МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«Национальный исследовательский университет ИТМО»

ФАКУЛЬТЕТ ПРОГРАММНОЙ ИНЖЕНЕРИИ И КОМПЬЮТЕРОНОЙ ТЕХНИКИ

ЛАБОРАТОРНАЯ РАБОТА №2

по дисциплине
«Алгоритмы и структуры данных»
БАЗОВЫЕ ЗАДАЧИ

Выполнил работу:
Студент группы P3218
Рамеев Тимур Ильгизович
Преподаватель:
Косяков М. С.
Тараканов Д. С.

Санкт-Петербург 2024

Оглавление

Коровы в стойло

Исходный код

```cpp
#include <iostream>
#include <vector>
#include <algorithm>

using namespace std;

int main() {
    int n, k;
    cin >> n >> k;
    vector<int> arr;
    int minimum = 10000000;
    for (int i = 0; i < n; i++) {
        int input;
        cin >> input;
        arr.push_back(input);
        if (i != 0) {
            minimum = min(arr[i] - arr[i - 1], minimum);
        }
    }
    int maximum = (arr[size(arr) - 1] - arr[0] + k - 2) / (k - 1);
    int current_value = 0;
    while (maximum != minimum) {
        int counter = 1;
        current_value = (minimum + maximum + 1) / 2;
        //cout << maximum << " " << current_value << " " << minimum << endl;
        int flag = 0;
        int temp = 0;
        for (int i = 1; i < n; i++) {
            int difference = arr[i] - arr[i - 1] + temp;
            if (difference >= current_value) {
                counter++;
                temp = 0;
            }
            else {
                temp = difference;
            }
            if (counter == k) {
                flag = 1;
                break;
            }
        }
        if (flag == 1) {
            minimum = current_value;
        }
        else {
            maximum = current_value - 1;
        }
    }
    cout << maximum;
    return 0;
}
```

Пояснения к примененному алгоритму

Входные данные:

- n – количество стойл, входной параметр
- k – количество коров
- maximum – верхняя граница отрезка для бинарного поиска максимального расстояния между стойлами
- minimum – нижняя граница отрезка для бинарного поиска максимального расстояния между стойлами
- arr – массив, содержащий координаты стойл, входной параметр

Алгоритм выполнения:

Первым делом в цикле считаем все координаты из потока ввода, параллельно с этим найдем минимум для нашего максимального расстояния. Он будет равен минимальному расстоянию между координатами.

Далее рассчитаем максимум для максимального расстояния. Это будет расстояние между первым и последним стойлами, деленное на k – 1. Так как коров k промежутков между ними должно быть k – 1.

Затем осуществляем бинарный поиск. Проверка на то, подходит ли текущее расстояние для ответа или нет, заключается в следующем. Сначала мы заводим счетчик counter, для количества элементов, находящихся на расстоянии difference большем или равном текущему current_value. Изначально он равен единице, так как первое стойло всегда будет занято коровой (надеюсь, понятно почему). Затем мы пробегаемся по координатам стойл (элементам массива arr) и когда расстояние difference становится большим или равном текущему current_value, увеличиваем счетчик. Когда counter достиг количества коров k, можно сказать, что промежуток current_value подходит и попробовать рассмотреть расстояния больше текущего current_value. Если мы прошли весь массив arr и counter так и не достиг количества коров k, значит расстояние current_value слишком большое, и следует взять его более меньшим.

Таким образом бинарный поиск точно рассчитаем нам значение максимального расстояния между стойлами, и все, что нам нужно сделать, это вывести его на экран.

Сложность:

$$O(\mathrm{n} * \log_2 T)\ ,$$

где $\mathrm{T} = \left(\frac{n_{max} - n_{min}}{k} - d\right)$,

где d – минимальное расстояние между стойлами

Число

Исходный код

```
#include <iostream>
#include <vector>
#include <string>
using namespace std;
int main() {
	string input;
	vector<string> arr;
	for (int i = 0; cin >> input; i++)
		arr.push_back(input);
	vector<string> sorted_arr;
	int index = 0;
	for (int i = 0; i < size(arr); i++) {
		string max_str = "!";
		for (int j = 0; j < size(arr); j++) {
			if ((max_str + arr[j]).compare(arr[j] + max_str) < 0) {
				max_str = arr[j];
				index = j;
			}
		}
		arr[index] = "!";
		//cout << max_str << " -  max" << endl;
		sorted_arr.push_back(max_str);
	}
	for (string str : sorted_arr) {
		cout << str;
	}
	return 0;
}
```

Пояснения к примененному алгоритму

Входные данные:

- arr – входной массив чисел
- sorted_arr – отсортированный массив чисел

Алгоритм выполнения:

В первом цикле заполняем массив arr входными данными, пока они есть.

Во втором цикле заполняем массив sorted_arr, уже отсортированными значениями. Сравнение двух чисел, на определение большего из них происходит по следующему алгоритму. Сравниваются две строки: (str1 + str2) и (str2 + str1), если строка (str1 + str2) больше, значит str1 считается больше str2, иначе наоборот. Так мы сравниваем все элементы в массиве arr во вложенном цикле. Таким образом мы определяем текущий максимум в массиве arr и именно его добавляем в sorted_arr. Спустя n^2 операций цикл завершается.

Теперь, все, что нам осталось – это вывести отсортированный массив без разделителей между элементами.

Сложность:

$$O(n^2)$$

Кошмар в замке

Исходный код

```cpp
#include <iostream>
#include <map>
#include <vector>

using namespace std;

int main() {

	struct label {
		long long size;
		int count;
	};

	string str;
	cin >> str;
	map<char, struct label> labels;
	char current_label = 'a';
	long long input;
	for (int i = 0; i < 26; i++) {
		cin >> input;
		labels[current_label++] = (struct label) { input, 0};

	}
	vector<char> was_added; // contains keys
	for (int i = 0; i < str.length(); i++) {
		labels[str[i]].count++;
		if (labels[str[i]].count == 2) {
			was_added.push_back(str[i]);
			char cur_symbol = str[i];
			str.erase(i, 1);
			str.erase(str.find(cur_symbol), 1);
			i -= 2;
		}
		if (size(was_added) == 26) {
			break;
		}
	}
	string str_to_add = "";
	for (int i = 0; i < size(was_added); i++) {
		int current_max = -1;
		int index = -1;
		for (int j = 0; j < size(was_added); j++) {
			if (labels[was_added[j]].size > current_max) {
				index = j;
				current_max = labels[was_added[j]].size;
			}
		}
		str_to_add += was_added[index];
		was_added.erase(was_added.begin() + index);
		i--;
	}
	str = str_to_add + str;
	reverse(str_to_add.begin(), str_to_add.end());
	cout << str << str_to_add;
	return 0;
}
```

Пояснения к примененному алгоритму

Входные данные:

- struct label – структура, описывает букву
  - поле size – вес буквы, задается входными данными
  - поле count – количество букв, встретившихся при проходе по строке
- str – входная строка
- labels – map содержащий имя буквы (char) и соответствующую структуру label для этой буквы
- was_added – массив, содержащий все буквы, которые встретились хотя бы 2 раза
- str_to_add – строка, которая добавляется слева и справа от основной строки, в алгоритме рассказано, зачем это нужно.

Алгоритм выполнения:

В первом цикле мы инициализируем map labels, присваивая значениям size значения с потока ввода, а полям count – 0.

Далее, во втором цикле, мы проходим по строке, и если вдруг встречаем букву 2 раза, удаляем эти два вхождения в строке, и добавляем букву в массив was_added. Таким образом, по окончании цикла в массиве was_added расположены буквы, которые при правильном расположении дадут максимально большой вес строки.

Теперь несколько слов о том, как именно нужно располагать эти элементы. Понятно, что достаточно расположить только одну из пар конкретной буквы на максимальное расстояние, так как в условии учитывается только максимальное расстояние между буквой. Именно поэтому мы удаляем только два вхождения в строке, а остальные нам не нужны. Теперь эти элементы нужно добавить обратно в строку. Логично располагать их как можно дальше друг от друга, чтобы увеличить вес строки. Так как все буквы имеют разный вес, логично, что самое большое расстояние должна иметь самая большая буква. Поэтому, ее мы расположим по краям нашей строки. На расстоянии, меньшем максимального на 2 будет находиться буква, чуть меньшего веса, то есть на второй позиции с начала и с конца.

Таким образом, все что нам нужно сделать, это отсортировать массив was_added по невозрастанию элементов и составить из отсортированных элементов строку. Именно это и делает третий цикл в программе. После него мы получаем строку str_to_add, содержащую буквы в порядке невозрастания по весу, которые мы исключили из основной строки в предыдущем цикле.

Последним шагом мы добавляем эту строку к основной в начале, затем переворачиваем ее (надеюсь понятно зачем) и добавляем к концу основной строки. Эту основную строку выводим на экран.

Сложность:

$$O(n)$$

Магазин

Исходный код

```cpp
#include <iostream>
#include <vector>
using namespace std;
int main() {
	int n, k;
	cin >> n >> k;
	vector<int> arr;
	int ret_sum = 0;
	for (int j = 0; j < n; j++) {
		int input;
		cin >> input;
		bool flag = false;
		for (int i = 0; i < size(arr); i++) {
			if (input >= arr[i]) {
				auto iter = arr.begin();
				arr.insert(iter + i, input);
				flag = true;
				break;
			}
		}
		if (!flag)
			arr.push_back(input);
		ret_sum += input;
	}
	for (int i = 1; i <= n; i++) {
		if (i % k == 0) {
			ret_sum -= arr[i - 1];
		}
	}
	cout << ret_sum;
	return 0;
}
```

Пояснения к примененному алгоритму

Входные данные:

- n – количество товаров
- k – номер бесплатного товара в чеке
- arr – отсортированный массив размера n содержащий цены товаров

Алгоритм выполнения:

Алгоритм прост. В первом цикле мы заполняем массив arr входными данными, соблюдая порядок невозрастания. Таким образом, к окончанию цикла, мы получаем полностью отсортированный массив arr в порядке невозрастания.

Далее, мы должны пробежать еще раз уже по отсортированному массиву. Идея в том, чтобы исключать из чека максимально большие цены. Такие цены находятся в отсортированном массиве с индексами кратными k. Таким образом исключая из общей суммы эти цены мы получаем минимально возможный чек.

Сложность:

$$O(n^2)$$