Университет ИТМО

Факультет программной инженерии и компьютерной техники

Лабораторная работа №2

по «Алгоритмам и структурам данных»

Базовые задачи

Выполнил:

Студент группы P3232

Чмурова М. В.

Преподаватели:

Косяков М.С.

Тараканов Д.С.

Санкт-Петербург

2024

Задача №E «Коровы в стойла»

Пояснение к примененному алгоритму:

Для наискорейшего решения алгоритма используется бинарный поиск, который позволяет сузить область поиска. Проверяется возможно ли разместить с минимальным расстоянием, которое является серединой между первым и последним стойлом.

Если всех коров невозможно разместить с таким расстоянием, то текущее значение середины слишком большое и рассматривается значение от начала координат до середины.

Иначе же пробуем увеличить максимальное расстояние между коровами, рассматривая отрезок координат от середины до конца.

Пока начало координат не станет больше или равным концу, повторяем данный алгоритм.

Сложность алгоритма:

O(n * log(last_element - first_element)) , где n – количество чисел. last_element – координата последнего стойла, first_element – координата первого стойла

Код алгоритма:

```
#include<iostream>
using namespace std;

int main() {

    //количество стойл и коров соответственно
    int n, k;
    cin >> n >> k;

    //координаты стойл
    int array_stall[n];
    for (int i = 0; i < n; i++) {
        cin >> array_stall[i];
    }

    int first_element = 0;
    int last_element = array_stall[n - 1] - array_stall[0];

    while (first_element < last_element) {

        //первая корова всегда стоит в первом стойле
        int cows_in_the_stall = 1;

        int middle = (first_element + last_element + 1) / 2;
        int first_element_tmp = array_stall[0];

        for (int i = 1; i < n; i++) {
```

```
            if ((array_stall[i] - first_element_tmp) >= middle) {
                //ставим корову в стойло
                cows_in_the_stall++;
                //считаем расстояние от нового стойла
                first_element_tmp = array_stall[i];
            }
        }

        if (cows_in_the_stall >= k) {
            first_element = middle;
        }
        else {
            last_element = middle - 1;
        }

    }
    cout << first_element;
    return 0;
}
```

Задача №F «Число»

Пояснение к примененному алгоритму:

Для оптимального нахождения максимального числа используется сортировка пузырьком для чисел при которой рассматривается каждая пара чисел с продвижением по массиву.

Рассматривается какое число в паре будет больше: объединение первого и второго чисел в одно или наоборот. В соответствии с этим числа меняются местами. Такой проход по числам производится несколько раз, пока все числа не будут переставлены.

Сложность алгоритма:

$O(n^2)$, где n – количество чисел

Код алгоритма:

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

using namespace std;

//для сравнения двух строк
bool is_right_position(string& a, string& b) {
    return (a + b) > (b + a);
}
```

```
int main() {

    //считывание числа в файл
    vector<string> numbers;
    ifstream inputFile("number.in");
    string num;
    while (inputFile >> num) {
        numbers.push_back(num);
    }
    inputFile.close();

    //длина массива
    int length = numbers.size();

    //сортировка чисел
    for (int i = 0; i < length - 1; ++i) {
        for (int j = 0; j < length - i - 1; ++j) {

            if (!is_right_position(numbers[j], numbers[j + 1])) {
                string tmp = numbers[j];
                numbers[j] = numbers[j + 1];
                numbers[j + 1] = tmp;
            }
        }
    }

    //объединение числа в одну строку
    string max_of_numbers;
    for (string& num : numbers) {
        max_of_numbers += num;
    }

    cout << max_of_numbers << endl;

}
```

Задача №G «Кошмар в замке»

Пояснение к примененному алгоритму:

Суть алгоритма сводится к тому, чтобы найти буквы, которые встречаются более одного раза и их крайние местонахождения удалить из строки и расположить в порядке убывания по весу их индексов. После этого мы вставляем между оставшейся изначальной строкой наши отсортированные по убыванию буквы справа от строки и отсортированные по возрастанию буквы слева от строки.

За счет этого буквы между которыми мы будем считать расстояния будут иметь максимальное расстояния для максимальных весов, что позволит получить максимальную

сумму – так как для максимальной суммы необходимо, чтобы для максимального расстояния были максимальные индексы.

Сложность алгоритма:

O(n + m log m), где n – количество символов в строке, а m – количество букв которые встречаются в строке более одного раза

Код алгоритма:

```
#include<iostream>
#include<fstream>
#include<vector>
#include <map>
#include <algorithm>
#include <unordered_map>

using namespace std;

int main() {

    //переменные
    string input_string;
    int weights[26];
    unordered_map<char, int> weights_for_letters;
    char letter = 'a';
    vector<int> word_weights;

    //считывается слово
    cin >> input_string;

    //считываются веса
    for (int i = 0; i < 26; ++i) {
        cin >> weights[i];
    }

    //заполняются веса для каждой буквы
    for (int weight : weights) {
        weights_for_letters[letter++] = weight;
    }

    //1. создать ассоциативный массив для каждой цифры (буквы)
    map<char, vector<int>> letter_indexes;

    //2. написать для каждой буквы ее индекс, когда она встречается
    for (int i = 0; i < input_string.size(); ++i) {
        char letter = input_string[i];
        letter_indexes[letter].push_back(i);
    }
```

```
    //3. если встречается более одного раза, то удалить ее первый и
последний индекс, при этом сохранив информацию о цифре в отдельный массив
    vector<char> repeated_chars;
    for (map<char, vector<int>>::iterator it = letter_indexes.begin(); it
!= letter_indexes.end(); ++it) {
        if (it->second.size() > 1) {
            repeated_chars.push_back(it->first);
            letter_indexes[it->first].erase(letter_indexes[it-
>first].begin());
            letter_indexes[it->first].pop_back();
        }
    }

    //4. массив с буквами которые встречаются более одного раза
отсортировать, реверснуть и между ними вставить оставшиеся цифры из
строки
    sort(repeated_chars.begin(), repeated_chars.end(),
[&weights_for_letters](char a, char b) {
        return weights_for_letters[a] > weights_for_letters[b];
    });

    //строка, содержащая буквы из repeated_chars в обратном порядке
    string reversed_repeated_chars;
    for (auto it = repeated_chars.rbegin(); it != repeated_chars.rend();
++it) {
        reversed_repeated_chars += *it;
    }

    //формируем результат
    string result(repeated_chars.begin(), repeated_chars.end());

    //восстановим строку по новым индексам
    for (map<char, vector<int>>::const_iterator it =
letter_indexes.begin(); it != letter_indexes.end(); ++it) {
        char letter = it->first;
        const vector<int>& indexes = it->second;
        for (size_t i = 0; i < indexes.size(); ++i) {
            result += letter;
        }
    }

    //добавить реверснутую строку
    result += reversed_repeated_chars;

    //5. готово!
    cout << result << endl;
}
```

Задача №H «Магазин»

Пояснение к примененному алгоритму:

Для получения максимально скидки и, соответственно, минимальной суммы, которую нужно будет потратить на оплату необходимо, чтобы мы удаляли каждый k-ый элемент из чека только один раз за весь чек. То есть в одном чеке необходимо иметь по k продуктов.

Тогда, чтобы получить максимальную скидку нужно, чтобы каждый чек состоял из максимально дорогих продуктов – для этого отсортировываются суммы по убыванию и каждый k-ый наибольший по сумме элемент получит скидку – то есть его стоимость будет равна 0.

Сложность алгоритма:

O(n), где n – количество продуктов в чеке

Код алгоритма:

```
#include<iostream>
#include <vector>
#include <algorithm>
using namespace std;

int main() {

    //переменные
    int n;
    int k;
    vector<int> prices(n);
    int sum = 0;

    //считываем кол-во товаров (n) и номер бесплатного товара (k)
    cin >> n >> k;

    //считываем цены товаров
    for (int i = 0; i < n; i++) {
        cin >> prices[i];
    }

    //сортируем цены по убыванию
    sort(prices.begin(), prices.end(), greater<int>());

    //считаем цену
    int num = 0;
    for (int i = 0; i < n; i++) {
        num++;
        if (num == k) {
            num = 0;
```

```
            continue;
        }
        sum += prices[i];
    }

    cout << sum;
}
```