2018 International Conference on Computational, Modeling, Simulation and Mathematical Statistics (CMSMS 2018)
ISBN: 978-1-60595-562-9

An Improved Algorithm of Binary Balanced Tree with Super Large-scale Data Set

Zhong-ming YANG[1,*], Ya-ping CHANG[1] and Ya-ru YANG[2]

[1]College of Computer Engineering Technical Guangdong Polytechnic of Science and Technology, Zhuhai, Guangdong, P.R. China 519090

[2]Zhuhai Vocational School of Polytechnics, Zhuhai, Guangdong, P.R. China 519090

[*]Corresponding author

Keywords: Balanced binary tree, Large scale data, Size balanced tree, Multi-line merging sort.

Abstract. Realization of quick search still relies on ordered data in the application scenarios of large-scale data search and analysis. This paper analyses an improved algorithm of realizing super large-scale balanced tree efficiently in the concurrent environment. Size Balanced Tree, which is hard to build a balanced tree in the processing of mass data, can group the mass data and then combined groups into a SBT by multi path conflation algorithm. Experiment shows compared with other binary sorting trees(AVL, Treap, random BST), there is no detailed leap variation in time along with the increasing knots. This algorithm has a low time complexity and high running efficiency. It can be applied in the application scenarios of operating quick ordered search of super large-scale data in the artificial intelligence field.

Introduction

Balanced binary tree, an optimized form of binary search tree, is to utilize the optimized strategy of reducing average path length and average search time to avoid over-deformed tree structure. Quick searching still relies on ordered data in the application scenarios of large-scale of data search and analysis, thus how to realize super large-scale balanced tree efficiently in concurrence environment seems very important. [1]

Paper [2] presents a unified framework of realization of balanced binary tree. Sleator [3] presents a quite classic sorting algorithm and Splay tree namely self-adjusting balanced tree, whose algorithm has a complexity rate of O(logn). Lai[4] present a self-adjusting heuristic upper and lower bound method to maintain the binary sorting tree and operate on the tree with pointer constant. Paper[5] presents multi Splay tree and make optimization of operation complexity.

In the aspect of Large Scale Data, Paper[6] presents a conflation algorithm by utilizing parallel processor, whose algorithm can make the ordered data distribution completely balanced and achieve a higher loading balance, extensibility and distribution stability. Paper[7] presents an odd bitonic sequence algorithm which is improved after adding CCI operation. The improved algorithm can make correct sorting of random odd bitonic sequence without adding the storage space and keeping calculation complexity level grade unchanged. The method in Paper[8] is to use hardware parallel processing sorting algorithm. The algorithm in Paper[9] can make the ordered data distribution completely balanced and achieves a higher loading balance, extensibility and distribution stability.

Thoughts of Size Balanced Tree

Size Balanced Tree

Binary sorting tree(BST), called as binary search tree, is a binary tree in which knots are ordered well. After in order traversals of BST will gain the sequence data. Due to the characteristics of trees, binary sorting tree is still applied in dynamic generation and also applied in the situation of continuously extending data. Assumed BST has N knots with max tree depth as N and minimum treed depth as logN (tree here refers to a balanced tree). So the worst situation of sorting is O(N2) and the best situation is O(NlogN).

Size Balanced Tree (SBT), suggested by Chen Qifeng, is the improvement of BST method. In this method, the tree balance is kept during the sorting procedure of binary sorting tree to make the time cost always be O(NlogN). So the important thought of this algorithm is to maintain binary sorting tree as balanced tree.

First of all, so as to define SBT, the concept of tree size is used here. The tree size refers to the number of total knots in tree i. Assumed the root knot of the tree as T, and the size of tree is marked as s[T]. Right sub-tree of one knot t in Tree T is marked as right[t] and left sub-tree is marked as left[t]. And each knot t of SBT will meet two natures as below.

Nature a: s[right[t]] ≥ s[left[left[t]]],and s[right[t]] ≥ s[right[left[t]]]

Nature b: s[left[t]] ≥ s[right[right[t]]],and s[left[t]] ≥ s[left[right[t]]]

s[L] ≥ s[C] and s[L] ≥ s[D], here T, L and R are knots and A, B,C and D are sub-tree.

When inserting or deleting the knots, SBT tree will not meet nature a and nature b. The rotary operation of tree refers to the adjustment operation of sub-tree in binary tree. And each time's rotation will not affect the result of in order traversals of binary tree. Tree rotation includes left rotation and right rotation and these two rotation will be in mirror image and be reversal operation.

When using rotation to maintain the nature of SBT tree, the tee will be restored by the maintain operation, that is to recalculate the tree size. Assumed that function maintain(T) is used to restore SBT root on T. Precondition for calling maintain(T) is sub-tree of T belongs to SBT.

The Details of the BST

Here are 4 situations need to be discussed. Since nature a and nature b is symmetric, only two situations of nature a will be discussed here.

Situation 1: if s[Left[Left[T]]>s[Right[T]],then s[A] > s[R], (1) right rotate T, namely rotate_right(T), (2)restore size of T, namely maintain(T), (3)restore size of L, namely maintain(L).

Situation 2: if s[right[left[t]]>s[right[t]], then s[B] > s[R], (1) left rotate L sub-tree (2) right rotate T (3) restore size of L (4) restore size of T, (5) restore size of B.

Main operation of SBT are as below.

insert(T,k); insert k into tree T;

delete(T,k); insert k into tree T;

find(T,k); seek k in tree T, return to k knot if k is found, otherwise return to NULL;

maintain(T,flag); restore size of T, flag is judging mark of situation 1 and situation 2;

right_rotate(T); right rotate tree T;

left_rotate(); left rotate tree T;

As SBT is balanced, the tree height is logN. Searching should be done before inserting and deleting operation. Time complexity of search operation is O(logN) and time complexity of SBT sorting is O(NlogN).

Improvement of SBT

If mass knots data is need to be processed, then it is hard to build a SBT by using SBT. So the mass data can be grouped firstly, then combined the groups into a SBT by mufti path conflation algorithm.

Assumed the number of knots is N, number of groups M is (we take the upper bound of root N), the number of knots of each group is n=N/M, the conflation algorithm of two SBT is as below.

Assumed two SBT respectively as T1 and T 2. Key(T) is the key code value of knot T, depth(k,T) is the depth value of key code k in tree T. Multi-Line Merging Sort algorithm of M groups is as the code.

we take the upper bound of root N), the number of knots of each group is n=N/M, the conflation algorithm of two SBT is as below.

Assumed two SBT respectively as T1 and T 2. Key(T) is the key code value of knot T, depth(k,T) is the depth value of key code k in tree T. Multi-Line Merging Sort algorithm of M groups is as below.

Reason for degeneration of binary sorting tree is that since the data to be ordered is sequent, the tree degenerate to be linear links. The solution is to generate sequence number by using random

numbers and data size inserting the tree is random. Analogously, measure SBT to be insert and be inserted by depth of tree. So the search complexity of conflation algorithm with N data and M group is O(logN/M)=O(logN-logM) and sorting complexity is O(N/MlogN/M)=O(N/M(logN-logM), whose complexity is obviously lower than O(NlogN).

```
while( M > 1) {
// number of Multi-Line Merging
Sort
for( i = 1 – M/2) {
// group number of two paths
conflation
If ( s[T1] > s[T2] )
// the knot number of T1 is larger
than T2
Merge(T1, T2);
// insert T2 into T1
Else if ( s[T1] < s[T2])
// knot number of T2 is larger than
T1
Merge(T2, T1);
// insert T1 into T2
Else if( all( depth(
key(T1),T2)) > all(depth(
key(T1),T2)))
//total depth of key code of T1 is
larger than T1
Merge(T1, T2);
// insert T2 into T1
Else
Merge(T1, T2);
// insert T2 into T1
Merge(T2, T1);
// insert T1 into T2
M = upper(M/2);
// update M as upper bound
of M/2
}
Algorithm of inserting T2 into
T1 Merge(T1, T2):
for( each node n in T1)
If (not find(T2, key(n)))
//n isn't in T2
Insert(T2, key(n));
}
```

Algorithm Experiment

Definition of Data Type of Algorithm

Realization of SBT is firstly using template type in SBT and then it can be easily applied in different types of data and realize SBT tree by chain table.

Data knots are ordered by key code K key. The data installed with E type can be used to define knots data randomly in corresponding to knots in SBT.

Members of the newly added data of this type should be the number of sub-knots size 1, namely size of sub-tree, involved in this knot.

Operation Definition of Algorithm

In operation of comparison key code, we use comparison type to realize integer type's key code and comparison type.

The definition of template type of SBT tree is as below. Root knot and head knot of tree, pointer of end knot and core mender function of SBT is as follows. Right rotation, left rotation and maintain operation.

The rest member function of this template type is the main operation of SBT including inserting, deleting and searching which will cause imbalance of tree. So we need to judge if it could meet the conditions of SBT. Otherwise, main operation will be called to maintain conditions of SBT.

Experimental Data Set-up

When verifying the algorithm, we generate 10 thousand to 100 thousand knots data which will be storage in the file of sbt_file.txt in data sub-catalog by process. Special attention paid to be that the type of key word and knot size should be long type in case of transboundary mistake due to huge number of knots. In addition, we haven't considered the situation of same key code, thus this algorithm doesn't consider the stability issue.

The experimental environment is the personnel computer with double core 2.60GHz and 4GM internal storage, which operate 10 thousand to 100 thousand knots inserting (sorting) on SBT tree. The average running time is as below. It shows no leap variation of time along with the increasing of knots. The improved SBT tree is showed as Table 1.

Table 1. Average time test for knots inserting.

Number of sorted knots	10000	20000	30000	40000	50000
SBT running time	0.135	0.281	0.437	0.608	0.764
Number of sorted knots	60000	70000	80000	90000	100000
SBT running time	0.92	1.108	1.279	1.45	1.622

Experimental Result Analysis

SBT Experimental Result

AVL algorithm self-balance binary sorting tree is to judge if it is balance tree or not through comparison of tree height. Treap algorithm is a Tree+Heap balanced binary sorting tree. The building of balanced tree is through the comparison of knots priority. Since the binary sorting tree built by random numbers is most likely balanced, the worst situation of binary sorting tree is to insert a group of ordered numbers in sequence. Then binary sorting tree is degenerated to a linear chain table. So the algorithm of producing random key code to generate BST is called as random BST algorithm. Here we sort two million data to make a comparison between SNT algorithm, AVL algorithm and Treap algorithm. Seen from Figure 1, we will find that SBT algorithm achieves the minimum time complexity and the best running efficiency.

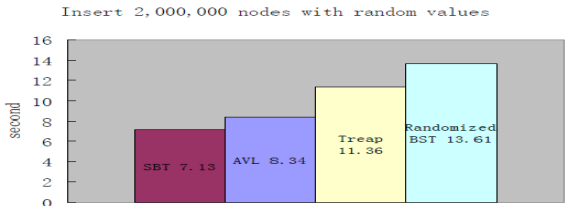

Figure 1. Comparison between SBT and other binary sorting trees.

Two million ordered value knots and two million random value knots are operated. Table 2 presents a comparison of tree's depth, height and rotation times between SBT algorithm and other algorithms.

Table 2. Experimental result of inserting two million knots data.

Inserting two million ordered value knots					
Type	**SBT**	**AVL**	**Treap**	**Random BST**	**Splay**
average depth	18.9514	18.9514	25.6528	26.2860	999999.5
height	20	20	51	53	1999999
rotation times	1999979	1999979	1999985	1999991	0
Inserting two million random value knots					
average depth	19.2415	19.3285	26.5062	25.5303	37.1953
height	24	24	50	53	78
rotation times	1568017	1395900	3993887	3997477	25151532

SBT Experiment Analysis

Seen from Figure 1, compared with other binary sorting trees(AVL, Treap and random BST), SBT achieves the minimum time complexity and the best running efficiency. SBT will juggle if occurring imbalance before each operation, so the tree will always be in optimal state. The maintenance of algorithm is to make comparison of knots size, with low cost of storage and high efficiency.

Seen from Table 2, random value sorting is faster than ordered value sorting for random value will be more likely to produce balanced binary tree and the cost of maintenance will decrease. The tree depth of ordered value sorting tree is lower than that of random value sorting tree for the maintenance operation in algorithm will make tree more balanced and tree height will be lower.

Conclusions

This paper presents SBT Multi-Line Merging Sort algorithm based on the improved SBT sorting algorithm. N knots are grouped into M groups and it offers Multi-Line Merging Sort algorithm of SBT tree and description of two SBT conflation algorithm. This algorithm has time complexity of O(N/M(logN-logM)) in the operation of sorting and kinds of operations like inserting, deleting and searching and it is especially applied in the sorting handling of large-scale mass data and shows a good experimental result.

Acknowledgment

This study was supported by Guangzhou science and technology project (201804010276), Guangdong Polytechnic of Science and Technology "research and social service platform" project.

References

[1] Shavit N. Data structures in the multicore age [J].Communications of the ACM,201,54(3): 76-84.

[2] Guibas, L.J. and Sedgewick, R. A Dichromatic Framework for Balanced Trees, Proceedings of the Nineteenth Annual IEEE Symposium on Foundations of Computer Science, 1978, 8-21.

[3] Sleator, D.D. and Tarjan, R.E. Self-adjusting Binary Search Trees, JACM, 1985, 32, 652-686.

[4] Lai, T.W. and Wood, D. Adaptive Heuristics for Binary Search Trees and Constant Linkage Cost. SIAM Journal on Computing 1998, 27: 6, 1564-1591.

[5] Wang, C.C., Derryberry, J. and Sleator, D.D. O(log log n)-Competitive Dynamic Binary Search Trees. Proceedings of the Seventeenth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA) 2006, January 22-26.

[6] Wang Wenyi, Qiou Yong. A New Algorithm for Parallel Mergesorting [J]. Computer Engineering and Applications. 2005, 41(5): 71-72.

[7] Zhang Jian-ping, Du Xue-dong. Parallel Sorting Algorithm of Odd Sequence [J]. Computer Engineering, 2007, 33(15): 96-97.

[8] Shi Y., Zhang L., Liu P. THSORT: A single-processor parallel sorting algorithm [J]. Journal of Software, 2003, 14(2): 159-165.

[9] Wang Ying, Li Kenli etl. A Vertical and Horizontal Multiway Algorithm for Parallel-Merging [J]. Journal of Computer Research and Development, 2006, 43(12): 2180-2186.