2021_DS_Fall_HW3

> 繳交日期：繳交截止日期：2021年12月18日 11:59 PM

> homework 21: B-heap (441, 6)
> homework 22: F-heap (449, 6)
> homework 23: SMMH? (469, 5)
> homework 24: AVL tree and Red-black tree (518, 14)
> homework 25: B-tree? (550, 12)
> homework 26: B+-tree (560, 8)

Program Requirement

- Programming Language: C
- Execution Environment
  - CPU core: 1
  - Memory: 2 GB
  - Execution time limit: 1 second
  - C Compiler: `GCC`
    - Compiled with `-O2 -std=c11 -Wall`
  - C Standard: C 11
  - Read input from stdin and write your output to stdout.
  - Use header files from C Standard Library only.
- Submit your answer as a **C source code** file, no executable file.
  - Your program will be compiled by us.

Homework #21 *

Compare the performance of leftist trees and B-heaps under the assumption that the only permissible operations are insert and delete min. For this, do the following:

1. Create a random list of n elements and a random sequence of insert and delete min operations of length m. The number of delete mins and inserts should be approximately equal. Initialize a min-leftist tree and a B-heap to contain the n elements in the first random list. Now, measure the time to perform the m operations using the min-leftist tree as well as the B-heap. Divide the time by m to get the average time per operation. Do this for $n = 100, 500, 1000, 2000, \ldots, 5000$. Let m be 5000. Tabulate your computing times.
2. Based on your experiments, make some statements about the relative merits of the two data structures?

Technical Specification

- Perform the experiment. Tabulate the computing times in your homework report.
- Based on your experiments, make some statements about the relative merits of the two data structures.

Homework #22

Write C functions to do the following:

1. Create an empty F-heap
2. Insert element x into an F-heap
3. Perform a delete min from an F-heap. The deleted element is to be returned to the invoking function.
4. Delete the element in node b of an F-heap a. The deleted element is to be returned to the invoking function.
5. Decrease the key in the node b of an F-heap a by some positive amount c.

Note that all operations must leave behind properly structured F-heaps. Your functions for (4) and (5) must perform cascading cuts. Test the correctness of your procedures by running them on a computer using suitable test data.

Input Format

There are instructions in each line:

1. `insert x val`: Insert an element with key `x`
2. `extract`: Print out the minimum in the heap, and delete it from the heap.
3. `delete x val`: Delete the node with key `x` and value `val`. It is guaranteed that there will be **at most 1 node** matching the key and value.
4. `decrease x val y`: Decrease the key by `y` on the node that has key `x` and value `y`. It is guaranteed that there will be **at most 1 node** matching the key and value.
5. `quit`: Terminate your program.

Output Format

Print out `(<key>)<value>\n`, if you encounter `extract` and the F-heap is not empty.

Technical Specification

- $-2147483648 \le x, \text{val} \le 2147483647$
- $1 \le y \le 2147483647$
- $2 \le n \le 10^5$, n is number of instructions
- The key after decreasing will not exceed the bound of 32-bit signed integer.

Sample Input

```
insert 1 1001
insert 2 1002
insert 100 1001
delete 100 1001
insert 3 1003
insert 4 1004
insert 100 1001
delete 100 1001
insert 5 1005
insert 6 1006
insert 100 1001
delete 100 1001
insert 7 1007
insert 8 1008
insert 100 1001
delete 100 1001
delete 6 1006
delete 7 1007
decrease 4 1004 1000
extract
extract
extract
extract
extract
extract
extract
insert 1 7
insert 1 9
insert 1 13
delete 1 13
delete 1 7
extract
extract
quit
```

Sample Output

```
(-996)1004
(1)1001
(2)1002
(3)1003
(5)1005
(8)1008
(1)9
```

Homework #23

Develop the code for all SMMH operations. Test all functions using your own test data.

Input Format

1. `insert x`: Add an integer `x` into the SMMH tree.
2. `delete min`: Delete the minimum element in the tree.
3. `delete max`: Delete the maximum element in the tree.
4. `show`: Show your tree.
5. `quit`: Terminate your program.

Output Format

Print your tree level by level.

Technical Specification

- $-2147483648 \leq x \leq 2147483647$
- $1 \leq \text{number of instructions} \leq 5 * 10^5$

Sample Input

```
insert 13
insert 25
insert 18
insert 30
insert 40
insert 50
insert 22
insert 16
insert 19
insert 96
delete min
delete min
show
delete max
delete max
show
insert 12
insert 88
insert 63
insert 78
insert 83
delete max
delete min
show
insert 55
insert 99
insert 45
delete min
delete max
show
quit
```

Sample Output

```
NULL
18 96
19 50 30 40
22 25
NULL
18 40
19 25 22 30
NULL
18 83
19 78 22 30
25 63 40
NULL
19 83
25 78 22 30
45 63 40 55
```

Homework #24 *

Program the **search**, **insert** and **delete** operations for AVL trees and red-black trees.
(a) Test the correctness of your functions.
(b) Generate a random sequence of n inserts of distinct values. Use this sequence to initialize each of the data structures. Next, generate a random sequence searches, inserts, and deletes. In this sequence, the probability of a search should be 0.5, that of an insert 0.25, and that of a delete 0.25. The sequence length is m. Measure the time needed to perform the m operations in the sequence using each of the above data structures.
(c) Do part (b) for n = 100, 1000, 10,000, and 100,000 and $m = n$, $2n$, and $4n$.
(d) What can you say about the relative performance of these data structures.

Input Format

There are lines of input. The first line is `AVL` or `red_black`. `AVL` means implement the "instructions" by AVL-tree. `red_black` means implement the "instructions" by red-black tree. The instructions are:

1. `insert x`: Add an integer `x` to the red-black/avl tree. If `x` already exists, do nothing.
2. `search x`: Print out the $balance factor$ (or $color$) of the tree node if the element `x` exists. If the element `x` does not exist, print out `Not found\n`.
3. `delete x`: Delete the element `x` in the tree if the element `x` exists.
4. `exist x`: Print out `exist\n` if the element `x` is in the tree. If the element `x` does not in the tree, print out `Not exist\n`.
5. `quit`: Terminate your program.

(Note: The text book did not describe how to implement `delete` on AVL or red-black tree. Students can write your own `delete` **to satisfy the constraint** of the AVL or red-black tree. It is guaranteed that there is **no** `search y` after `delete x` in the testcase.)

Output Format

Print out the $balance factor$ (or $color$) or `Not found` in a line, if you encounter an instruction `search x`.
Print out `exist` or `Not exist`, if you encounter an instruction `exist x`.

Technical Specification

- $-2147483648 \leq x \leq 2147483647$
- $1 \leq \text{number of instructions} \leq 5 * 10^5$
- Upload your report. In your report, you **must**
    1. Tabulate the time you measured from (c).
    2. Try to say about the relative performance of these data structures.

Sample Input

```
red_black
delete 0
exist 0
insert 0
insert 0
exist 0
delete 0
exist 0
quit
```

Sample Output

```
Not exist
exist
Not exist
```

Homework #25

Write algorithms to search and delete keys from a B-tree by position; that is, `get(k)` finds the kth smallest key, and `delete(k)` deletes the kth smallest key in the tree (**Hint:** To do this efficiently, additional information must be kept in each node. Write each pair (E_i, A_i) keep $N_i = \sum_{j=0}^{i-1} (\text{number of elements in the subtree } A_j + 1)$.) What are the worst-case computing times of your algorithm?

Input Format

The input consists of $N + 1$ lines.

The first line contains one integer N. It indicates how many B-tree operations in the following input.

The next N lines declare N B-tree operation. For every line i, it consists of a string S_i and an integer I_i; each separated by one whitespace. The S_i can be `add`, `get`, `getk`, `remove` or `removek`. Each corresponds to one B-tree operation.

- $\text{add } I_i$: add an integer I_i into the B-tree.
- $\text{get } I_i$: find any integer I_i in the B-tree.
- $\text{getk } I_i$: find the $I_i\text{-th}$ item in the B-tree.
- $\text{remove } I_i$: remove a integer I_i from the B-tree.
- $\text{removek } I_i$: remove the $I_i\text{-th}$ item from the B-tree.

> Usually we use B-tree as a key-value store where keys can be duplicated. But in this homework we doesn't store any value(or key equals to the value). Doing so avoid tying the answer to specific implementation detail or increase the burden of evaluate the correctness of B-tree implementation.

Output Format

The output consists of N lines. Each line i corresponding to the result of $i\text{-th}$ B-tree operation $S_i\ I_i$.

- $\text{add } x$:
  - Output `add(x) = ok`
- $\text{get } x$:

    - Output `get(x) = x` if x exists in the B-tree.
    - Output `get(x) = not found` if `x` doesn't exists in the B-tree.
- $\text{getk } k$:
    - Output `getk(k) = x` where `x` is the $k\text{-th}$ item in the B-tree.
    - Output `getk(k) = not found` if `k` is an illegal position (exceeded the size of the B-tree).
- $\text{remove } x$:
    - Output `remove(x) = x` if a `x` is removed from the B-tree successfully.
    - Output `remove(x) = not found` if `x` is not in the B-tree.
- $\text{removek } k$:
    - Output `removek(k) = x` where `x` is the $k\text{-th}$ item in the B-tree.
    - Output `removek(k) = not found` if `k` is an illegal position (exceeded the size of the B-tree).

Technical Specification

- $1 < N \leq 10^4$
- $S_1, S_2, \ldots, S_N \in \{\text{add, get, getk, remove, removek}\}$
- $0 \leq I_1, I_2, \ldots, I_N \leq 10^6$
- Describe the worst-case time complexity of your `getk` implementation and `removek` implementation.

Sample Input

```
12
add 0
add 10
add 20
get 0
get 10
get 20
getk 1
getk 2
getk 3
remove 10
removek 1
removek 1
```

```
8
add 1
add 1
add 2
add 2
removek 2
removek 1
removek 2
removek 1
```

Sample Output

```
add(0) = ok
add(10) = ok
add(20) = ok
get(0) = 0
get(10) = 10
get(20) = 20
getk(1) = 0
getk(2) = 10
getk(3) = 20
remove(10) = 10
removek(1) = 0
removek(1) = 20
```

```
add(1) = ok
add(1) = ok
add(2) = ok
add(2) = ok
removek(2) = 1
removek(1) = 1
removek(2) = 2
removek(1) = 2
```

Homework #26

Program B$^+$-tree functions for exact the range search as well as for insert and delete. Test all functions using your own test data.

Input Format

The input consists of $N + 1$ lines.

The first line consists of one integer N. It indicates how many B$^+$-tree operations are in the following input.

The rest of the N lines input; each line declares a B$^+$-tree operation. Line i consists of a string S_i and one to two integers. S_i can be `add`, `remove` or `range-search`. The operation comes with 1 to 2 integers as arguments.

- `add x`: add an integer `x` into the B$^+$-tree.
- `remove x`: remove an integer `x` from the B$^+$-tree.
- `range-search l r`: Search for all the integers in the B$^+$-tree that are within the range $[l, r)$. That is, given all the integers in the B$^+$-tree $x_0, x_1, \ldots, x_n$. Search for all the x that satisfied $l \leq x < r$.

Output Format

For each `range-search` operation, your program should output one line of text `range [l,r) = [x0,x1,x2,...]`. Where `l` and `r` is the argument specified in the given `range-search` operation. And `x0,x1,x2,...` are the integers that are in the B$^+$-tree and within the range l and r.

You can refer to the sample output section for a more specific output format.

Technical Specification

- $1 < N \leq 10^4$
- $S_0, S_1, \ldots, S_N \in \{\text{add}, \text{remove}, \text{range-search}\}$.
- For any operation `add x` or `remove x`. The `x` satisfied $0 \leq x \leq 10^6$.
- For any operation `range-search l r`. The `l` and `r` satistifed $0 \leq l \leq r \leq 10^6$.

Sample Input

```
14
add 11
add 11
add 23
add 35
add 56
add 79
add 99
range-search 10 30
range-search 60 90
range-search 0 100
remove 99
range-search 0 100
range-search 11 11
range-search 11 12
```

Sample Output

```
range [10,30) = [11,11,23]
range [60,90) = [79]
range [0,100) = [11,11,23,35,56,79,99]
range [0,100) = [11,11,23,35,56,79]
range [11,11) = []
range [11,12) = [11,11]
```