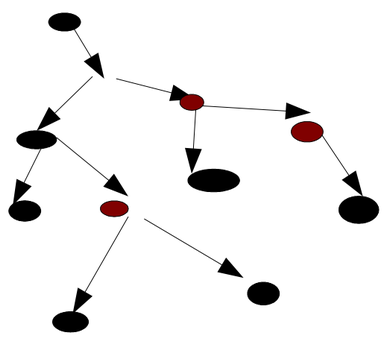
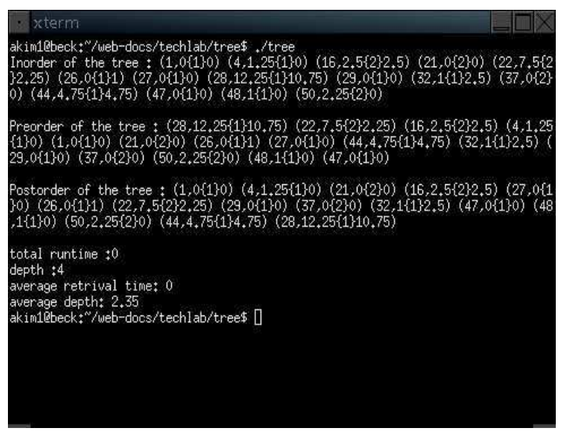
A Study of Balanced Search Trees: Brainstorming a New Balanced Search Tree

Anthony Kim, 2005
Computer Systems Techlab

Abstract

This project investigates four different balanced search trees for their advantages and disadvantages, thus ultimately their efficiency. Runtime and memory space management are two main aspects under the study. Statistical analysis is provided to distinguish subtle difference if there is any. A new balanced search tree is suggested and compared with the three balanced search trees. Balanced search trees are implemented in C++ extensively using pointers and structs.





Background

The order of data input is important to the structure of a binary search tree (or general search tree). In a optimal binary tree, the data are input so that insertion occurs just right which makes the tree "balanced," the size of left subtree is approximately equal to the size of right subtree at each node in the tree. In an optimal binary tree, the insertion, deletion, and search function occur in O(log N) with N as the number of data in the tree. This follows from that whenever data comparison occurs and subsequent traversal (to the left or to the right) the number of possible subset divides in half at each turn. However that's only when the input is nicely ordered and the search tree is balanced. It's also possible that the data are input so that only right nodes are added. (Root -> right -> right -> right ...) It's obvious that the search tree now looks like just a linear array. And it is. And this give O(N) to do insertion, deletion and search operation. This is not efficient. Thus balanced search trees are developed to perform its functions efficiently regardless of data input.

Procedure & methodology

Evaluating binary search trees can be done in various ways because they can serve number of purposes. For this project, a binary search tree was developed to take some advantage of random nature of statistics with some assumption. Therefore it is reasonable to do evaluation on this basis. With this overall purpose, several behaviors of balanced search trees will be examined. Those are:
(1) Time it takes to process a data set
(2) Average time retrieval of data
(3) Height of the binary tree
(4) Average retrieval depth of data

```
#include<fstream.h>
#include<iostream.h>
#include<stdlib.h>
struct NODE
{
  int key, freq;//key value, frequency counter
  double leftw, rightw;//weight value calculated based on R-S weighting
  int lefti, righti;//interval left max and right max
  NODE *left, *right, *parent;
};
NODE *ROOT;
int DEBUG=0;
int N;
double startT, endT;
double rstartT, rendT, rtotal;
long D=0, array[2000000];
const double threshold = 2.0, R=5.0, S=4.0;
void LeftRotate(NODE *pivot);
void RightRotate(NODE *pivot);
double Mix_insert(NODE *node, int newkey);
double Mix_insert_fix(NODE *node);
void inorder(NODE *current);
void preorder(NODE *current);
void postorder(NODE *current);
int search(NODE *current, int target);
int max(int a, int b);
NODE *create_node(int newkey);
void print(void);
void find_D(NODE *current, int d);
int check(NODE *current, long value);
void find_D(NODE *current, int d);
int check(NODE *current, long value, int d);
int main()
{
```

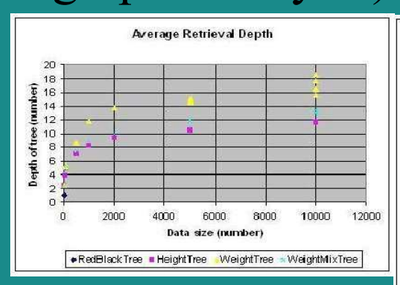
Median-weight-mix tree

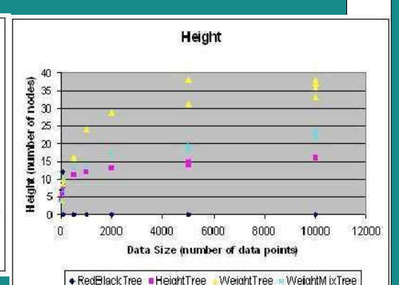
Median-weight-mix tree probably serves no theoretical purpose because it's not perfect. It has no well defined behavior that obeys a set of properties. Rather it serves practical purpose mostly likely in statistics. Median-weight-mix tree is based on following assumption in data processing:
(1) Given lower bound and upper bound of total data input, random behavior is assumed.
(2) Multiple "bells" is assumed to be present in the interval.
Algorithm:
Each node will have a key (data number), an interval (with lower and upper bounds of its "assigned" interval) and weights of left subtree and right subtree. The weights of each subtree are calculated is based on constants R and S. Constant R represents the importance of focusing frequency heavy data points. Constant S represents the importance of focusing frequency weak data points. So the ratio R/S consequently represents the relative importance of frequency heavy vs. frequency weak data points. Then tree will be balanced to adjust to a favorable R/S ratio at each node by means of rotating, left rotating and right rotating.

Results (data table/graphs/analysis)

Average Retrieval Depth

Test Case	Redblack	Height	Weight	MWM
1	3.5	2.35	2.35	2.35
2	5.68	3.9	5.08	4.42
3	N/A	4.85	5.44	5.31
4	N/A	7.01	8.68	7.42
5	N/A	8.22	11.82	8.82
6	N/A	9.24	13.72	9.96
7	N/A	10.38	14.67	12.02
8	N/A	10.48	14.52	11.65
9	N/A	10.47	15.16	11.65
10	N/A	11.57	18.59	13.22
11	N/A	11.59	15.68	13.02
12	N/A	11.65	16.43	13.12
13	N/A	11.67	16.57	13.37
14	N/A	11.54	17.77	13.13
r1	49.03	4.87	8.51	5.19
r2	18.56	4.4	4.55	4.74
r3	N/A	0	0	N/A
r4	2368.98	1.69	2.7	2.63





It is interesting to note that each tree shows logarithmically increasing curve as the data size increased. This logarithmic trendline was expected because the height of a perfect binary search should be lg(n). This characteristic was proved rigorously using induction for red-black tree. Height-balanced tree and weight-balanced tree should yield logarithmic curve because they are self-balancing trees after all. Self-balancing allows trees to obtain the optimal structure with "filled" branches and thus all nodes closer to the root. Median-weight-mix tree also shows logarithmic curve. Though its special R/S ratio makes it deviates from its mother the weight balanced tree. It seems to stabilize the original weight-balanced tree, interestingly. This is probably caused by R/S ratio. I think R/S ratio skews the data points little bit so that in a way it behaves like interval balanced, or height balanced. As result, the logarithmic curve of median-weight-mix tree lies between the logarithmic curve of weight-balanced tree and the curve of height-balanced tree. The red black tree should give some insight into this analysis, but unfortunately, the red black tree fails to run on large test cases due to a seemingly minor bug in the program.

As mentioned earlier, average retrieval depth is proportional to average retrieval time, assuming constant time between each recursive steps. Fortunately, the average retrieval depth data is more readable than the average retrieval time. It's interesting that the average retrieval depth data shows logarithmic regression line. This is not obvious, but probably the fact that height of search tree follows logarithmic curve somehow forces the average retrieval depth to behave similarly. The average retrieval depth data also supports the observation made based on the height data. The median-weight-mix tree's curve lies between height-balanced tree's and weight-balanced tree's.

Conclusion

This independent study of four balanced binary search trees yields some insight into the data structure. First, it has experimentally verified that balanced search trees' logarithmic characteristics. The logarithmic characteristics include the height of the tree and the average retrieval depth. Although, no reliable experimental data was produced, but the average retrieval time and the total run time are also expected to behave logarithmically.

Another observation is that height-balanced tree performs better than weight-balanced tree in almost all aspects. I have an explanation for the unexpected result. I think the answer lies in the recursive functions of the two search trees. Even though both are structured in the same way, recursive balancing function of weight-balanced tree is cumulative. This is because weight of each node is the sum of the weight of left subtree, the weight of right subtree plus one. On the other hand, the recursive balancing function of height-balanced tree is somewhat conservative. The height at each node is the greater of heights of left subtree and right subtree plus one.

The median-weight-mix tree showed an interesting performance. Though with specialized weighting system, the tree nevertheless shows logarithmic characteristics like red-black tree, height-balanced tree and weight-balanced tree. This experimentally proves that median-weight-mix tree is one of many of balanced search trees. More interestingly, the median-weight-mix tree's logarithmic characteristics lies between height-balanced tree's and weight-balanced tree's.

More studies can be done on balanced search trees. One suggestion is to expand the study to other balanced search trees such as B-tree etc. Also most test cases of various type can be generated and tested to give more accurate result. One thing I wanted to do but did not have time to is varying R/S ratio. Varying R/S ratio changes median-weight-mix tree and ultimately its performance. Varying R/S ratio is probably linked to logarithmic characteristics of the balanced search tree. It is possible that varying R/S ratio will change the logarithmic curve in the range defined by height-balanced tree's curve and weight-balanced tree's curve.