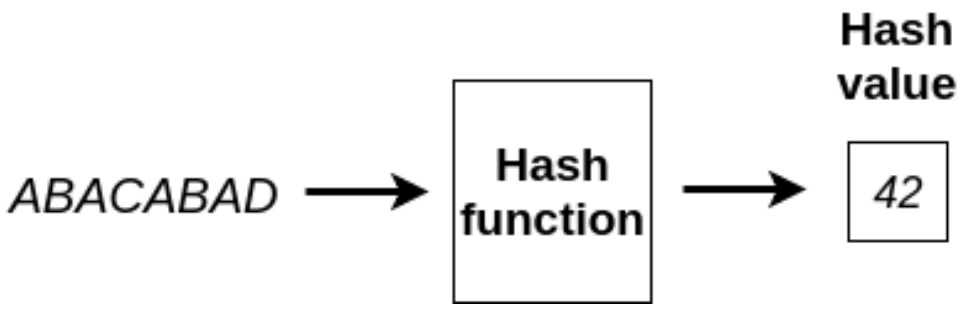
Theory: String hashing

32 minutes 6 / 6 problems solved

Start practicing

String hashing is a technique that allows us to represent a string as a number. A rule used to associate a string with a number is called a **hash function**, and the resulting number is called a **hash value** (or simply a **hash**):



An important advantage of string hashing is that it makes it possible to compare two strings in $O(1)$ since we simply need to compare the strings' hash values. This property is used to efficiently solve some substring processing problems.

Usually, a hash for a string is calculated as follows: each symbol of the string is associated with a number, then a hash value is computed as a sum of these numbers with some coefficients. There are several ways to associate a symbol with a number. In this topic, we will use the following rule: A corresponds to 1, B corresponds to 2, ..., Z corresponds to 26. That is, each symbol is associated with its order number in the alphabet. As for string hashing functions, there are a few of them as well. Our next step is to learn some of them and understand their pros and cons.

§1. Linear hashing

For a string $s = s_0 s_1 ... s_{n-1}$, a linear hash function h_L is defined as a sum of the symbols' associated values:

$$h_L(s) = s_0 + s_1 + ... + s_{n-1}$$

For example, a hash value for $s = ABAC$ is

$$h_L(ABAC) = 1 + 2 + 1 + 3 = 7.$$

This is one of the simplest hash functions for strings. A disadvantage of the linear hash function is that a hash value does not depend on the order of symbols. This means that if we reorder the symbols of a string, the hash value for the string won't change. For example, strings $s_1 = ABAC$ and $s_2 = CBAA$ are not equal, but they consist of the same symbols and thus have equal hash values:

$$h_L(ABAC) = 1 + 2 + 1 + 3 = 7,\ h_L(CBAA) = 3 + 2 + 1 + 1 = 7$$

A situation when two different strings have equal hash values is called a **collision**. An important property of any hash function is how many strings it maps to the same hash value. The less the number of such strings, the better the hash function. At this point, linear hashing is not the best choice since the limitation described above results in many collisions.

§2. Polynomial hashing

For a string $s = s_0 s_1 ... s_{n-1}$, a polynomial hash function is defined as follows:

$$h_P(s) = \left(s_0 \cdot a^0 + s_1 \cdot a^1 + ... + s_{n-1} \cdot a^{n-1}\right)\ mod\, m$$

Here, a is a constant, usually a prime number approximately equal to the total number of different symbols in the alphabet; m is a constant as well, usually a big prime number. Let's consider how we can calculate the polynomial hash for $s = ACDC$. For simplicity, we will use $a = 3$ and $m = 11$.

$$h_P(ACDC) = \left(1 \cdot 3^0 + 3 \cdot 3^1 + 4 \cdot 3^2 + 3 \cdot 3^3\right)\ mod\, 11$$

Current topic:

String hashing

Topic depends on:

String basics

Topic is required for:

Rabin-Karp algorithm

Table of contents:

$$= (1 \cdot 1 + 3 \cdot 3 + 4 \cdot 9 + 3 \cdot 27) \ mod\, 11$$

$$= 127 \ mod\, 11 = 6$$

Although the polynomial hash depends on the order of symbols in a string, collisions are still possible. For example, $s_1 = BBAB$ and $s_2 = ABCC$ are different strings with equal hash values:

$$h_P(BBAB) = \left(2 \cdot 3^0 + 2 \cdot 3^1 + 1 \cdot 3^2 + 2 \cdot 3^3\right) \ mod\, 11 = 71 \ mod\, 11 = 5,$$

$$h_P(ABCC) = \left(1 \cdot 3^0 + 2 \cdot 3^1 + 3 \cdot 3^2 + 3 \cdot 3^3\right) \ mod\, 11 = 115 \ mod\, 11 = 5.$$

However, the probability of a collision for the polynomial hash function is $\approx \frac{1}{m}$, which is quite low for a big m. Thus, the polynomial hash function is a good choice for string hashing.

§3. Rolling hashing

This section describes not a distinct function, but rather a property of hash functions that is applicable to both linear and polynomial hashing.

Consider a string $s = ABBCCCB$. Let's use the linear hash function to calculate a hash for a prefix of s of length 4:

$$h_L(ABBC) = s_0 + s_1 + s_2 + s_3 = 1 + 2 + 2 + 3 = 8.$$

Now, assume we need to calculate a hash value for the next substring of s of length 4. This can be done as follows:

$$h_L(BBCC) = s_1 + s_2 + s_3 + s_4 = 2 + 2 + 3 + 3 = 10.$$

Note that the second sum can also be obtained if we subtract s_0 from the first sum and then add s_4:

$$h_L(BBCC) = h_L(ABBC) - s_0 + s_4 = 8 - 1 + 3 = 10.$$

Thus, if we know a hash value $h_L(s_i s_{i+1} ... s_j)$, a hash value for the next substring can be calculated in $O(1)$ as follows:

$$h_L(s_{i+1} s_{i+2} ... s_{j+1}) = h_L(s_i s_{i+1} ... s_j) - s_i + s_{j+1}.$$

This property of a hash function is called a rolling hash. The same property holds to the polynomial hash function: the only difference is that we need to start the calculations with the last substring of s:

$$h_P(BCCB) = \left(s_2 \cdot 3^0 + s_3 \cdot 3^1 + s_4 \cdot 3^2 + s_5 \cdot 3^3\right) \ mod\, 11$$

$$= \left(2 \cdot 3^0 + 3 \cdot 3^1 + 3 \cdot 3^2 + 2 \cdot 3^3\right) \ mod\, 11$$

$$= 92 \ mod\, 11 = 4.$$

The next substring of s from the end has the following hash value:

$$h_P(BBCC) = \left(s_1 \cdot 3^0 + s_2 \cdot 3^1 + s_3 \cdot 3^2 + s_4 \cdot 3^3\right) \ mod\, 11$$

$$= \left(2 \cdot 3^0 + 2 \cdot 3^1 + 3 \cdot 3^2 + 3 \cdot 3^3\right) \ mod\, 11$$

$$= 116 \ mod\, 11 = 6$$

Using the hash value for the first substring, the second one can be obtained as follows:

$$h_P(BBCC) = \left(\left(h_P(BCCB) - s_5 \cdot 3^3\right) \cdot 3 + s_1\right) \ mod\, 11$$

$$= \left((4 - 2 \cdot 27) \cdot 3 + 2\right) \ mod\, 11$$

$$= -148 \ mod\, 11 = 6$$

So, if we know the hash value $h_P(s_{i+1} s_{i+2} ... s_j)$, a hash value for a neighboring substring can be calculated as follows:

$$h_P(s_i s_{i+1} ... s_{j-1}) = \left(\left(h_P(s_{i+1} s_{i+2} ... s_j) - s_j \cdot a^{j-i-1}\right) \cdot a + s_i\right) \ mod\, m.$$

The rolling hash property allows us to efficiently solve some substring processing problems. A good example is a Rabin-Karp algorithm, which is based on string hashing. It allows us to find a substring in linear time, which

is faster than the naive approach. We will consider this algorithm in more detail in the next topic.

§4. Summary

String hashing is a way to represent a string as a number. It is useful for some string processing algorithms since hash values can be compared in $O(1)$. There are several ways to hash a string, linear and polynomial hashing being among them. The latter is a better choice for string hashing since it has fewer collisions. Both hash functions are rolling hashes. That is, if we know the hash value for some substring, a hash value for a neighboring substring can be calculated in $O(1)$.

Report a typo

28 users liked this theory. **1** didn't like it. **What about you?**

Start practicing

This content was created over 1 year ago and updated about 12 hours ago. Share your feedback below in comments to help us improve it!

Comments (4) Hints (0) Useful links (0) Show discussion