**STRING HASHING**

Hashing is a technique that is used to uniquely identify a specific object from a group of similar objects.

Assume that you have an object and you want to assign a key to it to make searching easy. To store the key/value pair, you can use a simple array like a data structure where keys (integers) can be used directly as an index to store values. However, in cases where the keys are large and cannot be used directly as an index, you should use hashing.

In hashing, large keys are converted into small keys by using **hash functions**. The values are then stored in a data structure called **hash table**. The idea of hashing is to distribute entries (key/value pairs) uniformly across an array. Each element is assigned a key (converted key). By using that key you can access the element in **O(1)** time. Using the key, the algorithm (hash function) computes an index that suggests where an entry can be found or inserted.

A simple would be counting frequencies of alphabets in a string and storing in it separate array (of size 26), so that when you need to print frequency of any particular alphabet then you can directly access that value without having to iterate all over the string again.

However hashing could lead to collisions as well which might dissolve the purpose of hashing itself. Below are few collision reducing techniques,

1. [Separate Chaining](http://www.geeksforgeeks.org/hashing-set-2-separate-chaining/)

2. [Open Addressing](http://www.geeksforgeeks.org/hashing-set-3-open-addressing/)

String hashing is a technique of mapping string to an integer, suppose we have a string **S** then it can be assigned an integer value **H** such that for some base **B,**

**H = s[0] \* B(m – 1) + s[1] \* B(m – 2) + … + s[m - 2] \* B1 + s[m - 1] \* B0**

The task of comparing two strings then reduces to just comparing the hash values of the two strings. But sometimes string. A problem arises when m and B are big enough and the number H becomes too large to fit into the standard integer types. To overcome this, instead of the number H itself we use its remainder when divided by some other number M. Applying the basic rules of modular arithmetic to the above expression:

**A + B = C => (A % M + B % M) % M = C % M  
A \* B = C => ((A % M) \* (B % M)) % M = C % M**

We get:

**H % M = (((s[0] % M) \* (B(m – 1) % M)) % M + ((s[1] % M) \* (B(m – 2) % M)) % M +……+ ((s[m - 2] % M) \* (B1 % M)) % M +**

**((s[m - 1] % M) \* (B0 % M)) % M) % M.**

The drawback of using remainders is that it may turn out that two different strings map to the same number (it is called a collision). This is less likely to happen if M is sufficiently large and B and M are prime numbers.

In order to understand the applications of string hashing visit [String Hashing for competitive programming](https://threads-iiith.quora.com/String-Hashing-for-competitive-programming) and [Rabin-Karp string searching algorithm](https://www.topcoder.com/community/data-science/data-science-tutorials/introduction-to-string-searching-algorithms/).

Here are few questions that you should practice to get a proper grasp of the concept.

[**https://www.hackerearth.com/practice/data-structures/hash-tables/basics-of-hash-tables/practice-problems/algorithm/a-needle-in-the-haystack-1/description/**](https://www.hackerearth.com/practice/data-structures/hash-tables/basics-of-hash-tables/practice-problems/algorithm/a-needle-in-the-haystack-1/description/)

<https://www.hackerearth.com/practice/data-structures/hash-tables/basics-of-hash-tables/practice-problems/algorithm/little-monk-and-the-matchmaker/description/>

<https://www.hackerearth.com/challenge/competitive/code-monk-hashing/algorithm/monk-and-match-making/>

<https://www.codechef.com/problems/SSTORY>

<http://codeforces.com/contest/245/problem/H>