Университет ИТМО

Факультет программной инженерии и компьютерной техники

«Алгоритмы и структуры данных»

**отчет по блоку задач №2 (Яндекс.Контест)**

**Выполнил:**

Студент группы P3210

Федоров Евгений Константинович

**Преподаватель:**

Тараканов Денис Сергеевич

Санкт-Петербург, 2025г.

**Задача №1 «E. Коровы в стойла»**

**Пояснение к примененному алгоритму:**

Задача решается с использованием алгоритма “Binary Search”, где за искомое число будем брать наибольшее расстояние между коровами. Минимальное значение – 0, максимальное – разница между последней и первой координатой. Далее в цикле проверяем, можно ли уместить всех коров с расстоянием больше либо равным, чем average, где average равен половине суммы минимального и максимального расстояний. Если не удалось расставить всех коров, значит среднее значение должно быть ниже => верхняя граница становится средним. Если мы уложили всех коров, то фиксируем это расстояние в результирующей переменной и пробуем уместить коров с большим расстоянием (minRange = average).

**Алгоритмическая сложность:**

* **По времени**: Алгоритм по времени отрабатывает как обычный бинарный поиск со сложностью **O(n log n).**
* **По памяти**: Так как используется один вектор, то сложность по памяти составляет **O(n).**

Код:

|  |
| --- |
| #include <cmath>  #include <iostream>  #include <queue>  #include <vector>  using namespace std;  int main() {  int places, cows;  cin >> places >> cows;  vector<int> v(places);  for (int i = 0; i < places; i++) {  cin >> v[i];  }  int best = 0;  int minRange = 0, maxRange = v.back() - v.front(), average;  while (minRange <= maxRange) {  average = (maxRange + minRange) / 2;  int remainingCows = cows - 1;  int lastPlacesCow = v[0];  for (size\_t i = 0; i < v.size(); i++) {  if (abs(v[i] - lastPlacesCow) >= average) {  lastPlacesCow = v[i];  remainingCows--;  }  }  if (remainingCows <= 0) {  minRange = average + 1;  best = average;  } else {  maxRange = average - 1;  }  }  cout << best << endl;  return 0;  } |

**Задача №2 «F. Число»**

**Пояснение к примененному алгоритму:**

Данная задача решается с использованием алгоритма “Bubble sort”, сравнивания соседние элементы и переставляя их местами при определенном условии. В данном случае условием является максимальная сумма элементов при их конкатенации. То есть если int(str(A) +str(B)) < int(str(B) + str(A)), то меняем их местами. Выполнив такую сортировку, мы получим максимально возможное число.

**Алгоритмическая сложность:**

1. **По времени:** Алгоритм выполняется за стандартное для данного алгоритма время: **O(N2)**, так как для полной сортировки требуется вложенный цикл.
2. **По памяти:** По памяти алгоритм имеет сложность **O(N)**, так как используется вектор для хранения и сортировки данных.

Код:

|  |
| --- |
| #include <iostream>  #include <vector>  using namespace std;  int main() {  vector<string> numbers;  string line;  for (string input; cin >> input;) {  numbers.push\_back(input);  if (numbers.size() >= 2) {  for (size\_t j = 0; j < numbers.size(); j++) {  for (size\_t i = 1; i < numbers.size(); i++) {  if ((numbers[i - 1] + numbers[i]) < (numbers[i] + numbers[i - 1])) {  string buffer = numbers[i - 1];  numbers[i - 1] = numbers[i];  numbers[i] = buffer;  }  }  }  }  }  if (!numbers.empty()) {  for (string e : numbers) {  cout << e;  }  }  return 0;  } |

**Задача №3 «G. Кошмар в замке»**

**Пояснение к примененному алгоритму:**

В данной задаче от нас требуется расставить буквы в строке таким образом, чтобы строка имела наибольший вес. В данной задаче можно не учитывать буквы, которые встречаются меньше двух раз, так как их положение абсолютно не влияет на конечный вес строки, все внимание уделяется буквам, которые встречаются 2 раза. Добавляем букву и ее вес в мапу, после чего создаем мапу, где будут хранится количество раз, которое встретилась буква, после чего копируем это мапу в вектор и сортируем ее так по весу. Создаем три вектора для сохранения конечного результата. Если буква встречается хотя бы 2 раза, то добавляем ее в начало вектора leftSide и в конец вектора rightSide, остальные буквы сгружаем в вектор middle. Для завершения задачи последовательно выводим содержимое векторов.

**Алгоритмическая сложность:**

1. **По времени:** **O(N log N)** , так как сортировка вектора занимает именно такое время в данной задаче.
2. **По памяти:** **O(N),** так как используются простые вектор, каждый из которых в худшем случае хранит n данных.

Код:

|  |
| --- |
| #include <algorithm>  #include <deque>  #include <iostream>  #include <unordered\_map>  #include <vector>  using namespace std;  int main() {  vector<pair<char, int>> weights(26);  unordered\_map<char, int> freq\_map;  string input;  cin >> input;  for (int i = 0; i < 26; i++) {  int weighty;  cin >> weighty;  weights[i] = {'a' + i, weighty};  }  for (char c : input) {  freq\_map[c]++;  }  vector<pair<char, int>> frequency(freq\_map.begin(), freq\_map.end());  auto cmp = [&](const pair<char, int> a, const pair<char, int> b) {  return weights[a.first - 'a'].second > weights[b.first - 'a'].second;  };  sort(frequency.begin(), frequency.end(), cmp);  deque<char> leftSide, rightSide;  vector<char> middle;  for (const auto& p : frequency) {  if (p.second >= 2) {  leftSide.push\_back(p.first);  rightSide.push\_front(p.first);  for (int j = 0; j < p.second - 2; ++j) {  middle.push\_back(p.first);  }  } else {  middle.push\_back(p.first);  }  }  string result;  for (char e : leftSide)  result += e;  for (char e : middle)  result += e;  for (char e : rightSide)  result += e;  cout << result << endl;  return 0;  } |

**Задача №4 «H. Магазин»**

**Пояснение к примененному алгоритму:**

В данной задаче достаточно отсортировать чек в порядке убывания, так можно имитировать разбиение товаров на разные чеки для того, чтобы посчитать минимальную стоимость. То есть при каждом k – ом минимальном товаре, его цена становится равной нулю, нужно стараться обнулять как можно крупные товары, а значит их нужно ставить в один чек с товарами, которые дороже их. Сортируем товары в обратном порядке, каждый k-ый товар – бесплатный.

**Алгоритмическая сложность:**

1. **По времени:** **O(N log N)**, так как сортировка вектора в убывающем порядке занимает именно такое время.
2. **По памяти:** **O(N)**, так как используется лишь один вектор с максимальным количеством данных n

Код:

|  |
| --- |
| #include <algorithm>  #include <iostream>  #include <map>  #include <vector>  using namespace std;  int main() {  int n, k;  cin >> n >> k;  vector<int> nums(n);  for (int i = 0; i < n; i++) {  cin >> nums[i];  }  sort(nums.begin(), nums.end(), greater<int>());  for (size\_t i = 0; i <= nums.size(); i += k) {  if (i != 0) {  nums[i - 1] = 0;  }  }  int result = 0;  for (auto num : nums) {  result += num;  }  cout << result << endl;  return 0;  } |