МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
имени М.В. ЛОМОНОСОВА

ОЛИМПИАДНАЯ РАБОТА

Наименование олимпиады школьников: **«Ломоносов»**

Профиль олимпиады: **Информатика**

ФИО участника олимпиады: **Сидоров Владислав Олегович**

Класс: **11 класс**

Технический балл: **73**

Дата проведения: **17 марта 2022 г.**

Результаты проверки:

Оценка участника строится из 3 частей:

1. оценка за задание - рассчитывается путем запуска тестов и определения правильности работы программы на тестах, до 100 баллов по каждой задаче;
2. дополнительные баллы за полностью правильное решение задания со 2 по 5 - в случае прохождения всех тестов по заданию к оценке прибавляется 55 баллов;
3. нормализация оценки - если полученная из пунктов 1 и 2 сумма баллов превышает 500, то итоговая оценка - 100, если не превышает 500, но превышает 400 - 99 баллов, если не превышает 400 - делится на 3.9 и округляется до целого.

Оценки за задания:

№	1	2	3	4	5
Оценка	100	94	92	0	0

Дополнительный балл: 0

Задание 1. Попытка 1.

```
#ifndef LOCAL
#pragma GCC optimize("Ofast")
#pragma GCC optimize ("O3")
#endif

#include <iostream>
#include <cmath>
#include <ctime>
#include <vector>
#include <iomanip>
#include <vector>
#include <set>
#include <unordered_set>
#include <map>
#include <unordered_map>
#include <queue>
#include <deque>
#include <stack>
#include <algorithm>
#include <numeric>
#include <bitset>
#include <list>
#include <forward_list>
#include <tuple>
```

```
#include <string>
#include <complex>
#include <random>
#include <array>
#include <sstream>

#define len(a) (int)a.size()
#define all(a) a.begin(), a.end()
#define forn(n, i) for (int i = 0; i < n; ++i)
#define fornrev(n, i) for (int i = n - 1; i > -1; --i)
#define accuracy(a) fixed << setprecision(a)

typedef long long ll;
typedef long double ld;

const int INF = (int)1e9;
const ll LINF = (ll)1e18;
const double EPS = 0.000000000000001;
const double PI = acos(-1.0);
const int MOD = 1e9 + 7;

using namespace std;

void fast_io()
{
```

```
    ios::sync_with_stdio(false);
    cin.tie(nullptr);
    cout.tie(nullptr);
}

void file_in(const string& filename)
{
    freopen(filename.c_str(), "r", stdin);
}

void file_out(const string& filename)
{
    freopen(filename.c_str(), "w", stdout);
}

double diff_clock(clock_t clock1, clock_t clock2)
{
    return (double)(clock2 - clock1) * 1000 / CLOCKS_PER_SEC;
}

void sleep(int milliseconds)
{
    clock_t start = clock();
    for (clock_t end = clock(); diff_clock(start, end) < milliseconds; end = clock()) {}
}
```

```
int randint(int a, int b)
{
    static random_device rd;
    static mt19937 rnd(rd());
    return (int)(rnd() % (b - a + 1) + a);
}

ll randint64(ll a, ll b)
{
    static random_device rd;
    static mt19937_64 rnd(rd());
    return (ll)(rnd() % (b - a + 1) + a);
}

template<typename T>
void uniquize(vector<T>& vec)
{
    vec.resize(distance(vec.begin(), unique(vec.begin(), vec.end())));
}

#ifdef LOCAL
#include "Code/Utilities/Debug output.cpp"
#define debug(...) cerr << "debug(" << #__VA_ARGS__ << "): ", debug_out(cerr, __VA_ARGS__)
```

```
#else
#define debug(...)
#endif

inline string input_filename();

void run();
void test();
void check();
void gen();
void check_stupid();

signed main()
{
#ifdef LOCAL
    file_in("input.txt");
    file_out("output.txt");
    clock_t start = clock();
#else
    fast_io();
    if (!input_filename().empty())
    {
        file_in(input_filename() + ".in");
        file_out(input_filename() + ".out");
    }
```

```
#endif

    run();
//    test();
//    gen();
//    check();
//    check_stupid();

#ifdef LOCAL
    cerr << "TIME = " << accuracy(3) << diff_clock(start, clock()) << endl;
#endif
    return 0;
}

inline string input_filename()
{
    return "";
}

struct SegmentTree
{
    struct Node
    {
        int max_prefix_val;
        int max_prefix;
```

```
int max_suffix_val;
int max_suffix;

tuple<int, int, int> max_segment;

int block_size;
int block_sum;

Node()
{
    max_prefix_val = -INF;
    max_prefix = 0;

    max_suffix_val = -INF;
    max_suffix = 0;

    max_segment = {-INF, 0, 0};

    block_size = 0;
    block_sum = 0;
}

explicit Node(int value)
{
```

```
        max_prefix_val = value;
        max_prefix = 1;

        max_suffix_val = value;
        max_suffix = 0;

        max_segment = {value, 0, 1};

        block_size = 1;
        block_sum = value;
    }
};

vector<Node> tree;
vector<int>& data;

SegmentTree(vector<int>& data) : data(data)
{
    tree.resize(4 * len(data));
    build(0, 0, len(data));
}

void rebuild()
{
    build(0, 0, len(data));
```

```
}

Node merge(const Node& left, const Node& right)
{
    Node result;
    result.block_size = left.block_size + right.block_size;
    result.block_sum = left.block_sum + right.block_sum;

    result.max_prefix_val = left.max_prefix_val;
    result.max_prefix = left.max_prefix;
    if (left.block_sum + right.max_prefix_val > result.max_prefix_val)
    {
        result.max_prefix_val = left.block_sum + right.max_prefix_val;
        result.max_prefix = left.block_size + right.max_prefix;
    }

    result.max_suffix_val = right.max_suffix_val;
    result.max_suffix = left.block_size + right.max_suffix;
    if (left.max_suffix_val + right.block_sum > result.max_suffix_val)
    {
        result.max_suffix_val = left.max_suffix_val + right.block_sum;
        result.max_suffix = left.max_suffix;
    }

    result.max_segment = max(left.max_segment,
```

```
                    max(make_tuple(left.max_suffix_val + right.max_prefix_val,
left.max_suffix, right.max_prefix + left.block_size),
                        make_tuple(get<0>(right.max_segment),
get<1>(right.max_segment) + left.block_size, get<2>(right.max_segment) +
left.block_size)));
    return result;
}

void build(int p, int left, int right)
{
    if (right - left == 1)
    {
        tree[p] = Node(data[left]);
        return;
    }
    int mid = (left + right) / 2;
    build(2 * p + 1, left, mid);
    build(2 * p + 2, mid, right);
    tree[p] = merge(tree[2 * p + 1], tree[2 * p + 2]);
}

Node get_max(int p, int left, int right, int left_, int right_)
{
    if ((right <= left_) || (right_ <= left))
    {
        return {};
```

```
        }
        if ((left_ <= left) && (right <= right_))
        {
            return tree[p];
        }
        int mid = (left + right) / 2;
        return merge(
                get_max(2 * p + 1, left, mid, left_, right_),
                get_max(2 * p + 2, mid, right, left_, right_)
        );
    }
};

void gen()
{
    int n = randint(1, 10);
    cout << n << '\n';
    for (int i = 0; i < n; ++i)
    {
        cout << randint(-10, 10) << ' ';
    }
    cout << '\n';
}

void check_stupid()
```

```
{
    int n;
    cin >> n;
    vector<int> arr(n);
    for (int i = 0; i < n; ++i)
    {
        cin >> arr[i];
    }

    tuple<int, int, int> max_segment = {-INF, -1, -1};
    for (int i = 0; i < n; ++i)
    {
        int cur_sum = 0;
        for (int j = i; j < n; ++j)
        {
            cur_sum += arr[j];
            max_segment = max(max_segment, {cur_sum, i, j + 1});
        }
    }

    auto [max_val, max_left, max_right] = max_segment;
    cout << max_val << ' ' << max_left + 1 << ' ' << max_right + 1 << '\n';
}

void check()
```

```
{
    int n;
    cin >> n;
    vector<int> arr(n);
    for (int i = 0; i < n; ++i)
    {
        cin >> arr[i];
    }
    SegmentTree tree(arr);

    tuple<int, int, int> max_segment = tree.get_max(0, 0, n, 0, n).max_segment;
    auto [max_val, max_left, max_right] = max_segment;
    cout << max_val << ' ' << max_left + 1 << ' ' << max_right + 1 << '\n';
}

void test()
{
    int n;
    cin >> n;
    vector<int> arr(n);
    for (int i = 0; i < n; ++i)
    {
        cin >> arr[i];
    }
    SegmentTree tree(arr);
```

```
    int q;
    cin >> q;
    for (; q > 0; --q)
    {
        int left, right;
        cin >> left >> right;
        --left;
        SegmentTree::Node node = tree.get_max(0, 0, n, left, right);

        auto [max_sum, max_left, max_right] = node.max_segment;
        max_left += left;
        max_right += left;
        cout << max_sum << '\n';
        cout << max_left + 1 << ' ' << max_right << '\n';
    }
}


int solve_half(int n, int max_segs, int base_cost, char base_char, SegmentTree&
max_tree, SegmentTree& min_tree, vector< tuple<int, int, char> >& operations)
{
    priority_queue< tuple<int, int, int, int, int, char> > q;
    {
        auto [max_val, max_left, max_right] = max_tree.get_max(0, 0, n, 0,
n).max_segment;
        q.emplace(max_val, 0, n, max_left, max_right, 1 - base_char);
```

```
    }
    operations.reserve(n / 2 + 1);
    operations.emplace_back(0, n, base_char);
    for (int i = 0; i < max_segs; ++i)
    {
        auto [val, left, right, max_left, max_right, op] = q.top();
        q.pop();
        if (val < 0)
        {
            break;
        }
        base_cost -= val;
        operations.emplace_back(max_left, max_right, op);
        if (op == base_char)
        {
            {
                auto [new_max_val, new_max_left, new_max_right] = max_tree.get_max(0, 0,
n, max_left, max_right).max_segment;
                q.emplace(new_max_val, max_left, max_right, new_max_left + max_left,
new_max_right + max_left, 1 - base_char);
            }
            if (left != max_left)
            {
                auto [new_max_val, new_max_left, new_max_right] = min_tree.get_max(0, 0,
n, left, max_left).max_segment;
                q.emplace(new_max_val, left, max_left, new_max_left + left, new_max_right
+ left, base_char);
```

```
        }
        if (right != max_right)
        {
            auto [new_max_val, new_max_left, new_max_right] = min_tree.get_max(0, 0,
n, max_right, right).max_segment;
            q.emplace(new_max_val, max_right, right, new_max_left + max_right,
new_max_right + max_right, base_char);
        }
    }
    else
    {
        {
            auto [new_max_val, new_max_left, new_max_right] = min_tree.get_max(0, 0,
n, max_left, max_right).max_segment;
            q.emplace(new_max_val, max_left, max_right, new_max_left + max_left,
new_max_right + max_left, base_char);
        }
        if (left != max_left)
        {
            auto [new_max_val, new_max_left, new_max_right] = max_tree.get_max(0, 0,
n, left, max_left).max_segment;
            q.emplace(new_max_val, left, max_left, new_max_left + left, new_max_right
+ left, 1 - base_char);
        }
        if (right != max_right)
        {
            auto [new_max_val, new_max_left, new_max_right] = max_tree.get_max(0, 0,
n, max_right, right).max_segment;
```

```
                q.emplace(new_max_val, max_right, right, new_max_left + max_right,
new_max_right + max_right, 1 - base_char);
            }
        }
    }
    return base_cost;
}

void get_ans(int n, vector< tuple<int, int, char> >& operations)
{
    vector<char> ans(n);
    for (auto [left, right, c] : operations)
    {
        for (int i = left; i < right; ++i)
        {
            ans[i] = c;
        }
    }
    for (int i = 0; i < n; ++i)
    {
        cout << (char)(ans[i] + '0');
    }
    cout << '\n';
}
```

```
void solve_even(int n, int k)
{
    vector<char> photo(n + 1);
    for (int i = 1; i <= n; ++i)
    {
        char c;
        cin >> c;
        photo[i] = (char)(c - '0');
    }

    vector< vector<int> > prefix(2, vector<int>(n + 1));
    for (int c = 0; c < 2; ++c)
    {
        for (int i = 1; i <= n; ++i)
        {
            prefix[c][i] = prefix[c][i - 1] + (photo[i] == c);
        }
    }

    vector< vector< vector<int> > > dp(2, vector< vector<int> >(k + 1, vector<int>(n + 1, INF)));
    vector< vector< vector<int> > > parent(2, vector< vector<int> >(k + 1, vector<int>(n + 1, -1)));
    dp[0][0][0] = dp[1][0][0] = 0;
    for (int i = 1; i <= n; ++i)
    {
```

```
    for (int j = 1; j <= k; ++j)
    {
        for (int c = 0; c < 2; ++c)
        {
            int min_val, min_val_parent;
            min_val = INF;
            min_val_parent = -1;
            for (int m = 1; m <= i; ++m)
            {
                int cur_val = dp[!c][j - 1][m - 1] - prefix[!c][m - 1];
                if (cur_val < min_val)
                {
                    min_val = cur_val;
                    min_val_parent = m;
                }
            }
            if (min_val_parent != -1)
            {
                dp[c][j][i] = min_val + prefix[!c][i];
                parent[c][j][i] = min_val_parent;
            }
        }
    }
}
```

```
    int min_cost, min_cost_groups, min_cost_color;
    min_cost = INF;
    min_cost_groups = min_cost_color = -1;
    for (int c = 0; c < 2; ++c)
    {
        for (int i = 1; i <= k; ++i)
        {
            if (dp[c][i][n] < min_cost)
            {
                min_cost = dp[c][i][n];
                min_cost_groups = i;
                min_cost_color = c;
            }
        }
    }

    int cur_prefix = n;
    while (cur_prefix)
    {
        for (int i = parent[min_cost_color][min_cost_groups][cur_prefix]; i <= cur_prefix;
++i)
        {
            photo[i] = (char)min_cost_color;
        }
        cur_prefix = parent[min_cost_color][min_cost_groups][cur_prefix] - 1;
```

```
        min_cost_color = !min_cost_color;
        --min_cost_groups;
    }

    for (int i = 1; i <= n; ++i)
    {
        cout << ((char)(photo[i] + '0'));
    }
    cout << '\n';
}

void solve_test()
{
    int n, k;
    cin >> n >> k;
    vector<int> max_arr(n);
    vector<int> min_arr(n);
    int zeros = 0;
    char last_c;
    for (int i = 0; i < n; ++i)
    {
        char c;
        cin >> c;
        max_arr[i] = c == '0' ? -1 : 1;
        zeros += c == '0';
```

```
    min_arr[i] = -max_arr[i];
    last_c = c;
}

if (k % 2 == 0)
{
    if (last_c == '1')
    {
        max_arr.back() = INF + 1;
        min_arr.back() = -INF - 1;
    }
    else
    {
        max_arr.back() = INF - 1;
        min_arr.back() = -INF + 1;
    }
}

SegmentTree max_tree(max_arr);
SegmentTree min_tree(min_arr);

vector< tuple<int, int, char> > operations1, operations2;

int cost1 = solve_half(n, k / 2, n - zeros, 0, max_tree, min_tree, operations1);
if (k % 2 == 0)
```

```
    {
        if (last_c == '0')
        {
            min_arr.back() = INF + 1;
            max_arr.back() = -INF - 1;
        }
        else
        {
            min_arr.back() = INF - 1;
            max_arr.back() = -INF + 1;
        }
        min_tree.rebuild();
        max_tree.rebuild();
    }
    int cost2 = solve_half(n, k / 2, zeros, 1, min_tree, max_tree, operations2);

    if (cost1 < cost2)
    {
        get_ans(n, operations1);
    }
    else
    {
        get_ans(n, operations2);
    }
}
```

```
void run()
{
    int t;
    cin >> t;
    for (; t > 0; --t)
    {
        solve_test();
    }
}
```

Задание 1. Попытка 2.

```
#ifndef LOCAL
#pragma GCC optimize("Ofast")
#pragma GCC optimize("O3")
#endif

#include <iostream>
#include <vector>
#include <algorithm>
#include <map>
#include <string>

#define len(a) (int)a.size()

const int INF = (int)1e9 + 10;

typedef long long ll;

using namespace std;

void fast_io()
{
    ios_base::sync_with_stdio(0);
    cin.tie(0);
    cout.tie(0);
```

```
}

void file_in(const string& filename)
{
    freopen(filename.c_str(), "r", stdin);
}

void file_out(const string& filename)
{
    freopen(filename.c_str(), "w", stdout);
}

void run();

signed main()
{
#ifdef LOCAL
    file_in("input.txt");
    file_out("output.txt");
#else
    fast_io();
#endif

    run();
```

```
}

unordered_map<char, char> cost;

void prepare()
{
    char cur_cost = 0;
    for (char c = '0'; c <= '9'; ++c)
    {
        cost[c] = cur_cost++;
    }
    for (char c = 'a'; c <= 'z'; ++c)
    {
        cost[c] = cur_cost++;
    }
    for (char c = 'A'; c <= 'Z'; ++c)
    {
        cost[c] = cur_cost++;
    }
}

int compare(string& str1, string& str2)
{
    if (len(str1) > len(str2))
    {
```

```
        return 1;
    }
    else if (len(str1) < len(str2))
    {
        return -1;
    }
    for (int i = 0; i < len(str1); ++i)
    {
        if (cost[str1[i]] > cost[str2[i]])
        {
            return 1;
        }
        else if (cost[str1[i]] < cost[str2[i]])
        {
            return -1;
        }
    }
    return 0;
}

void delete_zeros(string& str)
{
    int zeros_count = 0;
    for (char i : str)
    {
```

```
        if (i != '0')
        {
            break;
        }
        ++zeros_count;
    }
    str = str.substr(min(zeros_count, len(str) - 1));
}

void run()
{
    prepare();

    int k, n;
    cin >> k >> n;

    bool is_max_str = false;
    string max_str, cur_str;
    vector<int> max_pos;
    for (int i = 0; i < n; ++i)
    {
        cin >> cur_str;
        delete_zeros(cur_str);
        bool is_ok = true;
        if (cur_str == "0")
```

```
{


}
else
{
    if (len(cur_str) <= k - 1)
    {
        is_ok = false;
    }
    else
    {
        for (int j = 0; j < k - 1; ++j)
        {
            if (cur_str[len(cur_str) - 1 - j] != '0')
            {
                is_ok = false;
                break;
            }
        }
    }
}
if (!is_ok)
{
    continue;
}
```

```
    if (!is_max_str)
    {
        is_max_str = true;
        max_str = cur_str;
        max_pos.push_back(i);
    }
    else
    {
        int cmp_res = compare(max_str, cur_str);
        if (cmp_res == -1)
        {
            max_str = cur_str;
            max_pos.clear();
            max_pos.push_back(i);
        }
        else if (cmp_res == 0)
        {
            max_pos.push_back(i);
        }
    }
}

if (!is_max_str)
{
    cout << "-1\n";
```

```
        return;
    }
    cout << max_str << '\n';
    for (int i : max_pos)
    {
        cout << i + 1 << '\n';
    }
}
```

Задание 1. Попытка 3.

```
#ifndef LOCAL
#pragma GCC optimize("Ofast")
#pragma GCC optimize("O3")
#endif

#include <iostream>
#include <vector>
#include <algorithm>
#include <map>
#include <string>
#include <unordered_map>

#define len(a) (int)a.size()

const int INF = (int)1e9 + 10;

typedef long long ll;

using namespace std;

void fast_io()
{
    ios_base::sync_with_stdio(0);
    cin.tie(0);
```

```
    cout.tie(0);
}

void file_in(const string& filename)
{
    freopen(filename.c_str(), "r", stdin);
}

void file_out(const string& filename)
{
    freopen(filename.c_str(), "w", stdout);
}

void run();

signed main()
{
#ifdef LOCAL
    file_in("input.txt");
    file_out("output.txt");
#else
    fast_io();
#endif

    run();
```

```
}

unordered_map<char, char> cost;

void prepare()
{
    char cur_cost = 0;
    for (char c = '0'; c <= '9'; ++c)
    {
        cost[c] = cur_cost++;
    }
    for (char c = 'a'; c <= 'z'; ++c)
    {
        cost[c] = cur_cost++;
    }
    for (char c = 'A'; c <= 'Z'; ++c)
    {
        cost[c] = cur_cost++;
    }
}

int compare(string& str1, string& str2)
{
    if (len(str1) > len(str2))
```

```
    {
        return 1;
    }
    else if (len(str1) < len(str2))
    {
        return -1;
    }
    for (int i = 0; i < len(str1); ++i)
    {
        if (cost[str1[i]] > cost[str2[i]])
        {
            return 1;
        }
        else if (cost[str1[i]] < cost[str2[i]])
        {
            return -1;
        }
    }
    return 0;
}

void delete_zeros(string& str)
{
    int zeros_count = 0;
    for (char i : str)
```

```
    {
        if (i != '0')
        {
            break;
        }
        ++zeros_count;
    }
    str = str.substr(min(zeros_count, len(str) - 1));
}

void run()
{
    prepare();

    int k, n;
    cin >> k >> n;

    bool is_max_str = false;
    string max_str, cur_str;
    vector<int> max_pos;
    for (int i = 0; i < n; ++i)
    {
        cin >> cur_str;
        delete_zeros(cur_str);
        bool is_ok = true;
```

```
if (cur_str == "0")
{

}
else
{
    if (len(cur_str) <= k - 1)
    {
        is_ok = false;
    }
    else
    {
        for (int j = 0; j < k - 1; ++j)
        {
            if (cur_str[len(cur_str) - 1 - j] != '0')
            {
                is_ok = false;
                break;
            }
        }
    }
}
if (!is_ok)
{
    continue;
```

```
    }
    if (!is_max_str)
    {
        is_max_str = true;
        max_str = cur_str;
        max_pos.push_back(i);
    }
    else
    {
        int cmp_res = compare(max_str, cur_str);
        if (cmp_res == -1)
        {
            max_str = cur_str;
            max_pos.clear();
            max_pos.push_back(i);
        }
        else if (cmp_res == 0)
        {
            max_pos.push_back(i);
        }
    }
}

if (!is_max_str)
{
```

```
        cout << "-1\n";
        return;
    }
    cout << max_str << '\n';
    for (int i : max_pos)
    {
        cout << i + 1 << '\n';
    }
}
```

Задание 2. Попытка 1.

```
#ifndef LOCAL
#pragma GCC optimize("Ofast")
#pragma GCC optimize("O3")
#endif

#include <iostream>
#include <vector>
#include <algorithm>
#include <map>
#include <string>
#include <unordered_map>

#define len(a) (int)a.size()

const int INF = (int)1e9 + 10;

typedef long long ll;

using namespace std;

void fast_io()
{
    ios_base::sync_with_stdio(0);
    cin.tie(0);
```

```
    cout.tie(0);
}

void file_in(const string& filename)
{
    freopen(filename.c_str(), "r", stdin);
}

void file_out(const string& filename)
{
    freopen(filename.c_str(), "w", stdout);
}

void run();

signed main()
{
#ifdef LOCAL
    file_in("input.txt");
    file_out("output.txt");
#else
    fast_io();
#endif

    run();
```

```
}

unordered_map<char, char> cost;

void prepare()
{
    char cur_cost = 0;
    for (char c = '0'; c <= '9'; ++c)
    {
        cost[c] = cur_cost++;
    }
    for (char c = 'a'; c <= 'z'; ++c)
    {
        cost[c] = cur_cost++;
    }
    for (char c = 'A'; c <= 'Z'; ++c)
    {
        cost[c] = cur_cost++;
    }
}

int compare(string& str1, string& str2)
{
    if (len(str1) > len(str2))
```

```
    {
        return 1;
    }
    else if (len(str1) < len(str2))
    {
        return -1;
    }
    for (int i = 0; i < len(str1); ++i)
    {
        if (cost[str1[i]] > cost[str2[i]])
        {
            return 1;
        }
        else if (cost[str1[i]] < cost[str2[i]])
        {
            return -1;
        }
    }
    return 0;
}

void delete_zeros(string& str)
{
    int zeros_count = 0;
    for (char i : str)
```

```
    {
        if (i != '0')
        {
            break;
        }
        ++zeros_count;
    }
    str = str.substr(min(zeros_count, len(str) - 1));
}

struct Node
{
    char c;

    Node() : Node(127) {}
    Node(char c) : c(c) {}

    bool operator<(const Node& other) const
    {
        return cost[c] < cost[other.c];
    }
};

void run()
{
```

```
prepare();


map<Node, int> counts;
int n;
cin >> n;
for (int i = 0; i < n; ++i)
{
    char c;
    cin >> c;
    if (cost.contains(c))
    {
        ++counts[Node(c)];
    }
}


for (int size = 61; size >= 1; --size)
{
    string max_str;
    max_str.resize(size);
    map<Node, int> tmp = counts;
    bool is_ok = true;
    for (int i = 0; i < size; ++i)
    {
        bool is_placed = false;
        for (auto it = tmp.rbegin(); it != tmp.rend(); ++it)
```

```
    {
        if (cost[it->first.c] > size - i)
        {
            continue;
        }
        max_str[i] = it->first.c;
        --it->second;
        is_placed = true;
        if (it->second == 0)
        {
            tmp.erase(it->first);
        }
        break;
    }
    if (!is_placed)
    {
        is_ok = false;
        break;
    }
}
if (is_ok)
{
    cout << max_str << '\n';
    return;
}
```

```
    }


    cout << "-1\n";
}
```

Задание 2. Попытка 2.

```
#ifndef LOCAL
#pragma GCC optimize("Ofast")
#pragma GCC optimize("O3")
#endif

#include <iostream>
#include <vector>
#include <algorithm>
#include <map>
#include <string>
#include <unordered_map>

#define len(a) (int)a.size()

const int INF = (int)1e9 + 10;

typedef long long ll;

using namespace std;

void fast_io()
{
    ios_base::sync_with_stdio(0);
    cin.tie(0);
```

```
    cout.tie(0);
}

void file_in(const string& filename)
{
    freopen(filename.c_str(), "r", stdin);
}

void file_out(const string& filename)
{
    freopen(filename.c_str(), "w", stdout);
}

void run();

signed main()
{
#ifdef LOCAL
    file_in("input.txt");
    file_out("output.txt");
#else
    fast_io();
#endif

    run();
```

```
}

unordered_map<char, char> cost;

void prepare()
{
    char cur_cost = 0;
    for (char c = '0'; c <= '9'; ++c)
    {
        cost[c] = cur_cost++;
    }
    for (char c = 'a'; c <= 'z'; ++c)
    {
        cost[c] = cur_cost++;
    }
    for (char c = 'A'; c <= 'Z'; ++c)
    {
        cost[c] = cur_cost++;
    }
}

int compare(string& str1, string& str2)
{
    if (len(str1) > len(str2))
```

```
    {
        return 1;
    }
    else if (len(str1) < len(str2))
    {
        return -1;
    }
    for (int i = 0; i < len(str1); ++i)
    {
        if (cost[str1[i]] > cost[str2[i]])
        {
            return 1;
        }
        else if (cost[str1[i]] < cost[str2[i]])
        {
            return -1;
        }
    }
    return 0;
}

void delete_zeros(string& str)
{
    int zeros_count = 0;
    for (char i : str)
```

```
    {
        if (i != '0')
        {
            break;
        }
        ++zeros_count;
    }
    str = str.substr(min(zeros_count, len(str) - 1));
}

struct Node
{
    char c;

    Node() : Node(127) {}
    Node(char c) : c(c) {}

    bool operator<(const Node& other) const
    {
        return cost[c] < cost[other.c];
    }
};

void run()
{
```

```
prepare();


map<Node, int> counts;
int n;
cin >> n;
for (int i = 0; i < n; ++i)
{
    char c;
    cin >> c;
    if (cost.count(c))
    {
        ++counts[Node(c)];
    }
}


for (int size = 61; size >= 1; --size)
{
    string max_str;
    max_str.resize(size);
    map<Node, int> tmp = counts;
    bool is_ok = true;
    for (int i = 0; i < size; ++i)
    {
        bool is_placed = false;
        for (auto it = tmp.rbegin(); it != tmp.rend(); ++it)
```

```
    {
        if (cost[it->first.c] > size - i)
        {
            continue;
        }
        max_str[i] = it->first.c;
        --it->second;
        is_placed = true;
        if (it->second == 0)
        {
            tmp.erase(it->first);
        }
        break;
    }
    if (!is_placed)
    {
        is_ok = false;
        break;
    }
}
if (is_ok)
{
    cout << max_str << '\n';
    return;
}
```

```
    }


    cout << "-1\n";
}
```

Задание 3. Попытка 1.

```
#ifndef LOCAL
#pragma GCC optimize("Ofast")
#pragma GCC optimize("O3")
#endif

#include <iostream>
#include <vector>
#include <algorithm>
#include <map>
#include <string>
#include <unordered_map>

#define len(a) (int)a.size()

const int INF = (int)1e9 + 10;

typedef long long ll;

using namespace std;

void fast_io()
{
    ios_base::sync_with_stdio(0);
    cin.tie(0);
```

```
    cout.tie(0);
}

void file_in(const string& filename)
{
    freopen(filename.c_str(), "r", stdin);
}

void file_out(const string& filename)
{
    freopen(filename.c_str(), "w", stdout);
}

void run();

signed main()
{
#ifdef LOCAL
    file_in("input.txt");
    file_out("output.txt");
#else
    fast_io();
#endif

    run();
```

```
}

vector< vector<int> > graph;
vector<char> used;
vector<int> heights;
vector<int> ans;

void dfs(int v, int height)
{
    used[v] = 1;
    heights[v] = height;
    for (int to : graph[v])
    {
        if (used[to] == 1)
        {
            ++ans[v];
        }
        else if (used[to] == 2)
        {
            --ans[v];
        }
        else
        {
            dfs(to, height + 1);
```

```
            ans[v] += ans[to] - 1;
        }
    }
    used[v] = 2;
}

int height_border;
vector< pair<int, int> > edges;

void add_edge(int u, int v)
{
    if (u < v)
    {
        edges.emplace_back(u, v);
    }
    else
    {
        edges.emplace_back(v, u);
    }
}

void dfs_ans(int v)
{
    used[v] = 1;
    for (int to : graph[v])
```

```
    {
        if (heights[to] < height_border)
        {
            add_edge(v, to);
        }
        else
        {
            if (!used[to])
            {
                dfs_ans(to);
            }
        }
    }
}

void run()
{
    int n, m;
    cin >> n >> m;
    graph.resize(n);
    for (int i = 0; i < m; ++i)
    {
        int u, v;
        cin >> u >> v;
        --u, --v;
```

```
    graph[u].push_back(v);
    graph[v].push_back(u);
}

used.resize(n);
heights.resize(n);
ans.resize(n);
dfs(0, 0);

int min_value, min_subtree;
min_value = INF;
min_subtree = -1;
for (int i = 1; i < n; ++i)
{
    if (ans[i] < min_value)
    {
        min_value = ans[i];
        min_subtree = i;
    }
}

used.assign(n, 0);
edges.reserve(min_value);
height_border = heights[min_subtree];
dfs_ans(min_subtree);
```

```
    if (min_value != len(edges))
    {
        exit(-1);
    }

    sort(edges.begin(), edges.end());
    cout << min_value << '\n';
    for (auto [u, v] : edges)
    {
        cout << u + 1 << ' ' << v + 1 << '\n';
    }
}
```

Задание 4. Попытка 1.

```
#ifndef LOCAL
#pragma GCC optimize("Ofast")
#pragma GCC optimize("O3")
#endif

#include <iostream>
#include <vector>
#include <algorithm>
#include <map>
#include <string>
#include <unordered_map>

#define len(a) (int)a.size()

const int INF = (int)1e9 + 10;

typedef long long ll;

using namespace std;

void fast_io()
{
    ios_base::sync_with_stdio(0);
    cin.tie(0);
```

```
    cout.tie(0);
}

void file_in(const string& filename)
{
    freopen(filename.c_str(), "r", stdin);
}

void file_out(const string& filename)
{
    freopen(filename.c_str(), "w", stdout);
}

void run();

signed main()
{
#ifdef LOCAL
    file_in("input.txt");
    file_out("output.txt");
#else
    fast_io();
#endif

    run();
```

```
}

vector< vector<int> > graph;
vector<char> used;
vector<int> heights;
vector<int> ans;

void dfs(int v, int height)
{
    used[v] = 1;
    heights[v] = height;
    for (int to : graph[v])
    {
        if (used[to] == 1)
        {
            ++ans[v];
        }
        else if (used[to] == 2)
        {
            --ans[v];
        }
        else
        {
            dfs(to, height + 1);
```

```
            ans[v] += ans[to] - 1;
        }
    }
    used[v] = 2;
}

int height_border;
vector< pair<int, int> > edges;

void add_edge(int u, int v)
{
    if (u < v)
    {
        edges.emplace_back(u, v);
    }
    else
    {
        edges.emplace_back(v, u);
    }
}

void dfs_ans(int v)
{
    used[v] = 1;
    for (int to : graph[v])
```

```
    {
        if (heights[to] < height_border)
        {
            add_edge(v, to);
        }
        else
        {
            if (!used[to])
            {
                dfs_ans(to);
            }
        }
    }
}

void run()
{
    int n, m;
    cin >> n >> m;
    graph.resize(n);
    for (int i = 0; i < m; ++i)
    {
        int u, v;
        cin >> u >> v;
        --u, --v;
```

```
    graph[u].push_back(v);
    graph[v].push_back(u);
}

used.resize(n);
heights.resize(n);
ans.resize(n);
dfs(0, 0);

int min_value, min_subtree;
min_value = INF;
min_subtree = -1;
for (int i = 1; i < n; ++i)
{
    if (ans[i] < min_value)
    {
        min_value = ans[i];
        min_subtree = i;
    }
}

used.assign(n, 0);
edges.reserve(min_value);
height_border = heights[min_subtree];
dfs_ans(min_subtree);
```

```
    if (min_value != len(edges))
    {
        exit(-1);
    }

    sort(edges.begin(), edges.end());
    cout << min_value << '\n';
    for (auto [u, v] : edges)
    {
        cout << u + 1 << ' ' << v + 1 << '\n';
    }
}
```