfloor \log_2(l+1) \rfloor \end{aligned}$$

To sum up condition ① and ②, total matching times of searching a target key in half-full B+ tree have

$$2+(h-2)\lfloor \log_2(l+1) \rfloor \le u \le 1+(h-2)\lceil \log_2(l+1) \rceil + \lfloor \log_2(m+1) \rfloor \quad (13)$$

The proof of Lemma 10 is completed. □

4.2 Total Matching Times of B+ Tree

1) Total Matching Times of Keys in Full B+ Tree

Theorem 1. Let $P(h)$ be total matching times of keys in B+ tree with height of h. Then in a full B+ tree, $P(h)$ can be calculated as follow.

$$P = n(h-1)q^{h-2}T(n) + (S(n) - T(n) + t_1(n))\frac{q^{h-1}-1}{q-1} + q^{h-1}S(n) \qquad (14)$$

Proof of Theorem 1..

As shown in Figure 4, considering a full B+ tree with n keys per node and height of h+1, the root node has n+1 subtree, where every subtree is a full B+ tree with n keys per node and height of h.

Let N be total key number of h+1 tiers, and let M be total key number of h tiers. From Definition 1 and Lemma 1, we have $\mathrm{M} = (n+1)^{h-1}n$ and $\mathrm{N}=\mathrm{M}\cdot(n+1)$.

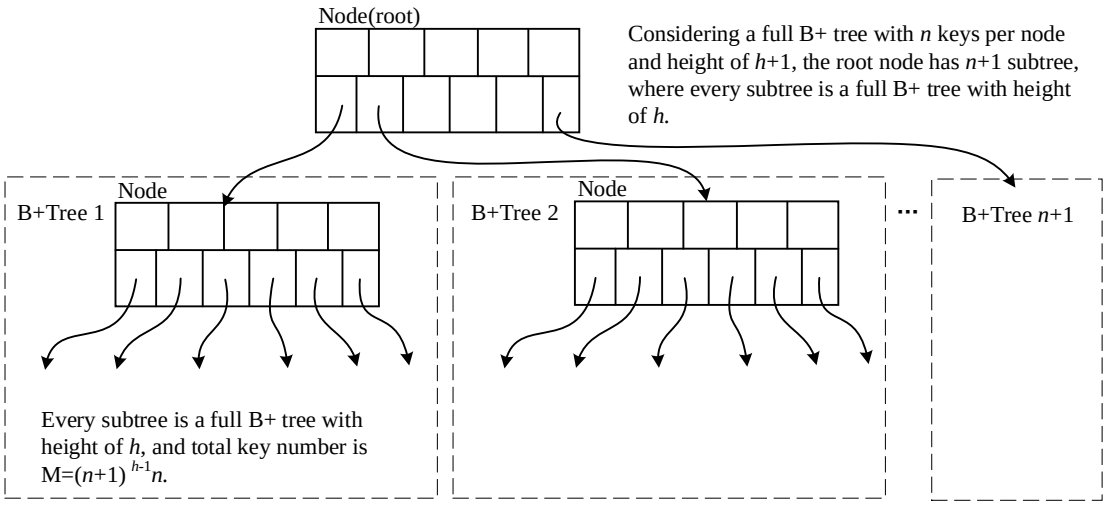


Figure 4. A full multi-keys B+ tree with height of h+1

Let total matching times of keys in B+ tree with height of h, be $P(h)$. Then we discuss following two conditions.

① When height of B+ tree is h=1, all keys are hit in nodes, and we have $P(1) = \sum_{i=1}^{n} s_i = S(n)$.

② When height of B+ tree is h+1, $P(h+1)$ can be calculated as the sum $P(h)$ of n+1 subtrees, and total matching times of N keys in root node. There are three parts of calculation, and we will illustrate in sequence.

A. The first subtree

For subtree 1 in Figure 4, let V be total matching times of M keys among h+1 tiers, and let v be total matching times of 1 key among h+1 tiers. Since the first subtree has keys smaller than all keys in root node, the M keys in root node of the h+1 tier all take up matching times of t_1. Then we have $v_i = P_i + t_1, i \in [1, M]$.

Thus, total matching times of M keys among h+1 tiers is as follow.

$$V_1 = \sum_{i=1}^{M} v_i = \sum_{i=1}^{M} P_i + t_1 \cdot M = P(h) + t_1 \cdot M \qquad (15)$$

B. The second subtree

For M keys in the second subtree, the first key needs matching times of s_1 in the root node, and other $M-1$ keys need t_2 times. Then we have calculation of v_i of M keys in the second subtree.

$$v_i = \begin{cases} P_1 + s_1, i = 1 \\ P_i + t_2, i \in [2, M] \end{cases} \qquad (16)$$

Thus, total matching times of M keys in the second subtree among h+1 tiers is as follow.

$$V_2 = \sum_{i=1}^{M} v_i = \sum_{i=1}^{M} P_i + s_1 + t_2 \cdot (M-1) = P(h) + s_1 + t_2 \cdot (M-1) \qquad (17)$$

C. Remaining subtrees

The remaining subtrees (subtree 3 to subtree n+1) have similar condition with the second subtree, that is, for subtree i, the first key of M keys takes up matching times of s_{i-1} among the h+1 tier, and the other $M-1$ keys takes up matching times of t_i among the h+1 tier.

Thus,

$$V_i = P(h) + s_{i-1} + t_i \cdot (M-1), i \in [3, n+1] \qquad (18)$$

From formula (15), (17) and (18), we can add up all the V_i and get the recurrence formula of $P(h+1)$.

$$P(h+1) = \sum_{i=1}^{n+1} V_i = (n+1) \cdot P(h) + S(n) + (M-1) \cdot T(n) + t_1$$

With $\mathrm{M} = (n+1)^{h-1} n$, we can get

$$P(h+1) = (n+1) \cdot P(h) + S(n) - T(n) + t_1 + (n+1)^{h-1} \cdot n \cdot T(n).$$

Here are the process of simplification.

Set $\begin{matrix} n+1 = x \\ nT(n) = y, \\ S(n) - T(n) + t_1 = z \end{matrix}$ （19）

Then we have $\begin{matrix} P(h) = x \cdot P(h-1) + y \cdot x^{h-2} + z \\ x \cdot P(h-1) = x^2 \cdot P(h-2) + y \cdot x^{h-2} + z \cdot x \\ \cdots \\ x^k \cdot P(h-k) = x^{k+1} \cdot P(h-k-1) + y \cdot x^{h-2} + z \cdot x^k \end{matrix}$.

Adding up the above k formulas, we have

$$\sum_{i=0}^{k} x^i \cdot P(h-i) = \sum_{i=1}^{k+1} x^i \cdot P(h-i) + (k+1) y \cdot x^{h-2} + z \cdot \sum_{i=0}^{k} x^k .$$

Eliminate the $\sum_{i=0}^{k} x^i \cdot P(h-i)$ in both sides of the equation, and calculate the sum up of $\sum_{i=0}^{k} x^k$, we have $P(h) = x^{k+1} \cdot P(h-k-1) + (k+1)y \cdot x^{h-2} + z \cdot \frac{x^{k+1}-1}{x-1}$.

Let k be $h-2$, then $P(h) = x^{h-1} \cdot P(1) + (h-1)y \cdot x^{h-2} + z \cdot \frac{x^{h-1}-1}{x-1}$.

With x, y and z in formula (19), we have

$$P = (n+1)^{h-1} \cdot S(n) + (h-1) \cdot (n+1)^{h-2} \cdot n \cdot T(n) + \left[S(n) - T(n) + t_1\right] \cdot \frac{(n+1)^{h-1}-1}{n}.$$

The proof of Theorem 1 is completed. □

2) Total Matching Times of Keys in Half-full B+ Tree

Theorem 2. In a half-full B+ tree, the total matching times of keys in B+ tree with height of h $P(h)$ can be calculated as follow.

$$P(h) = 2(r^{h-2}m + r^{h-2}S(m) + (h-2)r^{h-3}mT(l) + (S(l) - T(l) + t_1(l))\frac{r^{h-2}-1}{r-1}) \quad (20)$$

Proof of Theorem 2..

As shown in Figure 5, considering a half-full B+ tree with r keys per node and height of h, the root node has one key and two subtrees, where every subtree is a half-full B+ tree with m keys per node and height of h-1. Let N be total key number of h-1 tiers, and let M be total key number of $h-2$ tiers. From Definition 1 and Lemma 1, we have $\mathrm{M} = r^{h-2}m$ and $\mathrm{N}=\mathrm{M} \cdot (r+1)$. And total key number of h-tier B+ tree is 2N. Each subtree has r+1 branches, which are B+ tree with height of h-2 and total key number is M=$r^{h-3}m$.

Let $P(h)$ be total matching times of h-tier B+ tree. Then total matching times of h-1 tiers is $P(h-1)$, like subtree $\{h-1\}@1$ and $\{h-1\}@2$, as the red boxes shown in Figure 5. And total matching times of h-2 tiers is $P(h-2)$, like branch tree $\{h-2\}@r$, as the blue boxes shown in Figure 5. In following parts, we discuss the calculation of $P(h)$.

① When height of B+ tree is h=1, the root node has only one key, thus we have $P(1)=1$.

② When height of half-full B+ tree is $h-1$, $P(h\text{-}1)$ can be calculated as the sum $P(h\text{-}2)$ of r+1 branch trees, and total matching times of M keys in root node. When height of half-full B+ tree is h, $P(h)$ can be calculated as the sum $P(h\text{-}1)$ of 2 subtrees, and total matching times of N keys in root node. We can calculate $P(h)$ through two steps.

A. Calculation of $P(h-1)$

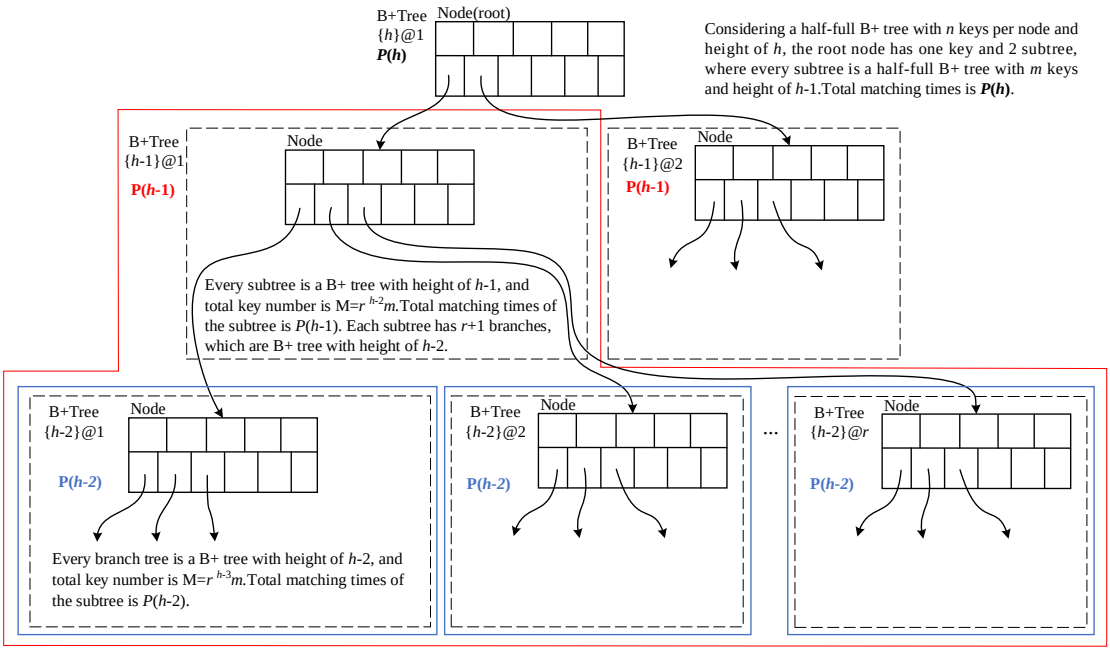


Figure 5. A half-full B+ tree with height of h

Let V be total matching times of N keys among h tiers, and let v be total matching times of 1 key among h tiers.

a) Branch Tree $\{h-2\}@1$

For branch tree $\{h-2\}@1$ in Figure 5, the M keys all need to match t_1 times in h-1 tier. Thus, $v_i = P_i + t_1, i \in [1, M]$.

Adding up all matching time of branch tree $\{h-2\}@1$ among h tiers, we have V_1 as follow.

$$V_1 = \sum_{i=1}^{M} v_i = \sum_{i=1}^{M} P_i + t_1 \cdot M = P(h-2) + t_1 \cdot M \qquad (21)$$

b) Branch Tree $\{h-2\}@2$

For M keys in branch tree $\{h-2\}@2$, the first key needs s_1 times of matching in the h-1 tier, and other M-1 keys all need t_2 times of matching. Then we have v_i for branch tree $\{h-2\}@2$.

$$v_i = \begin{cases} P_1 + s_1, i = 1 \\ P_i + t_2, i \in [2, M] \end{cases}$$

Thus, matching times of branch tree $\{h-2\}@2$ among $h-1$ tiers is as follow.

$$V_2 = \sum_{i=1}^{M} v_i = \sum_{i=1}^{M} P_i + s_1 + t_2 \cdot (M-1) = P(h-2) + s_1 + t_2 \cdot (M-1) \qquad (22)$$

c) Remaining Branch Trees

The remaining r-1 branch trees of $\{h-1\}@1$ (branch tree $\{h-2\}@3$ to $\{h-2\}@(r+1)$) have similar situation with the branch tree $\{h-2\}@2$, that is, for branch tree $\{h-2\}@i$, the first key of M keys takes up matching times of s_{i-1} among the $h-1$ tier, and the other $M-1$ keys takes up matching times of t_i among the $h-1$ tier. Thus, we have V_i for remaining branch trees.

$$V_{n+1} = P(h-2) + s_n + t_{n+1} \cdot (M-1) \tag{23}$$

From formula (21) to (23), we can add up all the V_i and get the recurrence formula of $P(h-1)$.

$$P(h-1) = \sum_{i=1}^{n+1} V_i = (n+1) \cdot P(h-2) + S(n) + (M-1) \cdot T(n) + t_1 \tag{24}$$

B. Calculation of $P(h)$

As analyzed before, $P(h)$ can be calculated as sum of two $P(h-1)$, and matching times of N keys in the h tier, as follow.

$$P(h) = 2\left[P(h-1) + (r+1)^{h-2} \cdot m \right] \tag{25}$$

With formula (24), it can be simplified as follow.

$$P = 2(r^{h-2}m + r^{h-2}S(m) + (h-2)r^{h-3}mT(l) + (S(l) - T(l) + t_1(l))\frac{r^{h-2}-1}{r-1}) \tag{26}$$

The proof of Theorem 2 is completed. □

3) Average Matching Times in B+ Tree

In this subsection, we calculate the average matching times of full B+ tree and half-full B+ tree, respectively.

Theorem 3. Average matching times of a full B+ tree, can be calculated as follow.

$$Q_{Full} = \frac{n(h-1)q^{h-2}T(n) + (S(n) - T(n) + t_1(n))\frac{q^{h-1}-1}{q-1} + q^{h-1}S(n)}{q^{h-1}n} \tag{27}$$

Proof of Theorem 3..

From Lemma x, we have the total key number of a full B+ tree as $\mathrm{N} = q^{h-1}n$.

According to Theorem 1, total matching time of the full B+ tree is

$$P_{Full} = (n+1)^{h-1} \cdot S(n) + (h-1) \cdot (n+1)^{h-2} \cdot n \cdot T(n) + \left[S(n) - T(n) + t_1 \right] \cdot \frac{(n+1)^{h-1} - 1}{n}.$$

Thus, the average matching times of a full B+ tree is as follow.

$$Q_{Full} = \frac{n(h-1)q^{h-2}T(n) + (S(n) - T(n) + t_1(n))\frac{q^{h-1}-1}{q-1} + q^{h-1}S(n)}{q^{h-1}n} \quad (28)$$

The proof of Theorem 3 is completed. □

Theorem 4**.** Average matching times of a half-full B+ tree, can be calculated as follow.

$$Q_{half} = \frac{r^{h-2}m + r^{h-2}S(m) + (h-2)r^{h-3}mT(l) + (S(l) - T(l) + t_1(l))\frac{r^{h-2}-1}{r-1}}{r^{h-2}m} \quad (29)$$

Proof of Theorem 4..

From Lemma x, we have the total key number of a half-full B+ tree as $\mathrm{M} = r^{h-2}m$.

According to Theorem 2, total matching time of the half-full B+ tree is

$$P_{half} = 2(r^{h-2}m + r^{h-2}S(m) + (h-2)r^{h-3}mT(l) + (S(l) - T(l) + t_1(l))\frac{r^{h-2}-1}{r-1}).$$

Thus, the average matching times of a half-full B+ tree is as follow.

$$Q_{half} = \frac{r^{h-2}m + r^{h-2}S(m) + (h-2)r^{h-3}mT(l) + (S(l) - T(l) + t_1(l))\frac{r^{h-2}-1}{r-1}}{r^{h-2}m} \quad (30)$$

The proof of Theorem 4 is completed. □

With Theorem 3 and Theorem 4, we can construct B+ trees and verify the correctness, results are shown in Section 5.

Lemma 11. With same height *h* and key number per node *n*, when total number of keys $N \neq 2^n - 1$,the minimum average matching times of full B+ tree is same as the maximum average matching times of half-full B+ tree; when total number of keys N is equal with $2^n - 1$, the minimum average matching times of full B+ tree is two more than the maximum average matching times of half-full B+ tree.

Proof of Lemma 11..

From Lemma x, since we have certain situation of total key number .

$$u_{Full-\min} = 1 + (h-1)\lfloor \log_2(\mathrm{N}+1) \rfloor$$

$$u_{Half-\max} = 1 + (h-2)\left\lceil \log_2\left(\left\lfloor \frac{\mathrm{N}}{2} \right\rfloor + 1\right) \right\rceil + \left\lceil \log_2\left(\left\lfloor \frac{\mathrm{N}+1}{2} \right\rfloor + 1\right) \right\rceil$$

Then we subtract above two formula.

$$\Delta = u_{Full-\min} - u_{Half-\max} = (h-1)\lfloor \log_2(\mathrm{N}+1) \rfloor - (h-2)\left\lceil \log_2\left(\left\lfloor \frac{\mathrm{N}}{2} \right\rfloor + 1\right) \right\rceil - \left\lceil \log_2\left(\left\lfloor \frac{\mathrm{N}+1}{2} \right\rfloor + 1\right) \right\rceil$$

Thus, $\Delta = \begin{cases} h-2, N = 2^n - 1 \\ 0, N \neq 2^n - 1 \end{cases}$.

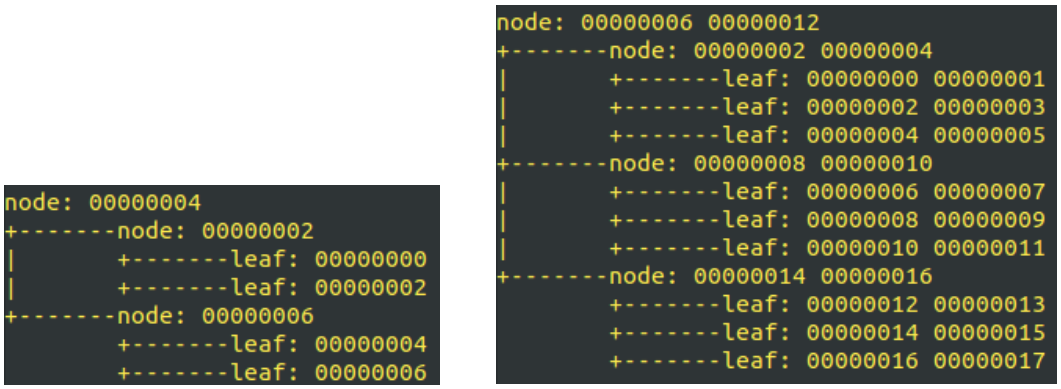
The proof of Lemma 11 is completed. □

5 Numerical Simulations

5.1 Total Key Number

To verify Lemma 1 and Lemma 2, we construct a half-full B+ tree and a full B+ tree, with their height is 3, and calculate the total key number. The outcome of simulation is shown in Figure 6, where maximum keys of node n is set as n=2, and height of B+ tree is limited to h=3. Thus the least key number of leaf nodes is $m=\lfloor(n+1)/2\rfloor=1$, the least children number of internal nodes is $r=\lfloor n/2\rfloor+1=1$, and the most children number of internal nodes is $q=n+1=3$. From Lemma 1, we can theoretically calculate the $N_{min}=2mr^{h-2}=2\times2\times1^{3-2}=4$ and $N_{max}=nq^{h-1}=2\times3^{3-1}=18$.

From Figure 6, we can see that (1) in half-full B+ tree, total key number is 4, and (2) in full B+ tree, total key number is 18. The simulation results are in conformity with Lemma 1.



(a) Half-full Binary Tree (b) Full Binary Tree

Figure 6. Simulated Binary Trees with Height of 3

5.2 Average Matching Times

To validate Theorem 3 and Theorem 4, we conduct a simulation on 241 well-constructed B+ trees of full entries and half full respectively, where height h is 3 and the maximum number of keys in node n ranges from 2 to 242. We traverse all keys of each tree and obtain the matching times of each key. Figure 6 illustrates the relation between theoretical upper/lower bound of matching times for half tree and full tree. From Figure 7, we can see that: The average matching times of keys are between the upper bound and the lower bound of theoretical value. This means that our theoretical analysis and experiments are correct.

References

[1] Bayer R , Mccreight E M . Organization and maintenance of large ordered indexes[J]. Acta Informatica, 1972, 1(3):173-189.

[2] Knuth D E.The art of computer programming 3: Sorting and searching[M]. Addison-Wesley, 1973.

[3] Knuth D E. Optimum binary search trees[J]. Acta informatica, 1971, 1(1): 14-25.

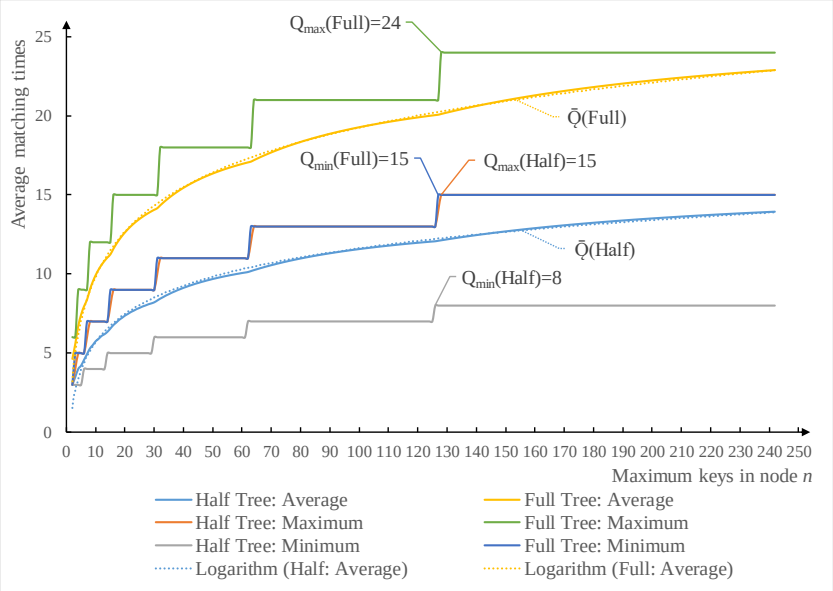


Figure 7 Verification of average matching time in memory (The solid line is the experimental data drawing, and the dashed line is the fitting formula)