Министерство науки и высшего образования Российской Федерации Федеральное государственное автономное образовательное учреждение высшего образования «Национальный исследовательский университет ИТМО»

Факультет Программной инженерии и компьютерной техники

**Отчёт по задачам блока №2**

**По дисциплине «Алгоритмы и структуры данных»**

Группа: P3212

Выполнил: Балин А. А.

Проверил: Косяков М. С.

# Задача E

## Решение

Изначальное решение состояло в том, чтобы вычислить массив – массив расстояний между стойлами. Зная, что нужно поставить k коров, я искал минимум в массиве dx и старался от него избавиться. В какой-то момент логика добавления исчезнувшего минимального dx к расстоянию слева или справа стала слишком сложной и всё ещё не работала – у меня кончились идеи по решению дилемм, возникших на этом этапе. С новой силой приступив к задаче, я решил воспользоваться одним из методов: деление отрезка поиска пополам. Однако в первый раз я воспользовался им, чтобы ставить корову в такой позиции, чтобы минимизировать расстояния между позициями слева и справа. Вполне очевидно, что первых двух коров надо ставить в первое и последнее стойло, так как при любом исходе ставить крайнюю справа корову в не крайнее справа место получится менее выгодно (для левой коровы так же). То есть, третью корову я ставил на i-тое место так, чтобы . Столкнулся с той же проблемой, что и в предыдущем решении, поэтому крайнее решение оказалось в поиске самого ответа бинарным поиском. Мне казалось, что решение очень «тупое», однако оно легко написалось, отладилось и прошло тесты.

Очевидно, что решение с бинарным поиском можно оценить как .

## Код

#include <algorithm>

#include <iostream>

#include <vector>

using namespace std;

int main() {

int N, K;

cin >> N;

cin >> K;

vector<int> x(N);

for (size\_t i = 0; i < N; ++i) {

cin >> x[i];

}

int left = 1;

int right = x[N - 1];

int mid = 0;

bool T = false;

while (left <= right) {

if (left - right == -1) {

T = true;

}

mid = (left + right) / 2;

int last = x[0];

int k = K - 1;

for (int i = 1; i < N; i++) {

if (x[i] - last >= mid) {

last = x[i];

k--;

if (k == 0) {

left = mid;

break;

}

}

}

if (k != 0) {

right = mid;

}

if (T) {

break;

}

}

cout << mid;

}

# Задача F

## Решение

Идея: создать компаратор для того, чтобы отсортировать встроенной функцией . Изначально компаратор сравнивал «в тупую» значения и , но, как оказалось, на такое сравнение нужно много различных проверок и я в них настолько запутался, что не стал работать с этим дальше. Следующий компаратор сравнивал и посимвольно, что оказалось верной стратегией. Самым тяжелым было понять, как я у себя в голове строил максимальное число из кусков, чтобы перенести это на код (это и было первым решением).

Согласно стандарту C++11/14, std::sort() имеет асимптотику . Тогда можно оценить количество операций между двумя константами , поэтому .

## Код

#include <algorithm>

#include <iostream>

#include <vector>

using namespace std;

bool customComparator(const string& a, const string& b) {

string s1 = a + b;

string s2 = b + a;

if (s1 == s2) {

return false;

}

for (size\_t i = 0; i < s1.size(); i++) {

if (s1[i] > s2[i]) {

return true;

}

if (s1[i] < s2[i]) {

return false;

}

}

}

int main() {

string x;

vector<string> nums;

while (cin >> x) {

nums.push\_back(x);

}

sort(nums.begin(), nums.end(), customComparator);

bool zeros = true;

for (int i = 0; i < nums.size(); i++) {

cout << nums[i];

}

}

# Задача G

## Решение

Заметим две вещи: неповторяющиеся символы можно ставить по центру строки в любом порядке; повторяющиеся больше двух раз символы записываем в неповторяющиеся, дальше см. замечание 1. Таким образом, мы составляем строку из символов из вектора pairs (максимум 26 штук), далее ставим в любом порядке символы неповторяющиеся из вектора alone, и закрываем (как скобки) символы из pairs (выводя в обратном порядке). Почему работает замечание 1? Можно доказать, что, меняя любой неповторяющийся символ местами с повторяющимся, мы не получаем выигрыша (если считать сумму для конкретной пары и для остальных, в обоих случаях она получится меньше). Замечание два работает из-за условия, что вес пары рассчитывается из максимального расстояния между двумя одинаковыми символами, таким образом, выполнив замечание два, мы увеличим общее расстояние между всеми парными символами и увеличим итоговую сумму. Осталось только отсортировать символы по убыванию веса и вывести.

С учётом того, что максимальный размер сортируемого массива равен 26, получается, что максимальное количество операций это , где некоторая константа (заполнение массивов, отделение массива alone и pairs). Тогда , и .

## Код

#include <algorithm>

#include <iostream>

#include <vector>

#define ALPHABET\_LEN 26

using namespace std;

int main() {

string s;

cin >> s;

vector<int> weights(ALPHABET\_LEN);

vector<int> counts(ALPHABET\_LEN);

vector<char> alone;

vector<char> pairs;

for (size\_t i = 0; i < ALPHABET\_LEN; i++) {

cin >> weights[i];

}

for (size\_t i = 0; i < s.size(); i++) {

counts[(size\_t)s[i] - 'a']++;

}

for (size\_t i = 0; i < ALPHABET\_LEN; i++) {

if (counts[i] == 1) {

alone.push\_back((int)i + 'a');

}

else if (counts[i] > 1) {

pairs.push\_back((int)i + 'a');

while (counts[i] > 2) {

alone.push\_back((int)i + 'a');

counts[i] -= 1;

}

}

}

sort(pairs.begin(), pairs.end(), [&weights](char& a, char& b) {

return weights[(size\_t)a - 'a'] > weights[(size\_t)b - 'a'];

});

for (size\_t i = 0; i < pairs.size(); i++) {

cout << pairs[i];

}

for (size\_t i = 0; i < alone.size(); i++) {

cout << alone[i];

}

for (size\_t i = pairs.size(); i > 0; i--) {

cout << pairs[i - 1];

}

}

# Задача H

## Решение

Заметим, что можно разделить все покупки на L чеков. Тогда легче считать, что количество товаров в чеке равно k, иначе: если , то мы покупаем эти товары отдельной покупкой L+1; если , тогда разбиваем на покупки по k товаров. Допустим, мы сделали какое-то количество покупок, у нас остались товары с ценами в порядке возрастания, тогда получить скидку больше, чем не получится, и такую скидку можно сделать, только выбрав товары с до . Применяя эту логику в программе, мы проходимся по отсортированному по убыванию массиву цен и не учитываем в конечной сумме при .

Из-за сортировки, .

## Код

#include <algorithm>

#include <iostream>

#include <vector>

using namespace std;

bool cmp(int& a, int& b) { return a > b; }

int main() {

int n, k;

cin >> n >> k;

vector<int> c(n);

for (int i = 0; i < n; i++)

cin >> c[i];

int ans = 0;

sort(c.begin(), c.end(), cmp);

for (int i = 0; i < n; i++) {

if ((i + 1) % k != 0)

ans += c[i];

}

cout << ans;

}