Algorithms Report2 (Due date: 5 PM, Nov. 7)

Problem solving manually

1. Using Figure 7.1 as a model, illustrate the operation of PARTITION in quicksort on the array A = <15, 22, 9, 16, 12, 8, 17, 14, 1, 2, 16, 13> and
A = <1, 3, 4, 5, 6, 7, 8, 9>. The pivot is the first element in A.

2. Consider inserting the keys 28, 17, 6, 2, 9, 17, 7, 22, 10, 12, 13, 8, 1, 21 from left to right into a hash table of length m = 5 using separate chaining where $h(k) = k \bmod m$. Draw the hash table after all the keys are inserted.

3. Consider inserting the keys 9, 18, 7, 21, 10, 12, 11, 8 from left to right into a hash table of length **m = 11** using open addressing with the auxiliary hash function $h'(k) = k \bmod m$. Draw the hash tables after inserting these keys

  a) using linear probing whith $h(k,i) = (h'(k) + i) \bmod m$

  b) using quadratic probing with$h(k,i) = (h'(k) + c_1 + c_2 i^2) \bmod m$,
  where $c_1 = 1$ and $c_2 = 3$

  c) using double hashing with $h(k,i) = (h'(k) + ih_2(k)) \bmod m$,
  where$h_2(k) = 1 + (k \bmod (m-1))$,

for i=0,1,...,m-1.
(Must show all the intermediate steps involving the hash value calculations.)

4. Answer the following questions for the keys
19, 20, 8, 6, 16, 3, 9, 7, 15, 14, 5

  a. Draw the final structure of binary search tree T when above keys are inserted from left to right.

  b. Draw the tree that results after successively executing the following functions.
  TREE-DELETE(T,19), TREE-DELETE(T,9), TREE-DELETE(T,14).

5. Write pseudo code for TREE-PREDECESSOR procedure

6. In the style of Figure 13.1(a), draw the complete binary search tree of height 3 on the keys {1, 2, . . . , 15}. Add the NIL leaves and color the nodes in three different ways such that the black-heights of the resulting red-black trees are 2, 3, and 4.

7. Draw the red-black tree that results after TREE-INSERT is called on the tree in Figure 13.1(c) with key 5. If the inserted node is colored red, is the resulting tree a red-black tree? What if it is colored black? Answer without TREE-INSERT-FIXUP execution.

8. Draw the red-black trees that result after successively inserting the keys in the order 2, 5, 8, 7, 9, 11, 13 into an initially empty red-black tree. Also, count the number of color changes, left rotations, and right rotations.

9. Draw the red-black trees that result from the successive deletion of the keys in the order 8, 11, 13, 5, 2, 9, 7 on the tree generated in exercise 8. Also, count the number of color changes, left rotations, and right rotations.

Programming

1. Construct the open address hash table according to the following description.

- m = 37
- Hash functions

  linear probing: $h(k, i) = (h'(k) + i) \bmod m$

  quadratic probing: $h(k, i) = (h'(k) + c_1 i + c_2 i^2) \bmod m$

  where, $h'(k) = k \bmod m$, $c_1 = 1$, $c_2 = 3$.

  double hashing: $h(k, i) = (h_1(k) + i h_2(k)) \bmod m$,

  where, $h_1(k) = k \bmod m$, $h_2(k) = 1 + (k \bmod (m - 1))$.

- **Insert 30 keys** that are randomly generated.

The key consists of three characters from a~z (string), i.e., sum of three ASCII values. (Ignore duplicate keys, that is, the keys with same sum of three ASCII values.)

1) Print the contents of the hash table for above three different hash functions. (Must show the **position and key value** of the inserted keys).

2) Print the average number of probes for the three different hash functions.

3) What is size of the largest cluster for each of the three different hash functions.

2. BST (Binary Search Tree)

- ■ **Node x** in a BST must have three pointers **left[x], right[x], parent[x],** and **key[x].**

1. Program the following functions.

   a. TREE-INSERT(T, x) /* Do not attach x if it is already in T */

b. TREE-SEARCH(T, k)
c. NEAREST-NEIGHBOR(T, k)
d. TREE-DELETE(T, x)
e. PRINT-BST(T) /* Prints the key values of the BST. */
/* The BST structure must be preserved. */

2. Using TREE-INSERT(T, x) construct a BST with the keys in A[20] and print the BST.

Fill in A[20] by rand()%50. Execute srand(time(NULL)) first.
Avoid duplicate values.

3. Execute TREE-SEARCH(T, 10), TREE-SEARCH(T, 9), TREE-SEARCH(T, 15) sequentially and for each search print all the keys of the nodes visited during the search.
4. Execute NEAREST-NEIGHBOR(T, 5), NEAREST-NEIGHBOR(T, 9), NEAREST-NEIGHBOR(T, 17) and print the outputs.
5. Execute TREE-INSERT(T, 6), TREE-INSERT(T, 29), TREE-INSERT(T, 17), TREE-INSERT(T, 21), sequentially.
PRINT-BST(T), if a key is actually inserted, each time after the insertion is performed. (Check whether the insertions are corrected performed.)
If a key is already in T then the insertion is ignored.
6. Execute TREE-DELETE(T, 6), TREE-DELETE(T, 17), TREE-DELETE(T, 21), TREE-DELETE(T, 7) sequentially.
PRINT-BST(T) each time after the deletion is performed.
(Check whether the deletions are corrected performed.)

3. RBT (Red-Black Tree)

1. Program the following functions.
a. RB-INSERT(T, z) /* Do not attach z if it is already in T */
b. RB-DELETE(T, z)

2. Using RB-INSERT(T, z) construct a RBT with the keys in A[20] and print the BST.

Fill in A[20] by rand()%50. Execute srand(time(NULL)) first.
Avoid duplicate values.

3. Execute RB-INSERT(T, 6), RB-INSERT(T, 29), RB-INSERT(T, 17),
RB-INSERT(T, 21), sequentially.
PRINT-BST(T), if a key is actually inserted, each time after the insertion
is performed. (Check whether the insertions are corrected performed.)
If a key is already in T the insertion ignored.

4. Execute RB-DELETE(T, 6), RB-DELETE(T, 17), RB-DELETE(T, 21),
RB-DELETE(T, 7) sequentially.
PRINT-BST(T) each time after the deletion is performed.
(Check whether the deletions are corrected performed.)

5. What are the heights of BST and RBT just after insertions are made?

- Submit (27319) the following in hardcopy (printout).
  a) solution of problem solving manually part
  b) the program (source code) and test results of programming part

- Mail the program (source code) only to sc4217@skku.edu (41 class)
- Mail the program (source code) only to wonjin12@skku.edu (43 class)

The program file format is as follows :
Student_ID_Report2_Name
ex) 2019123456_Report2_홍길동