Строки

Сапожников Денис

Contents

1 Хеши

Большинство задач на строки сильно бы упростились, если бы можно было проверять на равенство две строки за $O(1)$, а не за их длину. Хеши как раз предлагают этот метод.

1.1 Полиномиальный хеш

1.1.1 Определение

Определение. Полиномиальным хешом строки $s = s_0s_1s_2\dots s_{n-1}$ называется сумма

$$hash(s) = (s_0 + s_1 \cdot Q + s_2 \cdot Q^2 + \dots + s_{n-1}Q^{n-1})\%P$$

для некоторых чисел P и Q, называемых **модулем хеша** и **основанием хеша** соответственно.

Зачем нам это нужно? Теперь мы будем говорить, что если $hash(s) = hash(t)$, то строки s и t равны[1].

Возникает резонный вопрос: Строк много, а чисел мало, как быть? А именно, если число P порядка 10^9 (а именно такое число обычно и будет использоваться), то мы отображаем набор из экспоненты строк (строк длины 100 из 26 символов 26^{100}) в числа из интервала $[0; P-1]$. Это значит, что разным строкам будут сопоставлены одни и те же числа.

Не надо паники, это нормально. Ситуация, когда две строки имеют одинаковый хеш называется **коллизия**. Идея в том, что коллизии бывают не так часто, если правильно всё сделать.

Утверждение (Бездоказательно). *Отображение строк в их полиномиальные хеши преобразует строку в случайное число* [2]. *А про случайные числа есть интересное наблюдение, именуемое парадоксом дней рождений.*

1.1.2 Парадокс дней рождений

Если в классе 23 ученика или более, то более вероятно то, что у какой-то пары одноклассников дни рождения придутся на один день, чем то, что у каждого будет свой неповторимый день рождения (то есть верятность совпадения дней рождений в классе из 23 и более людей превышает 50%).

Эту вероятность вычислить совсем не сложно.

- Если в классе один ученик, то вероятность того, что все ДР в разные дни равна 1.
- Если в классе есть два ученика, то вероятность того, что ДР 2го ученика попадет на ДР первого равна $\frac{1}{365}$. То есть вероятность того, что у всех ДР в разные дни равна $1 - \frac{1}{365}$
- Если в классе есть три ученика, то вероятность того, что ДР третьего ученика не попадет на ДР первых двух равна $1 - \frac{2}{365}$ (если их ДР в разные дни). Таким образом, вероятность того, что у трех учеников ДР в разные дни равна $\left(1 - \frac{1}{365}\right)\left(1 - \frac{2}{365}\right)$
- И т.д. получаем формулу вида

$$\left(1 - \frac{1}{365}\right)\left(1 - \frac{2}{365}\right)\left(1 - \frac{3}{365}\right) \times \dots \times \left(1 - \frac{23}{365}\right) < 0.5$$

[1] с большой вероятностью

[2] при наличии некоторых дополнительных условий, о которых в разделе о выборе P и Q

Если бы в году было n дней, то потребовалось бы примерно $\sqrt{n}$ чисел. Эмпирически это означает, что если вы делаете k сравнений строк, то P должно быть порядка k^2.

1.1.3 О выборе P и Q

1. Чтобы отображение было случайным, необходимо, чтобы P было простым числом

2. Чтобы не было «глупых» коллизий, необходимо следующие условия:

   (a) $s_i < Q$
   Если $Q = 2$, а $s = 012, t = 221$, то $hash(s) = hash(t) = 10$.

   (b) $s_i > 0$
   Без этого хеши для строк 000 и 00 будут равны. То есть если вы делаете $c \to c -$'a', то вы как раз отображаете букву 'a' в ноль, не надо так!

3. Чтобы было мало коллизий необходимо брать большое P, но чтобы вычисления помещались в long long, необходимо чтобы $P \sim 10^9$. Таким образом, хорошими простыми числами являются $10^9+7, 10^9+9$. Рекомендую найти своё простое число в районе 10^9 и использовать его на олимпиадах.

4. Если сравнений подстрок больше, чем 10^5 (например, $10^5 \log 10^5$), и у вас есть есть подозрение на коллизии (например, при замене Q/P случаются разные WA), то для каждой строки храните не один хеш, а два (суммы разным модулям).

   Пример, где хеш по двойному модулю необходим – это если у вас есть множество из n строк и вы хотите проверить, есть ли в нём пара равных. Тогда вы засунете хеши всех строк в set и проверите их на равенство, **но неявно вы сравните каждую строку с каждой**, то есть у вас будет n^2 сравнений на равенство.

1.1.4 Хеши подстрок

Помимо сравнения строк, можно сравнивать и подстроки. Для этого запомним не только хеш строки, но и хеш всех ее префиксов. Тогда
$hash(s[0 \dots r]) - hash(s[0 \dots l-1]) = Q^{l-1}s_l + Q^{l+1}s_{l+1} + \dots + Q^{r-1}s_r =$
$= Q^{l-1}(s_l + Qs_{l+1} + \dots + s_r Q^{r-l-1}) = Q^{l-1} \cdot hash(s[l \dots r])$

То есть мы можем получить хеш произвольной подстроки, домноженной на Q^{l-1}. Не забудьте, что все вычисления происходят по простому модулю, значит разница $hash(s[0 \dots r]) - hash(s[0 \dots l-1])$ **может быть отрицательной**, так что не забудьте в таком случае добавить P. Далее, если мы хотим сравнить две подстроки на равенство, то есть два способа:

1. Сложный, но удобный. Можно делить число $Q^{l-1} \cdot hash(s[l \dots r])$ по простому модулю P на Q^{l-1}. Это будет занимать $O(\log P)$ или $O(1)$ и $O(n \log P)$ предпросчета.

2. Простой, но иногда неудобный. Пусть есть $Q^{l_1-1} \cdot hash(s[l_1 \dots r_1])$ и $Q^{l_2-1} \cdot hash(t[l_2 \dots r_2])$. Мы хотим понять, равны ли $s[l_1 \cdots r_1]$ и $t[l_2 \cdots r_2]$.

   Тогда домножим первое число на величину Q^{l_2-1}, а вторую – на Q^{l_1-1}. Теперь получится: $Q^A \cdot hash(s[l_1 \dots r_1])$ и $Q^A \cdot hash(t[l_2 \dots r_2])$, где $A = l_1 + l_2 - 2$. То есть теперь если $hash(s[l_1 \dots r_1]) = hash(t[l_2 \dots r_2])$, то и $Q^A \cdot hash(s[l_1 \dots r_1]) = Q^A \cdot hash(t[l_2 \dots r_2])$; в обратную сторону равенство тоже верно.

1.1.5 Задачи

Предположим, вы забыли как считать z-функцию. Тогда будем делать бинпоиск по длине z-блока для каждой позиции и проверять на равенство подстроки.

```cpp
const int P = 1e9 + 7;
const int Q = 543;

const int N = 1e5 + 1;

int degq[N], h[N];

bool is_equal(int l1, int r1, int l2, int r2) {
  int h1 = h[r1] - (l1 == 0 ? 0 : h[l1 - 1]);
  if (h1 < 0) h1 += P;
  int h2 = h[r2] - (l2 == 0 ? 0 : h[l2 - 1]);
  if (h2 < 0) h2 += P;
  return h1 * 1LL * degq[l2] % P == h2 * 1LL * degq[l1] % P;
}

int main() {
  // h(s1s2s3...) = s1 + s2*Q + s3*Q^2 + ...
  degq[0] = 1;
  for (int i = 1; i < N; ++i)
    degq[i] = degq[i - 1] * 1LL * Q % P;

  string s;
  cin >> s;

  int n = s.size();

  for (int i = 0; i < n; ++i) {
    h[i] = ((i == 0 ? 0 : h[i - 1]) + s[i] * 1LL * degq[i]) % P;
  }

  for (int i = 0; i < n; ++i) {
    int l = 0, r = n - i + 1;

    while (r - l > 1) { // [l; r)
      int m = (r + l) / 2;

      if (is_equal(0, m - 1, i, i + m - 1))
        l = m;
      else
        r = m;
    }

    cout << l << ' ';
  }
}
```

1.2 Хеши для множеств и мультимножеств

Пусть мы теперь хотим сравнивать не строки на равенство за $O(1)$, а множества или мультимножества (множества, в которых каждый элемент может встречаться больше одного раза).

Способ 1: Каждому числу x сопоставим случайное число $r(x)$. Важно, чтобы это сопоставление было одинаковое для x всегда.

Хешом множества $s = \{s_1, s_2, \ldots, s_n\}$ назовем значение выражения

$$h(s) = r(s_1) \oplus r(s_2) \oplus \ldots \oplus \ldots r(s_n)$$

Тогда вставка и удаление из хеша – это $h(s + s_{n+1}) = h(s) \oplus r(s_{n+1})$ и $h(s \setminus s_k) = h(s) \oplus r(s_k)$. Это очень простой и легко пишущийся хеш, рекомендую.

Способ 2: В прошлом способе мы не могли дважды добавить одно и то же число, это проблема. Теперь хешом множества $s = \{s_1, s_2, \ldots, s_n\}$ назовем следующую величину:

$$h(s) = (Q^{s_1} + Q^{s_2} + \ldots + Q^{s_n})\%P$$

Тут тоже нечего комментировать: добавление элемента во множество — это $+=$, удаление – это $-=$.

1.3 Хеши для табличек

Аналогично полиномиальному хешу для строки введем полиномиальный хеш для таблицы:

$$h\left(\begin{bmatrix} a_{11} & a_{12} & \ldots & a_{1n} \\ a_{21} & a_{22} & \ldots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \ldots & a_{mn} \end{bmatrix}\right) = \sum_{ij} Q_1^i Q_2^j a_{ij}$$

Не сложно заметить, что с помощью этого хеша лего вычислять хеши подматриц, если сохранить все префиксные хеши, всё делается аналогично обычному полиномиальному хешу.

1.4 Взлом хеша

Об этой теме можно говорить очень много, буквально этой теме посвящен огромный раздел в Компьютерной Безопасности. О прикладных аспектах (с точки зрения ломания решений на кф) можно прочитать в этом посте на codeforces.