Programming Assignment 3: Hash Tables and Hash Functions

Revision: May 4, 2018

Introduction

In this programming assignment, you will practice implementing hash functions and hash tables and using them to solve algorithmic problems. In some cases you will just implement an algorithm from the lectures, while in others you will need to invent an algorithm to solve the given problem using hashing.

Learning Outcomes

Upon completing this programming assignment you will be able to:

1. Apply hashing to solve the given algorithmic problems.
2. Implement a simple phone book manager.
3. Implement a hash table using the chaining scheme.
4. Find all occurrences of a pattern in text using Rabin–Karp's algorithm.

Passing Criteria: 2 out of 3

Passing this programming assignment requires passing at least 2 out of 3 code problems from this assignment. In turn, passing a code problem requires implementing a solution that passes all the tests for this problem in the grader and does so under the time and memory limits specified in the problem statement.

Contents

1 Problem: Phone book

Problem Introduction

In this problem you will implement a simple phone book manager.

Problem Description

Task. In this task your goal is to implement a simple phone book manager. It should be able to process the following types of user's queries:

- **add** `number` `name`. It means that the user adds a person with name `name` and phone number `number` to the phone book. If there exists a user with such number already, then your manager has to overwrite the corresponding name.
- **del** `number`. It means that the manager should erase a person with number `number` from the phone book. If there is no such person, then it should just ignore the query.
- **find** `number`. It means that the user looks for a person with phone number `number`. The manager should reply with the appropriate name, or with string "not found" (without quotes) if there is no such person in the book.

Input Format. There is a single integer N in the first line — the number of queries. It's followed by N lines, each of them contains one query in the format described above.

Constraints. $1 \le N \le 10^5$. All phone numbers consist of decimal digits, they don't have leading zeros, and each of them has no more than 7 digits. All names are non-empty strings of latin letters, and each of them has length at most 15. It's guaranteed that there is no person with name "not found".

Output Format. Print the result of each **find** query — the name corresponding to the phone number or "not found" (without quotes) if there is no person in the phone book with such phone number. Output one result per line in the same order as the **find** queries are given in the input.

Time Limits.

language	`C`	`C++`	`Java`	`Python`	`JavaScript`	`Scala`
time (sec)	3	3	6	6	9	9

Memory Limit. 512MB.

Sample 1.

Input:

```
12
add 911 police
add 76213 Mom
add 17239 Bob
find 76213
find 910
find 911
del 910
del 911
find 911
find 76213
add 76213 daddy
find 76213
```

Output:

```
Mom
not found
police
not found
Mom
daddy
```

Explanation:
76213 is Mom's number, 910 is not a number in the phone book, 911 is the number of police, but then it was deleted from the phone book, so the second search for 911 returned "not found". Also, note that when the daddy was added with the same phone number 76213 as Mom's phone number, the contact's name was rewritten, and now search for 76213 returns "daddy" instead of "Mom".

Sample 2.

Input:

```
8
find 3839442
add 123456 me
add 0 granny
find 0
find 123456
del 0
del 0
find 0
```

Output:

```
not found
granny
me
not found
```

Explanation:
Recall that deleting a number that doesn't exist in the phone book doesn't change anything.

Starter Files

The starter solutions for C++, Java and Python3 in this problem read the input, implement a naive algorithm

to look up names by phone numbers and write the output. You need to use a fast data structure to implement a better algorithm. If you use other languages, you need to implement the solution from scratch.

What to Do

Use the direct addressing scheme.

Need Help?

Ask a question or see the questions asked by other learners at this forum thread.

2 Problem: Hashing with chains

Problem Introduction

In this problem you will implement a hash table using the chaining scheme. Chaining is one of the most popular ways of implementing hash tables in practice. The hash table you will implement can be used to implement a phone book on your phone or to store the password table of your computer or web service (but don't forget to store hashes of passwords instead of the passwords themselves, or you will get hacked!).

Problem Description

Task. In this task your goal is to implement a hash table with lists chaining. You are already given the number of buckets m and the hash function. It is a polynomial hash function

$$h(S) = \left(\sum_{i=0}^{|S|-1} S[i]x^i \bmod p \right) \bmod m\,,$$

where $S[i]$ is the ASCII code of the i-th symbol of S, $p = 1\ 000\ 000\ 007$ and $x = 263$. Your program should support the following kinds of queries:

- **add** `string` — insert `string` into the table. If there is already such string in the hash table, then just ignore the query.
- **del** `string` — remove `string` from the table. If there is no such string in the hash table, then just ignore the query.
- **find** `string` — output "yes" or "no" (without quotes) depending on whether the table contains `string` or not.
- **check** i — output the content of the i-th list in the table. Use spaces to separate the elements of the list. **If i-th list is empty, output a blank line.**

When inserting a new string into a hash chain, you must insert it in the beginning of the chain.

Input Format. There is a single integer m in the first line — the number of buckets you should have. The next line contains the number of queries N. It's followed by N lines, each of them contains one query in the format described above.

Constraints. $1 \le N \le 10^5$; $\frac{N}{5} \le m \le N$. All the strings consist of latin letters. Each of them is non-empty and has length at most 15.

Output Format. Print the result of each of the **find** and **check** queries, one result per line, in the same order as these queries are given in the input.

Time Limits.

language	`C`	`C++`	`Java`	`Python`	`JavaScript`	`Scala`
time (sec)	1	1	5	7	7	7

Memory Limit. 512MB.

Sample 1.

Input:

```
5
12
add world
add HellO
check 4
find World
find world
del world
check 4
del HellO
add luck
add GooD
check 2
del good
```

Output:

```
HellO world
no
yes
HellO
GooD luck
```

The ASCII code of 'w' is 119, for 'o' it is 111, for 'r' it is 114, for 'l' it is 108, and for 'd' it is 100. Thus, $h(\text{“world"}) = (119 + 111 \times 263 + 114 \times 263^2 + 108 \times 263^3 + 100 \times 263^4 \bmod 1\,000\,000\,007) \bmod 5 = 4$. It turns out that the hash value of *HellO* is also 4. Recall that we always insert in the beginning of the chain, so after adding “world" and then “HellO" in the same chain index 4, first goes “HellO" and then goes “world". Of course, “World" is not found, and “world" is found, because the strings are case-sensitive, and the codes of 'W' and 'w' are different. After deleting “world", only “HellO" is found in the chain 4. Similarly to “world" and “HellO", after adding “luck" and “GooD" to the same chain 2, first goes “GooD" and then “luck".

Sample 2.

Input:

```
4
8
add test
add test
find test
del test
find test
find Test
add Test
find Test
```

Output:

```
yes
no
no
yes
```

Adding “test" twice is the same as adding “test" once, so first **find** returns “yes". After del, “test" is no longer in the hash table. First time **find** doesn't find “Test" because it was not added before, and

strings are case-sensitive in this problem. Second time "Test" can be found, because it has just been added.

Sample 3.

Input:

```
3
12
check 0
find help
add help
add del
add add
find add
find del
del del
find del
check 0
check 1
check 2
```

Output:

```
no
yes
yes
no

add help
```

Explanation:
Note that you need to output a blank line when you handle an empty chain. Note that the strings stored in the hash table can coincide with the commands used to work with the hash table.

Starter Files

There are starter solutions only for C++, Java and Python3, and if you use other languages, you need to implement solution from scratch. Starter solutions read the input, do a full scan of the whole table to simulate each **find** operation and write the output. This naive simulation algorithm is too slow, so you need to implement the real hash table.

What to Do

Follow the explanations about the chaining scheme from the lectures. Remember to always insert new strings in the beginning of the chain. Remember to output a blank line when **check** operation is called on an empty chain.

Some hints based on the problems encountered by learners:

- Beware of integer overflow. Use `long long` type in C++ and `long` type in Java where appropriate. Take everything $\pmod p$ as soon as possible while computing something $\pmod p$, so that the numbers are always between 0 and $p-1$.
- Beware of taking negative numbers $\pmod p$. In many programming languages, $(-2)\%5 \neq 3\%5$. Thus you can compute the same hash values for two strings, but when you compare them, they appear to

be different. To avoid this issue, you can use such construct in the code: $x \leftarrow ((a\%p) + p)\%p$ instead of just $x \leftarrow a\%p$.

Need Help?

Ask a question or see the questions asked by other learners at this forum thread.

3 Problem: Find pattern in text

Problem Introduction

In this problem, your goal is to implement the Rabin–Karp's algorithm.

Problem Description

Task. In this problem your goal is to implement the Rabin–Karp's algorithm for searching the given pattern in the given text.

Input Format. There are two strings in the input: the pattern P and the text T.

Constraints. $1 \le |P| \le |T| \le 5 \cdot 10^5$. The total length of all occurrences of P in T doesn't exceed 10^8. The pattern and the text contain only latin letters.

Output Format. Print all the positions of the occurrences of P in T in the ascending order. Use 0-based indexing of positions in the the text T.

Time Limits.

language	`C`	`C++`	`Java`	`Python`	`JavaScript`	`Scala`
time (sec)	1	1	5	5	3	3

Memory Limit. 512MB.

Sample 1.

Input:

```
aba
abacaba
```

Output:

```
0 4
```

Explanation:
The pattern *aba* can be found in positions 0 (**aba**caba) and 4 (abac**aba**) of the text *abacaba*.

Sample 2.

Input:

```
Test
testTesttesT
```

Output:

```
4
```

Explanation:
Pattern and text are case-sensitive in this problem. Pattern *Test* can only be found in position 4 in the text *testTesttesT*.

Sample 3.

Input:

```
aaaaa
baaaaaaa
```

Output:

```
1 2 3
```

Note that the occurrences of the pattern in the text can be overlapping, and that's ok, you still need to output all of them.

Starter Files

The starter solutions in C++, Java and Python3 read the input, apply the naive $O(|T||P|)$ algorithm to this problem and write the output. You need to implement the Rabin–Karp's algorithm instead of the naive algorithm and thus significantly speed up the solution. If you use other languages, you need to implement a solution from scratch.

What to Do

Implement the fast version of the Rabin–Karp's algorithm from the lectures.

Some hints based on the problems encountered by learners:

- Beware of integer overflow. Use `long long` type in C++ and `long` type in Java where appropriate. Take everything $\pmod p$ as soon as possible while computing something $\pmod p$, so that the numbers are always between 0 and $p-1$.
- Beware of taking negative numbers $\pmod p$. In many programming languages, $(-2)\%5 \neq 3\%5$. Thus you can compute the same hash values for two strings, but when you compare them, they appear to be different. To avoid this issue, you can use such construct in the code: $x \leftarrow ((a\%p)+p)\%p$ instead of just $x \leftarrow a\%p$.
- Use operator `==` in Python instead of implementing your own function `AreEqual` for strings, because built-in operator `==` will work much faster.
- In C++, method `substr` of `string` creates a new string, uses additional memory and time for that, so use it carefully and avoid creating lots of new strings. When you need to compare pattern with a substring of text, do it without calling `substr`.
- In Java, however, method `substring` does NOT create a new `String`. Avoid using `new String` where it is not needed, just use `substring`.

Need Help?

Ask a question or see the questions asked by other learners at this forum thread.

4 Solving a Programming Challenge in Five Easy Steps

4.1 Reading Problem Statement

Start by reading the problem statement that contains the description of a computational task, time and memory limits, and a few sample tests. Make sure you understand how an output matches an input in each sample case.

If time and memory limits are not specified explicitly in the problem statement, the following default values are used.

Time Limits.

language	`C`	`C++`	`Java`	`Python`	`JavaScript`	`Scala`
time (sec)	1	1	1.5	5	5	3

Memory limit: 512 Mb.

4.2 Designing an Algorithm

After designing an algorithm, prove that it is correct and try to estimate its expected running time on the most complex inputs specified in the constraints section. If you laptop performs roughly 10^8–10^9 operations per second, and the maximum size of a dataset in the problem description is $n = 10^5$, then an algorithm with quadratic running time is unlikely to fit into the time limit (since $n^2 = 10^{10}$), while a solution with running time $O(n \log n)$ will. However, an $O(n^2)$ solution will fit if $n = 1\,000$, and if $n = 100$, even an $O(n^3)$ solutions will fit. Although polynomial algorithms remain unknown for some hard problems in this book, a solution with $O(2^n n^2)$ running time will probably fit into the time limit as long as n is smaller than 20.

4.3 Implementing an Algorithm

Start implementing your algorithm in one of the following programming languages supported by our automated grading system: `C`, `C++`, `Haskell`, `Java`, `JavaScript`, `Python2`, `Python3`, or `Scala`. For all problems, we provide starter solutions for `C++`, `Java`, and `Python3`. For other programming languages, you need to implement a solution from scratch. The grading system detects the programming language of your submission automatically, based on the extension of the submission file.

We have reference solutions in `C++`, `Java`, and `Python3` (that we don't share with you) which solve the problem correctly under the given constraints, and spend at most 1/3 of the time limit and at most 1/2 of the memory limit. You can also use other languages, and we've estimated the time limit multipliers for them. However, we have no guarantee that a correct solution for a particular problem running under the given time and memory constraints exists in any of those other languages.

In the Appendix, we list compiler versions and flags used by the grading system. We recommend using the same compiler flags when you test your solution locally. This will increase the chances that your program behaves in the same way on your machine and on the testing machine (note that a buggy program may behave differently when compiled by different compilers, or even by the same compiler with different flags).

4.4 Testing and Debugging

Submitting your implementation to the grading system without testing it first is a bad idea! Start with small datasets and make sure that your program produces correct results on all sample datasets. Then proceed to checking how long it takes to process a large dataset. To estimate the running time, it makes sense to implement your algorithm as a function like `solve(dataset)` and then implement an additional procedure `generate()` that produces a large dataset. For example, if an input to a problem is a sequence of integers of length $1 \le n \le 10^5$, then generate a sequence of length 10^5, pass it to your `solve()` function, and ensure that the program outputs the result quickly.

Check the boundary values to ensure that your program processes correctly both short sequences (e.g., with 2 elements) and long sequences (e.g., with 10^5 elements). If a sequence of integers from 0 to, let's say, 10^6 is given as an input, check how your program behaves when it is given a sequence $0, 0, \ldots, 0$ or a sequence $10^6, 10^6, \ldots, 10^6$. Afterwards, check it also on randomly generated data. Check degenerate cases like an empty set, three points on a single line, a tree which consists of a single path of nodes, etc.

After it appears that your program works on all these tests, proceed to stress testing. Implement a slow, but simple and correct algorithm and check that two programs produce the same result (note however that this is not applicable to problems where the output is not unique). Generate random test cases as well as biased tests cases such as those with only small numbers or a small range of large numbers, strings containing a single letter "a" or only two different letters (as opposed to strings composed of all possible Latin letters), and so on. Think about other possible tests which could be peculiar in some sense. For example, if you are generating graphs, try generating trees, disconnected graphs, complete graphs, bipartite graphs, etc. If you generate trees, try generating paths, binary trees, stars, etc. If you are generating integers, try generating both prime and composite numbers.

4.5 Submitting to the Grading System

When you are done with testing, submit your program to the grading system! Go to the submission page, create a new submission, and upload a file with your program (make sure to upload a source file rather than an executable). The grading system then compiles your program and runs it on a set of carefully constructed tests to check that it outputs a correct result for all tests and that it fits into the time and memory limits. The grading usually takes less than a minute, but in rare cases, when the servers are overloaded, it might take longer. Please be patient. You can safely leave the page when your solution is uploaded.

As a result, you get a feedback message from the grading system. You want to see the "`Good job!`" message indicating that your program passed all the tests. The messages "`Wrong answer`", "`Time limit exceeded`", "`Memory limit exceeded`" notify you that your program failed due to one of these reasons. If you program fails on one of the first two test cases, the grader will report this to you and will show you the test case and the output of your program. This is done to help you to get the input/output format right. In all other cases, the grader will *not* show you the test case where your program fails.

5 Appendix: Compiler Flags

C (`gcc 5.2.1`). File extensions: `.c`. Flags:

```
gcc -pipe -O2 -std=c11 <filename> -lm
```

C++ (`g++ 5.2.1`). File extensions: `.cc`, `.cpp`. Flags:

```
g++ -pipe -O2 -std=c++14 <filename> -lm
```

If your `C/C++` compiler does not recognize `-std=c++14` flag, try replacing it with `-std=c++0x` flag or compiling without this flag at all (all starter solutions can be compiled without it). On Linux and MacOS, you most probably have the required compiler. On Windows, you may use your favorite compiler or install, e.g., `cygwin`.

Java (`Open JDK 8`). File extensions: `.java`. Flags:

```
javac -encoding UTF-8
java -Xmx1024m
```

JavaScript (`Node v6.3.0`). File extensions: `.js`. Flags:

```
nodejs
```

Python 2 (`CPython 2.7`). File extensions: `.py2` or `.py` (a file ending in `.py` needs to have a first line which is a comment containing "python2"). No flags:

```
python2
```

Python 3 (`CPython 3.4`). File extensions: `.py3` or `.py` (a file ending in `.py` needs to have a first line which is a comment containing "python3"). No flags:

```
python3
```

Scala (`Scala 2.11.6`). File extensions: `.scala`.

```
scalac
```