Федеральное государственное автономное образовательное учреждение высшего образования «Национальный исследовательский университет ИТМО»

Факультет программной инженерии и компьютерной техники

Отчет по блоку №2 (яндекс контест)

По Алгоритмам и структурам данных

Выполнил:

Ступин Тимур Русланович

Группа № P3108

Поток № 1.3

Преподаватель:

Тараканов Денис Сергеевич

Санкт-Петербург 2025

Содержание

Задача 1 Коровы в стойла

Решение

Заметим, что если возможно расставить коров таким образом, чтобы расстояние между каждой парой было больше либо равно `d`, то такая расстановка существует для любого числа меньшего `d`. Также заметим, что если коров нельзя размещать таким образом чтобы расстояние между каждой парой было больше либо равно `D`, то этого нельзя сделать и для любого числа большего `D`. Таким образом получаем, что нужно найти максимальное возможное `d`, то есть такое, что для `d+1` искомого размещения уже не существует. Эта задача решается при помощи бинарного поиска. Для того чтобы проверять что текущее рассматриваемое значение `c` подходит или нет выполняется прямая проверка: каждая из коров ставится в стойло, расположенное на расстоянии больше либо равном `c` от предыдущего занятого стойла (`last`). Далее проверяется всех ли коров удалось расставить и в зависимости от этого устанавливаются новые значения `l` и `r`.

Сложность алгоритма

- По времени: Сложность бинарного поиска $logN$, при этом мы имеем внутренний цикл на N. В итоге получаем $O(NlogN)$
- По памяти: Память выделяется только для входных данных, дополнительно хранятся только локальные переменные. Получаем $O(1)$

Итого:

- По времени: $O(NlogN)$
- По памяти: $O(1)$

Код

```cpp
#include <bits/stdc++.h>
using namespace std;

int main() {
  int n;
  cin >> n;
  int k;
  cin >> k;
  vector<int> v(n);
  for (int i = 0; i < n; i++) {
    cin >> v[i];
  }
  int l = 0, r = v.back() + 1;
  while (r - l > 1) {
    int c = (l + r) / 2;
    int cnt = 1;
    int last = v[0];
    for (int i = 1; i < n && cnt < k; i++) {
      if (v[i] - last >= c) {
        last = v[i];
        cnt++;
      }
    }
    if (cnt == k) {
      l = c;
    } else {
      r = c;
    }
  }
  cout << l << endl;
  return 0;
}
```

Задача 2 Число

Решение

Идея решения в том, чтобы отсортировать числа в нужном порядке, для этого нужно правильно определить процедуру сравнения двух строк. Допустим мы сравниваем строки `s1` и `s2`. Очевидно, что в конечном итоге после их конкатенации мы хотим получить максимально возможное число. В таком случае просто сравним числа `s1 + s2` и `s2 + s1`. Это делается простым циклом с поэлементным сравнением.

Сложность алгоритма

- По времени: для сортировки использует функция `sort` и `stl`, которая реализует быструю сортировку за $O(NlogN)$. При этом внутри компаратора мы выполняем проход за $O(N)$. В итоге имеем $O(N^2logN)$
- По памяти: Дополнительной памяти кроме вводимых данных программа не использует. $O(1)$

Итого:

- По времени: $O(N^2logN)$
- По памяти: $O(1)$

Код

```cpp
#include <bits/stdc++.h>
using namespace std;

bool comp(const string& a, const string& b) {
  const string s1 = a + b;
  const string s2 = b + a;
  for (int i = 0; i < static_cast<int>(s1.size()); i++) {
    if (s1[i] == s2[i])
      continue;
    return s1[i] < s2[i];
  }
  return false;
}

int main() {
  vector<string> v;
  string s;
  while (cin >> s) {
    v.emplace_back(s);
  }
  sort(v.rbegin(), v.rend(), comp);
  string ans;
  for (const auto& a : v) {
    ans += a;
  }
  cout << ans << endl;
  return 0;
}
```

Задача 3 Кошмар в замке

Решение

Для начала для каждой буквы латинского алфавита посчитаем сколько раз она встречается в строке. В отдельном массиве сохраним соответствие между буквой и её весом, после чего произведем сортировку по убыванию веса. Приступим к формированию ответа. Заведём строки `l`, `r` и `c`, соответствующие левой, правой и центральной части результирующей строки. Далее будем просто обходить буквы в порядке убывания веса, и в случае если это возможно прибавлять по одному экземпляру текущей буквы к левой и правой части. Оставшиеся экземпляры будем помещать в центр. В конце нужно будет произвести конкатенацию строк `l`, `r` и `c` для получения ответа. Как итог, буквы с самым большим весом будут наиболее удалены друг от друга (в начале и конце строки), буквы вторые по весу будут следующие по удалённости и так далее. Тем самым в задаче использован жадный подход.

Сложность алгоритма

- По времени: Обходы по буквам и сортировка происходят за константное время, так как букв фиксированное количество. При этом формирование ответа происходит посимвольно, что эквивалентно проходу по входной строке. В итоге имеем сложность $O(N)$.
- Мы храним в памяти введенную строку, массив с количеством символов на 26 элементов, а также массив весов на 26 элементов. Итоговая сложность $O(N)$

Итого:

- По времени: $O(N)$
- По памяти: $O(N)$

Код

```cpp
#include <bits/stdc++.h>
using namespace std;

int main() {
  vector<int> used(26);
  vector<pair<int, int>> w(26);
  string s;
  cin >> s;
  for (const auto c : s) {
    used[c - 'a']++;
  }
  for (int i = 0; i < 26; i++) {
    int val;
    cin >> val;
    w[i] = {val, i};
  }
  sort(w.rbegin(), w.rend());
  string l, r, c;
  for (auto [_, a] : w) {
    const char sym = static_cast<char>(a + 'a');
    if (used[a] >= 2) {
      l += sym;
      r += sym;
    }
    for (int i = 0; i < (used[a] < 2 ? used[a] : used[a] - 2); i++) {
      c += sym;
    }
  }
  cout << l << c;
  for (int i = static_cast<int>(r.size()) - 1; i >= 0; i--) {
    cout << r[i];
  }
  cout << endl;
  return 0;
}
```

Задача 4 Магазин

Решение

Идея в том, что если отсортировать товары по убыванию стоимости, то выгоднее всего будет разделить все покупки на чеки по `k` товаров (возможно с некоторым остатком). Это так, потому что брать меньше `k` товаров очевидно не имеет смысла (бесплатных не будет), а брать блоками размера `m*k`, где `m` - целое число `>2` также неоптимально, так как получим `m` бесплатных товаров, но они будут самыми дешёвыми в это выборке и их сумма будет точно не больше суммы самых дешёвых товаров в каждой из `m` отдельных выборок по `k` товаров. Для формирования ответа достаточно в отсортированном массиве просуммировать все элементы кроме тех что стоят на позиция кратных `k`.

Сложность алгоритма

- По времени: Проходы по массиву выполнятся за линейной время, но в решении используется сортировка, поэтому получаем сложность $O(NlogN)$
- В памяти хранится только входной массив. Сложность $O(N)$

Итого:

- По времени: $O(NlogN)$
- По памяти: $O(N)$

Код

```cpp
#include <bits/stdc++.h>
using namespace std;

int main() {
  int n, k;
  cin >> n >> k;
  vector<int> v(n);
  for (int i = 0; i < n; i++) {
    cin >> v[i];
  }
  sort(v.rbegin(), v.rend());
  int sum = 0;
  for (int i = 0; i < static_cast<int>(v.size()); i++) {
    if ((i + 1) % k == 0)
      continue;
    sum += v[i];
  }
  cout << sum << endl;
  return 0;
}
```