Programming Assignment #4

Lecturer: Prof. Jaesik Park
Teaching Assistants: ByeongHoon So, Hyunmin Lee, Junha Lee

***** PLEASE READ THIS GRAY BOX CAREFULLY BEFORE STARTING THE ASSIGNMENT *****

Due date: 11:59 PM May 29, 2020

Evaluation policy:

- Late submission penalty.
  - 11:59 PM May 29 ~ 11:59 PM May 30.
    - Late submission penalty (30%) will be applied to the total score.
  - After 11:59 PM May 30.
    - 100% penalty is applied for that submission.
- Your code will be automatically tested using an evaluation program.
  - Each problem has the maximum score.
  - A score will be assigned based on the behavior of the program.
- We won't accept any submission via email - it will be ignored.

***** PLEASE READ THIS GRAY BOX CAREFULLY BEFORE STARTING THE ASSIGNMENT *****

Coding:

- Please do not use the containers in C++ standard template library (STL).
  - Such as <queue>, <vector>, and <stack>.
  - Any submission using the above headers will be disregarded.
  - Due to the many requests, <cstring> and <string> are fine to use.

Submission:

- Before submitting your work, compile and test your code using C++11 compiler in repl.it.
  - Please refer to the attached file named "PA_instructions_updated.pdf".
  - There might be a penalty if the submission would not work in the "repl.it + C++11" environment.
- What you need to submit.
  - A zip file named "pa4.zip" that contains
    - pa4.cpp
    - avl.cpp and avl.h
    - hash_function.cpp and hash_function.h
    - hash_table.cpp and hash_table.h

Any questions?

- Please use LMS - Q&A board.

1. Quiz (2 pts)

From given statements about the 2-3-4 tree, print **T** (true) or **F** (false) for each statement. You will get credit when all the answers are correct.

- Statements
  (1) Each internal node can have only 2 or 4 children.
  (2) Every node can hold only 2 or 3 or 4 data elements.
  (3) All keys in the left subtree are smaller than the first key in the parent node.
  (4) All keys in the right subtree are greater than the second key in the parent node.
  (5) A 2-3-4 tree is a B-tree of order 4.
  (6) 2-3-4 trees aim to have the largest tree height possible.

Print out a sequence of answers of each statement with the string separated with the spacebar. The answer is either T (true) or F (false). If you think the answers of the statements (1)~(3) are true and those of (4)~(6) are false then print "`T T T F F F`". You can modify `task_1` function in `pa4.cpp`.

- Example execution

```
>> ./pa4.exe 1
[Task 1]
T T T F F F
```

2. Quiz (3 pts)

Print out the answer of each sub-quiz in quiz order. You will get credit when all the answers of the sub quizzes are correct.

(1) B-tree of order n is a n-way tree in which each non-root node contains _____.

1. at most (n – 1)/2 keys
2. exact (n – 1)/2 keys
3. at least 2n keys
4. at least (n – 1)/2 keys

(2) B-tree is a tree data structure that can keep data sorted and allow searches, insertions, and deletions in _____ time.

1. O(1)
2. O(log n)
3. O(n)
4. O(2^n)

(3) A B-tree of order 4 and of height 3 will have a maximum of _____ leaves.

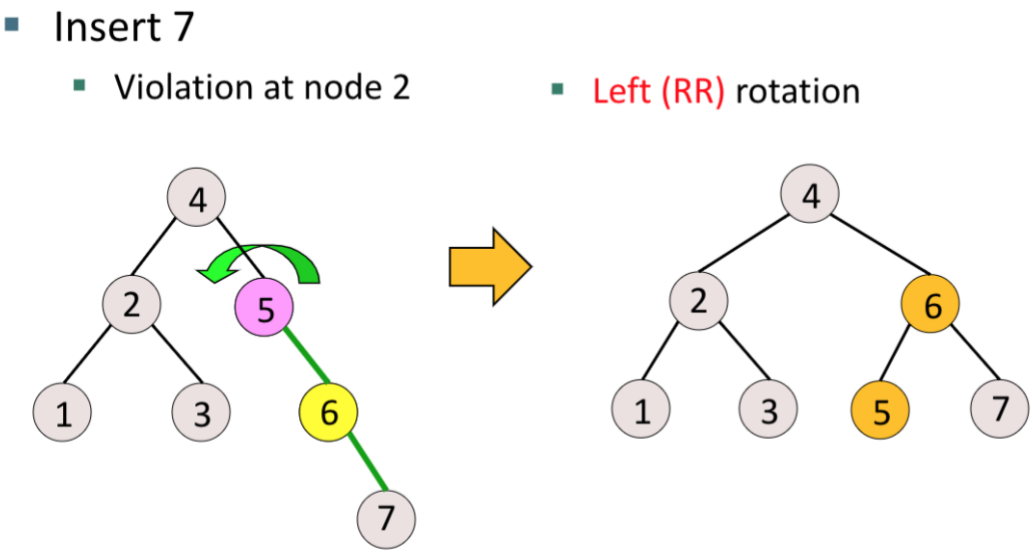
1. 64
2. 81
3. 255
4. 256

Print out a sequence of answers of each sub-quiz with the string separated with the spacebar. Each sub-quiz will have 1 correct answer. If you think the answers of the quiz 2-(1), 2-(2) and 2-(3) are 1, 4, 3, respectively, then print 1 4 3. You can modify `task_2` function in `pa4.cpp`.

- Example execution

```
>> ./pa4.exe 2
[Task 2]
1 4 3
```

3. AVL Tree Insertion (4 pts)



Example of left rotation to resolve RR imbalance

a. Implement a function that inserts an element into an AVL tree. The insertion might cause the AVL tree to violate its properties (imbalances). Your code should be able to **resolve the imbalances** of the AVL tree (LL, RR, LR, RL). You can modify `avl.cpp` and `avl.h` files for this problem.

b. Input & Output of `AVLTree::insert`
Input: key of element to be inserted
Output:
   - 0, if the insertion is successful.
   - 1, if the key already exists in the tree.

c. `task_3` prints
   i. The return value for each insertion and
   ii. The results of preorder and inorder traversal of the constructed tree.

d. Example Input & Output

Input	Output
[('insert',10), ('insert',30), ('insert',20)]	0 0 0 20 10 30 10 20 30
[('insert',4), ('insert',2), ('insert',10), ('insert',9), ('insert',15), ('insert',5)]	0 0 0 0 0 0 9 4 2 5 10 15 2 4 5 9 10 15
[('insert',1), ('insert',2), (‘insert’, 1), ('insert',3), ('insert',4), ('insert',5), ('insert',6)]	0 0 1 0 0 0 0 4 2 1 3 5 6 1 2 3 4 5 6

e. Example execution

```
>> ./pa4.exe 3 "[('insert',4), ('insert',2), ('insert',10),
('insert',9), ('insert',15), ('insert',5)]"
[Task 3]
0
0
0
0
0
0
9 4 2 5 10 15
2 4 5 9 10 15
```

4. AVL Tree Deletion (4 pts)

a. Implement a function that deletes an element from an AVL tree. Same with the previous task, your code should be able to **resolve the imbalances** after the deletion. You can modify `avl.cpp` and `avl.h` files for this problem.

b. Input & Output of `AVLTree::erase`
Input: key of element to be deleted.
Output:
- 0, if the deletion is successful.
- 1, if the key does not exist in the tree.

c. `task_4` prints
   i. The return value for each insertion & deletion and
   ii. The results of preorder and inorder traversal of the constructed tree

d. Example Input & Output

Input	Output
[('insert',4), ('insert',6), ('insert',0), ('delete',7)]	0 0 0 1 4 0 6 0 4 6
[('insert',4), ('insert',2), ('insert',10), ('insert',9), ('insert',15), ('insert',5), ('insert',0), ('delete',4), ('insert',10)]	0 0 0 0 0 0 0 0 1 9 2 0 5 10 15 0 2 5 9 10 15

e. Example execution

```
>> ./pa4.exe 4 "[('insert',4), ('insert',2), ('insert',10),
('insert',9), ('insert',15), ('insert',5), ('insert',0),
('delete',4), ('insert',10)]"
[Task 4]
0
0
0
0
0
0
0
0
1
9 2 0 5 10 15
0 2 5 9 10 15
```

5. Mid-square hashing (2 pts)

a. Implement a binary mid-square hash function. This function maps an n-bit integer key to an index of a 2^r-sized table. As a key is n bits, your code should treat the square of the key as 2n bits. You can assume that `r` is even. You can modify `hash_function.cpp` and `hash_function.h` files for this problem.

b. Input & output

Input: Three commands (The order is always `'n'`, `'r'`, and `'key'`)

- `('n', integer)`: the size of a key.
- `('r', integer)`: the size of an index.
- `('key', integer)`: a key to be hashed (in decimal).

Output: The result (i.e. index) of hashing in decimal.

c. Example Input & Output

Input	Output
`[('n', 4), ('r', 4), ('key', 10)]`	`9`
`[('n', 10), ('r', 4), ('key', 1023)]`	`8`
`[('n', 10), ('r', 4), ('key', 15)]`	`0`

d. Example execution

```
>> ./pa4.exe 5 "[('n', 4), ('r', 4), ('key', 10)]"
[Task 5]
9
```

6. Hash table (5 pts)

a. Implement **a closed hash table** with **rehashing** implementation. This hash table is used with n-bit integer keys and hashing into a table of size 2^r.
This hash table uses **linear probing** as a collision handling method. The index of the key k after i-th collision, $h_i(k)$, is:

$$h_i(k) = (h(k) + i) \mod 2^r$$

when h is the hash function implemented in `task_5`.
You don't need to consider an insertion when the table is full or a deletion of a key which does not exist or multiple insertions of the same key.
You can modify `hash_table.cpp` and `hash_table.h` files for this problem.

b. Input & output
Input: A sequence of commands
- `('n', integer)`: the size of a key.
  (The first command is always `'n'`)
- `('r', integer)`: the size of an index.
  (The second command is always `'r'`)
- `('insert', integer)`: insert `integer` into the hash table.
- `('delete', integer)`: delete `integer` from the hash table.

Output: For each slot of the hash table, print out
- the value if the state of the slot is occupied.
- the state if the state of the slot is empty or deleted.

c. Example Input & Output

Input	Output
[('n', 4), ('r', 2), ('insert', 15), ('insert', 2), ('insert', 3)]	0: 15 1: 2 2: 3 3: empty
[('n', 4), ('r', 2), ('insert', 15), ('insert', 2), ('insert', 3), ('delete', 2), ('delete', 3)]	0: 15 1: deleted 2: deleted 3: empty
[('n', 4), ('r', 2), ('insert', 15), ('insert', 2), ('insert', 3), ('delete', 2), ('delete', 3), ('insert', 0)]	0: 15 1: 0 2: deleted 3: empty

d. Example execution

```
>> ./pa4.exe 6 "[('n', 4), ('r', 2), ('insert', 15), ('insert',
2), ('insert', 3)]"
[Task 6]
0: 15
1: 2
2: 3
3: empty
```