INF280: Competitive programming

Strings

Louis Jachiet

String search

Typical string problem

String searching

You are given a text T (i.e. a long string) and a pattern P, and you need to find a/all positions in T where P appears.

Usual variations:

- T and P not necessarily made of chars
- multiple patterns $P_1, \ldots, P_k$
- different types of patterns (case insensitive, regexps, etc.)

Find a string pattern within another string

- Naive algorithm:

```
// s is the string, p is the pattern
for (int i=0, j; i < s.size() - p.size() + 1; ++i) {
  int j = 0;
  while(j < p.size() && s[i+j] == p[j])
    j++;
  if (j == p.size())
    printf("Match at position %d\n", i);
}
```

- Good on average
- Worst case time complexity $O(|s| \times |p|)$
- Can we do better?

Knuth–Morris–Pratt

Idea

If we matched j first letters of p at position i, we don't need to compare all of p for position $i + 1$.

Algorithm

For each prefix p' of p, compute the longest strict suffix p'' of p' that is a prefix of p.

A nice and efficient algorithm, but hard to code. Let us see something simpler.

String hash

Idea

Have a hash function h that can easily be computed over a sliding window.

In practice

Given the text $s_1 \dots s_n$ and the pattern $p_1 \dots p_k$ we compute $o = h(p_1 \dots p_k)$, then for each $i \in n - k$ we compare $h(s_i \dots s_{k+i})$ with o. If they match there is a high probability there is a match at position i.

String hash

Idea

Have a hash function h that can easily be computed over a sliding window.

In practice

Given the text $s_1 \dots s_n$ and the pattern $p_1 \dots p_k$ we compute $o = h(p_1 \dots p_k)$, then for each $i \in n - k$ we compare $h(s_i \dots s_{k+i})$ with o. If they match there is a high probability there is a match at position i.

To get linear time

$h(s_{i+1} \dots s_{k+i+1})$ can be computed from $h(s_i \dots s_{k+i})$ by adding s_{k+i+1} at the end and removing s_i at the beginning.

What hash function has the right properties?

String hash

Use $\mathbb{Z}/2^{64}\mathbb{Z}$ with any odd number $g > 1$!

$$h(s_0 \ldots s_k) = \sum_i s_i g^{k-i}$$

Update $h(s_0 \ldots s_k)$

- $h(s_1 \ldots s_k) = h(s_0 \ldots s_k) - s_0 \times g^k$
- $h(s_0 \ldots s_k s_{k+1}) = g \times h(s_0 \ldots s_k) + s_{k+1}$

String hash

Use $\mathbb{Z}/2^{64}\mathbb{Z}$ with any odd number $g > 1$!

$$h(s_0 \ldots s_k) = \sum_i s_i g^{k-i}$$

Notes:

- $\mathbb{Z}/2^{64}\mathbb{Z}$ is just `unsigned long long`!
- You can precompute g^k for all useful k
- Small g often have random collisions (BA=AF with $g = 5$)
- to make collisions unlikely, you also need to make sure that g is bigger than the values manipulated and that $min(k \mid g^k = 1)$ is big ($g = 10^6 + 3$ usually works)

String hash alternative

Use $\mathbb{Z}/2^{64}\mathbb{Z}$ with some g!

$h(s_0 \ldots s_k) = \sum_i s_i g^i$

Update $h(s_0 \ldots s_k)$

- $h(s_1 \ldots s_k) = (h(s_0 \ldots s_k) - s_0) \times g^{-1}$
- $h(s_0 \ldots s_k s_{k+1}) = h(s_0 \ldots s_k) + g^{k+1} s_{k+1}$

String hash alternative

Use $\mathbb{Z}/2^{64}\mathbb{Z}$ with some g!

$h(s_0 \ldots s_k) = \sum_i s_i g^i$

Update $h(s_0 \ldots s_k)$

- $h(s_1 \ldots s_k) = (h(s_0 \ldots s_k) - s_0) \times g^{-1}$
- $h(s_0 \ldots s_k s_{k+1}) = h(s_0 \ldots s_k) + g^{k+1} s_{k+1}$

Note that $g^{-1} = g^{2^{64}-1}$ which can be precomputed with python, e.g. with $g = 27$:

```
pow(27,2**64-1,2**64)=9564978408590137875
```

Exercises

Exercise 0.

If string hash acts as a perfect hash function (i.e. as a random function) what is the probability of a collision? Given n distinct strings, for what values of n, it is likely that two of those hashes are equal?

Exercise 1.

Solve string matching with string hash. What is the complexity?

Exercise 2.

Solve string matching with many patterns $p_1, \ldots, p_k$. What is the complexity?

Exercise 3*.

Use string hash to search the longest palindrome in a string.