Homework #6.

Associative Container for Sorted Data with Locality of Reference

(Due: 2021-12-30 23:59:59)

A. Overview

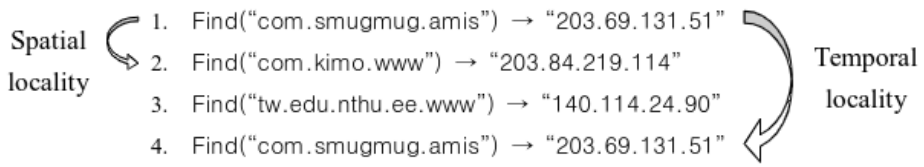
In this homework, you need to implement an associative container that stores data entries in the form of (key, value) pairs. The container can be used to store data such as Figure 1, where we have five data entries of domain name and IP address pairs.

Key	Value
com.kimo.www	203.84.219.114
com.smugmug.amis	203.69.131.51
net.akamai.b.a539	203.69.131.51
tw.edu.nctu.www	140.113.40.36
tw.edu.nthu.ee.www	140.114.24.90

Figure 1. IP addresses keyed by the associated domain names

The data entries are sorted by their keys. The keys are all unique, and in your container, there should never be two data entries with the same key. The values do not need be unique.

Look-ups (e.g. search for a data entry by its key) in the container are known to exhibit both spatial locality (*neighbors of a key just accessed are very likely to be accessed subsequently*) and temporal locality (*keys that are recently accessed are likely to be accessed again*). Following is an example of such an access pattern:



B. Description

For this homework, there are five kinds of commands you need to implement, there format and the corresponding output are specified as below.

INSERT

Input Format:

INSERT[\t]key:***key***[\t]value:***value***

Output Format:

INSERT[space][***key***][space]SUCCESS!

INSERT[space][***key***][space]FAIL!

Description:

This command inserts new data entry (***key***, ***value***) to your container.

If a data entry with the same ***key*** already exists in the container, you should output "INSERT[space][***key***][space]FAIL!" and leave the existing entry unchanged.

If the memory is full and no more data can be inserted to your container, you should also output "INSERT[space][***key***][space]FAIL!".

If the insertion is successful, you should output

"INSERT[space][***key***][space]SUCCESS!".

FIND

Input Format:

FIND[\t]key:***key***

Output Format:

FIND[space][***key]***:[value with the key equal to ***key***]!

[***key***][space]NOT[space]FOUND!

Description:

This command finds in your container the data entry with the key value equal to ***key***.

If the data entry is found, you should output

“FIND[space][***key]***:[value with the key equal to ***key***]!”.

If no data entry in the container matches ***key***, you should output

“[***key***][space]NOT[space]FOUND!”.

ERASE

Input Format:

ERASE[\t]key:***key***

Output Format:

ERASE[space][***key***][space]SUCCESS!

ERASE [space][***key***][space] FAIL!

Description:

This command erases the data entry with ***key*** from the container.

If the corresponding data entry is indeed erased from the container, you should output “ERASE[space][***key***][space]SUCCESS!”.

Else you should output “ERASE [space][***key***][space] FAIL!”.

ENUM

Input Format:

ENUM[\t]key:***szLowKey***[\t]key:***szHighKey***

Output Format:

ENUM[space][***szLowKey***]:[value with the key equal to ***key***]

.

.

.<output between ***szLowKey*** and ***szHighKey*** >

.

ENUM[space][***szHighKey***]:[value with the key equal to ***key***]

Description:

This command enumerates the data

entries with keys between ***szLowKey*** and ***szHighKey*** (inclusively on both ends) .

Your output should be the data entries with keys between ***szLowKey*** and ***szHighKey*** (inclusively on both ends) ,and its corresponding value.

The output should be sorted lexically by their keys.

The output format was shown as below.

“ENUM[space][***szLowKey***]:[value with the key equal to ***key***]”

.

.

.<output between ***szLowKey*** and ***szHighKey*** >

.

“ENUM[space][***szHighKey***]:[value with the key equal to ***key***]”

If the data entry with the ***szLowKey*** is not found, you should substitute it with the smallest key in your container. Similarly, If the data entry with the ***szHighKey*** is not found, you should substitute it with the largest key in your container.

CLEAR

Input Format:

CLEAR[\t]

Output Format:

None

Description:

For this command you should clear all the data entries in the container and release all the memory used by your container.

You don’t need to output any message for this command.

Constraints

1. For this homework, both the keys and the values are character strings of lengths that are at most 128 bytes long including the null terminator '\0'.

2. 0< number of input lines <= 300000

Preloaded Input Data

```
struct tTestData {
    int line_num;
    char data[max_data_size][buffer_size];
};
```

line_num: the number of commands

data: each line of command

Input.txt

Each line in input.txt matches one of the command format mention in B.

Example:

```
ERASE    key:aaa
ERASE    key:123
INSERT                                                    key:1235
value:52593705-3f511e3e-6852fd64-71b9644e-747dc7e8
FIND     key:1235
INSERT   key:a    value:6c84f722-8df7c480-f9f54fa9-605cc5-582eac3c
INSERT   key:b    value:bd6858e4-768de7c7-1b061bf-cf3eca05-dd0dd2cf
INSERT   key:c    value:a0d1ddff-c1f462e5-3e86647-3beabf68-facb990d
INSERT   key:d    value:8acec4e1-1174b734-4b30137e-70e48e42-112bd69e
ENUM     key:c    key:d
```

Output.txt

Each line in input.txt matches one of the output format mention in B.

Example:

```
ERASE aaa FAIL!
ERASE 123 FAIL!
INSERT 1235 SUCCESS!
FIND 1235:52593705-3f511e3e-6852fd64-71b9644e-747dc7e8!
INSERT a SUCCESS!
INSERT b SUCCESS!
INSERT c SUCCESS!
INSERT d SUCCESS!
ENUM c:a0d1ddff-c1f462e5-3e86647-3beabf68-facb990d
ENUM d:8acec4e1-1174b734-4b30137e-70e48e42-112bd69e
```

C. Submission of Homework

1. Implement Associative Container for Sorted Data with Locality of Reference

2. Upload your code.cpp or code_shm.cpp to Code Sensor

3. Use the “View” and “Scoreboard” function on CODE SENSOR to make sure your submission is successful.

D. Grading Guideline

Following are some highlights on the grading guideline:

0. Correctness of your code (as calculated from the number of test cases passed.)
1. Performance of your code including memory usage and total instruction counts consumed.
2. The total grade points for the performance part are competitively shared by all the students.

3. Compilation error means 0 points for the homework.
4. You code has to go through the whole analysis process on CODE SENSOR without any interruption. If not, you also get 0 point for the homework.
5. You should not use the following forbidden (illegal) headers
   “stl_tree.h”, “map”, "mman.h", "valgrind.h", "callgrind.h", "ptrace.h", "signal.h"
6. You should not use the following library calls
   "mmap", "mprotect", "munmap", "syscall", "fork", "vfork", "pthread_create", "system", “exec family”
7. You should not make direct system calls or use any inline assembly.
8. Do not copy & paste code from others. Your code will be open for peer-review after the grading is finalized.

Reminders and hints

Splay tree seems like a perfect candidate for this homework as it supports ordering of data and has the cache-like property to deal with localities in the access pattern.