Analysis of Forest of Hashed Exponential Trees

Nikita
Lovely University
Phagwara

Neha Arora
Thapar University
Patiala

Puneet Kumar
Lovely University
Phagwara

ABSTRACT

Exponential Tree in the form of forest is proposed in such a manner that- (a) it provides faster access of a node and, (b) it becomes more compatible with the parallel environment. Empirically, it has been show that the proposed method decreases the total internal path length of an Exponential Tree quite considerably. The experiments were conducted by creating three different data structures using the same input- a conventional binary tree, a forest of hashed binary trees and a forest of hashed exponential trees. It has been shown that a forest of hashed exponential trees so produced has lesser internal path length and height in comparison of other two. It also increases the degree of parallelism.

General Terms

Algorithms, Parallelism, Traversing.

Keywords

Exponential Tree, Binary Search Tree, Internal Path Length, Parallel Processing, Balanced Tree

1. INTRODUCTION

The exponential tree was first introduced by Andersson in his research of Fast deterministic sorting and searching in linear space [1]. In exponential Tree, the number of children increases exponentially. It is almost identical to a binary search tree, with the exception of dimension. The dimension of exponential tree is not the same at all levels. In a normal binary tree, each node has 2^d children with a dimension of 1. In an exponential tree, the dimension equals to the depth of the node with a root node having a dimension equals to 1. Root node at level 1 holds two children (2^1), each node at level 2 holds four children (2^2), each node at level 3 holds 8 children (2^3) and so on. Therefore, the second level can hold two nodes, the third level can hold eight nodes and fourth one can hold 64 nodes and so on.

Anderson has shown that if integers are passed down in an exponential tree one by one than insertion takes O (√logn) for each integer [1]. This is improvement over the result given by Raman which takes O (n√lognloglogn) expected time [2]. Yijie Han has given an idea in Deterministic sorting in O (n log logn) time and linear space [3]. In this, complexity is reduced to O (nloglogn) expected time in linear space. The technique used by him is that, instead of inserting one integer at a time in an exponential tree as done by Anderesson [1], he passed down all integers at one level of the exponential tree at a time. This idea may provide a speedup, but in practical implementation it is difficult to handle integers in the form of batches.

Later on, Ajit Singh presented a way to implement exponential trees [4]. Modified concept of exponential trees has been used to implement the sorting as it is difficult to handle the pointers in actual Anderesson's exponential tree [1]. The modified exponential tree has following properties:

1. Each node at level j will hold j number of keys.
2. Each node at level j will hold j+1 children.
3. All the keys in every node must be sorted.

As far as concurrent processing is concerned, considerable work has been done to develop concurrent algorithms, refer to [5]-[11]. In a tree, root is the only gateway; it makes it difficult for all the active processes to achieve maximum parallelism. No matter how optimal our concurrent algorithms are; other active processes have to wait until the previous releases the lock. Ellis [9], [10] presented ways for concurrent searches and insertions in the AVL and 2-3 trees. Most of the presented solutions use some kind of locking scheme to allow concurrent processing on a single tree. The common goal is to increase the degree of concurrency. It is achieved by having a lesser portion of the tree locked and thus available a major portion to the rest of the active processes. These algorithms can increase the degree of concurrency up to a considerable extent. However, results could be better if the underlying data structure is modified to support large number of processes. Substantial work has been carried out on algorithms, but hardly an attempt has been made to create an optimal data structure.

In this paper, we propose a forest of hashed exponential trees in order to provide better compatibility with the concurrent environment. To check the overall balance

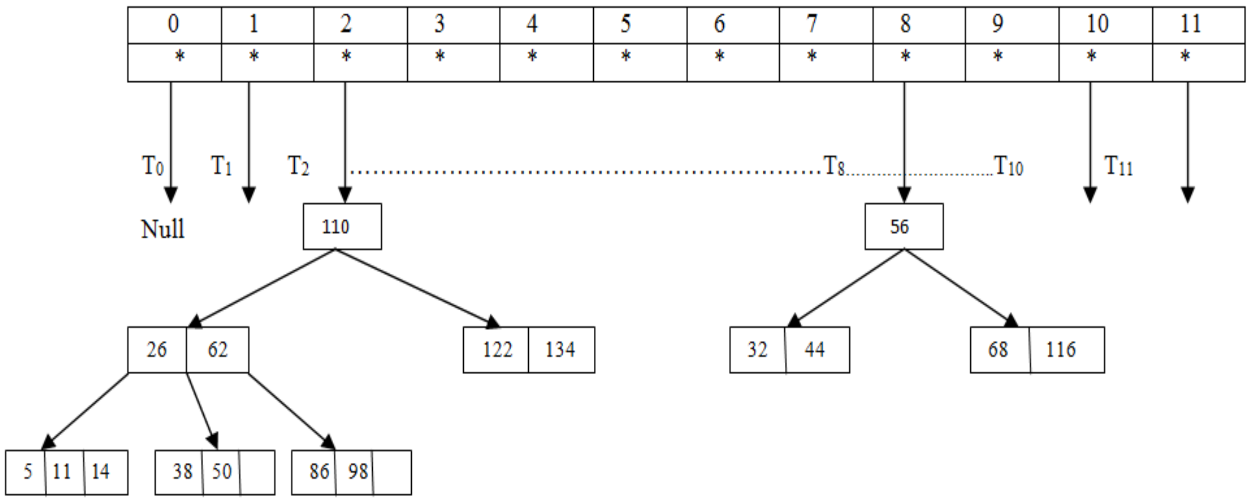

Fig 1: A Forest of Exponential trees

Of the proposed forest, parameters used are internal path length (IPL) and the height of the tree. The height of the tree is defined as the length of the longest path from the root to the leaf.IPL is defined as the sum of the depth of all the nodes in the tree

2. CREATION OF FOREST

A random exponential tree can be maintained in the set of trees called ‘forest’. The number of exponential trees in the forest would depend on the application and requirement. For the purpose of simulation of simulation, we have used a forest of 10 trees. There will be a multiple roots to hold multiple trees. An array of pointers will be there of size k=12. Each cell of the array acts as a root of tree. In case the tree is empty, array cell points to a null value (refer to figure 1). In order to find out the location of a key in the array, hash function is used. A Key to be inserted or deleted is first divided by the size of the array and its remainder will provide the location of that key in the array. Therefore, hash function used is given by: loc=key % k. Where ‘loc’ is the location of the array to/from which the key has to be inserted or deleted. In this way, we get a set of k possible exponential trees, which resembles a forest, but acts as a single exponential tree. The only difference between a single exponential tree and a forest is determining the tree location using hashing. In figure 1, the forest of two exponential trees is shown. The collective Internal Path Length of the forest = the sum of the IPLs of all the trees in the forest = 18+4= 22. Maximum height of the tree is 2.

3. METHODOLOGY

The same random input is used to construct the three data structures namely a conventional tree, a forest of Hashed binary trees and a forest of hashed exponential trees. A random number is generated for every insertion which serves as an input to three different algorithms: one creates a conventional tree that is also perfectly balanced, second one constructs an exponential tree and the third one creates a forest. Duplicates are avoided. The Internal Path Length (IPL) of all the three structures is calculated. For a Conventional tree, IPL is calculated by summing the depths of all nodes in the tree. In the case of a forest whether Exponential or Binary, the IPL is calculated by adding all the IPLs of the individual trees in the forest [4].The heights of all the trees in the forest as well are recorded. For example, the worst case height of tree in the above given forest is 2 and collective IPL is 18+4=22.The array size is 12 i.e. roughly equal to 1% of the input size i.e.1023.

4. RESULTS

The parameters used are as follows:

Average Height of forest = Sum of all twelve trees in the forest/12.

Height of worst tree in the forest = A tree with maximum height in the forest.

Collective Path length of forest = the sum of path lengths of all 12 trees in the forest

Table 1 is obtained by conducting several tests under Borland C++ compiler 5.5 for windows and GNU C++ compiler 3.4.3(g++) under Linux [4],[12].This table shows the result for n=1023 and k=12.the average height of exponential trees in a forest was roughly equal to 5,which is quite close to perfectly balanced exponential tree. On comparing the average height of the exponential trees in forest there is a whopping reduction of 90% as compared to conventional tree and a reduction of almost 50% when compared with forest of hashed binary trees. Though comparing both the forests with the conventional tree is not justifiable because the number of nodes changes randomly and is not the same in all the three structures. However, we get the worst case behavior of the structure through this analysis. And the result so obtained is more than considerable .The worst case IPL of exponential tree is 1715 as compared to 12174. IPL (I_n) is related to the number of comparisons as follows:

$$C_{an} = I_n/n+1 \qquad (1)$$

For Conventional tree, the number of comparisons is (12174/1023) + 1 = 12.9 approximately, for forest of hashed binary trees it is (6623/1023) +1=7 and for forest of exponential trees the comparisons are (1715/1023) +1=3. Hence, it is quite obvious that the forest of exponential trees require less number of comparisons. Therefore, behavior of exponential tree forest is far better than conventional tree and forest of hashed binary trees

5. TIME AND SPACE REQUIREMENTS OF FOREST

A random BST with n nodes requires log (n) time for almost all operations on an average. But forest requires an additional effort of hashing so an overhead of order O (1) is added to jump to correct tree. Therefore, total time required in forest is of order O (log (n/k)) +O (1). However the case is entirely different for hashed exponential tree forest. A random exponential tree requires log (log (n)) time for all operations on an average which implies O (log (log (n)) +O (1) time for forest of exponential trees [11]. The space whatever is used is in the form of Array which does not have any overhead. Overhead occurs when restructuring and balancing is needed in trees as the random BST or exponential trees have been assumed to be perfectly balanced [5].

6. PARALLEL PROCESSING OF FORESTS

The forest of trees is compatible with the concurrent environment. Each tree can be operated independently and it is quite obvious that a forest allows more number of processes to act upon .There is no need of any proof or demonstration without requiring use of any locking schemes [6],[8]. It can be concluded that by maintaining the trees in this way increases the degree of parallelism by k times which is of the order of size of array. For huge data sets and massively parallel systems, large number of trees can be used, requiring large array size. For better results, tree restructuring techniques can be used. In case of global restructuring, entire tree is taken as input and restructured. However in case of forest, only a part of forest needs to be restructuring. It can be done with series of other operations like insertions or deletions [10].It has been proved that a series of such operation leads to O (nlg^3 (n)) increase in size of IPL, though forests are immune to such problems

.

Table 1: Comparison between conventional tree, forest of hashed BST and forest of hashed exponential trees

Sample Run	Conventional Tree		Forest of Hashed Binary Trees			Reduction (%)		Forest of Hashed Exponential trees			Reduction (%)	
	Height	Path Length	Average Height	Worst Tree Height	Collective Path Length	Height	Path Length	Average Height	Worst Height	Collective Path Length	Height	Path Length
1	18	11018	10.8	12	6193	40	44	3.5	5	1574	72	86
2	20	10782	12.0	14	6594	40	39	3.7	5	1715	75	84
3	21	11299	11.6	13	6623	44	41	3.2	4	1660	80	85
4	24	13728	12.0	17	6542	50	52	2.3	4	1349	83	90
5	21	11840	11.1	14	6081	47	48	2.9	4	1462	81	87
6	22	11158	12.2	15	6567	44	41	3.5	4	1659	81	85
7	20	11185	11.4	16	6467	43	42	4.5	5	1630	75	85
8	20	10814	12.2	17	6533	39	40	3.7	5	1687	76	84
9	22	11359	12.0	16	6481	45	43	2.6	4	1602	82	86
10	24	12174	11.4	13	6414	52	47	2.8	4	1490	83	87

7. CONCLUSION AND FUTURE WORK

We have concluded that a forest of exponential trees shows better results than that of conventional trees as well as a forest of hashed binary trees. A tree can be converted into a forest with reduced IPL and good compatibility with concurrent environment, without compromising with structural information. It has been shown in results that the height of forest of exponential trees is smaller than that of forest of binary search trees. It provides more optimal searching. Time Complexity in case of the forest of Exponential Tree is less than that of Binary Search Trees. In nutshell, small modifications in data structure leads to faster node access and higher degree of parallelism.

In future, this forest can also helps in the field of sorting in a significant way. As amount of data is increased, complexity of sorting algorithm also gets increased significantly. But this forest can provide better results in field of sorting also. Forest has to be traversed in such a manner that result will provide a list of sorting elements. A way to traverse the forest needs to be found, so that parallel processing can contribute in the field of sorting also.

8. REFERENCES

[1] A. Anderson "Fast deterministic sorting and searching in linear space", IEEE Symposium on Foundations of Computer Science, 1996.

[2] R. Raman "Priority queues: small, monotone and trans-dichotomous" in European Symposium on Algorithms, 1996.

[3] Y. Han. 2002 "Deterministic sorting in O (n log logn) time and linear space" in 34th STOC.

[4] Ajit Singh, Dr. Deepak Garg "Implementation and Performance Analysis of Exponential Tree Sorting ", International Journal of Computer Applications, 2011.

[5] Samadhi "B Trees in System with Multiple Users" in Information Processing Letters, 107-112, 1976.

[6] Eswaran, K. P. , Gray, J.N Lorie, R. A. and Traiger, I.L "The notions of Consistency and Predicate Locks in a Database System" in Communication of the ACM, 1976.

[7] Bayer, R, Schkolnick "Concurrency of Operations on B Trees" in Acta Information, 1977.

[8] Ries, D.R., Stonebreaker "Effects of Locking Granularity in a Database Management System" in ACM Transaction on Database Systems, 1977.

[9] Ellis, C.S. "Concurrency search and insertion in AVL trees" in IEEE Transactions on Computers, 1980.

[10] Ellis, C.S. "Concurrency search and insertion in 2-3 trees". Acta Information, 1980.

[11] Kung, H. T., Lehman, P.L. "Concurrent manipulation of binary search trees" in ACM Transaction on Database Systems, 1980.

[12] Knuth, D. E. "The Art of Computer Programming" in Pearson Education, Vol.3, Searching and Sorting, 2005.

[13] Y. Han, M. Thorup. "Sorting integers in O (n√loglogn) expected time and linear space" in IEEE Symposium on Foundations of Computer Science, Vol-43, 2002.

[14] Day, A.C. "Balancing a Binary Tree" in Computer Journal, 1976.

[15] Stout, F, Bette, L.W. "Tree Rebalancing in Optimal Time and Space in Communication of the ACM", 1986.

[16] S. Alberts, T. Hagerup. "Improved parallel integer sorting without concurrent writing in Information and Computation, 1997.