Алгоритм решения задачи "Три вхождения"

Антон Дроздовский

1 Условие задачи

Input file name standard input

Output file name standard output

Time limit 2 s

Memory limit 256 MB

Условие

Дана строка s, состоящая из строчных латинских букв.

Необходимо найти максимальную по длине строку w, которая встречается в s хотя бы трижды без перекрытий.

Input

Первая и единственная строка входного файла содержит одну строку s ($3 \le |s| \le 10\,000$).

Output

Выведите одну строку — ответ на задачу. Если ответов несколько, выведите любой.

Гарантируется, что существует такая непустая строка, которая встречается хотя бы три раза в строке s.

Examples

standard input	standard output
eetuppeetuppetupp	etupp
bcacbacbaabacbbbacac	acb

2 Описание алгоритма

Строим суффиксный массив. По суффиксному массиву строим LCP (Longest Common Prefix) массив. LCP массив содержит на i-ой позиции длину общего префикса i-ого и $i+1$-ой суффиксов в суффиксном массиве.

Будем использовать дихотомию по длине суффикса. Это позволит в худшем случае рассмотреть $\log \frac{|s|}{3}$ суффиксов вместо $\frac{|s|}{3}$.

Имея всё на руках, сравниваем значения LCP массива с длиной рассматриваемого суффикса. Если дважды подряд у LCP массива значения не меньше рассматриваемой длины, то это означает, что у строки имеется три суффикса с общим префиксом, потенциальным на три вхождения без пересечений. Но, к сожалению, на пересечение нужно проверить. Для этого используем алгоритм Кнута-Морриса-Пратта.

3 Решение задачи на C++

```
#include <algorithm>
#include <ios>
#include <iostream>
#include <string>
#include <utility>
#include <vector>

std::vector<int> buildSuffixArray(const std::string& s) {
        int n = s.size();
        std::vector<int> suffixArray(n), classes(n), c(n),
            cnt(std::max(n, 256), 0);

        for (int i = 0; i < n; ++i) ++cnt[s[i]];
        for (int i = 1; i < 256; ++i) cnt[i] += cnt[i - 1];
        for (int i = n - 1; i >= 0; --i) suffixArray[--cnt[s
            [i]]] = i;

        classes[suffixArray[0]] = 0;
        int numClasses = 1;
        for (int i = 1; i < n; ++i) {
                if (s[suffixArray[i]] != s[suffixArray[i -
                    1]]) ++numClasses;
                classes[suffixArray[i]] = numClasses - 1;
        }

        std::vector<int> temp(n);
        for (int h = 0; (1 << h) < n; ++h) {
                for (int i = 0; i < n; i++) {
                        temp[i] = suffixArray[i] - (1 << h);
                        if (temp[i] < 0) temp[i] += n;
                }

                std::fill(cnt.begin(), cnt.begin() +
                    numClasses, 0);
                for (int i = 0; i < n; ++i) ++cnt[classes[
                    temp[i]]];
                for (int i = 1; i < numClasses; i++) cnt[i]
                    += cnt[i - 1];
```

```
                for (int i = n - 1; i >= 0; i--) suffixArray
                    [--cnt[classes[temp[i]]]] = temp[i];

                temp[suffixArray[0]] = 0;
                numClasses = 1;
                for (int i = 1; i < n; ++i) {
                        std::pair<int, int> cur = { classes[
                            suffixArray[i]], classes[(
                            suffixArray[i] + (1 << h)) % n]
                            };
                        std::pair<int, int> prev = { classes
                            [suffixArray[i - 1]], classes[(
                            suffixArray[i - 1] + (1 << h)) %
                            n] };
                        if (cur != prev) ++numClasses;
                        temp[suffixArray[i]] = numClasses -
                            1;
                }
                classes.swap(temp);
        }

        return suffixArray;
}

std::vector<int> buildLCP(const std::string& s, const std::
    vector<int>& suffixArray) {
        int n = s.size();
        std::vector<int> rank(n), lcp(n);

        for (int i = 0; i < n; ++i) rank[suffixArray[i]] = i
            ;

        int h = 0;
        for (int i = 0; i < n; i++) {
                if (rank[i] > 0) {
                        int j = suffixArray[rank[i] - 1];
                        while (i + h < n && j + h < n && s[i
                             + h] == s[j + h]) ++h;
                        lcp[rank[i]] = h;
                        if (h > 0) --h;
                }
        }

        return lcp;
}

bool check(const std::string& s, const std::vector<int>&
    suffixArray, const std::vector<int>& lcp, int len, std::
    string& result) {
        int count = 0;
```

```
        int counter = 0;
        std::string temp;
        for (int i = 1; i < s.size(); i++) {
                if (lcp[i] >= len) {
                        ++count;
                        if (count == 2) {
                                temp = s.substr(suffixArray[
                                    i], len);
                                temp += '#' + s;
                                std::vector<int>
                                    prefixFunction(temp.
                                    length());
                                prefixFunction[0] = 0;
                                int k;
                                for (int i = 1; i < temp.
                                    length(); ++i) {
                                        k = prefixFunction[i
                                             - 1];
                                        while (k > 0 && temp
                                            [i] != temp[k]) {
                                                k =
                                                    prefixFunction
                                                    [k - 1];
                                        }

                                        if (temp[i] == temp[
                                            k]) {
                                                ++k;
                                        }

                                        prefixFunction[i] =
                                            k;
                                }

                                for (int i = len; i < temp.
                                    length(); ++i) {
                                        if (prefixFunction[i
                                            ] == len) {
                                                ++counter;
                                                if (counter
                                                    == 3) {
                                                        break
                                                            ;
                                                }

                                                i += len -
                                                    1;
                                        }
                                }

```

```
                                if (counter >= 3) {
                                        result = s.substr(
                                            suffixArray[i],
                                            len);

                                        return true;
                                }
                                else {
                                        counter = 0;
                                }
                        }
                }
                else {
                        count = 0;
                }
        }
        return false;
}

int main() {
        std::ios_base::sync_with_stdio(false);
        std::cin.tie(nullptr);

        std::string s;
        std::cin >> s;

        s += '$';
        std::vector<int> suffixArray = buildSuffixArray(s);
        std::vector<int> lcp = buildLCP(s, suffixArray);

        int left = 1, right = s.size() / 3;
        std::string result;

        while (left <= right) {
                int mid = (left + right) / 2;
                std::string temp;
                if (mid <= (s.size() - 1) / 3 && check(s,
                    suffixArray, lcp, mid, temp)) {
                        result = temp;
                        left = mid + 1;
                }
                else {
                        right = mid - 1;
                }
        }

        std::cout << result << '\n';

        return 0;
}
```

4 Оценка временной сложности

Построение суффиксного массива

Суффиксный массива можно построить за линейное время используя алгоритм Карккайнена-Сандерса. В моём же коде строю суффиксный массив, используя классы эквивалентности. $O(|s| \log |s|)$.

Построение LCP массива

Используя суффиксный массив, LCP массив строится за линейное время. $O(|s|)$.

Нахождение нужного суффикса

Дихотомия вызывает функцию, в которой имеется цикл, проходящий по всему массиву LCP. При нахождении потенциально нужной подстроки используется метод std::string::substr и алгоритм Кнута-Морриса-Пратта, которые работают за линейное время. $O(|s|^2 \log |s|)$.

Итого

$O(|s|^2 \log |s|)$

5 Оценка сложности по памяти

LCP массив, суффиксный массив, суффиксы, строка: $O(|s|)$