Министерство образования и науки Российской Федерации федеральное государственное
автономное образовательное учреждение высшего образования
НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО

Факультет «Программной инженерии и компьютерной техники.»

Алгоритмы и структуры данных

Лабораторная работа №2
Базовые задачи

Выполнил
Григорьев Давид Владимирович
Группа: P3215
Преподаватели
Косяков Михаил Сергеевич
Тараканов Денис Сергеевич

Санкт-Петербург 2025

Содержание

1 Дополнительное задание. 1

2 Задача E. Коровы в стойла 3

3 Задача F. Число 5

4 Задача G. Кошмар в замке 6

5 Задача H. Магазин 8

1 Дополнительное задание.

Пояснение к примененному алгоритму

Аналитическое решение задания D.

Временная сложность

- **Худший случай**: $O(d^2)$, где d – вместимость контейнера.
  - Цикл выполняется $O(d)$ раз (ограничение $d + 2$)
  - Поиск в истории: $O(d)$ на итерацию

Пространственная сложность

$O(d)$ для хранения истории состояний.

Практическая эффективность

- Для $d \leq 1000$: 10^6 операций
- Реальные сценарии редко достигают худшего случая

Код алгоритма

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

int main() {
  size_t a_val = 0;
  size_t b_val = 0;
  size_t c_val = 0;
  size_t d_val = 0;
  size_t k_val = 0;
  std::cin >> a_val;
  std::cin >> b_val;
  std::cin >> c_val;
  std::cin >> d_val;
  std::cin >> k_val;

  std::vector<size_t> history;
  size_t current = a_val;
  history.push_back(current);

  for (size_t day = 1; day <= k_val; ++day) {
    size_t new_val = current * b_val;
    if (new_val < c_val) {
      std::cout << 0 << '\n';
      return 0;
    }
    size_t temp = new_val - c_val;
    if (temp <= 0) {
      std::cout << 0 << '\n';
```

```
      return 0;
    }
    temp = std::min(temp, d_val);

    if (temp == current) {
      std::cout << temp << '\n';
      return 0;
    }

    auto it = std::find(history.begin(), history.end(), temp);
    if (it != history.end()) {
      size_t pos = it - history.begin();
      size_t cycle_length = day - pos;
      size_t remaining_days = k_val - day;
      size_t final_pos = pos + (remaining_days % cycle_length);
      std::cout << history[final_pos] << '\n';
      return 0;
    }

    if (history.size() > d_val + 2) {
      break;
    }

    history.push_back(temp);
    current = temp;
  }

  std::cout << current << '\n';
  return 0;
}
```

2 Задача E. Коровы в стойла

Пояснение к примененному алгоритму

Решение этой задачи очень похоже на нахождение нуля функции.

Можно сказать, что мы используем метод половинного деления, где функция $CanPlace$ похожа на знак функции.

Сложность по времени:

- $O(N \log D)$, где N — количество стойл, D — максимальное расстояние между стойлами.
  - Функция `CanPlace` выполняется за $O(N)$ (линейный проход по всем стойлам).
  - Бинарный поиск по расстоянию (от 1 до D) имеет $O(\log D)$ итераций.
  - Итог: $O(N) \times O(\log D) = O(N \log D)$.

Сложность по памяти:

- $O(N)$ — хранение вектора `stalls` с координатами стойл.
- Остальные переменные — константный расход памяти ($O(1)$).

Код алгоритма

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

namespace {

bool CanPlace(const std::vector<int>& stalls, size_t cows, int min_dist) {
  size_t count = 1;
  int last_cow_coord = stalls[0];
  for (size_t i = 1; i < stalls.size(); ++i) {
    if (stalls[i] - last_cow_coord >= min_dist) {
      count++;
      last_cow_coord = stalls[i];
      if (count >= cows) {
        return true;
      }
    }
  }
  return count >= cows;
}

}  // namespace

int main() {
  size_t number_of_stalls = 0;
  size_t number_of_cows = 0;

  std::cin >> number_of_stalls >> number_of_cows;

  std::vector<int> stalls(number_of_stalls);
```

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

  int left = 1;
  int right = stalls.back() - stalls.front();
  int answer = 0;

  while (left <= right) {
    int mid = left + ((right - left) / 2);

    if (CanPlace(stalls, number_of_cows, mid)) {
      answer = mid;
      left = mid + 1;
    } else {
      right = mid - 1;
    }
  }

  std::cout << answer << '\n';
  return 0;
}
```

3 Задача F. Число

Пояснение к примененному алгоритму

Самое сложное было написать компаратор. Обычно мы работаем с переменными по отдельности, но проще всего оказалось складывать строки и сразу применять “>”. Учитывая, что строки одной длины, такое сравнение строк просто является численным сравнением.

Сложность по времени:

- $O(N \cdot K \cdot \log N)$, где:
  - N — количество строк
  - K — средняя длина строки
  - Чтение данных: $O(N)$ (линейное заполнение вектора)
  - Сортировка: $O(N \log N \cdot K)$ (каждое сравнение строк требует $O(K)$ операций)
  - Вывод: $O(N \cdot K)$ (конкатенация и вывод строк)

Сложность по памяти:

- $O(N \cdot K)$ — хранение всех строк в векторе `lists_of_paper`
- $O(K)$ — временные переменные (например, `last_string`)

Код алгоритма

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

int main() {
  std::vector<std::string> lists_of_paper{};
  std::string last_string;
  while (std::cin >> last_string) {
    lists_of_paper.emplace_back(last_string);
  }

  std::sort(
      lists_of_paper.begin(),
      lists_of_paper.end(),
      [](const std::string& var1, const std::string& var2) { return var1 + var2 > var2 +
  );

  for (std::string& str : lists_of_paper) {
    std::cout << str;
  }
}
```

4 Задача G. Кошмар в замке

Пояснение к примененному алгоритму

Самое сложное в этой задаче было расшифровать условие. Данная задача рассчитана дипломированных специалистов в лингвистике и шифровании.

Самый важный инсайт, что нужно сразу перемещать первые две буквы, которые встречаются больше двух раз.

Сложность по времени:

- $O(N)$, где N — длина входной строки.
  - Чтение данных: $O(N)$ (входная строка и веса символов)
  - Подсчёт частот символов: $O(N)$
  - Сортировка весов: $O(1)$ (сортировка 26 элементов)
  - Обработка символов: $O(N)$ (суммарный обход строки для удаления символов)
  - Формирование и вывод результата: $O(N)$ (конкатенация строк)

Сложность по памяти:

- $O(N)$ — хранение входной строки, промежуточных строк (`start_of_the_result`, `end_of_the_resul`
- $O(1)$ — хранение вектора весов `weights` (26 элементов) и словаря частот `char_frequency` (26 ключей)

Код алгоритма

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

namespace {
void RemoveFirstTwoChars(std::string& input_string, char char_to_remove) {
  int removed = 0;
  size_t current_index = 0;

  while (current_index < input_string.size() && removed < 2) {
    if (input_string[current_index] != char_to_remove) {
      current_index++;
      continue;
    }

    removed++;
    input_string.erase(current_index, 1);
  }
}
}  // namespace
int main() {
  std::string input_string;
  std::cin >> input_string;
```

```
  std::vector<std::pair<char, int>> weights;
  for (int i = 0; i < 26; ++i) {
    int weight = 0;
    std::cin >> weight;
    weights.emplace_back(static_cast<char>('a' + i), weight);
  }

  std::map<char, int> char_frequency;
  for (char c : input_string) {
    char_frequency[c]++;
  }

  std::stable_sort(
      weights.begin(),
      weights.end(),
      [](const std::pair<char, int>& a, const std::pair<char, int>& b) {
        return a.second > b.second;
      }
  );

  std::string start_of_the_result;

  for (auto& weight : weights) {
    char current_char = weight.first;
    if (char_frequency[current_char] >= 2) {
      RemoveFirstTwoChars(input_string, current_char);

      char_frequency[current_char] -= 2;
      start_of_the_result += current_char;
    }
  }

  std::string end_of_the_result(start_of_the_result);

  std::reverse(end_of_the_result.begin(), end_of_the_result.end());

  std::cout << start_of_the_result + input_string + end_of_the_result << '\n';
  return 0;
}
```

5 Задача H. Магазин

Пояснение к примененному алгоритму

Группируем товары с конца массива (самые дорогие) по k товаров в группе. В каждой группе самый дешевый товар (первый элемент группы) добавляется к скидке.

Сложность по времени:

- $O(n \log n)$ — доминирующий фактор
  - Сортировка: $O(n \log n)$
  - Расчет скидки: $O(\lfloor \frac{n}{k} \rfloor) \approx O(n)$

Сложность по памяти:

- $O(n)$ — хранение вектора `item_prices`
- Остальные переменные — $O(1)$

Код алгоритма

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

int main() {
  int items_total = 0;
  int free_every_kth = 0;
  std::cin >> items_total >> free_every_kth;

  std::vector<int> item_prices(items_total);
  for (int i = 0; i < items_total; ++i) {
    std::cin >> item_prices[i];
  }
  std::sort(item_prices.begin(), item_prices.end());

  size_t total_cost = 0;
  for (int price : item_prices) {
    total_cost += price;
  }

  size_t free_discount = 0;
  int current_position = items_total - 1;  // Start from the most expensive item

  // Process groups from expensive to cheap to maximize free items
  while (current_position >= 0) {
    const int group_start = current_position - free_every_kth + 1;

    if (group_start >= 0) {
      // The cheapest item in current group becomes free
      free_discount += item_prices[group_start];
      current_position = group_start - 1;  // Move to next group
    } else {
      break;  // Not enough items left to form a full group
```

```
    }
  }

  std::cout << total_cost - free_discount << '\n';

  return 0;
}
```