Rolling Hash

Danny Mittal

7 February 2020

1 Problem

You are given $N \leq 10^6$ strings of total length $L \leq 10^5$. What is the largest l such that two of the strings have the same prefix of length l?

2 How do we solve this?

Naively, for each value of l, compare all the prefixes of length l to each other and output the largest l that works. For a given l, there can be at most N prefixes of length l, so we have to do $O(N^2)$ comparisons, and each comparison is $O(l)$, so the total complexity for a given l is $O(N^2 l)$, making the overall complexity $O(N^4)$.

This is obviously too slow. If you make certain observations about the total length of the strings, you can prove that this will actually be $O(N^2 \log n)$, but that's still too slow, so we need a better approach, and that approach is rolling hash.

3 Hash

A hash is a function that converts some kind of object, in this case a string, into an integer. The use of this is that if two objects are equal, they will have the same hash, so we can compare two objects by first comparing their hash: if the hashes aren't equal, then they're not the same, so we don't have to compare the objects themselves, which can be time consuming. As long as we choose a good enough hash function so that two different objects will almost never have the same hashes, we can essentially disregard the possibility that two unequal objects will need to be fully compared; the only time we will ever need to fully compare two objects is if they are equal.

4 Rolling Hash

For strings, we usually use a certain type of hash known as a rolling hash. The rolling hash of a string s of length l is defined as follows: for some integer p and a mod m,

$$hash(s) = (\sum_{j=0}^{l-1} p^j s_j)\%m$$

This can be computed in $O(l)$ time. The useful aspect of this hash, however, comes when we consider appending a character to s.

Let $t = s + c$, where c is a single character. Then:

$$hash(t) = (\sum_{j=0}^{l} p^j t_j)\%m = (\sum_{j=0}^{l-1} p^j s_j + p^l c)\%m = (hash(s) + p^l c)\%m$$

p^l can be precomputed for all values of l, so if we know $hash(s)$, then we can compute $hash(t)$ in constant time, hence the name "rolling hash": we can easily "roll in" a character c. This allows us to do many things much more quickly; for example, given a string s of length l, we can compute the hashes of all of its prefixes in $O(l)$ time by computing the hash of every prefix based on the hash of the previous prefix.

Going back to the problem, this means that can now compute the hashes of every prefix of every string in $O(L)$ time. Given the hashes, we can now solve the problem as follows: for each value of l, create a map from hash values to the prefixes with that hash value. For each prefix s of length l, if $hash(s)$ is in the map, then check if any of the other strings with hash equal to $hash(s)$ is equal to s. If so, then l is a possible answer. There are L prefixes, for each of which we perform an $O(1)$ computation. By iterating through values of l in decreasing order, we can immediately stop once a value of l works, so we only compare two equal strings, a worst case $O(L)$ operation, once. Since other collisions are very unlikely, the complexity of this solution is $O(L)$.

5 Another Problem

Given a string S of length L and some integer $l \leq L$, find the first index $j > 0$ such that the substring $S_j ... S_{j+l-1}$ is equal to the substring $S_k ... S_{k+l-1}$ for some k such that $0 \leq k < j$.

6 Unrolling Hash

Similarly to how we can "roll in" an additional character in constant time, we can easily "unroll" a character, meaning compute the hash of a string after removing a character from the beginning (or end) in constant time. Let s be a string of length l and let t be the substring of s after the first character s_0 of s. Then:

$$hash(s) = (\sum_{j=0}^{l-1} p^j s_j)\%m = (s_0 + p(\sum_{j=0}^{l} p^j t_j))\%m = (s_0 + phash(t))\%m$$

$$hash(t) = (\frac{hash(s) - s_0}{p})\%m$$

If we choose m to be a prime (and not choose p to be a multiple of m), then we can compute p's mod m modular inverse p^{-1}, so that our above equation reduces to

$$hash(t) = (p^{-1}(hash(s) - s_0))\%m$$

allowing us to unroll in constant time.

We can now apply this to our new problem as follows: keep a map from hash values to the substrings with that hash value. For each value of j from 0 to $L - l$, compute the hash of $S_j ... S_{j+l-1}$. If $j = 0$, we can do this in $O(l)$; otherwise, we can do this in constant time by rolling and unrolling the hash for $j - 1$. Then, simply query the map with the hash value as we did before, and return j when you find two substrings that are equal.

7 Tips

Always use longs for hashes.

You should always choose p to be an odd prime. In particular, the optimal choice of p is generally the smallest prime that is at least as large as the range of characters that the input can contain, so for example:

- If the input can only contain lowercase or only uppercase letters, the range would be 26 and you'd choose $p = 29$.
- If the input can contain uppercase and lowercase letters, the range would be 52 and you'd choose $p = 53$.
- If the input can contain alphanumeric characters, the range would be 62 and you'd choose $p = 67$.

The choice of m depends on what you need: If you need to unroll, then m should be a prime, and $m = 10^9 + 7$ is a good choice. Make sure that you don't choose m to be larger than $2 \cdot 10^9$, as then you can run into long overflow issues when multiplying. If you don't need to unroll, then you don't need to use m at all - long overflow will automatically provide you with modding for $m = 2^64$, which will give you the largest range of hash values and thus the smallest chance of collision.

Some problems may allow test cases that force you to compare equal strings many times, making checking for string equality when the hashes are equal inefficient. In these cases, since collisions are so rare, you can assume that two strings with the same hash are equal so as to make your code efficient. Collisions can still happen, so it may be necessary to randomly generate the values of p and/or m (in USACO, you can instead just submit your code and manually change the values of p or m when you get Wrong Answer; note that USACO does not allow nondeterministic code).

8 Problems

8.1

Given a string S of length at most 10^6 and at most 10^5 pairs (a, b) representing substrings $S_a ... S_b$, find any two pairs that represent equal substrings or determine that there are none.

8.2

Given two strings S, T each of length at most 10^5, find the first occurrence of S in T or determine that T does not occur in S.

8.3

Given two strings S, T each of length at most 10^5, find the largest string which is a substring of both S and T.