Compound Binary Search Tree and Algorithms

Yong Tan

yongtan@navigation@outlook.com

Abstract

The Binary Search Tree (BST) is average in computer science which supports a compact data structure in memory and oneself even conducts a row of quick algorithms, by which people often apply it in dynamical circumstance. Besides these edges, it is also with weakness on its own structure specially with poor performance at worst case\[5\]. In this paper, we will develop this data structure into a synthesis to show a series of novel features residing in. Of that, there are new methods invented for raising the performance and efficiency nevertheless some existing ones in logarithm or linear time.

Keywords: binary search tree; algorithm

1 Introduction

Binary Search Tree (BST) is a common data structure broad elaborated in many literatures and textbooks as that regular. At first, the construct on it can be referred to a binary tree in which besides each unit incident a key (or value), each even carries three links mutually to comprise a compact structure, whose pointers respectively point to its own members in family the parent and two children that resides in the left side and the right side but maybe innull for link in open\[1\] 2 5\]; especially the one without parent as root or ancestor to all others.

Inside a BST, analogous constitution may occur by generating roots and trees recursively—each can as minor root on which new twigs can bloom from although there has existed a chief root to all items in tree.

If refer to maintain or build a tree, which must comply a Protocol of constitution; say the least, all incident keys must obey the clause that each at left link or right link which in charge by its parent should smaller than or larger than parent’s. Consider within a more large rank, of two subtrees the left one or the right one and their common root, correspondently at the left or the right, each key of descendant in tree is smaller or larger than ancestors.

Hence that law strongly conducts the operation of adding a fresh item into a BST, upon that, building a tree is actually accounted as a row of item insertions. Meanwhile, the single insertion can be outlined as a course of comparison as a path\[5\], called depth by us. In theory, with the longest one among them, we can use to measure the shape of a BST.

We can define a proper tree with log \( n \) depth by such a bifurcated structure above-mentioned on each item as a standard pattern, where variable \( n \) is the number of items inside tree. For an accessing in a tree to achieve an operation, complexity can be estimated for lower bound in \( \Theta(\log n) \) or for upper bound in \( O(n) \)[5], clearly, both are decided by the shape of tree, frankly speaking, by a temporal series of insertions in building period.

Thus the flavor of cognition becomes interesting when we study the shape of BST: the future shape of BST has actually been destined by the permutation of that ready sequence in advance; in contrast, we either have not got any way to carry an arbitrary permutation suitable to guarantee the proper shape of building.

Worst still, that is a challenge to us so far; upon that, people turned to reduce the estimation of shape refer to a conception which surveys major likelihood of average depth if the BST made up with a random sequence. In [5] reported by the empirical of many retrials through, the average depth can into \( 2 \ln n \) in most cases, which approximate to \( 1.39 \log n \).

In substance, the conclusion Robert Sedgewick et al made is enough preferable to solve many estimations applied on algorithms executing on BST, at that in the book authors yet conceded those existing methods in a poor performance at worst case. In practice
the tractability about manipulating a BST sometimes becomes vulnerable in some perhaps, at least to render the performance with instability.

1.1 Results

(1) Develop the BST into a synthesis by integrating distinct structures and to survey those novel correspondences among them. (2) Discuss the operations involved so as to exert the advantages that have embedded in those new components and make them support mutually to raise whole performance. (3) Completely solve the issue of building a proper tree when with a stochastic sequence as input and guarantee the cost of building invested by a logarithm time. (4) Estimate the batch works on BST than the traditional.

1.2 Perliminated

of a tree, it must refer to a set

1.2 Perliminated

batch works on BST than the traditional.

1.2 Perliminated

of a tree, it must refer to a set

1.2 Perliminated

batch works on BST than the traditional.

1.2 Perliminated

of a tree, it must refer to a set

1.2 Perliminated

batch works on BST than the traditional.

1.2 Perliminated

of a tree, it must refer to a set

1.2 Perliminated

batch works on BST than the traditional.

1.2 Perliminated

of a tree, it must refer to a set
Third. (1) A terminal cannot be ancestor to another. (2) Contrary to \( t_1 \) and \( t_2 \) both as CKs, further suppose \( t_i \) is an ancestor to \( t_i \).

Thus \( t_i \) in the left subtree that in charge by root \( t_i \). Over there at least an item \( t \) as child at \( t_i \) right link such that there is existence of \( \mathcal{K}, t_i(\rho_i) < \mathcal{K}, t_i(\rho_i) \) and \( \mathcal{K}, t_i(\rho_i) < \mathcal{K}, t_i(\rho_i) \), then \( t_i \)’s ref will among \( \rho_i \) and \( \rho_i+1 \) to lead to a contradiction. The third holds.

In 1979, J. H. Morris had invented a similar linear sequence made in \( O(n \log n) \) the lower boundary, he aimed to traverse a BST in a convenience. Robert Sedgewick et al even had projected the all items in a tree to charge the vacant position made by deleting; the deleted PK left down. At the worst case, the process will be deleting one and moving one, the cost in \( O(2) \) at the worst case.

Therefore, of deleting a CK, if a terminal as the alternate, then equivalently to delete two and move one, the cost in \( O(3) \) time; instead, for a PK as the alternate, the cost will become deleting two and move two. Eventually, the cost may in \( O(4) \) at the worst case for deleting a CK.

Now in ARA model, we advise the two members in ARA with their preimages as alternates which by that CK, either at the right side or at the left side, both contiguous to CK; actually they are the successor and predecessor in tree to that CK which mentioned by Hibbard or Sedgewick; their opinions both are right.

Our advice may at least involves two grounds: (1) Either of two alternates in ARA, the key on it is the extremum in LS (maximum) or RS (minimum) that means it has the qualification as new root to charge that tree which in the past to rooting CK, since the relationship for each key on descendant in that tree to new root likewise consists with that protocol of constitution of BST; the alternative whether as a parent or whether as an ancestor. (2) By our proof, there is not any likelihood for these alternates being CKs concurrently.

Via the dimension of ARA we readily seek out the alternate so we gain a constant complexity that of \( O(4) \) on deleting operation, which far less than logarithm.

3 Operations

3.1 Deletion

The existing method of deleting an item off a BST is fairly in perplexity to people. The central issue is the deletion concerns the perhaps of damage to logical structure. In fact, the focus about this problem is on deleting a CK off, upon that it needs an alternate in the tree to charge the vacant position made by deleting; the selected condition at alternate clearly requires that one with the capable to hold on those trees again which ever rooted in that CK before.

T. Hibbard in 1962 proposed the successor as alternate which at the leftmost position in the right subtree rooting CK which needs a progress of examination like \( (>,<,\ldots,<) \) for seeking. Analogously to \( (,<,>,\ldots,) \), Robert Sedgewick et al suggested the predecessor in left subtree the rightmost.

However for both routes, the cost at least is involved to \( \log n \); In ARA model, we have an analysis about this selection as follows.

Deleting a terminal, that may in \( O(1) \) without overplus from other actions since no child in charge by terminal. For a PK with unique child, its child actually can as a root to hold on a subtree or null. However, the child can as the alternate for charge the vacant position that the deleted PK left down. At the worst case, the process will be deleting one and moving one, so the cost can in the \( O(2) \) at the worst case.

3.2 Insertion

In contrast to deletion, insertion is more important that relating to building our synthesis that composed of a BST and an ARA; besides these, the insertion yet concerns to the function of offline manage a linear list. For example, the thread binary tree invented by J. H. Morris in 1979, that can be referred to the result in executing an offline method to obtain that list. When a set of online accessing with frequent insertions upon that tree, the offline will pay off a high price for a plenty of requirements of resorting.

What will changes happen in ARA when an insertion accomplishes in the BST? That we will survey is the key point that concerns if it in a proper tractability to us. The following proof will describe this evolution between the new item and its parent in BST, and their refs in ARA.
Lemma 1. As an item $t$ added into a BST as $t$’s child, consider their refs $\rho(t), \rho_s(t_i) \in T$, the ref $\rho(t)$ will by $\rho_s(t_i)$ at the left side or the right side in ARA.

Proof. If the fresh $t$ as left child of its parent $t_i$. If $t_i$’s LS is empty in ARA, then the $\rho(t)$ will be interpolated by $\rho_s(t_i)$ at left side, the lemma holds.

Instead, assume there is an item $\rho_{s-1}(t')$, the fresh $\rho(t)$ by it at the left side in ARA. Since the fresh is a terminal in tree, then by those corollaries above-mentioned there should be an assertion come true: $t'$ is an ancestor to $t$.

Also, the assertion will further lead to three cases about item $t_i$ and $t'$: (1) the $t'$ also is the ancestor to parent $t_i$; (2) or conversely; (3) $t_i$ and $t'$ are the same one.

If the first case holds, $t_i$ should have stayed in the right subtree that in charge by root $t'$ for $T, t_i > T', t_i$ then it is clearly a contradiction the fresh impossible as the left child of its parent since its parent in right subtree. Inversely for (2), then someone has been occupied the left link of $t_i$, maybe $t'$.

Finally the third case is truth--$\rho(t)$ by $\rho_s(t_i)$ at left side. Analogously to fresh $t$ as $t_i$’s right child, the lemma holds.

This lemma presents a clear correspondence that a sorted system with two dimensions the BST and the ARA. That not only makes our two building works concurrent on a routine to easy–an adding operation in BST also being an insertion on ARA, both are fresh by its parent; on the other hand, this lemma has exposed the affinity of two structures: the BST can carry the information over ARA. In next subsection, we will exploit this feature.

Herein, we lay the Doubly Linked List (DLL) on ARA as data structure to condition the dynamical insertion that may be caused from the frequently online. To the new data structure $T \oplus T$, we call Compound Binary Search Tree or CBST.

3.3 A Simple Query

Although ARA is a strictly sorted list by keys in ascent, yet there instantly appears to a challenge while an accessing on ARA merely with two ordinals, which try to obtain a piece-wise data like on a common sorted list whose items numbered by natural number; on account of such compact data structure and its kinds always with a fat chance in hashing or none the preferable to go.

At the aspect of maintaining a numbering system on an ARA, it never is none the easy: the incidence of point-wise renumbering in list which brought about by someone’s change may reaches all corners through the whole; especially worse still for online algorithms than you imagine. For this query, we can convert the operation of location with ordinal to calculate the position in ARA. The new model will supply the maintainability in logarithm complexity.

Of an insertion in ARA meanwhile as being adding item into BST, we learn two means to extend an ARA along axis: one is adding member at the left or right end of a present ARA; another, the fresh one interpolated between two. So we define a structure in BST.

Definition. Given an item family composed of a grandpa, a father and a grandson, we call the father Flexed Node (or FN), within their familial relationship, if and only if two links the grandpa–father and father–grandson, both in distinct sides.

For an instance: father at grandpa’s left link contrasted with grandson at father’s right link which to “replicate” the path at father; the two links can comprise a Flexed Pipe or FP; we call the left-right pattern of FP Clockwise inversely Anticlockwise.

Thus, we can observe an interest process: “Suppose an ordinal for ARA be known on grandpa, there is a visit occurs repetitively along such bearing that at parent bound for child. If this progress reflected on ARA, it can render the ordinal in cumulation or degression upon that the visiting in forward or retreat among those refs which are two distinct bearings. For this one-step the leaping does over those members in ARA, the ordinals change merely for correcting into one, we can reckon it as an invariance addressing the fixed one on numbering ordinal. Instead, on a FN, the grandson’s ref inserted among grandpa’s and father’s in ARA, the counting on ordinal should have to re-treat the one-point the grandson into one. At this time, the reckoning on ordinal must be yielded to the changes on both the tree and the queue, because once a tree rooted in the grandson, the treatment will be done with the variant maybe many that relevant to the scalar of that tree other than singleton of invariance.”

Hence, we measure the case on that replication with variable Flexion the number of items in that subtree; denoted by $T$. Of that, a progressive visit along the clue of parent-child in a tree actually evolves a leaping over an interval on that axis of ARA; the flexion can measure the thick of interval which embraced by grandpa and father.
We hence design the structure and functor in following, at first we suppose a visiting list \( \Phi = v_1, v_2, \ldots \) consisting of items that will be visited by functor \( \Phi \) and, the calculation is in the queue with ascending ordinal.

**The Structure**

1. For the root of BST, let flexion \( F \cdot \text{root} \) equal of the number of items in its left subtree.
2. The flexion on clockwise FN is of negative; positive for anticlockwise; none of the two into 0.
3. If item at the left link as child, the fixed counting for oneseft into –1, otherwise 1; herein denoted by \( S \), especially \( S \cdot \text{root} = 1 \).

The functor \( \Phi \) with visiting list \( \phi \) will start at root and assume the input ordinal is \( N \):

\[
\Phi(v_{i+1}) = \Phi(v_i) + F \cdot v_{i+1} + S \cdot v_{i+1};
\]

\[
\begin{align*}
&v_i = \text{root}; \quad \Phi(v_i) = 0 \quad \text{if } i = 0; \\
&\text{return } v_i \quad \text{if } \Phi(v_i) = N; \\
&v_{i+1} = v_i \cdot \text{Lchild} \quad \text{if } \Phi(v_i) > N; \\
&v_{i+1} = v_i \cdot \text{Rchild} \quad \text{if } \Phi(v_i) < N;
\end{align*}
\]

The search of route on functor \( \Phi \) is likewise drawn from root to someone inside a tree step-by-step without distinct difference to an ordinary query. Therefore the lower bound of cost definitely involves to \( \log n \). In addition to in ARA model, those intervals processed by functor \( \Phi \) in ARA incessantly shrinks over time, in this way the numeric can approach to the exact in the range as thin as possible till reach.

The same course can be inversed to follow the clue of child-parent applied on deletion certainly.

Because in need of the function of ordinal query on CBST, these flexions incident FNs on that route that involving to the item has been deleted must be corrected with 1 or -1. Hereby the cost on deleting operation becomes involved to \( \log n \) rather than ours above-mentioned.

We have introduced all principal operations in a CBST with FN model. Which these operations on each item, whether doing an insertion or whether doing a query or whether doing a deletion, this model can always conduct them to obtain the ordinal involved in ARA simultaneously.

On the other hand, it also maintains in a logarithm system a quick–algorithm set. Meanwhile, we solve a challenge for a compact linear list with a well responsiveness in logarithm times, however on implementation or on maintaining.

### 3.4 Building

We have got rid of the influence out from shape of BST in deletion, but the true of matter is not likelihood there for us to do some analogue things for other operations. Hence in this section we will discuss how to build a proper BST from the dimension of ARA. That conducts building a CBST is not concurrently to construct two structures other than the insertion above-mentioned instead to fabricate it, of a rather manner of industrialization.

As building a Pyramid as following pseudo code show over there assume \( n = 2^k - 1 \), algorithm recursively extracts items from ARA as parents to connect with their children that have been reserved down.

**Construct BST**

```cpp
/* Parameters */
\( \delta = \xi = 2; \quad \text{// the cursors, } \delta \text{ backup } \xi \)
\( \theta = 4; \quad \text{// offset for next extracted one} \)
\( \kappa = 1; \quad \text{// the width between parent & children} \)

/\* ** Building Module * * *\*/
01. Loop(\( \xi < n \))
02. \( dl = \xi - \kappa; \quad dr = \xi + \kappa; \quad \text{// addresses for left & right} \)
03. \( T(\xi) \cdot \text{Link} \approx T(dl); \quad T(dl) \cdot \text{Parent} \approx T(\xi); \)
04. \( T(\xi) \cdot \text{Rlink} \approx T(dr); \quad T(dr) \cdot \text{Parent} \approx T(\xi); \)
05. \( \xi \leftarrow \xi + \theta; \)
06. if \( \xi < n \) and \( \eta < n/2 \)
07. then \( \kappa = \delta; \quad \delta = \xi = \theta; \quad \theta \leftarrow 2\theta; \quad \text{// start next round} \)
08. return the tree \( T \).
```

In this algorithm, the process can swift convert the roles for items from parent to child. In every round, algorithm is equivalently to execute this converting module on an abstracted bed against the previous results. In this pattern, the primitive bed is ARA and, only the items on event–position in ARA can be permitted to participate.

For example, in the first round, initially these items at 2nd, 6th, 10th, \ldots, \((n - 1)\)th positions are involved in a new list as parents, where with offset the argument equals of 4 to pick up parents; on the other hand, with width the argument equals of 1, the odds are picked up as children.

Against the new sequence of parents, the second round will do the similar performance which chooses
the 4th, 12th, 20th, . . . , (n − 3)th in ARA to as parents by offset counted of 8, upon that those parents in first round, 2nd, 6th, . . . , now become children pointing to new parents, in which the width argument equals of 2 always half of offset argument. In this way, the course will terminate when the middle item at the 2\(^{k-1}\)th cups the rising pyramid as chief root to whole, which inverses the course of existing building method that begins at root of BST.

As to ARA as a sorted linear list, the parameter \(\kappa\) at 02th step can utilize this feature to control the selection of children—left element can but as left child, right child as well as left one; of that, this measure guarantees those keys comply the regulation about parent and children.

It wants us to answer a question about the detail of this kind of fabrication: There is any foul about each key in parent-child pattern, but not equivalent to in ancestor-descendent one. For example, one key in right subtree but smaller than the root’s by which that subtree in charge. The cause is the foul member at that position in ARA ahead of the root but selected as child by a lower descendent in that subtree. We so far cannot confirm if our tactic of picking children is reliable to avoid this foul, specifically on using parameter \(\kappa\). In [5], Robert Sedgewick et al ever mentioned this kind of error.

The proof is simply to prove the extremum in a tree, its ref impossibly outs the cordon the position at which the ancestor stays. Hereby, we will only discuss the case–ancestor versus its right subtree, thus it requires us prove the leftmost item in subtree that refs position in ARA always at its ancestors right side.

Given a \(j\)th round (for \(j > 1\)), and \(t_j\) as parent in this round, so it with a right width \(d_R(j) = 2^j-1\) for \(t_j\) to pick up right child. Suppose a tree is rooted in \(t_j\)’s right child \(t\) and, thus there is a path containing \(j – 2\) links to catenate the left extremum and the root that right child \(t\).

To measure the parameter \(\kappa\)’s change on every link in that path whose scalar can be quantified with the exponent on \(d_R(j)\) more than \(d_L(j - 1)\) for one degree. That means the foul is impossible in PM.

Meanwhile the proof does matter in another critical quality in PM that algorithm cannot pick up any item as parent or child repeatedly; because in very round, the candidates for in roots always outside those present trees that have been built up.

Of course, there is most likelihood to \(n ≠ 2^k – 1\) in practice; herein we can read it as \(n = m + m'\) for \(m = 2^k – 1\) and \(n/2 ≤ m ≤ m\); thus the “overplus” \(m'\) smaller than \(m\). It is clear that these overplus ones can be settled at the most bottom as children before proceeding PM. The shape only just depends on user’s tactic to arrange the positions for them maybe for balance. Anyway, the depth of CBST can be \(|\log n|\) at all.

It is certainly that the PM works on a sorted sequence. Hereby we will introduce a sorting algorithm which developed from Tournament Method that has been introduced in [6] whose complexity has been known in \(O(n \log n)\). We reformed it for condition the data structure the DLL, called Card Game Sorting Method or CGSM.

The pseudo code CGSM (1) about the engine is in following.

**CGSM (1)**

Function: Insert(x, t) // inserting in DLL, s precedes t
Function: Follow(s, t) // t follows s in DLL
/* * * Merger Method * * * */
Function: Merge(x, y) // x ∈ X; y ∈ Y; the heads of queues
01. \(H = x;\) if \(K_x > K_y\) then \(H = y;\) // elect the new head
02. \(\text{Loop} (y ≠ \varnothing)\) // not out the range of Y
03. if \(K_x < K_y\)
04. if \(x\) at the end of X
05. \(\text{then} \) Follow(x, y); break; // follow x, the y & RS
06. \(\text{else} \) y := x.Later; // continue on X.
07. \(\text{else} \) Insert(y, x); y := y.Later //insert & continue on Y
08. return \(H;\)

The sequence \(Y\) can as well as sorted heap in ascent where the member that with the min key among all always springs out from the top of heap; for the outside member \(y_j ∈ Y\), Merger Method attempts to seek out an appropriate interval in sequence \(X\) for inserting it by moving the cursor in sequence \(X\), whose process as sorting cards in card game.

\(^1\) The source code and files involving to test this algorithm has been hosted in this website: https://github.com/smartchampion/CGSM which encoded by C++ and executed in console platform.
Which maintains the kernel logic, by a pair of two neighbors the $x_{i-1}, x_i \in X$ and member $y_j \in Y$ commonly carry the inequality $Kx_{i-1} < Ky_j < Kx_i$ about three keys of theirs.

If functor to the end of $X$ and $Y$ no empty, then the rest subsequence in $Y$ would join to the right end of new $X$ together to compose the new sequence $X \cup Y$; than whole course end. Conversely none springing out $Y$ for search either leads to procedure terminate likewise.

Complexity. Let $|X| = t$ and $|Y| = s$, we can describe the process of comparisons by a set $Y'$ as:

$$Y' = \bigcup_{i=1}^{s} y_i \times X_i: X_i = x_k, \ldots, x_j \text{ for } 1 \leq k, j \leq t.$$ 

A Cartesian product in favor of present the detail on each member $y_i \in Y$. If a member $y_i \in$ inserted in sequence $X$ and ahead of member $x_j \in X$, then for shift to next member $y_{i+1}$, the proceeding will start at $x_j$. Thus there is a nonempty intersection $|X_i \cap X_{i+1}| = 1$, of that at worst case it is equivalent to $(s+t)$ times of comparisons implemented on this algorithm which tallies the sum of scalars of two sequences the $X$ and the $Y$; therefore the complexity can in $O(n)$ where $n = |X| + |Y|$; likewise, each member in that two also may be consider as equivalently being invoked for precisely once.

To a random sequence with $n$th members, certainly, by a way to scan the sequence through, it is easy on the level of procedure to yield a group of components within the sequence where each with a sorted subqueue. The sorting job eventually becomes a multi sequences merging. The pseudo code CGSM (2) in following, we only relate another means that has the treatment on each member in sequence as a singleton set in order to exhibit the course of merging multitude of components based on the pairwise:

```plaintext
/* Parameters */
Θ; k = |Θ|; // store the heads of subsequences.
s = 1; π = 0; // two cursors in Θ

/* Merger Rounds */
Loop (k > 1)
01. π + +; Θ[π] := Merger(Θ[π], Θ[s + 1]); s := s + 2
02. if s = k then Θ[π] := Θ[π]; // backup the last if k is odd
03. if s > k then k = π; s = 1; π = 0; // start the next round
```

It is easy to count of $|\log n|$ rounds for whole merger and, moreover by the analysis above-mentioned, the cost can in $O(n \log n)$.

Of course in this way, many CBSTs merger can be looked like a process of CGSM, the cost for BSTs merger can be in $O(n \log k + n)$ where variable $k$ is the number of trees. This means let we gets rid of much more annoying troubles that results from a mass of relations intertwined by tree’s shape.

In addition to about concurrently building a FN system in PM, we give a solution for a module in algorithm which in following.

```plaintext
/* Structure */
t \in T; t.ℓ = 0; t.r = 0; // two counters for left & right.

/* Module */
01. As Parent then record the number of items in subtree.
02. t.ℓ = λ.ℓ + λ.r + 1; // λ is the left child.
03. t.r = ρ.ℓ + ρ.r + 1; // ρ is the right child.
04. As Child if K.ℓ < K.π /\ 1 at the parent π left link
05. then F.t = (-1) \ast v.r; S.t = -1; // clockwise
06. otherwise F.t = v.ℓ; S.t = 1; // anticlockwise.
```

There is another alternative, the building course may begin at the median position in ARA in manner of top to bottom completely, by recursively bisection sequence to work out the BST. Here we don’t intend to have a length to introduce, over there it may occasion the tree depth in $|\log n| + 1$.

4 Bath of Works

It is apparently that we have reduced more trivial and unnecessary steps in our Merger method contrasted with merge sort introduced in [5]. Hence we naturally propose a theme the batch of works on CBST since the ARA also as another dimension supporting BST where they might in equivalence. Thus we can look the algorithm on two sequences in the manner of many-many other than the traditional method which everyone in a fixed sequence in turn invoked for accessing on the whole BST, the character of one-many.

Of course, in some occasion, such as one-time locking database for a bunch of jobs may spare much more resources than many-time ones. As a theoretical discussion, we merely reduce the issue simply to the preferable or not by the way of complexity analysis which is applicability. So we aim that: (1) Mark off the boundary for two methods, the batch and the traditional if instance in proper shape. (2) What is the index in a BST? By this guide we learn which alternative is more preferable with tree being inharmonic. (3) Analyze the instance of locality of accessing.
Here we only discuss the case of query inasmuch with others they are similarly one another. We firstly let sequence $X = x_1, \ldots, x_\lambda$ as an ARA; otherwise, refer sequence $Y = y_1, \ldots, y_\lambda$ to as the query sequence.

Then we add a module in Merger method and have a bit of reforming. The routine will process a success hit if referred to a query in BST with a ref, the ref succeeds to match the item inside that tree; now here a member in sequence $Y$ instead of that ref, it matches a member in set $X$ as well as $\mathcal{K}.x_i = \mathcal{K}.y_j$. Functor will return yes to sound the success hit.

Conversely, with no match and concurrently $\mathcal{K}.x_i > \mathcal{K}.y_j$ come true, it states the member $y_j$ failure for match; no hit happen. These additions have barely to increase the overall complexity nevertheless have added a conditional for execution.

Let $\kappa = \lambda \eta$ (for $0 < \lambda \leq 1$). In the case of sequence $Y$ been sorted in ascent, the cost of the batch could be in $(1 + \lambda)\eta a$ approach to $O(2n)$. 

Contrast to the traditional in the lower bound $\lambda n \log n$ with members in set $Y$ in turn for query on BST; if $\lambda n \log n \geq (1 + \lambda)\eta n$, we have boundary $\lambda \geq \log^{-1} n/2$ such that

$$\lambda = \rho^{-1} \text{ for } \rho = [\log n].$$

If the scalar of set $Y$ beyond, the batch is worthwhile.

Consider plus a sorting on sequence $Y$ in $\kappa \log \kappa$, as the measure with a delicate difference to $\kappa \log n$ the traditional, our task hence has to be altered to estimate the shape of BST.

Assume the cost is $\lambda \eta h$ of using traditional query in CBST where $h$ is the depth of tree. Thus an inequality

$$\lambda \eta h \geq (1 + \lambda)\eta n + \lambda n \log \lambda n = n + \lambda n \log 2\lambda n.$$ 

Eventually $h = \lambda^{-1} + \log 2\lambda n$. Consider $g = h - \log n$ then $g = \lambda^{-1} + \log 2\lambda n - \log n$.

Such that $g = 1 + \lambda^{-1} - \log \lambda^{-1}$. For $0 < \lambda \leq 1$, having $\log \lambda^{-1} \ll \lambda^{-1}$ so $g > 0$, means the existence of the depth $h$ in a CBST can as an index to measure that tree. For example, $\lambda = 1/4$ then $h \geq 4 + \log(2n/4)$, further if $h \geq \log 8n$, we can execute the batch.

It is interesting that if two extremums with extreme keys in sequence $Y$ in use to reduce a subqueue in ARA for query, where is boundary?

At first, the variable $\theta n$ (0 $\leq \theta \leq 1$) represents the scalar of that queue locked up in ARA, then the following equation marks out the boundary which has involved the cost of sorting on sequence $Y$

$$\lambda n \log n - \lambda n \log \lambda n = (\theta + \lambda)n.$$ 

Where the cost for the queries that with two extreme keys for lock a field inside a proper CBST, it can be considered to be ignored as negligible quantity.

Thus we have the boundary $\theta = \lambda \log(2\eta)^{-1}$. When $\lambda = 1$ the $\theta$ into null, the batch none the worthy; when $\lambda = 1/2$ the $\theta$ into $0$ that means the traditional still worthwhile; if $\lambda \leq 1/4$, then $\theta$ always larger than or equal of $\lambda$ by a constant-fold. The boundary indeed is rather volatile.

5 Summary

It is interest to develop a common data structure BST into a synthesis. In fact, more and more features covert in the CBST model needs us to reveal, such as maintain a proper BST at a lower cost or further reform the structure for special purpose for clients. On the other hand, we have improved the situation of poor performance at worst case on BST better than before to serve database management.

References

[1] Knuth (1997). The Art Of Computer Programming, Volume 1, 3/E. Pearson Education.

[2] Kenneth Rosen (2011). Discrete Mathematics and Its Applications 7th edition. McGraw-Hill Science. pp. 352353.

[3] Jyrki Katajainen and Jesper Larsson Traff (1997). A meticulous analysis of mergesort programs.

[4] Morris, Joseph H. (1979). Traversing binary trees simply and cheaply. Information Processing Letters. 9 (5).

[5] Robert Sedgewick, Kevin Wayne (2011). Algorithms Fourth Edition. Pearson Education, p.396-410.

[6] Donald Knuth (1973). The Art of Computer Programming, Sorting and Searching, Volume 3.