CS29003 ALGORITHMS LABORATORY
ASSIGNMENT 11
Date: 11th November, 2021

Important Instructions

1. **Format of files (in Moodle & HackerRank):** ROLLNO_A11_T1.c/.cpp, ROLLNO_A11_T2.c/.cpp, ROLLNO_A11_T3.c/.cpp

   Also For task 2, submit the output file (visible test cases) in moodle as well as in hackerrank. Do the same for task 3. The format (for moodle upload) should be ROLLNO_O_T2.txt, ROLLNO_O_T3.txt.
2. Solve the problems with least complexity (as possible). There will be separate marks for them.
3. **Write proper comments while coding. Failing which you will loose marks and it will be thought as copied.**
4. You have to **stick to the file input output formats strictly** as per the instructions.
5. Submission through **.zip files are not allowed**.
6. **Write your name, roll number and hacker rank id at the beginning of your program.**
7. Do not use any global variable unless you are explicitly instructed so.
8. **Use proper indentation in your code**.
9. **You are not allowed to use any library except *stdio* and *iostream*. Library *stdlib* is only allowed for dynamic memory allocation**.
10. **Copying from the Internet or copying among yourselves will be severely penalized**.

Hashing

Alice in Wonderland

Our friend, Alice went to a country named Wonderland to visit her friend Bob, in her newly purchased Airbike. Airbike is a special bike that can fly in the air with no smoke emission technology. But unfortunately, Bob was not at home. From his neighborhood, Alice got to know that Bob went on a vacation with his family members. Alice was very disappointed because she also wanted to spend the vacation at Wonderland. Alice was wandering in Bob's garden when suddenly she saw some magnets. As these magnets are known for their special abilities and are only available in Wonderland, hence Alice always wanted to collect them for her collection.

All these magnets have the same polar intensity. Here polar intensity is the repulsive power of the magnet. We can keep the magnets side-by-side if they are different. But for the same magnets, we have to keep them separately. More specifically we need to keep the same magnets at least P position(s) apart. That means between the same magnets, there should be at least P-1 magnets. E.g, in "aba bcde aba", two "aba" magnets are 2 positions apart. Now the problem is that Alice couldn't carry the magnet in her

Figure 1: Alice suddenly saw the magnets

Airbike since it causes magnetic interference with her electronic Airbike. But fortunately, at the time of arrival, she brought an insulator box with her, which does not allow magnetic flux to go outside of the insulator box. The box size is the same as the total number of magnets she collected from the garden. All magnets are denoted by a lowercase string (all characters in the string are lowercase English alphabet). Also, some magnets are superior to others. In our case, all the magnets which contain at least one vowel alphabet are superior to others. These magnets have a tendency to evolve. That means when one of these magnets is kept with some other magnet, two magnets are fused and forms a larger magnet, which finally loses its magnetic ability. Hence Alice decided to put some insulators in between when a superior magnet is kept with some other magnet. We should keep insulator '|' when we need to keep a superior magnet (containing vowel alphabet) and an ordinary (all consonants) magnet side-by-side. Similarly, we should keep insulator '#' when we need to keep two superior magnets side-by-side. Consider the insulators require no space in the insulator box. Two ordinary magnets can be kept side by side (without an insulator) if they are not the same.

Task 1: Design Hash Table

First design a hash table as stated below.

- ▷ Each entry of the hash table should contain a key and a value fields. Where key is of type string and value is of type integer.
- ▷ It should follow the Closed Hashing (Open Addressing) scheme. That is, the table size is fixed and separate chaining should not be used as collision resolution technique.
- ▷ Collision should be avoided by **Linear Probing only**
- ▷ For generating the hash function, you should have to use **polynomial hashing**. E.g, if the key is "abc", then the $hash = (c + b * p + a * p^2)\%hashTableSize$, where $p = 31$. While implementing this function, consider a as 1,b as 2, ... ,z as 26.
- ▷ While printing the table, you should have to print all entries in the hash table (including empty entries). E.g, a table of size 10 is shown below after insertion of some entries:

(,0) (,0) (mno,5) (jkl,4) (ghi,3) (def,2) (abc,1) (pq,6) (,0) (,0)

Here (,0) denotes empty entries.

▷ Each node of the hash table should follow the below structures.

- **HashNode structure**

```
typedef struct HashNode{
    string key;
    int value;
}HashNode;
```

Here equivalent structure could be used in case of C programming

- **HashTable structure**

```
typedef struct HashTable{
    int size;
    HashNode *array;
}HashTable;
```

▷ Implement the below functions for using the hash table.

- **Initialize** : This function initializes all keys with empty string and all value fields with 0.
- **Search**: This function searches a key, if key is found it should return the value field corresponding to a key. If not found, it should return -1.
- **Insert** : This function inserts a (key,value) pair in the hash table. If successful, it should return the index of the key. Otherwise (table size is full), it should return -1.
- **Delete** : This function deletes a (key,value) pair from the hash table. If the key is found, it should return the index of the key. If key is not found, it should return -1.
- **IncreaseValue** : This function increases the value field corresponding to a key by 1. Then it should return the index of the key. If the key is not found it should return -1.
- **Print** : This function should print all the (key,value) pairs (including empty fields) as described above.
- **DeleteTable** : Deletes the complete hash table.

NOTE: Use the above hash table, for solving the below problems, wherever hashing is needed.

Now let 'I def 3' denote 'Insert' operations with key as 'def' and value as '3'. 'D def' denotes the 'Delete' operation for key 'def'. 'IV def' denotes the "IncreaseValue' operation for the key 'def'.

Input/ Output format

▷ **Input :** The 1st line denotes the number of test cases, T. For every test case, The 1st line denotes the size of hash table, S. 2nd line denotes the number of operations, N. Each subsequent line denotes the operations. Let maximum length of name of the magnet is M.

$$1 <= T <= 100$$

$$1 <= S <= 1000$$

$$1 <= N <= 1000$$

$$1 <= M <= 5$$

▷ **Output :** The entries (tuples) of hash table, separated by spaces.

Example

FILE: *input-t1-1.txt*

```
2
10
5
I xyv 1
I erdp 1
I er 1
IV erdp
D er
3
1
I abcd 3
```

FILE: *output-t1-1.txt*

```
(,0) (xyv,1) (,0) (erdp,2) (,0) (,0) (,0) (,0) (,0) (,0)
(,0) (abcd,3) (,0)
```

Explanation

Input file :

There are 2 test cases (1st line).
For test case 1: The table size is 10 (2nd line). There are total 5 operations (3rd line). Next lines denote the operations, keys, values as mentioned above.
For test case 2: The table size is 3. There is only single operation.

Output file :

(,0) denotes empty entries.
For test case 1: Hash value of "xyv" is calculated as $(24 + 25 * 31 + 22 * 31^2)\%10 = 1$. Hence (xyz,1) is stored in 1st index of the table. Similarly, (erdp,1) mapped into $((5+18*31+4*31^2+16*31^3)\%10 = 3)$ 3rd index. As we have also performed *IncreaseValue* operation on erdp, hence we got 2 in the value field.
For test case 2: (abcd,3) is mapped to index 1 as described above.

Task 2: Help Alice with arranging the magnets

Help Alice to rearrange the magnets in such a way so that she could keep all the magnets inside the insulator box. If it is not possible to get such an arrangement, your function should print/ return -1. There can be many correct rearrangements. You have to provide any one of them.

Input/ Output format

▷ **Input :** The 1st line of the input is the number of test cases, T. For each test case, the next three lines denote the number of magnets N, the polar intensity of each magnet I and the magnets separated by spaces, respectively.

$$1 < T < 100$$

$$1 < N < 1000$$

$$1 < I < 50$$

▷ **Output :** For each test case, the output should be the rearranged magnets (if rearrangement is possible). Otherwise it should be -1.

Example

FILE: *input-t2-1.txt*

```
2
6
2
abc abc def def ca ca
5
3
abc abc abc def ca
```

FILE: *output-t2-1.txt*

```
abc # def # ca # abc # def # ca
-1
```

Explanation

Input

There are total 2 test cases (line 1).
In test case 1 : There are total 6 magnets (line 2). The polar intensity of each magnet is 2 (line 3). The magnets are separated by spaces (line 4).
In test case 2: There are total 5 magnets (line 5). The polar intensity of each magnet is 3 (line 6). The magnets are separated by spaces (line 7).

Output

For test case 1 : The rearrangement is possible. As each magnets are superior magnets (containing at least a vowel), hence they are separated by the insulator #.
For test case 2: The rearrangement is not possible.

Task 3: Help Alice with Sticky Magnets

Some magnets in the garden are sticky magnets. These magnets have all the above properties including the superiority property (magnets with vowel alphabets) but here the same magnets need to be kept exactly P position apart, where P is the magnetic intensity. These magnets are denoted by uppercase string (all characters in the string are uppercase alphabets). **We have not mixed sticky magnets with non-sticky magnets. That means in this task, we are only considering sticky magnets**. Help Alice to rearrange the sticky magnets in such a way so that she could keep all the sticky magnets inside the insulator box. If it is not possible to get such an arrangement, your function should print/ return -1. There can be many correct rearrangements. You have to provide any one of them.

Input/ Output format

- ▷ **Input :** The 1st line of the input is the number of test cases. For each test case, the next three lines denotes the number of magnets,the polar intensity of each magnet and the magnets separated by space respectively.
- ▷ **Output :** For each test case, the output should be the rearranged magnets (if rearrangement is possible). Otherwise it should be -1.

Example

FILE: *input-t3-1.txt*

```
2
6
3
ABC ABC ABC DEF CA CA
5
2
ABC ABC ABC DEF CA
```

FILE: *output-t3-1.txt*

```
-1
ABC # DEF # ABC # CA # ABC
```

Explanation

Input

There are total 2 test cases.
In test case 1: There are total 6 magnets, each with polar intensity 3.
In test case 2: There are total 5 magnets, with polar intensity 2

Output

For Test case 1 : The rearrangement is not possible.
For Test case 2: The rearrangement is possible. As all magnets are superior magnets (contains at least a vowel), hence they are separated by #.