DSA (CSE102) lab

26-03-25

Problem Statement

You must implement a basic **shift-based masking** (similar to a simple cipher) on each input string, then store and query them using **hashing** (open addressing, chaining, or direct addressing). You are allowed to program in **C or C++** *without* using any STL data structures (e.g., `unordered_map`, `map`, etc.). Instead, you will create and manage the hash table on your own.

Specifications

1. Shift Transformation

Let SHIFT be a fixed integer (for example, SHIFT = 1). For each character c in a string S, define:

$$c_{\text{shifted}} \;=\; (\text{ASCII}(c) + \text{SHIFT}) \bmod 128,$$

assuming the extended ASCII range $[0, 127]$. You can adapt this if you want only alphabetic characters or a different modulo range.

The *masked* version of S (denoted S_{masked}) is obtained by replacing each character c with c_{shifted}.

Example: If SHIFT = 1 and we have:

- `'c'` (ASCII 99) → ASCII 100 = `'d'`
- `'t'` (ASCII 116) → ASCII 117 = `'u'`

2. Hash Table (No STL)

You must store **only the masked version** of each string in a hash table:

- **No STL containers** allowed.
- Implement any collision resolution method (*open addressing*, *chaining*, or *direct addressing* if the range is small).

- Provide a suitable *hash function* for these masked strings (e.g., sum of ASCII codes, polynomial rolling, etc.).
- When a collision occurs, handle it according to your chosen scheme.

3. Querying

After inserting the masked versions of N original strings into your hash table:

1. Read Q query strings.
2. For each query q, compute its masked version q_{masked} using the same shift rule.
3. Check if q_{masked} is in your hash table.
4. Print `YES` if found, `NO` otherwise.

4. Language Constraints

- You must write your solution in **C or C++**, with *no STL* usage.
- Implement collision handling, arrays, and hashing logic manually.

Input & Output Format

Input Format:

1. An integer N (the number of original strings).
2. N lines, each containing one original string.
3. An integer Q (the number of queries).
4. Q lines, each containing one query string.

Output Format: For each query string, print `YES` or `NO` on its own line, depending on whether the hashed (masked) string is found.

Sample Example

Sample Input

```
4
cat
dad
bob
test
```

```
3
bob
dad
cot
```

SHIFT = 1, Explanation

- **Insertion Phase:**
  1. `"cat"` → `"dbu"`
  2. `"dad"` → `"ebe"`
  3. `"bob"` → `"cpc"`
  4. `"test"` → `"uftu"`

  These four *masked* strings (`dbu`, `ebe`, `cpc`, `uftu`) are inserted into your hash table.

- **Query Phase:**
  1. `"bob"` → `"cpc"` (found in the table) $\implies$ `YES`
  2. `"dad"` → `"ebe"` (found) $\implies$ `YES`
  3. `"cot"` → `"dpu"` (not found) $\implies$ `NO`

Sample Output

```
YES
YES
NO
```

Assignment

1. **Implement** the shift transformation for all strings.
2. **Create a suitable hash function** for these masked strings and an appropriate collision resolution scheme.
3. **Insert N masked strings** into the hash table.
4. **Process Q queries** by shifting each query, hashing it, and checking membership. Print `YES` or `NO`.
5. **Briefly discuss** how you handle collisions and the efficiency of your hash function.

Note: This simple shift-based masking is purely illustrative and not meant for robust security.