Университет ИТМО

Факультет программной инженерии и компьютерной техники

Лабораторная работа №2

по «Алгоритмам и структурам данных»

Базовые задачи (блок 2)

Выполнила:

Студентка группы P3230

Вавилина Е. А.

Преподаватели:

Косяков М.С.

Тараканов Д.С.

Санкт-Петербург

2025

Задача №1 «E. Коровы в стойла»

Пояснение к примененному алгоритму:

Будем считать, что максимальное расстояние между коровами лежит в промежутке (-1; координата конца – координата начала + 1). Воспользуемся бинпоиском по всем числам, которые лежат в этом интервале.

Что бы понять, в каком из интервалов надо искать число будем размещать коров начиная с первой коровы в первом стойле с расстоянием большим или равным текущей середине отрезка - дельте (т.е. как только расстояние от предыдущей коровы до следующей стало больше или равным дельты – ставим корову и переходим к следующей). Если удалось разместить всех коров – значит можем попробовать число больше и возьмем правый интервал (дельта; конец). Если всех коров не получилось разместить с нужным интервалом – переходим в левый интервал (начало; дельта).

Таким образом будем делать пока правая граница минус 1 больше левой границы (т.е. их разность больше 1). После этого выводим правую границу и это и есть искомое расстояние.

Оценки:

1. Время:

   Основной цикл while реализует бинпоиск по числам из интервала (-1; координата конца – координата начала + 1), назовем длину интервала k, сложность работы алгоритма O(log k), логарифмическая.

   Внутри цикла while вложен цикл, пробегающий по всем стойлам, пока не рассиавит всех коров или не закончатся стойла. В худшем случае работает за O(n), где n – количество пользователей.

   Следовательно, общий порядок сложности **O(n · log(k))**.

2. Память:

   Код использует фиксированное количество переменных и массив, хранящий стойла, длина которого равно n.

   Следовательно, сложность по памяти **константная**, **O(n)**

Код:

```cpp
#include <cstddef>
#include <iostream>
#include <vector>

int main() {
  size_t n = 0;
  std::cin >> n;

  size_t cows = 0;
  std::cin >> cows;
```

```
  std::vector<int> stalls(n);
  for (size_t i = 0; i < n; i++) {
    std::cin >> stalls[i];
  }

  int start = -1;
  int end = stalls[n - 1] - stalls[0] + 1;

  while (end - 1 > start) {
    size_t prev_cow_place = 0;
    size_t left_to_place = cows - 1;
    const int delta = (end + start) / 2;

    for (size_t i = 0; i < n; i++) {
      if (stalls[i] - stalls[prev_cow_place] >= delta) {
        prev_cow_place = i;
        left_to_place -= 1;
      }

      if (left_to_place == 0) {
        break;
      }
    }

    if (left_to_place > 0) {
      end = delta;
    }

    if (left_to_place == 0) {
      start = delta;
    }
  }

  std::cout << start;
}
```

Задача №2 «F. Число»

Пояснение к примененному алгоритму:

Используем встроенный алгоритм Introsort (реализованный через sort) из std. Что бы сортировать строки с условием «a+b>b+a» т.е. к a выгоднее добавить b, чем к b добавить a, напишем дополнительно функцию сравнения.

Оценки:

1. Время:

При сортировке скорость работы алгоритма O(n · log(n)), при этом каждое сравнение строки идет за k, где k – максимальная длинна строки (в худшем случае). Следовательно общая сложность O(n · k · log(n)).

Время ввода/вывода данных O(n · k).

Итоговый ответ: **O(n · k · log(n))**

2. Память:

Код хранит вектор из n строк длины не более k, т.е. O(n · k). Сортировка требует O(log n) дополнительной памяти. Временные строки при сортировке (суммы для сравнений) – O(k). Следовательно сложность по памяти **O(n · k)**

Код:

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

bool static CmpStrings(std::string const& left, std::string const& right) {
  return left + right > right + left;
}

int main() {
  std::vector<std::string> numbs;
  std::string current_srt;

  while (std::cin >> current_srt) {
    numbs.push_back(current_srt);
  }

  std::sort(numbs.begin(), numbs.end(), CmpStrings);

  for (const auto& numb : numbs) {
    std::cout << numb << "";
  }
}
```

Задача №3 «G. Кошмар в замке»

Пояснение к примененному алгоритму:

Отсортируем символы в строке по убыванию веса, для каждого символа посчитаем, сколько раз он встречается в строке. Далее пробежимся по всем символам. Проходя каждый символ, сделаем следующее:

1. Если символ встречается меньше 2 раз – его вклад в вес строки нулевой, добавим его в строку, которая попадет в середину.

2. Если символ встречается 2 раза или больше, есть 2 варианта:
    a. Если осталось встретить после этого символа только 1 такой же символ – добавим его в строку, которая будет добавлена слева.
    b. Если это последний такой символ – добавим его в строку для добавления справа.
    c. Все другие случаи – добавляем в среднюю часть, так как они на вес не влияют.

В конце получаем:

- Левую строку, в которой идут неповторяющиеся символы, встречающиеся 2 и более раз, отсортированные по убыванию весов.
- Правую строку, в которой идут неповторяющиеся символы, встречающиеся 2 и более раз, отсортированные по убыванию весов. Что бы после этого максимально удалить друг от друга символы с максимальным весом – развернем строку. Тогда символ с большим весом будет в конце.
- Среднюю строку, хранящую все числа, не влияющие на вес.

Полученная строка левая + средняя + правая – это итоговый искомый ответ.

Оценки:

1. Время:

   Считывание данных происходит за O(n), строка сортируется за O(n · log(n)), посчитать частоты - O(n), разворот и сложение сток за O(n). Итоговый ответ: **O(n · log(n))**

2. Память:

   Входная строка: O(n), набор векторов константной длинны, доп. строки left, right, no_weight: O(n) в сумме. Итоговый ответ: **O(n)**

Код:

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

int main() {
  std::string str;
  const int alphabed = 26;

  std::vector<int> weights(alphabed);
  std::vector<int> freq(alphabed, 0);
  std::vector<int> counted(alphabed, 0);

  std::cin >> str;

  for (int& weight : weights) {
```

```
    std::cin >> weight;
  }

  std::sort(str.begin(), str.end(), [&weights](char left, char right) {
    return weights[left - 'a'] > weights[right - 'a'];
  });

  std::string no_weight;
  std::string left;
  std::string right;

  for (const char current : str) {
    freq[current - 'a'] += 1;
  }

  for (const char current : str) {
    const int ind = current - 'a';
    counted[ind] += 1;

    if (freq[ind] - counted[ind] == 1 && freq[ind] >= 2) {
      left.push_back(current);
      continue;
    }

    if (freq[ind] - counted[ind] == 0 && freq[ind] >= 2) {
      right.push_back(current);
      continue;
    }

    if (freq[ind] == 1 || (freq[ind]) >= 2) {
      no_weight.push_back(current);
    }
  }

  std::reverse(right.begin(), right.end());
  std::cout << left + no_weight + right;
}
```

Задача №4 «Н. Магазин»

Пояснение к примененному алгоритму:
Отсортируем цены по возрастанию и будем суммировать их от большего к меньшему. Максимальная скидка будет достигнута, если в каждый чек попадет ровно k продуктов (за k-1 заплатим, один достанется бесплатно). Будем суммировать элементы от большего к меньшему, пропуская каждый k-тый по скидке. Когда дойдем до самого маленького элемента и обработает его – получим искомую сумму.

Оценки:

1. Время:

   Считывание данных идет за линию, обработка данных тоже за линию. Сортировка по возрастанию работает за O(n · log(n)). Общая сложность **O(n · log(n))**, линейная.

2. Память:

   Используется две константы, массив числе длинны n и дополнительную память размером log(n) при сортировке. Сложность по памяти **O(n)**, линейная.

Код:

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

int main() {
  int n = 0;
  int k = 0;
  std::cin >> n >> k;

  std::vector<int> prises(n);
  for (int& prise : prises) {
    std::cin >> prise;
  }

  std::sort(prises.begin(), prises.end());

  int sum = 0;
  int counter = 0;

  for (int i = n - 1; i >= 0; i--) {
    counter += 1;
    if (counter % k != 0) {
      sum += prises[i];
    } else {
      counter = 0;
    }
  }

  std::cout << sum;
}
```