Рекурсия

Denis Bakin

Рекурсия вокруг нас

- Рекурсия — ситуация, когда объект является частью себя

Фракталы как пример рекурсии

- Фракталы — самоподобные объекты
- Классический пример — **треугольник Серпинского**

Figure 2: Итерации треугольника Серпинского

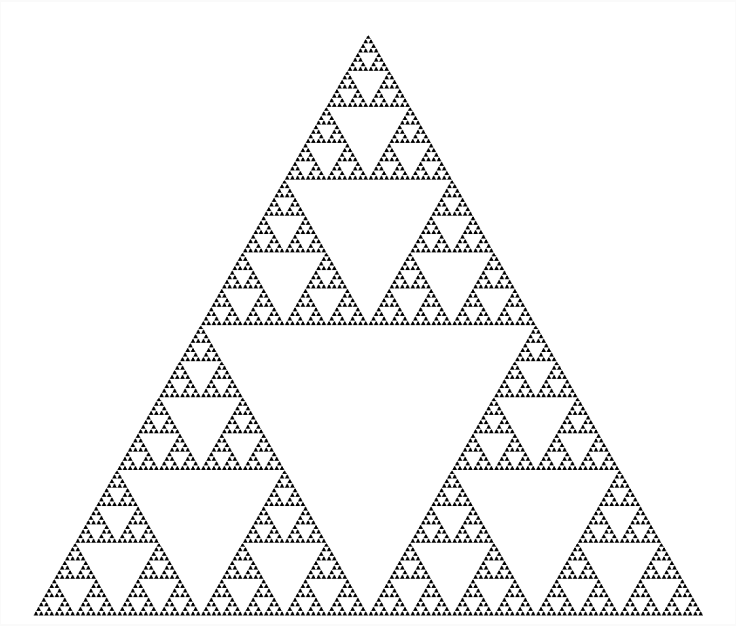

Figure 3: Треугольник Серпинского

Рекуррентные формулы

- Последовательности можно задавать через рекуррентное соотношение
- Пример — числа Фибоначчи:

$$\begin{cases} F_1 = 1 \\ F_2 = 1 \\ F_n = F_{n-1} + F_{n-2}, n > 2 \end{cases}$$

Рекурсия в программировании

- Рекурсия в коде — вызов функции из неё самой
- Применение:
  - обход графов (DFS)
  - генерация перестановок, слов
  - сортировки
  - вычислительная геометрия (например, выпуклая оболочка)

Иллюстрация DFS

Суть рекурсии

- Сведение большой задачи к нескольким подзадачам того же типа
- Решение подзадач $\rightarrow$ решение исходной задачи

Пример: сумма массива рекурсивно

Пример: сумма массива рекурсивно

- Сумма массива = первый элемент + сумма оставшихся

$$\text{sum}(i, j) = \begin{cases} a_i + \text{sum}(i+1, j), & i \neq j \\ a_i, & i = j \end{cases}$$

Альтернативное условие выхода

$$\mathrm{sum}(i, j) = \begin{cases} 0, & i > j \\ a_i + \mathrm{sum}(i+1, j), & \text{иначе} \end{cases}$$

Реализация в C++

```cpp
// args: {1, 2, 3, 4}, 0, 3
int recursive_sum(const std::vetor<int>& nums, int start, int stop) {
    if (start == stop) {
        return nums[start];
    }
    return nums[start] + recursive_sum(nums, start + 1, stop);
}
```

```
finding sum of 0 to 3
finding sum of 0 to 3 by summing 0 and sum of 1 to 3
finding sum of 1 to 3
finding sum of 1 to 3 by summing 1 and sum of 2 to 3
finding sum of 2 to 3
finding sum of 2 to 3 by summing 2 and sum of 3 to 3
finding sum of 3 to 3
found sum of 3 to 3: 4
10
```

Рекомендации по работе с рекурсией

1. Разбить задачу на меньшие подзадачи
2. Определить условие выхода
3. Проверить, что рекурсия всегда доходит до выхода
4. Думать о рекурсивной функции как о «чёрном ящике», который уже реализован

Пример: максимальная цифра числа

Пример: максимальная цифра числа

- Сведение: $\mathsf{max\_digit}(N) = \max(N\%10, \mathsf{max\_digit}(N/10))$
- Условие выхода: число из одной цифры
- $N < 10 \Rightarrow \mathsf{max\_digit}(N) = N$

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

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

Получение цифр числа рекурсивно

Вспомним: последнюю цифру числа получаем через `n % 10`, а «остаток» — через `n / 10`.

Получение цифр числа рекурсивно

Вспомним: последнюю цифру числа получаем через `n % 10`, а «остаток» — через `n / 10`.

Порядок вывода зависит от того, **когда** мы печатаем:

- **сначала печатаем, потом рекурсия** → цифры справа налево
- **сначала рекурсия, потом печатаем** → цифры слева направо

Цифры справа налево

```
void digitsReversed(int n) {
    if (n == 0) return;
    std::cout << n % 10 << ' ';   // сначала печатаем
    digitsReversed(n / 10);        // потом рекурсия
}
// digitsReversed(1234) → 4 3 2 1
```

Цифры справа налево

```
void digitsReversed(int n) {
    if (n == 0) return;
    std::cout << n % 10 << ' ';   // сначала печатаем
    digitsReversed(n / 10);        // потом рекурсия
}
// digitsReversed(1234) → 4 3 2 1
```

Вызовы:

- `digitsReversed(1234)`: печатает 4, вызывает `digitsReversed(123)`
- `digitsReversed(123)`: печатает 3, вызывает `digitsReversed(12)`
- `digitsReversed(12)`: печатает 2, вызывает `digitsReversed(1)`
- `digitsReversed(1)`: печатает 1, вызывает `digitsReversed(0)` → выход

Цифры слева направо

```
void digits(int n) {
    if (n == 0) return;
    digits(n / 10);                 // сначала рекурсия
    std::cout << n % 10 << ' ';    // потом печатаем
}
// digits(1234) → 1 2 3 4
```

Цифры слева направо

```
void digits(int n) {
    if (n == 0) return;
    digits(n / 10);                 // сначала рекурсия
    std::cout << n % 10 << ' ';    // потом печатаем
}
// digits(1234) → 1 2 3 4
```

Вызовы:

- `digits(1234)`: вызывает `digits(123)`, *затем* печатает 4
- `digits(123)`: вызывает `digits(12)`, *затем* печатает 3
- `digits(12)`: вызывает `digits(1)`, *затем* печатает 2
- `digits(1)`: вызывает `digits(0)` → выход, *затем* печатает 1

Собрать цифры в вектор

```
void collectDigits(int n, std::vector<int>& result) {
    if (n == 0) return;
    collectDigits(n / 10, result);  // сначала старшие
    result.push_back(n % 10);       // затем текущую
}
// 1234 → {1, 2, 3, 4}
```

- Условие выхода: `n == 0`
- Сведение: отделяем последнюю цифру, рекурсия обрабатывает остальные
- Порядок push_back после рекурсии → прямой порядок