Data Structure and Algorithm, Spring 2025
Mini-Homework L

TA E-mail: dsa_ta@csie.ntu.edu.tw

Due: 13:00:00, Tuesday, June 10, 2025

Rules and Instructions

- Any form of cheating, lying, or plagiarism will not be tolerated. Students can get zero scores and/or fail the class and/or be kicked out of school and/or receive other punishments for those kinds of misconducts.
- Discussions on course materials and homework solutions are encouraged. But you should write the final solutions alone and understand them fully. Books, notes, and Internet resources (including chatGPT) can be consulted, but not copied from.
- Since everyone needs to write the final solutions alone, there is **absolutely no need to lend your homework solutions and/or source codes to your classmates at any time.** In order to maximize the level of fairness in this class, lending and borrowing homework solutions are both regarded as dishonest behaviors and will be punished according to the honesty policy.
- In each mini-homework, we will have one programming problem that is closely connected to what we have taught in the class. The purpose of the problem is for you to *practice* what you have learned, before conquering the more challenging problems in other homework sets.
- Like other programming problems, you should write your code in C programming language, and then submit the code via the *DSA Judge*:

  https://dsa-2025.csie.org/.

  Each day, you can submit up to 5 times for each problem. The judge system will compile your code with

  ```
  gcc [id].c -static -O2 -std=c11
  ```

- For debugging in the programming part, we strongly suggest you produce your own test-cases with programs and/or use tools like `gdb` or compile with flag `-fsanitize=address` to help you solve issues like illegal memory usage. For `gdb`, you are strongly encouraged to use compile flag `-g` to preserve symbols after compiling.

  Note: For students who use IDE (VScode, DevC++...) you can modify the compile command in the settings/preference section. Below are some examples:

  - `gcc [id].c -O2 -std=c11 -fsanitize=address # Works on Unix-like OS`
  - `gcc [id].c -O2 -std=c11 -g # use it with gdb`

- The score of the part that is submitted after the deadline will get some penalties according to the following rule:

$$Late\ Score = \max\left(\left(\frac{5 \times 86400 - Delay\ Time(sec.)}{5 \times 86400}\right) \times Original\ Score, 0\right)$$

  Please note that the "gold medal" of the class cannot be used on mini-homework problems.

- If you have questions about this mini-homework, please go to the Discord channel and discuss the issues with your classmates. This is *strongly preferred* because it will provide everyone with a more interactive learning experience. If you really need email clarifications, you are encouraged to include the following tag in the subject of your email:

  - The subject should contain one tag, `"[hwl]"`, specifying the problem where you have questions. For example, `"[hwl] Will there be a repeated number?"`. Adding these tags potentially allows the TAs to track the status of each email and to provide faster responses to you.
  - If you want to provide your code segments to us as part of your question, please upload it to Gist or similar platforms and provide the link. Please remember to protect your uploaded materials with proper permission settings. Screenshots or code segments directly included in the email are discouraged and may not be reviewed.

Programming Part

Dynamic Hash Table (15 pts)

Problem Description

In this problem, we will ask you to implement the directoryless dynamic hash table introduced in the lecture. Please refer to the lecture notes and section 8.3.3 of Fundamentals of Data Structure in C by Horowitz et al, available on NTU COOL.

Here are the details of the implementation of the directoryless dynamic hash table.

- The item to be saved in the hash table is a string S formed by lowercase English alphabets, i.e., $\Sigma = \{a, b, \ldots, z\}$.
- We define the Rabin-Karp hash function $\tau(S)$ as follows:

$$\tau(S) = d^{m-1} \cdot S[1] + d^{m-2} \cdot S[2] + \cdots + d^1 \cdot S[m-1] + d^0 \cdot S[m] \tag{1}$$

  where $m = |S|$, $d = |\Sigma| = 26$. Note that because of the use of $LSB()$ function below, in the definition here we do not need to use the modulo operation.

- Here we define the hash function used by the hash table:

$$h(S, n) = LSB(\tau(S), n) \quad , \tag{2}$$

  where S is the input string, and $LSB(s, n)$ is a function that returns the n least significant bits of the input number s.

- r and q are the parameters determining the active buckets of the directoryless hash table. At any time, only buckets 0 through $2^r + q - 1$ are active. You can assume the hash table start with parameters $r = 2, q = 0$.
- To insert an item S into the hash table, calculate the the hash $h(S, r)$ using the current value of r. Then, depending on whether $h(S, r) < q$:
  - If yes, then store the item in bucket $h(S, r + 1)$.
  - If no, then store the item in bucket $h(S, r)$.

  Note that after insertion the bucket can overflow, which is normal.
- Each bucket can hold up to *two* items. The overflown bucket uses a linked list to store the additional items.

- If a new insertion triggers an overflow in a bucket, increase the value of q by 1. After the increment, $1 \leq q \leq 2^r$. This creates a new bucket at $2^r + q - 1$. The items in the original bucket $q - 1$ now should be rearranged into bucket $q - 1$ and bucket $2^r + q - 1$, using $h(S, r + 1)$. All other buckets should not change. Note that after the rearrangement, the buckets could still stay in the overflown state. Finally, if $q = 2^r$ after the increment, increase the value of r by 1 and set q to 0.

Input

The first line has a single positive integer number N, which specifies the number of strings to be inserted into the directoryless hash table.

Each of the following N lines include a single string S to be inserted. These N strings should be inserted in the given order.

Output

After inserting all N strings, you should print $2^r + q$ lines, where r and q are the values of these parameters.

In the k-th line, $1 \leq k \leq 2^r + q$, print strings which are stored in bucket $k - 1$ in the order that they are inserted. A single space character is used to separated consecutive items. If bucket $k - 1$ is empty, print -1 in the k-th line.

Constraint

- $1 \leq N \leq 2 \times 10^5$
- $1 \leq |s| \leq 64$

Sample Testcases

Sample Input 1

```
8
a
b
c
d
e
f
g
h
```

Sample Output 1

```
a e
b f
c g
d h
```

Sample Input 2

```
4
ae
ae
aa
aa
```

Sample Output 2

```
aa aa
-1
-1
-1
ae ae
```

Sample Input 3

```
4
c
c
c
c
```

Sample Output 3

```
-1
-1
c c c c
-1
-1
-1
```