Федеральное государственное автономное образовательное учреждение высшего образования Национальный исследовательский университет ИТМО

Факультет программной инженерии и компьютерной техники

Алгоритмы и структуры данных

Задачи E, F, G, H (Яндекс.Контест)

Выполнил: студент группы P3208, Васильев Н. А.

Преподаватель: Косяков М. С.

Санкт-Петербург 2025

# Задача E. Коровы в стойла

Код решает задачу разбиения маршрута на k или больше частей с учетом расстояний между координатами так, чтобы минимальное значение наибольшей длины части было максимально возможным.

Заполнение distances происходит на этапе ввода данных, чтобы избежать лишнего прохождения по массиву. Для бинарного поиска оценка количества итераций буден **O(log S)**, где S = sum(distances), для проверки каждой длины: **O(n)** — один проход по массиву расстояний. Итого эффективность времени – **O(n log S)**.

Память используется линейно: coordinates: **O(n)** distances: **O(n)**.

## Код:

#include <iostream>

#include <numeric>

#include <sstream>

#include <vector>

**using** **namespace** std;

**int** **main**() {

**int** n;

**int** k;

cin >> n >> k;

vector<**int**> coordinates(n);

vector<**int**> distances(n - **1**);

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

cin >> coordinates[i];

**if** (i > **0** && i <= n) {

distances[i - **1**] = (coordinates[i] - coordinates[i - **1**]);

}

}

**int** first = **1**;

**int** last = accumulate(distances.begin(), distances.end(), **0**);

**while** (first < last) {

**int** middle = last - (last - first) / **2**;

**int** parts = **1**;

**int** sum = **0**;

**for** (**int** d : distances) {

sum += d;

**if** (sum >= middle) {

parts++;

sum = **0**;

}

}

**if** (parts >= k) {

first = middle;

} **else** {

last = middle - **1**;

}

}

cout << first << endl;

**return** **0**;

}

# Задача F. Число

Все числа сортируются, используя компаратор, first + second > second + first; Таким образом, мы сравниваем две строки как потенциальных соседей в результирующем числе: какая комбинация даст большее значение. Время сортировки: количество сравнений: **O(n log n)**, каждое сравнение: **O(m)**, где m — средняя длина строки. Итого: **O(n log n \* m)**.

Вектор result хранит n строк, соответственно используемая память: **O(n \* m)**.

Код:

#include <algorithm>

#include <iostream>

#include <vector>

**using** **namespace** std;

**bool** **comparator**(**const** string& first, **const** string& second) {

**return** first + second > second + first;

}

**int** **main**() {

vector<string> result;

string num;

**while** (cin >> num) {

result.push\_back(num);

}

sort(result.begin(), result.end(), comparator);

**for** (**const** **auto**& n : result) {

cout << n << "";

}

cout << endl;

**return** **0**;

}

# Задача G. Кошмар в замке

Мы из входной строки и весов букв пытаемся сформировать максимально весомую итоговую строку. Вес буквы влияет на приоритет – чем больше вес, тем раньше используется пара в строке. Пара букв с наибольшим весом должна открывать и закрывать строку. Буквы, не имеющие пары, должны быть в центре строки в любом порядке, так как на вес они не влияют. Для этого отберем буквы, у которых count >= 2, и добавим их в вектор repeated\_letters как пары (вес, буква), а затем отсортируем его по убыванию веса. Используя края строки left и right вставим пары, а оставшиеся позиции заполним непарными символами.

Подсчёт частот выполняется за **O(n), с**ортировка repeated\_letters за **O(26 log 26) = O(1)**, проход по строке с расстановкой – **O(n)**. Итого: **O(n)** — линейная по длине строки

Использование памяти:

* s — массив размера n
* middle\_chars — максимум n
* free\_positions — максимум n

Итого: O(n) по памяти.

Код:

#include <algorithm>

#include <iostream>

#include <queue>

#include <vector>

**using** **namespace** std;

**int** **main**() {

string input;

cin >> input;

vector<**int**> weights(**26**);

**for** (**int** i = **0**; i < **26**; i++) {

cin >> weights[i];

}

vector<**int**> reps(**26**, **0**);

**for** (**char** c : input) {

reps[c - 'a']++;

}

vector<pair<**int**, **char**>> repeated\_letters;

**for** (**int** i = **0**; i < **26**; i++) {

**if** (reps[i] >= **2**) {

repeated\_letters.push\_back({weights[i], 'a' + i});

}

}

sort(repeated\_letters.rbegin(), repeated\_letters.rend());

vector<**char**> s(input.size(), '?');

vector<**char**> middle\_chars;

**int** left = **0**;

**int** right = input.size() - **1**;

**for** (**auto** [weight, ch] : repeated\_letters) {

**int** count = reps[ch - 'a'];

**if** (count >= **2**) {

**if** (left <= right)

s[left++] = ch;

**if** (left <= right)

s[right--] = ch;

**for** (**int** i = **0**; i < count - **2**; i++) {

middle\_chars.push\_back(ch);

}

}

reps[ch - 'a'] = **0**;

}

queue<**int**> free\_positions;

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

**if** (s[i] == '?') {

free\_positions.push(i);

}

}

**for** (**size\_t** i = **0**; i < **26**; i++) {

**if** (reps[i] == **1** && !free\_positions.empty()) {

s[free\_positions.front()] = 'a' + i;

free\_positions.pop();

}

}

**for** (**char** ch : middle\_chars) {

**if** (!free\_positions.empty()) {

s[free\_positions.front()] = ch;

free\_positions.pop();

}

}

**for** (**char** c : s)

cout << c;

cout << endl;

**return** **0**;

}

# Задача H. Магазин

Для того, чтобы выбрать товары так, чтобы максимизировать сумму их цен, при том, что каждый k-й товар идёт «бесплатно» нужно отсортировать prices по убыванию, а затем просуммировать их, пропуская каждый k-й товар.

Сортировка занимает **O(n log n)**, проход по вектору и суммирование – **O(n)**. Итого: **O(n log n)**.

Основное влияние на память составляет вектор prices размера n: **O(n)**.

Код:

#include <algorithm>

#include <iostream>

#include <vector>

**using** **namespace** std;

**int** **main**() {

**int** n;

**int** k;

cin >> n >> k;

vector<**int**> prices(n);

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

cin >> prices[i];

}

sort(prices.rbegin(), prices.rend());

**int** sum = **0**;

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

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

sum += prices[i];

}

}

cout << sum << endl;

**return** **0**;

}