

Хеширование в задачах на строки

Постановка задачи

Имеются две строки: S и T . Проверить, входит ли строка T в S как подстрока.

Наивное решение

```
m = |T|
n = |S|
for i = 0 .. n - m:
    if S[i .. i + m - 1] == T:
        всё хорошо
```

Наивное решение

```
m = |T|
n = |S|
for i = 0 .. n - m:
    if S[i .. i + m - 1] == T:
        всё хорошо
```

Время работы – $O(nm)$

$m = |T|$

$n = |S|$

$S \rightarrow h(S)$

$T \rightarrow h(T)$

for $i = 0 \dots n - m$:

if $h(S[i \dots i + m - 1]) == h(T)$:

 всё хорошо

```
m = |T|
n = |S|
S -> h(S)
T -> h(T)
for i = 0 .. n - m:
    if h(S[i .. i + m - 1]) == h(T):
        всё хорошо
```

$h(S)$ – хеш-функция от строки, которая вычисляется один раз (за один пробег по строке), хеши подстрок вычисляются за $O(1)$, итоговое время работы – $O(n + m)$

Вычисление хеш-функции строки

Есть 2 основных способа вычисления значений хеш-функции:

$$1) \quad h(S) = s_0 * P^0 + s_1 * P^1 + \dots + s_{n-1} * P^{n-1}$$

$$2) \quad h(S) = s_0 * P^{n-1} + s_1 * P^{n-2} + \dots + s_{n-1} * P^0$$

P – основание хеша

Вычисление хеш-функции строки

Есть 2 основных способа вычисления значений хеш-функции:

$$1) \quad h(S) = s_0 * P^0 + s_1 * P^1 + \dots + s_{n-1} * P^{n-1}$$

$$2) \quad h(S) = s_0 * P^{n-1} + s_1 * P^{n-2} + \dots + s_{n-1} * P^0$$

P – основание хеша

Так как значение $h(S)$ может быть достаточно большим, то её значение вычисляется по некоторому модулю ($h(S) \% \text{mod}$)

Коллизия

Коллизией называют две различные строки S и T , хеш-функции которых совпадают, при этом строки различаются.

Цель – выбрать модуль и основание таким образом, чтобы минимизировать возможность коллизий.

Выбор модуля и основания

Выбор модуля и основания

Операция взятия остатка от деления затратна по времени, поэтому имеет смысл использовать модуль типа `int`

Выбор модуля и основания

Операция взятия остатка от деления затратна по времени, поэтому имеет смысл использовать модуль типа `int`

Альтернативный подход – использование переполнения типов `int` и `long long` (что аналогично взятию остатков по модулям 2^{32} и 2^{64})

Выбор модуля и основания

Операция взятия остатка от деления затратна по времени, поэтому имеет смысл использовать модуль типа `int`

Альтернативный подход – использование переполнения типов `int` и `long long` (что аналогично взятию остатков по модулям 2^{32} и 2^{64})

Минус – хеши, использующие переполнение, дают коллизии на строках Туэ-Морса

Строки Туэ-Морса

Строки, получаемые по правилу:

$$1) S_1 = 'A'$$

$$2) S_i = S_{i-1} + \text{not}(S_{i-1})$$

называют строками Туэ-Морса

$\text{not}(S)$ – строка, получаемая из исходной заменой всех символов 'A' на символ 'B'

Строки Туэ-Морса

Первые несколько строк:

1) A

2) AB

3) ABBA

4) ABBAABAAB

5) ABBAABAABBAABAABBA

6) ...

Строки Туэ-Морса

Первые несколько строк:

- 1) A
- 2) AB
- 3) ABBA
- 4) ABBAABAAB
- 5) ABBAABAABBAABAABBA
- 6) ...

Подробнее:

<https://codeforces.com/blog/entry/4898>

Выбор модуля

Операция взятия остатка от деления затратна по времени, поэтому имеет смысл использовать модуль типа `int`.

Выбор модуля

Операция взятия остатка от деления затратна по времени, поэтому имеет смысл использовать модуль типа `int`.

Чтобы уменьшить вероятность коллизий, будем выбирать простой модуль

Выбор модуля

Операция взятия остатка от деления затратна по времени, поэтому имеет смысл использовать модуль типа `int`.

Чтобы уменьшить вероятность коллизий, будем выбирать простой модуль (число порядка 10^9).

Выбор модуля

Минус использования одного простого модуля:
Достаточно сгенерировать $O(\sqrt{n})$ случайных строк
(n – размер модуля), чтобы получить коллизию

Выбор модуля

Минус использования одного простого модуля:
Достаточно сгенерировать $O(\sqrt{n})$ случайных строк
(n – размер модуля), чтобы получить коллизию

Объяснение – парадокс дней рождения.

Парадокс дней рождения

В группе, состоящей из 23 или более человек, вероятность совпадения дней рождения (число и месяц) хотя бы у двух людей превышает 50 %. Например, 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

Выбор модуля

Выход из ситуации – использование двух простых модулей.

Возможные варианты:

$10^9 + 7$, $10^9 + 9$, $10^9 + 33$, $10^9 + 123$, 179179179

Выбор модуля

Выход из ситуации – использование двух простых модулей.

Возможные варианты:

$10^9 + 7$, $10^9 + 9$, $10^9 + 33$, $10^9 + 123$, 179179179

Пара модулей тоже ломается, если взломщику известно основание хеша

Выбор модуля

Выход из ситуации – использование двух простых модулей.

Возможные варианты:

$10^9 + 7$, $10^9 + 9$, $10^9 + 33$, $10^9 + 123$, 179179179

Пара модулей тоже ломается, если взломщику известно основание хеша

Подробнее:

<https://codeforces.com/blog/entry/4900>

<https://pastebin.com/JfTEUwCe>

Выбор основания

Выбор основания

$$1) \ h(S) = s_0 * P^0 + s_1 * P^1 + \dots + s_{n-1} * P^{n-1}$$

$$2) \ h(S) = s_0 * P^{n-1} + s_1 * P^{n-2} + \dots + s_{n-1} * P^0$$

P – основание хеша

Выбор основания

$$1) h(S) = s_0 * P^0 + s_1 * P^1 + \dots + s_{n-1} * P^{n-1}$$

$$2) h(S) = s_0 * P^{n-1} + s_1 * P^{n-2} + \dots + s_{n-1} * P^0$$

P – основание хеша

Значение P стоит брать таким, чтобы оно было больше $|\alpha|$, $|\alpha|$ – размер алфавита (количество различных символов, которые могут использоваться в строках)

Выбор основания

$$1) h(S) = s_0 * P^0 + s_1 * P^1 + \dots + s_{n-1} * P^{n-1}$$

$$2) h(S) = s_0 * P^{n-1} + s_1 * P^{n-2} + \dots + s_{n-1} * P^0$$

P – основание хеша

Значение P стоит брать таким, чтобы оно было больше $|\alpha|$, $|\alpha|$ – размер алфавита (количество различных символов, которые могут использоваться в строках).

P нужно брать случайным образом.

Выбор основания

$$1) h(S) = s_0 * P^0 + s_1 * P^1 + \dots + s_{n-1} * P^{n-1}$$

$$2) h(S) = s_0 * P^{n-1} + s_1 * P^{n-2} + \dots + s_{n-1} * P^0$$

P – основание хеша

Значение P стоит брать таким, чтобы оно было больше $|\alpha|$, $|\alpha|$ – размер алфавита (количество различных символов, которые могут использоваться в строках).

P нужно брать случайным образом, причем оно должно быть простым.

Выбор направления хеша

$$1) h(S) = s_0 * p^0 + s_1 * p^1 + \dots + s_{n-1} * p^{n-1}$$

$$2) h(S) = s_0 * p^{n-1} + s_1 * p^{n-2} + \dots + s_{n-1} * p^0$$

Будет реализован второй способ, так как в этом случае удобнее брать хеши подстрок

Реализация

Заведем массив h , определяемый следующим образом:

$$h[i] = s[0] * p^{i-1} + s[1] * p^{i-2} + \dots + s[i - 1]$$

Реализация

Заведем массив h , определяемый следующим образом:

$$h[i] = s[0] * P^{i-1} + s[1] * P^{i-2} + \dots + s[i - 1]$$

$$h[0] = 0$$

$$h[1] = s[0]$$

$$h[2] = s[0] * P + s[1]$$

$$h[3] = s[0] * P^2 + s[1] * P + s[2]$$

...

Реализация

Заведем массив h , определяемый следующим образом:

$$h[i] = s[0] * P^{i-1} + s[1] * P^{i-2} + \dots + s[i-1]$$

$$h[0] = 0$$

$$h[1] = s[0]$$

$$h[2] = s[0] * P + s[1] = h[0] * P + s[1]$$

$$h[3] = s[0] * P^2 + s[1] * P + s[2]$$

...

Реализация

Заведем массив h , определяемый следующим образом:

$$h[i] = s[0] * P^{i-1} + s[1] * P^{i-2} + \dots + s[i-1]$$

$$h[0] = 0$$

$$h[1] = s[0]$$

$$h[2] = s[0] * P + s[1] = h[0] * P + s[1]$$

$$h[3] = s[0] * P^2 + s[1] * P + s[2] = P * (s[0] * P + s[1]) + s[2]$$

...

Реализация

Заведем массив h , определяемый следующим образом:

$$h[i] = s[0] * P^{i-1} + s[1] * P^{i-2} + \dots + s[i-1]$$

$$h[0] = 0$$

$$h[1] = s[0]$$

$$h[2] = s[0] * P + s[1] = h[0] * P + s[1]$$

$$h[3] = s[0] * P^2 + s[1] * P + s[2] = P * (s[0] * P + s[1]) + s[2] = P * h[2] + s[2]$$

...

Реализация

Заведем массив h , определяемый следующим образом:

$$h[i] = s[0] * P^{i-1} + s[1] * P^{i-2} + \dots + s[i - 1]$$

$$h[i] = h[i - 1] * P + s[i - 1]$$

Реализация

```
vector < pair <int, int> > p_pow;
const int mod1 = 1e9 + 7, mod2 = 1'000'000'009;
int p;

void make_pows(string &s){
    int n = s.size() + 1;
    p_pow.resize(n);
    p_pow[0] = {1, 1};
    for(int i = 1; i < n; ++i){
        p_pow[i].first = (p_pow[i - 1].first * 111 * p) % mod1;
        p_pow[i].second = (p_pow[i - 1].second * 111 * p) % mod2;
    }
}

void make_hashes(vector < pair <int, int> > &h, string &s){
    int n = s.size() + 1;
    h.resize(n);
    h[0] = {0, 0};
    for(int i = 1; i < n; ++i){
        h[i].first = (h[i - 1].first * 111 * p + s[i - 1]) % mod1;
        h[i].second = (h[i - 1].second * 111 * p + s[i - 1]) % mod2;
    }
}
```

Реализация

```
int main() {  
    string s, t;  
    cin >> s >> t;  
    int x = rand();  
    p = max(257, x + (x % 2 == 0));  
    vector < pair <int, int> > h_s, h_t;  
    make_pows(s);  
    make_hashes(h_s, s);  
    make_hashes(h_t, t);  
}
```

Хеш подстроки

Научимся искать хеш подстроки $h[l..r]$

Хеш подстроки

Научимся искать хеш подстроки $h[l..r]$

$h(s[l..r]) =$

Хеш подстроки

Научимся искать хеш подстроки $h[l..r]$

$$h(s[l..r]) = s[l] * P^{r-l} + s[l+1] * P^{r-l-1} + \dots + s[r]$$

Хеш подстроки

Научимся искать хеш подстроки $h[l..r]$

$$h(s[l..r]) = s[l] * P^{r-l} + s[l+1] * P^{r-l-1} + \dots + s[r]$$

Хеш подстроки

Научимся искать хеш подстроки $h[l..r]$

$$h(s[l..r]) = s[l] * P^{r-l} + s[l+1] * P^{r-l-1} + \dots + s[r]$$

$$h(s[0..r]) = h[r+1]$$

Хеш подстроки

Научимся искать хеш подстроки $h[l..r]$

$$h(s[l..r]) = s[l] * P^{r-l} + s[l+1] * P^{r-l-1} + \dots + s[r]$$

$$h(s[0..r]) = h[r+1]$$

$$h[r+1] = s[0] * P^r + s[1] * P^{r-1} + \dots + s[l-1] * P^{r-l+1} + \dots + s[r]$$

Хеш подстроки

Научимся искать хеш подстроки $h[l..r]$

$$h(s[l..r]) = s[l] * P^{r-l} + s[l+1] * P^{r-l-1} + \dots + s[r]$$

$$h(s[0..r]) = h[r+1]$$

$$h[r+1] = s[0] * P^r + s[1] * P^{r-1} + \dots + s[l-1] * P^{r-l+1} \\ + \dots + s[r]$$

$$s[l..r] =$$

Хеш подстроки

Научимся искать хеш подстроки $h[l..r]$

$$h(s[l..r]) = s[l] * P^{r-l} + s[l+1] * P^{r-l-1} + \dots + s[r]$$

$$h(s[0..r]) = h[r+1]$$

$$h[r+1] = s[0] * P^r + s[1] * P^{r-1} + \dots + s[l-1] * P^{r-l+1} + \dots + s[r]$$

$$s[l..r] = s[0..r] - s[0..l-1]$$

Хеш подстроки

Научимся искать хеш подстроки $h[l..r]$

$$h(s[l..r]) = s[l] * P^{r-l} + s[l+1] * P^{r-l-1} + \dots + s[r]$$

$$h(s[0..r]) = h[r+1]$$

$$h[r+1] = s[0] * P^r + s[1] * P^{r-1} + \dots + s[l-1] * P^{r-l+1} + \dots + s[r]$$

$$s[l..r] = s[0..r] - s[0..l-1]$$

$$h[l] = s[0] * P^{l-1} + s[1] * P^{l-2} + \dots + s[l-1]$$

Хеш подстроки

Научимся искать хеш подстроки $h[l..r]$

$$h(s[l..r]) = s[l] * P^{r-l} + s[l+1] * P^{r-l-1} + \dots + s[r]$$

$$h(s[0..r]) = h[r+1]$$

$$h[r+1] = s[0] * P^r + s[1] * P^{r-1} + \dots + s[l-1] * P^{r-l+1} + \dots + s[r]$$

$$s[l..r] = s[0..r] - s[0..l-1]$$

$$h[l] = s[0] * P^{l-1} + s[1] * P^{l-2} + \dots + s[l-1]$$

Хеш подстроки

Научимся искать хеш подстроки $h[l..r]$

$$h(s[l..r]) = s[l] * P^{r-l} + s[l+1] * P^{r-l-1} + \dots + s[r]$$

$$h(s[0..r]) = h[r+1]$$

$$h[r+1] = s[0] * P^r + s[1] * P^{r-1} + \dots + s[l-1] * P^{r-l+1} + \dots + s[r]$$

$$s[l..r] = s[0..r] - s[0..l-1]$$

$$h[l] = s[0] * P^{l-1} + s[1] * P^{l-2} + \dots + s[l-1]$$

Хеш подстроки

Научимся искать хеш подстроки $h[l..r]$

$$h(s[l..r]) = s[l] * P^{r-l} + s[l+1] * P^{r-l-1} + \dots + s[r]$$

$$h(s[0..r]) = h[r+1]$$

$$h[r+1] = s[0] * P^r + s[1] * P^{r-1} + \dots + s[l-1] * P^{r-l+1} + \dots + s[r]$$

$$s[l..r] = s[0..r] - s[0..l-1]$$

$$h[l] = s[0] * P^{l-1} + s[1] * P^{l-2} + \dots + s[l-1]$$

$$h(s[l..r]) =$$

Хеш подстроки

Научимся искать хеш подстроки $h[l..r]$

$$h(s[l..r]) = s[l] * p^{r-l} + s[l+1] * p^{r-l-1} + \dots + s[r]$$

$$h(s[0..r]) = h[r+1]$$

$$h[r+1] = s[0] * p^r + s[1] * p^{r-1} + \dots + s[l-1] * p^{r-l+1} + \dots + s[r]$$

$$s[l..r] = s[0..r] - s[0..l-1]$$

$$h[l] = s[0] * p^{l-1} + s[1] * p^{l-2} + \dots + s[l-1]$$

$$h(s[l..r]) = h[r+1] - h[l] * p^{r-l+1}$$

Нахождение хеша подстроки

```
pair <int, int> get_hash(vector < pair <int, int> > &h, int l, int r){
    pair <int, int> ans;
    ans.first = (h[r + 1].first - h[l].first * 111 * p_pow[r - l + 1].first) % mod1;
    if (ans.first < 0){
        ans.first += mod1;
    }
    ans.second = (h[r + 1].second - h[l].second * 111 * p_pow[r - l + 1].second) % mod2;
    if (ans.second < 0){
        ans.second += mod2;
    }
    return ans;
}
```

Сравнение хешей подстрок

```
pair <int, int> get_hash(vector < pair <int, int> > &h, int l, int r){
    pair <int, int> ans;
    ans.first = (h[r + 1].first - h[l].first * 111 * p_pow[r - l + 1].first) % mod1;
    if (ans.first < 0){
        ans.first += mod1;
    }
    ans.second = (h[r + 1].second - h[l].second * 111 * p_pow[r - l + 1].second) % mod2;
    if (ans.second < 0){
        ans.second += mod2;
    }
    return ans;
}

bool equal(vector < pair <int, int> > &h, int l1, int r1, int l2, int r2){
    if (r2 - l2 != r1 - l1){
        return 0;
    }
    return get_hash(h, l1, r1) == get_hash(h, l2, r2);
}
```

Алгоритм Рабина-Карпа

Имеются две строки: S и T . Проверить, входит ли строка T в S как подстрока.

Алгоритм Рабина-Карпа

Имеются две строки: S и T . Проверить, входит ли строка T в S как подстрока.

1) Найдем хеш строки T ($O(m)$)

Алгоритм Рабина-Карпа

Имеются две строки: S и T . Проверить, входит ли строка T в S как подстрока.

- 1) Найдем хеш строки T ($O(m)$)
- 2) Найдем хеш строки S ($O(n)$)

Алгоритм Рабина-Карпа

Имеются две строки: S и T . Проверить, входит ли строка T в S как подстрока.

- 1) Найдем хеш строки T ($O(m)$)
- 2) Найдем хеш строки S ($O(n)$)
- 3) Пробежимся с окном длины m по строке S . Если хеш одной из подстрок совпал с хешем строки T , то ответ – да, если не нашлось ни одной такой подстроки, ответ – нет ($O(n)$)

Алгоритм Рабина-Карпа

Имеются две строки: S и T . Проверить, входит ли строка T в S как подстрока.

- 1) Найдем хеш строки T ($O(m)$)
- 2) Найдем хеш строки S ($O(n)$)
- 3) Пробежимся с окном длины m по строке S . Если хеш одной из подстрок совпал с хешем строки T , то ответ – да, если не нашлось ни одной такой подстроки, ответ – нет ($O(n)$)

Итоговое время работы – $O(n + m)$

```
m = |T|
hT = hT[m]
n = |S|
for i = 0 .. n - m:
    if get_hash(i, i + m - 1) == hT:
        всё хорошо
```

Применения хешей

1. Лексикографическое сравнение подстрок/строк
2. Нахождение количества различных подстрок
3. Проверка подстроки/строки на палиндромность
4. Нахождение минимального циклического сдвига строки
5. Поиск наидлиннейшей общей подстроки

Сравнение строк

Даны две строки, определить какая из них
больше/меньше

Сравнение строк

Даны две строки, определить какая из них больше/меньше

1. Если строки совпадают в первых i символах, то они совпадают и на меньшем префиксе

Сравнение строк

Даны две строки, определить какая из них больше/меньше

1. Если строки совпадают в первых i символах, то они совпадают и на меньшем префиксе
2. Если строки не совпадают в первых i символах, то они не совпадают и на большем префиксе

Сравнение строк

Решение:

Сравнение строк

Решение:

бинарным поиском ищем самый первый символ, в котором строки не совпадают, и сравниваем их (следует учесть, что длины строк могут не совпадать)

Сравнение строк

Решение:

бинарным поиском ищем самый первый символ, в котором строки не совпадают, и сравниваем их (следует учесть, что длины строк могут не совпадать)

Время работы – $O(\log N)$, $N = \min(|S|, |T|)$

```

int cmp(vector < pair <int, int> > &h, string &s, int l1, int r1, int l2, int r2){
    // 0 - подстроки равны
    // 1 - первая больше
    // -1 - первая меньше
    int len1 = r1 - l1 + 1, len2 = r2 - l2 + 1, len = min(len1, len2);
    int l = 0, r = len + 1;
    while (r - l > 1){
        int m = (l + r) / 2;
        if (get_hash(h, l1, l1 + m - 1) == get_hash(h, l2, l2 + m - 1)){
            l = m;
        }
        else {
            r = m;
        }
    }
    if (l == len){ // префиксы полностью совпали
        if (len1 < len2){
            return -1;
        }
        if (len1 == len2){
            return 0;
        }
        if (len1 > len2){
            return 1;
        }
    }
    if (s[l1 + l] < s[l2 + l]){
        return -1;
    }
    return 1;
}

```

```

bool cmp(vector < pair <int, int> > &h, string &s, int l1, int r1, int l2, int r2){
    // 1 - первая подстрока меньше второй
    // 0 - в противном случае
    int len1 = r1 - l1 + 1, len2 = r2 - l2 + 1, len = min(len1, len2);
    int l = 0, r = len + 1;
    while (r - l > 1){
        int m = (l + r) / 2;
        if (get_hash(h, l1, l1 + m - 1) == get_hash(h, l2, l2 + m - 1)){
            l = m;
        }
        else {
            r = m;
        }
    }
    if (l == len){ // префиксы полностью совпали
        return len1 < len2;
    }
    return s[l1 + l] < s[l2 + l];
}

```

Минимальный циклический сдвиг

Дана строка S . Найти её лексикографически минимальный циклический сдвиг.

Минимальный циклический сдвиг

Дана строка S . Найти её лексикографически минимальный циклический сдвиг.

Пример: `ісрс`

Минимальный циклический сдвиг

Дана строка S . Найти её лексикографически минимальный циклический сдвиг.

Пример: `ісрс`

Циклические сдвиги:

`ісрс`

`сіср`

`рсіс`

`срсі`

Минимальный циклический сдвиг

Дана строка S. Найти её лексикографически минимальный циклический сдвиг.

Пример: ісрс

Циклические сдвиги:

ісрс

сіср

рсіс

срсі

Ответ: сіср

Минимальный циклический сдвиг

Решение:

Минимальный циклический сдвиг

Решение:

1. Положим $n = |S|$ и удвоим строку S .

Минимальный циклический сдвиг

Решение:

1. Положим $n = |S|$ и удвоим строку S .
2. Пробежимся по S с окном размера n , взяв первые n символов в качестве минимального сдвига

Минимальный циклический сдвиг

Решение:

1. Положим $n = |S|$ и удвоим строку S .
2. Пробежимся по S с окном размера n , взяв первые n символов в качестве минимального сдвига
3. Будем сравнивать минимальный сдвиг с текущим с помощью хешей

Минимальный циклический сдвиг

Решение:

1. Положим $n = |S|$ и удвоим строку S .
2. Пробежимся по S с окном размера n , взяв первые n символов в качестве минимального сдвига
3. Будем сравнивать минимальный сдвиг с текущим с помощью хешей
4. Если текущий сдвиг меньше минимального, обновим ответ

Минимальный циклический сдвиг

Решение:

1. Положим $n = |S|$ и удвоим строку S .
2. Пробежимся по S с окном размера n , взяв первые n символов в качестве минимального сдвига
3. Будем сравнивать минимальный сдвиг с текущим с помощью хешей
4. Если текущий сдвиг меньше минимального, обновим ответ
5. Время работы – $O(N \log N)$, $N = |S|$

Проверка на палиндромность

Проверка на палиндромность

1. Найдем хеши для S и $r(S)$, $r(S)$ – перевёрнутая строка S .

Проверка на палиндромность

1. Найдем хеши для S и $r(S)$, $r(S)$ – перевёрнутая строка S .
2. Чтобы проверить строку на палиндромность, достаточно сравнить хеши строк S и $r(S)$ ($h[n]$ и $rh[n]$)

Проверка на палиндромность

1. Найдем хеши для S и $r(S)$, $r(S)$ – перевернутая строка S .
2. Чтобы проверить строку на палиндромность, достаточно сравнить хеши строк S и $r(S)$ ($h[n]$ и $rh[n]$)
3. Для проверки на палиндромность подстроки $[l..r]$ строки S достаточно сравнить хеши $get_hash(l, r)$ и $get_rhash(n - r - 1, n - l - 1)$

Нахождение подпалиндромов строки

1. Найдем хеши для S и $r(S)$, $r(S)$ – перевернутая строка S .

Нахождение подпалиндромов строки

1. Найдем хеши для S и $r(S)$, $r(S)$ – первернутая строка S .
2. Чтобы найти все подпалиндромы нечетной длины, пробежимся по строке и для каждой позиции найдем такое максимальное l , что строка $s[i-l], \dots, s[i], \dots, s[i+l]$ – палиндром

Нахождение подпалиндромов строки

1. Найдем хеши для S и $r(S)$, $r(S)$ – перевернутая строка S .
2. Чтобы найти все подпалиндромы нечетной длины, пробежимся по строке и для каждой позиции найдем такое максимальное l , что строка $s[i-l], \dots, s[i], \dots, s[i+l]$ – палиндром
3. Аналогично для нахождения подпалиндромов четной длины рассмотрим все пары позиций $(i, i+1)$ и найдем такое максимальное l , что строка $s[i-l], \dots, s[i], s[i+1], s[i+l+1]$ – палиндром

Нахождение подпалиндромов строки

1. Найдем хеши для S и $r(S)$, $r(S)$ – перевернутая строка S .
2. Чтобы найти все подпалиндромы нечетной длины, пробежимся по строке и для каждой позиции найдем такое максимальное l , что строка $s[i-l], \dots, s[i], \dots, s[i+l]$ – палиндром
3. Аналогично для нахождения подпалиндромов четной длины рассмотрим все пары позиций $(i, i+1)$ и найдем такое максимальное l , что строка $s[i-l], \dots, s[i], s[i+1], s[i+l+1]$ – палиндром
4. Время работы – $O(N \log N)$, $N = |S|$.

Нахождение числа подстрок

Нахождение числа подстрок

Дана строка. Найти количество её различных подстрок.

Нахождение числа подстрок

Решение:

Нахождение числа подстрок

Решение:

1. Будем рассматривать подстроки в порядке возрастания длины

Нахождение числа подстрок

Решение:

1. Будем рассматривать подстроки в порядке возрастания длины
2. На каждом шаге заведем сет, в котором будем хранить количество различных хешей среди подстрок фиксированной длины

Нахождение числа подстрок

Решение:

1. Будем рассматривать подстроки в порядке возрастания длины
2. На каждом шаге заведем сет/мап, в котором будем хранить количество различных хешей среди подстрок фиксированной длины
3. Ответ – сумма размеров сетов/мапов по всем итерациям алгоритма

Нахождение числа подстрок

Решение:

1. Будем рассматривать подстроки в порядке возрастания длины
2. На каждом шаге заведем сет/мап, в котором будем хранить количество различных хешей среди подстрок фиксированной длины
3. Ответ – сумма размеров сетов/мапов по всем итерациям алгоритма
4. Время работы – $O(N^2 \log N)$

Структуры данных, использующие хеши в C++

`unordered_set`

`unordered_map`

Плюсы – добавление элементов и их поиск
занимают $O(1)$ времени

Описание работы:

http://www.cplusplus.com/reference/unordered_set/

http://www.cplusplus.com/reference/unordered_map/

Вопросы?

Ссылки

Тест, ломающий хеши с e-maхх:

<https://codeforces.com/blog/entry/4898>

Взлом хешей по двум модулям:

<https://codeforces.com/blog/entry/4900>

<https://pastebin.com/JfTEUwCe>

Реализация базовых функций:

<https://ideone.com/0157OI>

unordered_set/map:

http://www.cplusplus.com/reference/unordered_set/

http://www.cplusplus.com/reference/unordered_map/

Примечание:

В задачах А, В, С требуется работа с файлами