Строки

1 A. Мультимножество Василия

Разбор

Общая идея решения заключается в построении бора для двоичных представлений чисел. Это позволит нам удобно находить наилучший возможный XOR. Добавление чисел в бор реализуется тривиально. Как искать ответ? Получим двоичное представление числа x и для каждого его бита будем пытаться совершать переход в противоположный бит. Если такого перехода нет, делаем переход по умолчанию.

Асимптотика решения: $O(\log(MAXN) \times q)$

Решение

```
#define ll long long int

#include <math.h>
#include <iostream>
#include <algorithm>
#include <vector>
#include <map>
#include <set>
#include <unordered_map>
#include <unordered_set>

using namespace std;

struct TrieNode {
    char v;
    ll cnt = 0;
    ll terminal = 0;
    TrieNode* z = 0; // 0
    TrieNode* o = 0; // 1

    TrieNode(char v) {
        this->v = v;
    }
    TrieNode* add(char v) {
        if (v == '0') {
            if (!this->z) {
                this->z = new TrieNode(v);
            }
            this->z->cnt++;
            return this->z;
        }
        if (!this->o) {
            this->o = new TrieNode(v);
        }
        this->o->cnt++;
        return this->o;
    }
    TrieNode* get(char v) {
        if (v == '0' && this->z) {
            return this->z;
        }
        if (v == '1' && this->o) {
            return this->o;
        }
        return 0;
    }
};

struct Trie {
```

```
TrieNode* root = new TrieNode('b');

string get_bits(ll num) {
    string s = "00000000000000000000000000000000"; // 31
    ll idx = s.size() - 1;
    while (num) {
        s[idx--] = '0' + (num % 2);
        num /= 2;
    }
    return s;
}

ll get_num(vector<char> bits) {
    ll res = 0;
    for (int i = 0; i < bits.size(); ++i) {
        res += (bits[i] - '0') * (1 << (bits.size() - i - 1));
    }
    return res;
}

bool has(string bits) {
    auto node = this->root;
    for (auto bit: bits) {
        node = node->get(bit);
        if (!node || !node->cnt) {
            return 0;
        }
    }
    return node->terminal;
}

void add(ll num) {
    auto aligned_bits = this->get_bits(num);
    auto node = this->root;
    for (auto bit: aligned_bits) {
        node = node->add(bit);
    }
    node->terminal++;
}

void del(ll num) {
    auto aligned_bits = this->get_bits(num);
    if (!this->has(aligned_bits)) {
        return;
    }
    auto node = this->root;
    for (int i = 0; i < aligned_bits.size(); ++i) {
        auto bit = aligned_bits[i];
        auto prev_node = node;
        node = prev_node->get(bit);
        node->cnt--;
        if (i == aligned_bits.size() - 1) {
            node->terminal--;
        }
        if (!node->cnt) {
            if (prev_node->o == node) {
                prev_node->o = 0;
            } else {
                prev_node->z = 0;
            }
            delete node;
            return;
        }
    }
}

ll xor_(ll num) {
    auto node = this->root;
    if (!node->z && !node->o) {
        return num;
```

```
        }
        auto res_bits = vector<char>();
        auto bits = this->get_bits(num);
        for (auto bit: bits) {
            auto prev_node = node;
            auto n_bit = bit == '0' ? '1' : '0';
            node = prev_node->get(n_bit);
            if (!node) {
                node = prev_node->get(bit);
            }
            res_bits.push_back(node->v);
        }
        return max(this->get_num(res_bits) ^ num, num);
    }
};

int main() {
    cin.tie(0);cout.tie(0);
    ios_base::sync_with_stdio(0);

    ll q, v;
    char op;
    cin >> q;
    auto t = Trie();
    while (q--) {
        cin >> op >> v;
        if (op == '+') {
            t.add(v);
        } else if (op == '-') {
            t.del(v);
        } else {
            cout << t.xor_(v) << endl;
        }
    }
}
```

2 B. A-функция от строчки

Разбор

Соединим обе строки друг с другом так, чтобы справа была перевёрнутая исходная строка: SS^R. Затем посчитаем z-функцию для всей строки, ответ будет лежать на отрезке от m до $n-1$ в обратном порядке.

Асимптотика решения: $O(N)$

Решение

```
n = int(input())
s = input()
m = 2 * n
ss = s + s[::-1]
z = [0] * m
l, r = 0, 0
for i in range(1, m):
    k = max(0, min(r - i, z[i - l]))
    while i + k < m and ss[k] == ss[i + k]:
        k += 1
    z[i] = k
    if i + k > r:
        l = i
        r = l + k
print(*z[m:n-1:-1])
```

3 C. Сортировка слов

Разбор

Выпишем все слова и построим по ним бор. Затем пройдёмся поиском в глубину по бору, выполняя переходы в лексикографическом порядке и выводя накопленное слово при достижении терминальной вершины.

Для этой задачи также существует простое решение, не использующее идею построения бора.

Асимптотика решения: $O(\sum_w |w| + N)$

Решение

```
#define ll long long int

#include <math.h>
#include <iostream>
#include <algorithm>
#include <vector>
#include <map>
#include <set>
#include <unordered_map>
#include <unordered_set>

using namespace std;

struct TrieNode {
    char v;
    ll cnt = 0;
    ll terminal = 0;
    unordered_map<char, TrieNode*> to  = unordered_map<char, TrieNode*>();

    TrieNode(char v) {
        this->v = v;
    }
    TrieNode* add(char v) {
        if (!this->to.count(v)) {
            this->to[v] = new TrieNode(v);
        }
        this->to[v]->cnt++;
        return this->to[v];
    }
};

struct Trie {
    TrieNode* root;

    Trie() {
        this->root = new TrieNode(0);
    }
    void add(string s) {
        auto node = this->root;
        for (char c: s) {
            node = node->add(c);
        }
        node->terminal++;
    }
};

int solve(TrieNode* n, string &w, string const &s, int pos = 0) {
    for (int i = 0; i < n->terminal; ++i) {
        while (s[pos] == '.') {
            cout << s[pos++];
        }
        cout << w;
        while (pos < s.size() && s[pos] != '.') {
            pos++;
        }
    }
```

```
    while (pos < s.size() && s[pos] == '.') {
        cout << s[pos++];
    }
    for (int i = 0; i < 26; ++i) {
        char c = 'a' + i;
        if (!n->to.count(c)) {
            continue;
        }
        auto next = n->to[c];
        w.push_back(c);
        pos = solve(next, w, s, pos);
        w.pop_back();
    }
    return pos;
}

int main() {
    string s;
    cin >> s;
    string w = "";
    auto t = Trie();
    for (int i = 0; i < s.size(); ++i) {
        if (s[i] != '.') {
            w += s[i];
        } else if (w != "") {
            t.add(w);
            w = "";
        }
    }
    if (w != "") {
        t.add(w);
        w = "";
    }
    solve(t.root, w, s);
    cout << endl;
}
```

4 D. Период строки

Разбор

Предположим, что период строки s равен некоторому значению p и нам нужно быстро проверить, верно ли это. Как мы можем это сделать? Давайте сравним значения z-функции подстрок $s_{0,n-p}$ и $s_{p,n}$. Само значение p находится очень просто - будем считать, что это максимальное значение z-функции.

Асимптотика решения: $O(N)$

Решение

```
s = input()
n = len(s)
z = [0] * n
l, r = 0, 0
max_z_i = 0
for i in range(1, n):
    z[i] = max(0, min(r - i, z[i - l]))
    while i + z[i] < n and s[z[i]] == s[i + z[i]]:
        z[i] += 1
    if i + z[i] > r:
        l = i
        r = l + z[i]
    if z[i] > z[max_z_i]:
        max_z_i = i

def period():
    d = z[max_z_i]
```

```
    if z[d] == n - d and n % (n - d) == 0:
        return n // (n - d)
    return 1

print(period())
```

5 E. Подпалиндромы

Разбор

Здесь требовалось реализовать алгоритм Манакера. Обратите внимание, что на emaxx-algo до сих пор существует ошибка в решении. Самый простой способ подсчета числа подпалиндромов, который не требует разделения подсчета подпалиндромов четной и нечетной длины, состоит в следующем:

1. Добавим между каждой парой символов дополнительный символ
2. Для каждого символа в строке будем вести подсчет подпалиндромов нечетной длины с текущим символом в середине
3. В зависимости от того, какой символ на данный момент в середине, будем брать отступ в 1 или 2 элемента от середины. В конце мы получим чистую сумму подпалиндромов, к которой требуется только добавить N, чтобы учесть все подпалиндромы длины 1.

Асимптотика решения: $O(N)$

Решение

```
def count(s):
    n = len(s)
    cnt = [0] * n
    l, r = 0, 0
    for i in range(n):
        shift = 1 if s[i] == '#' else 2
        k = max(0, min(cnt[2 * l - i], r - i))
        rad = shift + 2 * k
        while i - rad >= 0 and i + rad < n and s[i + rad] == s[i - rad]:
            k += 1
            rad = shift + 2 * k
        cnt[i] = k
        if i + k > r:
            l = i
            r = l + k
    return sum(cnt)

s_raw = input()
s = ''
for i in range(len(s_raw)):
    s += s_raw[i] + '#'
cnt = count(s[:-1])
print(cnt + len(s_raw))
```

6 F. Анаграммные подстроки

Разбор

От нас требуется найти такое возможное соответствие букв из одной строки буквам другой строки, что если мы для какой-то позиции установили соответствие букв, то это соответствие должно сохраниться и для других позиций. Идея: заменим буквы на расстояние до предыдущего вхождения этой буквы в строку. Если это первое вхождение, то заменим на INF. При этом сделаем это не для каждой строки в отдельности, а для их конкатенации: $s\#t$. Затем посчитаем z-функцию на получившейся строке с изменённой функцией сравнения символов:

1. Если символы равны (т.е. равны позиции их предыдущих вхождений) , то увеличим значение z-функции на 1
2. Если символы не равны, то один из них должен быть ещё не рассмотренным символом из искомой подстроки. Почему так? Потому что если мы уже ранее видели такой символ в подстроке, но не видели в сообщении, то такая подстрока не может являться частью сообщения. Если поставленное условие выполняется, увеличим значение z-функции на 1

Асимптотика решения: $O(|S| + |T|)$

Решение

```
INF = 10 ** 9 + 7

s = input()
t = input()

dist = []
char_pos = {}
for i in range(len(t)):
    c = t[i]
    if c not in char_pos:
        dist.append(INF)
    else:
        dist.append(i - char_pos[c])
    char_pos[c] = i
dist.append(-1)
char_pos.clear()
for i in range(len(s)):
    c = s[i]
    if c not in char_pos:
        dist.append(INF)
    else:
        dist.append(i - char_pos[c])
    char_pos[c] = i

n = len(dist)
z = [0] * n
l, r = 0, 0
max_z_i = 0
for i in range(1, n):
    z[i] = max(0, min(r - i, z[i - l]))
    while i + z[i] < n and (dist[z[i]] == dist[i + z[i]] or dist[z[i]] == INF and i + z[i] -
    dist[i + z[i]] < i):
        z[i] += 1
    if i + z[i] > r:
        l = i
        r = l + z[i]
    if z[i] > z[max_z_i]:
        max_z_i = i

ans = []
for i in range(len(t) + 1, n):
    if z[i] == len(t):
        ans.append(i - len(t))
print(len(ans))
print(*ans)
```

7 G. Поиск подстроки

Разбор

В этой задаче было необходимо найти все вхождения строки t длиной m в качестве подстроки в строку s длиной n. Воспользуемся алгоритмом Кнута-Морриса-Пратта - создадим строку $t\#s$, построим префикс-функцию и найдём все индексы, в которых значение π-функции равно длине строки t.

Асимптотика решения: $O(N + M)$

Решение

```
s = input()
t = input()
st = t + '#' + s
n, m = len(st), len(t)
p = [0] * n
ans = []
for i in range(1, n):
    j = p[i - 1]
    while j > 0 and st[i] != st[j]:
        j = p[j - 1]
    if st[i] == st[j]:
        j += 1
    p[i] = j
    if p[i] == m:
        ans.append(i - 2 * m)
print(*ans)
```

8 H. Словарь

Разбор

Построим бор на заданном нам словаре. Пользуясь тем, что длины слов в словаре достаточно маленькие, пойдём по строке и для каждой позиции проверим, начинается ли с неё какое-то слово из словаря.

Асимптотика решения: $O(\sum_w |w| + \max |w| \times |S|)$

Решение

```
#define ll long long int

#include <math.h>
#include <iostream>
#include <algorithm>
#include <vector>
#include <map>
#include <set>
#include <unordered_map>
#include <unordered_set>

using namespace std;

struct TrieNode {
    char v;
    ll cnt = 0;
    ll terminal = 0;
    unordered_map<char, TrieNode*> to  = unordered_map<char, TrieNode*>();

    TrieNode(char v) {
        this->v = v;
    }
    TrieNode* add(char v) {
        if (!this->to.count(v)) {
            this->to[v] = new TrieNode(v);
        }
        this->to[v]->cnt++;
        return this->to[v];
    }
};

struct Trie {
    TrieNode* root;
```

```
    Trie() {
        this->root = new TrieNode(0);
    }
    void add(string s) {
        auto node = this->root;
        for (char c: s) {
            node = node->add(c);
        }
        node->terminal++;
    }
};

int main() {
    cin.tie(0); cout.tie(0);
    ios::sync_with_stdio(0);

    string s;
    cin >> s;
    ll n;
    cin >> n;
    auto t = Trie();
    vector<string> words;
    unordered_map<string, bool> w2a;
    while (n--) {
        string w;
        cin >> w;
        t.add(w);
        words.push_back(w);
        w2a[w] = 0;
    }
    for (int i = 0; i < s.size(); ++i) {
        auto node = t.root;
        string w = "";
        ll max_len = 30;
        ll max_rem = s.size() - i;
        for (int j = 0; j < min(max_len, max_rem); ++j) {
            if (node->to.count(s[i + j])) {
                node = node->to[s[i + j]];
                w += s[i + j];
                if (node->terminal) {
                    w2a[w] = 1;
                }
            } else {
                break;
            }
        }
    }
    for (auto w: words) {
        cout << (w2a[w] ? "Yes" : "No") << endl;
    }
}
```

9 I. Лотерея

Разбор

Построим бор на числах, которые называют участники лотереи. Затем запустим поиск в глубину по такому бору. В процессе обхода поддерживаем необходимую сумму выплат. Когда мы рассматриваем потомков в каком-то узле, у нас есть два варианта:

1. Существует цифра, которую никто не назвал – в этом случае завершаем обход, т.к. найдено лучшее решение (мы не будем доплачивать никому за последующие цифры)
2. Такой цифры нет – тогда зайдём в каждое поддерево и попытаемся найти там лучшее решение

Как считаем ответ для какого-то узла? Умножим количество проходящих через эту цифру последовательностей на разницу между доплатой за угадывание цифры на этом шаге и на предыдущем. Если это

терминальная вершина, то попытаемся обновить ответ.

Асимптотика решения: $O(N \times M)$

Решение

```
#include <fstream>
#include <iostream>
#include <map>
#include <string>
#include <vector>

using namespace std;

long long n, m, k, ans = -1;
string genstrs;
vector<long long> cost;
vector<char> sans;

struct Trie {
    bool flag;
    long long count = 0;
    map<char, Trie *> children;
    Trie() : flag(false) {}
};

void insert(Trie *t, const string &str) {
    for (char c : str) {
        auto it = t->children.find(c);
        if (it == t->children.end()) {
            Trie *next = new Trie;
            t->children[c] = next;
            t = next;
        } else {
            t = it->second;
        }
        t->count++;
    }

    t->flag = true;
}

void dfs(Trie *t, long long sums, vector<char> &s, long long i) {
    if (t->flag) {
        if (sums < ans || ans == -1) {
            ans = sums;
            sans = s;
        }

        return;
    }

    for (char c : genstrs) {
        auto it = t->children.find(c);
        if (it == t->children.end()) {
            s[i] = c;
            if (sums < ans || ans == -1) {
                ans = sums;
                sans = s;

                for (long long j = i + 1; j < m; ++j)
                    sans[j] = '0';
            }

            return;
        } else {
            auto t1 = it->second;
            s[i] = c;
            if (i == 0)
                sums += (t1->count) * cost[i];
```

```
            else
                sums += (t1->count) * (cost[i] - cost[i - 1]);

            dfs(t1, sums, s, i + 1);
            if (i == 0)
                sums -= (t1->count) * cost[i];
            else
                sums -= (t1->count) * (cost[i] - cost[i - 1]);
        }
    }
}

int main() {

    cin >> n >> m >> k;
    cost.resize(m);
    for (int i = 0; i < k; ++i)
        genstrs += (char)('0' + i);

    for (long long &x : cost)
        cin >> x;

    Trie *t = new Trie;
    string str;
    for (long long i = 0; i < n; i++) {
        cin >> str;
        insert(t, str);
    }

    vector<char> x(m);
    dfs(t, 0, x, 0);
    for (char x : sans)
        cout << x;

    for (long long i = 0; i < m - sans.size(); i++)
        cout << '0';

    cout << endl << ans << endl;

    return 0;
}
```