Университет ИТМО

Факультет программной инженерии и компьютерной техники

Лабораторная работа №1

по «Алгоритмам и структурам данных»

Блок 2

Выполнил:
Студент группы P3216
Билошийцкий Михаил Владимирович

Преподаватели:
Косяков М.С.
Тараканов Д.С.

Санкт-Петербург
2024

E. Коровы и стойла

Решение:
Решение задачи достигается бинарным поиском. Вначале у нас есть переменная n – количество стойл, k – количество коров и coords – отсортированный массив координат стойл. Далее я запускаю бинарный поиск максимального минимального расстояния между коровами, помещёнными в стойла по массиву coords, где переменная left – это минимальная дистанция, а right – максимальная, а функция canPlace – говорит, можем ли мы разместить при такой дистанции коровы в стойла правильно или нет.

Дополнительные пояснения в коде:

```
#include <iostream>

bool canPlace(int32_t* coords, int32_t n, int32_t k, int32_t dist);

int main(int argc, char** argv) {
    // Количество стойл
    int32_t n;
    // Количество коров
    int32_t k;
    // n координат стойл
    int32_t* coords = new int32_t[n];

    std::cin >> n >> k;
    for (int32_t i = 0; i < n; i++) {
        std::cin >> coords[i];
    }

    int32_t left = 0;
    int32_t right = coords[n - 1] - coords[0];
    int32_t result = -1;

    // Бинарный поиск максимального минимального расстрояния
    while (left <= right) {
        int32_t mid = left + (right - left) / 2;
        if (canPlace(coords, n, k, mid)) {
            result = mid;
            left = mid + 1;
        } else {
            right = mid - 1;
        }
```

```
    }

    std::cout << result << std::endl;

    delete[] coords;
    return 0;
}

// Можно ли разместить коров при такой минимальной дистанции
bool canPlace(int32_t* coords, int32_t n, int32_t k, int32_t dist) {
    int32_t last_cord = coords[0];
    int32_t placed = 1;
    for (int32_t i = 0; i < n; i++) {
        if (coords[i] - last_cord >= dist) {
            placed++;
            if (placed == k) {
                return true;
            }
            last_cord = coords[i];
        }
    }
    return false;
}
```

F. Число

Решение:

Задача достаточно тривиальная. Единственное, что нужно сделать – это считать данные и отсортировать их, применяя собственный компаратор сортировки путём сцепления двух строк и проверки, что из них больше и большее число поставить первее, чем меньшее. Отсортировав вектор строк и соединив их, мы получим наибольшее число из введённых кусочков.

Дополнительные пояснения в коде:

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

bool comparator(std::string a, std::string b);
```

```
int main(int argc, char** argv) {
    using namespace std;
    vector<string> nums;
    string row;
    while (cin >> row) {
        nums.push_back(row);
    }

    // Сортировка, применяя свой компаратор
    sort(nums.begin(), nums.end(), comparator);

    for (size_t i = 0; i < nums.size(); i++) {
        cout << nums[i];
    }
    cout << endl;
    return 0;
}

// Функция-компаратор двух строк для верной сортировки
// Сцепляет строки и смотрит, где получилось больше число
bool comparator(std::string a, std::string b) {
    return a + b > b + a;
}
```

G. Кошмар в замке

Решение:

Так как в задаче от нас требуется получить строку с максимальным весом, где вес вычисляется как максимальное расстояние между позициями, в которых стоит эта буква, перемноженная на вес этой буквы, то становится очевидно, что для решения данной задачи имеет смысл рассматривать только те буквы, что встречаются 2 раза и более в строке.
Мой алгоритм заключается в следующем:
Сперва я считываю строку s и массив весов weights.
Далее создаю структуру map<char, pair<int32_t, int32_t> > letter_counter, где ключём является буква, а значением пара: количество и последний найденный индекс буквы в строке s.
Далее я пробегаюсь по строке, заполняю letter_counter и как только замечаю, что буква встречается уже ровно 2 раза, то добавляю её в строку, что будет стоять вначале, а по индексу этой буквы и тому, что был найден до этого в строке s я ставлю 0, чтобы потом этот символ пропустить при выводе, так как мы их уже переставили.
Далее я применяю сортировку к началу строки по весам, что были нам даны.
Потом вывожу начало строки, середину s с пропуском нулевых символов и конец строки, что является реверсом начала.

Дополнительные пояснения в коде:

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

#define ALPH_LENGTH 26
#define FIRST_LETTER 'a'

bool comparator(char a, char b, int32_t* weights);

int main(int argc, char** argv) {
    using namespace std;
    // Строка
    string s;
    // Веса
    int32_t weights[ALPH_LENGTH];
    // Сколько раз встречалась буква в строке:
    // буква -> количество, последний найденный индекс
    map<char, pair<int32_t, int32_t> > letter_counter;

    cin >> s;
    for (size_t i = 0; i < ALPH_LENGTH; i++) {
        cin >> weights[i];
        // Поначалу ещё ничего не найдено
        letter_counter[FIRST_LETTER + i] = make_pair(0, -1);
    }

    // Строка, что будет стоять вначале и реверсивно в конце
    string start = "";

    for (size_t i = 0; i < s.length(); i++) {
        char c = s[i];
        pair<int32_t, int32_t>* v = &letter_counter[c];
        // Увеличиваем счётчик у той буквы, что нам встретилась
        v->first++;
        int32_t last_index = v->second, cur_index = i;
        // Устанавливаем новый индекс
        v->second = cur_index;
```

```
        // Если буква встретилась 2 раза, то берём её в учёт тех, что будут
переставлены
        if (v->first == 2) {
            start.push_back(c);
            // Записываем туда нули, чтобы эти символы пропустить при выводе
            s[last_index] = '0';
            s[cur_index] = '0';
        }
    }

    // Сортировка начала строки по весам
    sort(
        start.begin(), start.end(),
        [&](char a, char b) {
            return comparator(a, b, weights);
        }
    );

    // Вывод начала строки
    cout << start;
    // Вывод центра с пропуском тех букв, что были переставлены
    for (size_t i = 0; i < s.length(); i++) {
        if (s[i] != '0') cout << s[i];
    }
    // Вывод конца строки (реверс начала)
    for (size_t i = start.length(); i > 0; i--) {
        cout << start[i - 1];
    }
    cout << endl;

    return 0;
}

// Сравнение двух символов по весам
bool comparator(char a, char b, int32_t* weights) {
    return weights[a - FIRST_LETTER] > weights[b - FIRST_LETTER];
}
```

H. Магазин

Решение:

Если мы отсортируем цену товаров в порядке убывания, то есть сделаем так, чтобы наиболее дорогие товары оказались в начале списка и пройдемся по ней циклом, каждый k-тый товар не прибавляя к итоговой сумме, то мы и получим самую выгодную стоимость с разбиением товаров на отдельные чеки в соответствии с параметром акции.

Делая сортировку в порядке убывания, мы можем удобно пройтись по массиву, пропуская каждый k-тый товар, будто бы мы покупаем N товаров N/K чеками, формируя чеки от самых дорогих товаров, к самым дешёвым. Например, у нас есть 11 товаров, k=5, мы сначала сформируем 2 чека из самых дорогих товаров, оставив 1 самый дешёвый товар на последний чек без скидки, тем самым максимально сэкономив.

Дополнительные пояснения в коде:

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

bool comparator(int32_t a, int32_t b);

int main(int argc, char** argv) {
    using namespace std;
    // Количество товаров
    int32_t n;
    // Параметр акции
    int32_t k;
    cin >> n >> k;
    // Цены на товары
    vector<int32_t> prices(n);
    for (size_t i = 0; i < n; i++) {
        cin >> prices[i];
    }

    // Сортировка
    sort(prices.begin(), prices.end(), comparator);

    // Подсчёт суммы со скидкой
    int32_t total = 0;
    for (size_t i = 0; i < n; i++) {
        if ((i + 1) % k != 0) {
            total += prices[i];
        }
    }
```

```
    cout << total << endl;
    return 0;
}

// Компаратор для сортировки в обратном порядке
bool comparator(int32_t a, int32_t b) {
    return a > b;
}
```