Университет ИТМО

Факультет программной инженерии и компьютерной техники

**Лабораторная работа №2**

по «Алгоритмам и структурам данных»

Timus

Выполнил:

Студент группы P32092

Голиков А.С.

Преподаватели:

Косяков М.С.

Тараканов Д.С.

Санкт-Петербург

2023

Задача №1322 «Шпион»

Пояснение к примененному алгоритму:

В голову приходит наивное решение - давайте сортанем все входные буквы, так мы получим первый столбец. (Визуально восстановим первый и последний столбец таблицы кодирования)

Потом если сделаем один циклический сдвиг (допишем перед каждой первой буквой ей соответствующую последнюю) - Получим уже ряд из двубуквенных последовательностей, которые действительно были в зашиврованном слове.

Если сортануть эту последовательность, мы получим уже первые два столбца кодирования, так как действительно для каждой из имеющихся пар букв есть перестановка, начинающаяся с нее.

Проделав такую операцию n раз (количество букв в слове), восстановим таблицу целиком - осталось выбрать нужный ряд и задача решена

К сожалению, это решение работает слишком медленно, так что надо думать..(

Методом пристального взгляда заметим, что сортировка всегда одинаковая - то есть, если после первого дописывания буква на 6 строке теперь возглавляет третью, то и потом после дописывания этой самой 6 буквы, после сортировки эта строка станет на 3 место.

Это верно, потому что правый столбец всегда отсортирован и при дописывании левого меняется порядок только так, как менялся бы при сортировке этого самого левого ряда.

Так что достаточно просто запомнить, куда уходит какая буква и восстановить этот путь.

Асимптотика - O(n), где n - длина слова

Код решения:

#include <iostream>

#include <vector>

using namespace std;

int main() {

int row;

string input;

cin >> row >> input;

int n = input.size();

vector<int> count('z' - 'A' + 1, 0);

int m = count.size();

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

count[input[i] - 'A']++;

}

int sum = 0;

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

sum += count[i];

count[i] = sum - count[i];

}

vector<int> t(n);

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

t[count[input[i] - 'A']] = i;

count[input[i] - 'A']++;

}

int j = t[row - 1];

vector<char> ans(n);

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

ans[i] = input[j];

j = t[j];

}

for (auto i: ans) {

cout << i;

}

return 0;

}

Задача №1604 «В Стране Дураков»

Задача - заполнить массив числами и минимизировать повторения.

Предлагаю разбить массив на подмассивы в количестве повторений самого частого числа, а потом просто по порядку убывания частоты чисел заполнять эти массивы. Пример:

Входные данные:

4

2 1 2 3

Самое частое число - 4, его будет целых 3. Так что разбиваем итоговый массив на 3 и начинаем заполнять:

[4], [4], [4]

Теперь заполняем следующим по частоте числом: 3 (2 повторения)

[4, 3], [4, 3], [4]

Теперь следующим (1 - 2 повторения)

[4, 3], [4, 3], [4, 1]

[4, 3, 1], [4, 3], [4, 1]

И осталось только одно число 2

[4, 3, 1], [4, 3, 2], [4, 1]

В такой расстановке в каждом подмассиве числа идут по убыванию их частоты, так что повториться внутри подмассива не могут.

Значит, если есть повторы (повторами я называю два подряд одинаковых числа), то они только между блоками.

Но такое может быть только если в первый блок (из соседних) попало только одно число - самое частое, то есть тот случай, когда самое большое n\_i > n/2 (обозначения из условия)

Но в таком случае, очевидно, невозможно избежать повторов. Оптимальная расстановка чисел - чередование этого частого числа и других, а наша расстановка именно так все и делает

Асимптотика - O(nlogn)

Код решения:

#include <iostream>

#include <vector>

#include <algorithm>

using namespace std;

int main() {

int k;

cin >> k;

vector<pair<int, int>> signs(k);

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

int x;

cin >> x;

signs[i] = make\_pair(x, i + 1);

}

sort(signs.begin(), signs.end(), greater<>());

int size = signs[0].first;

vector<vector<int>> locations(size);

int ind = 0;

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

while (signs[i].first) {

locations[ind].push\_back(signs[i].second);

--signs[i].first;

ind = (ind + 1) % size;

}

}

for (const auto &p: locations) {

for (auto t: p) {

cout << t << " ";;

}

}

return 0;

}