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 №3
Дополнительные задания.

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

Санкт-Петербург 2025

Содержание

1 Первое дополнительное задание.

Пояснение к примененному алгоритму

Аналитическое решение задачи D основано на обнаружении циклов в последовательности состояний системы. Алгоритм моделирует ежедневные изменения количества бактерий, но вместо полного прохода по всем k дням использует следующие оптимизации:

- **Моделирование дня**:
  - Утро: количество бактерий умножается на b ($x_{n+1} = x_n \cdot b$).
  - После обеда: вычитается c ($x_{n+1} - = c$). Если $x_{n+1} < c$, эксперимент завершается (вывод 0).
  - Вечер: оставшиеся бактерии ограничиваются значением d ($x_{n+1} = \min(x_{n+1}, d)$).
- **Обнаружение циклов**:
  - Хранение истории всех предыдущих состояний ($history$).
  - При обнаружении повторяющегося значения ($temp = current$) система стабилизировалась — вывод текущего значения.
  - Если новое состояние уже встречалось ранее, вычисляется длина цикла ($cycle\_length$) и позиция в истории (pos). Оставшиеся дни ($remaining\_days$) вычисляются через остаток от деления на длину цикла ($final\_pos$).
- **Оптимизация памяти**:
  - Прерывание цикла, если размер истории превышает $d + 2$, чтобы избежать переполнения.

Временная сложность

- Цикл выполняется $O(d)$ раз (ограничение $d + 2$).
- Поиск в истории (`std::find`): $O(d)$ на итерацию.
- Общая временная сложность: $O(d^2)$ в худшем случае.

Пространственная сложность

$O(d)$ для хранения истории состояний ($history$).

Сравнение с обычным решением

Обычное решение моделирует все k дней напрямую:

```
size_t current = a_val;
for (size_t day = 1; day <= k_val; ++day) {
    current = current * b_val - c_val;
    if (current <= 0) return 0;
    current = std::min(current, d_val);
}
std::cout << current;
```

Минусы: Работает за $O(k)$, для $k \leq 10^{18}$.

Практическая эффективность

- Для $d \leq 1000$: максимальное количество операций около 10^6 (при $d = 1000$, $d^2 = 10^6$).
- Примеры работы:
  - **Пример 1**: $a = 1, b = 3, c = 1, d = 5, k = 2 \rightarrow$ после 2 дней: 5.
  - **Пример 2**: $a = 1, b = 2, c = 0, d = 4, k = 3 \rightarrow$ после 3 дней: 4.
  - **Пример 3**: $a = 1, b = 2, c = 3, d = 5, k = 2 \rightarrow$ эксперимент завершается на 1-м дне ($x = 0$).

Частный случай: $b == 1$

```
/*
Частные случаи:
    b == 1:
        Если c == 0, то количество бактерий не меняется (x = x), поэтому выводим min
        Если c > 0, то каждый день x -= c. Если x станет  0, выводим 0.

    c == 0:
        Каждый день x *= b, но не более d. Если b == 1, выводим min(a, d).
        Иначе, считаем, сколько дней нужно, чтобы достичь d. Если k больше, выводим
*/

if (b_val == 1) {
    if (c_val == 0) {
        std::cout << std::min(a_val, d_val) << '\n';
        return 0;
    } else {
        if (a_val <= c_val) {
            std::cout << 0 << '\n';
            return 0;
        }
        size_t remaining = k_val;
        if (a_val - remaining * c_val <= 0) {
            std::cout << 0 << '\n';
        } else {
            std::cout << a_val - remaining * c_val << '\n';
        }
        return 0;
    }
}
```

Код алгоритма

```
#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;
```

```cpp
  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 Доказательство ассоциативности gcd.

Для доказательства ассоциативности операции НОД (gcd) покажем, что для любых целых чисел a, b, c выполняется равенство:

$$\gcd(a, \gcd(b, c)) = \gcd(\gcd(a, b), c).$$

Доказательство через разложение на простые множители

1. **Разложение чисел:** Представим числа a, b, c в виде произведения простых множителей:

$$a = \prod_p p^{\alpha_p}, \quad b = \prod_p p^{\beta_p}, \quad c = \prod_p p^{\gamma_p}.$$

2. **НОД и операция минимума:** НОД двух чисел содержит минимальные степени общих простых множителей. Например:

$$\gcd(a, b) = \prod_p p^{\min(\alpha_p, \beta_p)}.$$

3. **Левая часть:**

$$\gcd(a, \gcd(b, c)) = \prod_p p^{\min(\alpha_p, \min(\beta_p, \gamma_p))}.$$

4. **Правая часть:**

$$\gcd(\gcd(a, b), c) = \prod_p p^{\min(\min(\alpha_p, \beta_p), \gamma_p)}.$$

5. **Свойство минимума:** Для любых неотрицательных целых чисел α, β, γ выполняется:

$$\min(\alpha, \min(\beta, \gamma)) = \min(\min(\alpha, \beta), \gamma).$$

   Это следует из того, что обе части равенства представляют собой наименьшее из трех чисел α, β, γ.

6. **Вывод:** Поскольку степени всех простых множителей совпадают, то:

$$\gcd(a, \gcd(b, c)) = \gcd(\gcd(a, b), c).$$

3 Пример неассоциативной операции и разрушение структуры дерева отрезков

Рассмотрим операцию $f(a, b) = a - b$, которая **не ассоциативна**, так как:

$$(a - b) - c \neq a - (b - c)$$

Пример: Массив и дерево отрезков для вычитания

Пусть задан массив $A = [5, 3, 2]$. Построим дерево отрезков для операции вычитания.

- **Корень** покрывает интервал $[0, 2]$:

$$((5 - 3) - 2) = 0$$

- **Левый подынтервал** $[0, 1]$:

$$5 - 3 = 2$$

- **Правый подынтервал** $[2, 2]$:

$$2$$

Проблема: Некорректный результат при запросе

Запросим значение на интервале $[0, 2]$:

- Дерево отрезков возвращает корневое значение: 0.
- Однако порядок вычисления влияет на результат:
  - Слева направо: 5 - 3 - 2 = (5 - 3) - 2 = 0,
  - Справа налево: 5 - (3 - 2) = 5 - 1 = 4.

Конкретный момент "поломки"

Алгоритм дерева отрезков:

1. Разбивает интервал $[0, 2]$ на $[0, 1]$ и $[2, 2]$.
2. Получает значения 2 (из $[0, 1]$) и 2 (из $[2, 2]$).
3. Объединяет их как $2 - 2 = 0$.

Истинный результат зависит от порядка вычисления, и дерево не может его сохранить, так как разбивает задачу на независимые подынтервалы.

4 Универсальное хэширование

Универсальное семейство хэш-функций — это набор функций H, отображающих ключи из множества U в диапазон $\{0, \dots, M - 1\}$, удовлетворяющий условию:

$$\forall x \neq y \in U, \quad \Pr_{h \leftarrow H}[h(x) = h(y)] \leq \frac{1}{M}.$$

Это гарантирует, что для любого множества ключей $S \subset U$, среднее число коллизий при случайном выборе $h \in H$ ограничено $\mathbb{E}[C_x] \leq \frac{N}{M}$, где $N = |S|$. При $M = O(N)$ время операций (вставки, поиска, удаления) становится $O(1)$ в среднем.

5 Доказательство не оптимальности стратегий max-heap и min-heap в задачи K

Пример 1 (min-heap):

Входные данные:

```
7 6
3
1
2
-1
-2
5
```

Объяснение:

1. Запрос на 3: выделен блок 1–3.
2. Запрос на 1: выделен блок 4.
3. Запрос на 2: выделен блок 5–6.
4. Освобождение блока 1 (1–3): свободные 1–3.
5. Освобождение блока 2 (4): свободные 1–4.
6. Запрос на 5: свободные блоки 1–4 (4) и 7 (1) → недостаточно. Ответ −1.

Оптимальное выделение:

1. Запрос 3 → 5–7.
2. Запрос 1 → 4.
3. Запрос 2 → 1–2.
4. После освобождений 1–3 свободны → запрос 5 → выделить 1–5.

Пример 2 (max-heap):

Входные данные:

```
7 4
4
3
-1
4
```

Объяснение:

1. Запрос на 4: выделен 1–4.
2. Запрос на 3: выделен 5–7.
3. Освобождение блока 1 → свободные 1–4.
4. Запрос на 4: выделен 1–4.

Оптимальное выделение: Если бы max-heap выделил первый запрос в конец (4–7), тогда после освобождения можно было бы использовать блоки 1–3 и 4–7.