Homework #3

Deadline 11:59pm Saturday 23 November 2013

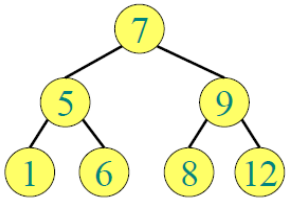
*Please present a **printed** report with all your answers, explanations, and sample plots. Submit also a soft copy of the source code and binaries used to generate these results. Please note that copying of any results or source code will result in ZERO credit for the whole homework.*

Problem 1

Describe a $\Theta(n)$-time algorithm that, given a set S of n integers and another integer x, determines whether or not there exists two elements in S whose elements sum to x. Analyze its running time to verify it takes $\Theta(n)$.

Problem 2

Implement a C++ function that, given a binary tree, will print out the tree with one line for each level, with nodes on the same level separated by a space. For example, the tree:



should be printed as:

```
7
5 9
1 6 8 12
```

Assume the node structure is defined as:

```
struct Node {
  Node *left, *right;
  int key;
}
```

You can use any data structure from the STL if you want to use any. Describe the running time and the space requirements of your algorithm.

Problem 3

1. Write pseudo-code for the `Tree-Predecessor` procedure.
2. Show that if a node in a binary search tree has two children, then its successor has no left child and its predecessor has no right child.

Problem 4

You are given a set of n **static** keys that need to be inserted in a hash table for fast search. You are required to implement a hash table, in C++, that supports search operations in constant time in the worst case. Use the universal hash functions that are defined by $h_{ab}(k)=((ak+b) \bmod p) \bmod m$ for $a \in (1,2,..,p-1)$ and $b \in (0,1,...,p-1)$ for some large prime number p. Set $m = n$ for all experiments below.

1. Implement the perfect hashing scheme described in the lectures (and textbook). Note that the size of the secondary hash table for each slot $m_i = n_i^2$.
2. Try inserting n random keys into the hash table for n = 1000, 10000, 100000. For each n, search 1000 keys (500 that are already inserted and 500 that are not). Plot the average search time for each n with n on the x-axis.
3. Repeat (1) and (2) for a normal hash table, with collisions resolved by chaining i.e. linked lists instead of secondary hash tables, with new keys inserted at the head of the lists. Compare the average search time as a function the number of keys in the hash table to that using perfect hashing. [Note: you can use a ready made implementation of a linked list, e.g. from the C++ STL].

Problem 5

Implement a Red-Black Tree (RB-Tree) data structure in C++. Specifically, you should implement *insertion* and *search* functions. For simplicity assume that the tree deals only with *integers* i.e. the data items are themselves the keys and they are all integers.

Perform the following experiments with your implementation:

1. Starting with an empty tree, insert random n integers into the RB-Tree of sizes 10^3, 10^4, 10^5, and 10^6. For each input size n, search for a random 500 integers. Repeat 5 times for each n and plot the search average search time (over the 5 runs) versus the input size n.
2. Repeat (1) but for sorted inputs i.e. sort the random n numbers before inserting them in the tree. Compare the average search time for the sorted and unsorted inputs.
3. Repeat (2) for a normal BST that is not forced to be balanced. Plot the search time for the sorted and unsorted inputs. Compare with that of the RB-Tree.

Instructions

Please submit a soft copy of the solutions together with source code and binaries in one zip file. The file should be named as **CMP461.HW##.First.Last.zip** where HW## is the homework number e.g. HW01, First and Last are your first and last name. So, if your name is Mohamed Aly, and this is homework #1, the file should be named **CMP461.HW01.Mohamed.Aly.zip.** Failing to follow these instructions will cost you points.

Acknowledgment: Some problems are adapted from Erik Demaine.