Разбор задач для практикума

Denis Bakin

Что такое матрица

- Матрица — прямоугольная таблица чисел размером $m \times n$
- Элемент в строке i, столбце j обозначаем $A_{i,j}$
- Частые операции: сложение, умножение, скалирование, возведение в степень

$$A = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{pmatrix}$$

Как умножить матрицы

- Умножить можно A размера $m \times k$ на B размера $k \times n$
- Результат — матрица C размера $m \times n$

$$C_{i,j} = \sum_{t=1}^{k} A_{i,t} \cdot B_{t,j}$$

Асимптотика умножения матриц

- Наивный алгоритм: $\mathcal{O}(mkn)$
- Для квадратных матриц $n \times n$: $\mathcal{O}(n^3)$
- Быстрые алгоритмы лучше n^3, но сложные
- Для 2×2 умножение — константное время $O(1)$

Асимптотика умножения матриц

Пример: умножение матриц 2×2

Пусть

$$A = \begin{pmatrix} a & b \\ c & d \end{pmatrix}, \qquad B = \begin{pmatrix} e & f \\ g & h \end{pmatrix}$$

Пример: умножение матриц 2×2

Пусть

$$A = \begin{pmatrix} a & b \\ c & d \end{pmatrix}, \qquad B = \begin{pmatrix} e & f \\ g & h \end{pmatrix}$$

Тогда

$$AB = \begin{pmatrix} ae + bg & af + bh \\ ce + dg & cf + dh \end{pmatrix}$$

Умножение матрицы 2×2 на вектор

Пусть $A = \begin{pmatrix} a & b \\ c & d \end{pmatrix}$, $v = \begin{pmatrix} x \\ y \end{pmatrix}$. Найдем Av

Умножение матрицы 2×2 на вектор

Пусть $A = \begin{pmatrix} a & b \\ c & d \end{pmatrix}$, $v = \begin{pmatrix} x \\ y \end{pmatrix}$. Найдем Av

$$\begin{pmatrix} a & b \\ c & d \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} ax + by \\ cx + dy \end{pmatrix}$$

Числа Фибоначчи — формула Бине

Определение: $F_0 = 0$, $F_1 = 1$, $F_n = F_{n-1} + F_{n-2}$

Формула Бине:

$$F_n = \frac{\varphi^n - \psi^n}{\sqrt{5}}, \qquad \varphi = \frac{1+\sqrt{5}}{2}, \qquad \psi = \frac{1-\sqrt{5}}{2}$$

Матрица Фибоначчи

- как выразить нахождения следующего числа с помощью matvec операции? n-тое?
- $\begin{pmatrix} F_n \\ F_{n-1} \end{pmatrix} \to \begin{pmatrix} F_{n+1} \\ F_n \end{pmatrix}$

Матрица Фибоначчи

- как выразить нахождения следующего числа с помощью matvec операции? n-тое?
- $\begin{pmatrix} F_n \\ F_{n-1} \end{pmatrix} \to \begin{pmatrix} F_{n+1} \\ F_n \end{pmatrix}$

Матрица перехода:

$$M = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}$$

Тогда:

$$M \begin{pmatrix} F_n \\ F_{n-1} \end{pmatrix} = \begin{pmatrix} F_{n+1} \\ F_n \end{pmatrix}$$

Получение F_n через матрицу

В частности:

$$M^n \begin{pmatrix} F_1 \\ F_0 \end{pmatrix} = \begin{pmatrix} F_{n+1} \\ F_n \end{pmatrix}$$

Значит

- $M^{n-1} \cdot \begin{pmatrix} 1 \\ 0 \end{pmatrix}$ даёт F_n
- Основная задача — быстро вычислить M^n

Схема реализации задачи

Нужно вычислять $M^n \bmod MOD$ и получать F_n.

- какая идея решения?

Используем такие функции:

- `struct Matrix`
- `makeIdentity(int n)`
- `multiply(const Matrix&, const Matrix&, long long MOD)`
- `fastPow(Matrix base, long long exp, long long MOD)`
- `nthFibonacci(long long n, long long MOD)`

Схема реализации задачи

Нужно вычислять $M^n \bmod MOD$ и получать F_n.

- какая идея решения?
- какие функции, структуры нам потребуются? какие аргументы они будут принимать?

Используем такие функции:

- `struct Matrix`
- `makeIdentity(int n)`
- `multiply(const Matrix&, const Matrix&, long long MOD)`
- `fastPow(Matrix base, long long exp, long long MOD)`
- `nthFibonacci(long long n, long long MOD)`

Схема реализации задачи

Нужно вычислять $M^n \bmod MOD$ и получать F_n.

- какая идея решения?
- какие функции, структуры нам потребуются? какие аргументы они будут принимать?
- оцените асимптотику решения

Используем такие функции:

- `struct Matrix`
- `makeIdentity(int n)`
- `multiply(const Matrix&, const Matrix&, long long MOD)`
- `fastPow(Matrix base, long long exp, long long MOD)`
- `nthFibonacci(long long n, long long MOD)`

Matrix

```
#include <vector>

struct Matrix {
    int n;
    std::vector<std::vector<long long>> a;
    Matrix(int _n = 0) : n(_n), a(_n, std::vector<long long>(_n, 0)) {}
};
```

Если не хочется использовать структуру, а краткое название хочется - `using Matrix = std::vector<std::vector<long long>>;`

```
Matrix makeIdentity(int n) {
    Matrix I(n);
    for (int i = 0; i < n; ++i) I.a[i][i] = 1;
    return I;
}
```

fastPow

- какая асимптотика у `fastPow`?
- как инициализируется результат в стандартном алгоритме?

fastPow

- какая асимптотика у `fastPow`?
- как инициализируется результат в стандартном алгоритме?

```
Matrix fastPow(Matrix base, long long exp, long long MOD) {
    int n = base.n;
    Matrix result = makeIdentity(n);
    while (exp > 0) {
        if (exp % 2 != 0) result = multiply(result, base, MOD);
        base = multiply(base, base, MOD);
        exp /= 2;
    }
    return result;
}
```

nthFibonacci

```
long long nthFibonacci(long long n, long long MOD) {
    if (n == 0) return 0 % MOD;
    Matrix M(2);
    M.a[0][0] = 1; M.a[0][1] = 1;
    M.a[1][0] = 1; M.a[1][1] = 0;
    Matrix R = fastPow(M, n-1, MOD);
    return R.a[0][0] % MOD;
}
```

Алгоритм Евклида

- $\gcd(a, b)$ — наибольший общий делитель (greatest common divisor)

Алгоритм Евклида

- $\gcd(a, b)$ — наибольший общий делитель (greatest common divisor)
-
$$d = \gcd(a, b) \Rightarrow \begin{cases} a \vdots d \\ b \vdots d \end{cases} \Rightarrow \forall k \in \mathbb{Z} \;:\; (a - k \cdot b) \vdots d \Leftrightarrow (a \bmod b) \vdots d$$

Алгоритм Евклида

- $\gcd(a, b)$ — наибольший общий делитель (greatest common divisor)
- $$d = \gcd(a, b) \Rightarrow \begin{cases} a \vdots d \\ b \vdots d \end{cases} \Rightarrow \forall k \in \mathbb{Z} \ : \ (a - k \cdot b) \vdots d \Leftrightarrow (a \bmod b) \vdots d$$
- Рекурсия:
$$\gcd(a, b) = \begin{cases} a, & b = 0 \\ \gcd(b, a \bmod b), & b > 0 \end{cases}$$

Алгоритм Евклида

- $\gcd(a, b)$ — наибольший общий делитель (greatest common divisor)
-
$$d = \gcd(a, b) \Rightarrow \begin{cases} a \vdots d \\ b \vdots d \end{cases} \Rightarrow \forall k \in \mathbb{Z} \ : \ (a - k \cdot b) \vdots d \Leftrightarrow (a \bmod b) \vdots d$$
- Рекурсия:
$$\gcd(a, b) = \begin{cases} a, & b = 0 \\ \gcd(b, a \bmod b), & b > 0 \end{cases}$$
- $a \leq b \Rightarrow a \bmod b \leq \frac{a}{2}$

Алгоритм Евклида

- $\gcd(a, b)$ — наибольший общий делитель (greatest common divisor)
- $$d = \gcd(a, b) \Rightarrow \begin{cases} a \vdots d \\ b \vdots d \end{cases} \Rightarrow \forall k \in \mathbb{Z} \ : \ (a - k \cdot b) \vdots d \Leftrightarrow (a \bmod b) \vdots d$$
- Рекурсия:
$$\gcd(a, b) = \begin{cases} a, & b = 0 \\ \gcd(b, a \bmod b), & b > 0 \end{cases}$$
- $a \leq b \Rightarrow a \bmod b \leq \frac{a}{2}$
- Работает за $O(\log \min(a, b))$

Реализация алгоритма Евклида (рекурсия)

```
int gcd(int a, int b) {
    if (b == 0) {
        return a;
    }
    return gcd(b, a % b);
}
```

Реализация алгоритма Евклида (цикл)

```
int gcd(int a, int b) {
    while (b != 0) {
        int temp = b;
        b = a % b;
        a = temp;
    }
    return a;
}
```

Идея: каноническая факторизация

- Любое целое $N > 1$ можно записать в виде

$$N = p_1^{e_1} \cdot p_2^{e_2} \cdot \dots \cdot p_k^{e_k} = \prod_{i=1}^{k} p_i^{e_i},$$

  ... где $p_1 < p_2 < \dots < p_k$ — простые, $e_i \geq 1$ — целые степени.
- Каноническая форма: вектор пар (p_i, e_i) — удобен для операций над большими числами
- Интуиция: как хранение отдельных разрядов при работ с большими числами
- Цель: представлять числа как `vector<Factor>` и выполнять умножение/деление без восстановления целого

Математика операций на факторизованной форме

- Пусть

$$A = \prod_i p_i^{a_i}, \qquad B = \prod_i p_i^{b_i}$$

Математика операций на факторизованной форме

- Пусть
$$A = \prod_i p_i^{a_i}, \qquad B = \prod_i p_i^{b_i}$$
- $p_i \in ?$, $a_i, b_i \geq ?$

Математика операций на факторизованной форме

- Пусть
$$A = \prod_i p_i^{a_i}, \qquad B = \prod_i p_i^{b_i}$$
- $p_i \in ?$, $a_i, b_i \geq ?$
- (координаты p_i — по объединённому множеству простых делителей), где $a_i, b_i \geq 0$

Математика операций на факторизованной форме

- Пусть
$$A = \prod_i p_i^{a_i}, \qquad B = \prod_i p_i^{b_i}$$
- $p_i \in ?$, $a_i, b_i \geq ?$
- (координаты p_i — по объединённому множеству простых делителей), где $a_i, b_i \geq 0$
- Тогда
$$A \cdot B = \prod_i p_i^{a_i + b_i}$$
$$\frac{A}{B} = \prod_i p_i^{a_i - b_i}, \quad \text{требуется } a_i \geq b_i \ \forall i$$

Математика операций на факторизованной форме

- Пусть
$$A = \prod_i p_i^{a_i}, \qquad B = \prod_i p_i^{b_i}$$
- $p_i \in ?$, $a_i, b_i \geq ?$
- (координаты p_i — по объединённому множеству простых делителей), где $a_i, b_i \geq 0$
- Тогда
$$A \cdot B = \prod_i p_i^{a_i + b_i}$$
$$\frac{A}{B} = \prod_i p_i^{a_i - b_i}, \quad \text{требуется } a_i \geq b_i \ \forall i$$
- Значит операции — это слияние отсортированных списков и суммирование/вычитание показателей

Формат данных

```
struct Factor { long long p; int e; };
```

- `p` — простое, `e` — экспонента
- Внутреннее требование: векторы отсортированы по `p`, без нулевых степеней

Интерфейс функций (декомпозиция)

- `vector<Factor> factorize(long long n);`

Интерфейс функций (декомпозиция)

- vector<Factor> factorize(long long n);
- vector<Factor> multiplyFactors(const vector<Factor>& A, const vector<Factor>& B);

Интерфейс функций (декомпозиция)

- `vector<Factor> factorize(long long n);`
- `vector<Factor> multiplyFactors(const vector<Factor>& A, const vector<Factor>& B);`
- `vector<Factor> divideFactors(const vector<Factor>& A, const vector<Factor>& B);` — вернуть пустой вектор или выбросить ошибку, если не делится

Интерфейс функций (декомпозиция)

- `vector<Factor> factorize(long long n);`
- `vector<Factor> multiplyFactors(const vector<Factor>& A, const vector<Factor>& B);`
- `vector<Factor> divideFactors(const vector<Factor>& A, const vector<Factor>& B);` — вернуть пустой вектор или выбросить ошибку, если не делится
- `long long toNumber(const vector<Factor>& F, long long MOD = -1);` — опционально, с модулем или аккуратно с проверкой переполнения

Факторизация — идея (trial division)

- Для n используем простой перебор делителей до $\sqrt{n}$:
  - для $d = 2$ отдельно, затем нечётные $d = 3, 5, ...$
  - считать степень, делить до тех пор, пока $n \% d == 0$
  - после цикла, если $n > 1$ — остаток прост и добавляется как фактор
- Сложность: $O(\sqrt{n})$
- Как ускорить можно?

Слияние (merge) — ключевая идея

- Пусть A и B — отсортированные по $\mathfrak{p}$ списки множителей. Как объединить списки, сохраняя сортировку?:

Слияние (merge) — ключевая идея

- Пусть A и B — отсортированные по p списки множителей. Как объединить списки, сохраняя сортировку?:
- если $p_A < p_B$ — в результат добавляется (p_A, e_A) (или $(p_A, e_A - e_B)$ для деления)

Слияние (merge) — ключевая идея

- Пусть A и B — отсортированные по p списки множителей. Как объединить списки, сохраняя сортировку?:
- если $p_A < p_B$ — в результат добавляется (p_A, e_A) (или $(p_A, e_A - e_B)$ для деления)
- если $p_A == p_B$ — складываем/вычитаем экспоненты

Слияние (merge) — ключевая идея

- Пусть A и B — отсортированные по p списки множителей. Как объединить списки, сохраняя сортировку?:
- если $p_A < p_B$ — в результат добавляется (p_A, e_A) (или $(p_A, e_A - e_B)$ для деления)
- если $p_A == p_B$ — складываем/вычитаем экспоненты
- если $p_A > p_B$ — аналогично с B

Слияние (merge) — ключевая идея

- Пусть A и B — отсортированные по p списки множителей. Как объединить списки, сохраняя сортировку?:
- если $p_A < p_B$ — в результат добавляется (p_A, e_A) (или $(p_A, e_A - e_B)$ для деления)
- если $p_A == p_B$ — складываем/вычитаем экспоненты
- если $p_A > p_B$ — аналогично с B
- Итог — линейное время $O(|A| + |B|)$

Слияние (merge) — иллюстрация

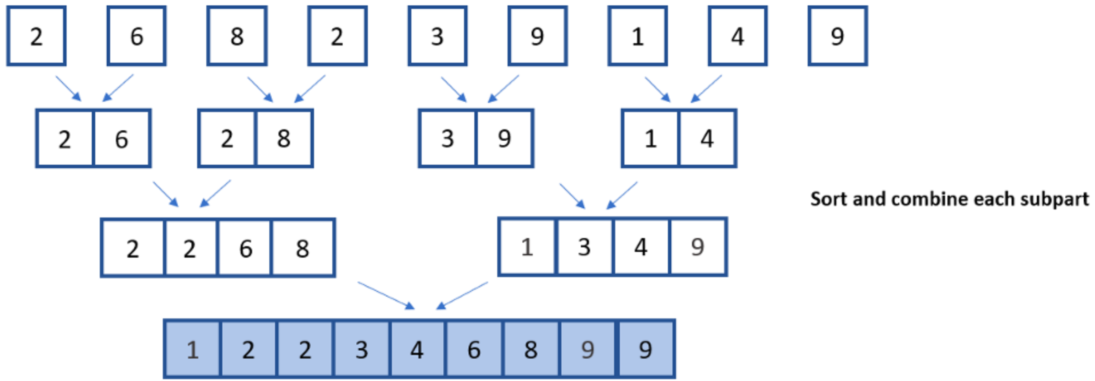

Figure 2: Слияние

Преобразование обратно: `toNumber`

- как будет реализована функция `toNumber`?

Преобразование обратно: toNumber

- как будет реализована функция toNumber?

```
long long toNumber(const vector<Factor>& F, long long MOD = -1) {
    int64_t acc = 1;
    int64_t tmp = 1;
    for (const Factor& f : F) {
        acc*=fastPow(f.p, f.e, MOD)
    }
    return acc;
}
```

Примеры и демонстрация

- Пример 1: $N = 360 = 2^3 \cdot 3^2 \cdot 5^1$

Примеры и демонстрация

- Пример 1: $N = 360 = 2^3 \cdot 3^2 \cdot 5^1$
    - `factorize(360)` $\to$ `[{2,3},{3,2},{5,1}]`

Примеры и демонстрация

- Пример 1: $N = 360 = 2^3 \cdot 3^2 \cdot 5^1$
  - `factorize(360)` $\rightarrow$ `[{2,3},{3,2},{5,1}]`
- Пример 2: Умножение:

Примеры и демонстрация

- Пример 1: $N = 360 = 2^3 \cdot 3^2 \cdot 5^1$
  - `factorize(360)` $\rightarrow$ `[{2,3},{3,2},{5,1}]`
- Пример 2: Умножение:
  - $A = 2^2 \cdot 3^1$ (`[{2,2},{3,1}]`), $B = 3^2 \cdot 5^1$ (`[{3,2},{5,1}]`)

Примеры и демонстрация

- Пример 1: $N = 360 = 2^3 \cdot 3^2 \cdot 5^1$
  - `factorize(360)` $\to$ `[{2,3},{3,2},{5,1}]`
- Пример 2: Умножение:
  - $A = 2^2 \cdot 3^1$ (`[{2,2},{3,1}]`), $B = 3^2 \cdot 5^1$ (`[{3,2},{5,1}]`)
  - `multiplyFactors(A,B)` $\to$ `[{2,2},{3,3},{5,1}]` $\to$ число $= 2^2 \cdot 3^3 \cdot 5^1 = 4 \cdot 27 \cdot 5 = 540$

Примеры и демонстрация

- Пример 1: $N = 360 = 2^3 \cdot 3^2 \cdot 5^1$
  - `factorize(360)` $\rightarrow$ `[{2,3},{3,2},{5,1}]`
- Пример 2: Умножение:
  - $A = 2^2 \cdot 3^1$ (`[{2,2},{3,1}]`), $B = 3^2 \cdot 5^1$ (`[{3,2},{5,1}]`)
  - `multiplyFactors(A,B)` $\rightarrow$ `[{2,2},{3,3},{5,1}]` $\rightarrow$ число $= 2^2 \cdot 3^3 \cdot 5^1 = 4 \cdot 27 \cdot 5 = 540$
- Пример 3: Деление:

Примеры и демонстрация

- Пример 1: $N = 360 = 2^3 \cdot 3^2 \cdot 5^1$
  - `factorize(360)` $\to$ `[{2,3},{3,2},{5,1}]`
- Пример 2: Умножение:
  - $A = 2^2 \cdot 3^1$ (`[{2,2},{3,1}]`), $B = 3^2 \cdot 5^1$ (`[{3,2},{5,1}]`)
  - `multiplyFactors(A,B)` $\to$ `[{2,2},{3,3},{5,1}]` $\to$ число $= 2^2 \cdot 3^3 \cdot 5^1 = 4 \cdot 27 \cdot 5 = 540$
- Пример 3: Деление:
  - `divideFactors( [{2,3},{3,2},{5,1}], [{2,1},{3,1}] )` $\to$ `[{2,2},{3,1},{5,1}]`

Вопросы?