



Рекурсия

Составитель: Рощупкин Александр

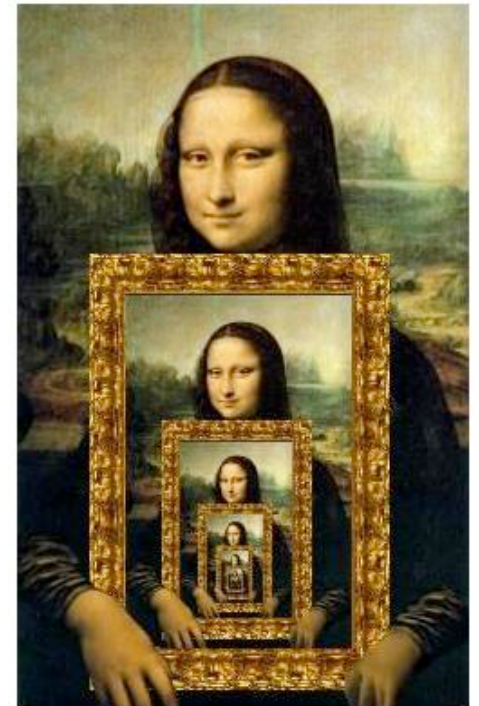


Рекурсия

Рекурсивный подход

“что бы понять рекурсию, нужно понять рекурсию...”

- * **Рекурсия** – описание объекта или процесса внутри него самого
- * Альтернатива итерации
- * Необходимо задачу представить в рекурсивном виде



Подход “Разделяй и властвуй”

- * Подход «разделяй и властвуй» заключается том, что выполнение одной сложной задачи можно разбить на несколько задач того же типа, но меньшего размера и комбинирования их решения для получения ответа к исходной задаче
- * Разбиения выполняются до тех пор, пока все подзадачи не окажутся элементарными

Пример рекурсии

- * Представим задачу возведения в степень так: «что бы возвести число в степень, нужно возвести в степень меньшее число» $x^n = x * x^{n-1}$
- * Возведём двойку в четвёртую степень: **pow(2, 4)**
- * $\text{pow}(2, 4) = 2 * \text{pow}(2, 3)$
- * $\text{pow}(2, 3) = 2 * \text{pow}(2, 2)$
- * $\text{pow}(2, 2) = 2 * \text{pow}(2, 1)$
- * $\text{pow}(2, 1) = 2$

Рекурсивный метод

- * Рекурсивным считается такой метод, который вызывает сам себя,
- * Простейший пример – это бесконечная рекурсия

```
public void infinit() {  
    System.out.println("Infinit");  
    infinit();  
}
```

- * Вызов рекурсивного метода происходит бесконечно, что приводит к переполнению стека, т.к. информация о каждом вызове помещается в системный стек

```
Exception in thread "main" java.lang.StackOverflowError
```

Механика работы рекурсии

- * **Стековый фрейм**
 - * При каждом рекурсивном вызове, в системный стек помещается информация о вызове (значения локальных переменных и адрес возврата)
- * **Рекурсивный спуск**
 - * Помещение стековых фреймов в стек. Выполнение операций перед вызовом – операции на спуске
- * **Рекурсивный подъём**
 - * Извлечение стековых фреймов из стека. Выполнение операций после вызова – операции на подъёме

Конечная рекурсия

- * В рекурсивном вызове должно быть условие выхода при условии окончания рекурсивного алгоритма

```
public void print(int count) {  
    if (count > 0) {  
        System.out.println(count);  
        print(count - 1);  
    }  
}
```

- * Напишите рекурсивный метод который будет выводить на экран возрастающую последовательность чисел
- * Написать метод, вычисляющий сумму элементов возрастающей последовательности

Пример факториала

* Итеративная формула $n! = 1 \cdot 2 \cdot \dots \cdot n = \prod_{k=1}^n k$

* Рекурсивная формула $n! = \begin{cases} 1 & n = 0, \\ n \cdot (n-1)! & n > 0. \end{cases}$

```
public int factorial(int n) {  
    if (n == 0) {  
        return 1;  
    }  
    return n * factorial(n - 1);  
}
```

Двойная рекурсия

- * Двойная рекурсия – это такой рекурсивный метод, который содержит два рекурсивных вызова

```
public void doubl () {  
    doubl ();  
    doubl ();  
}
```

Пример чисел Фибоначчи

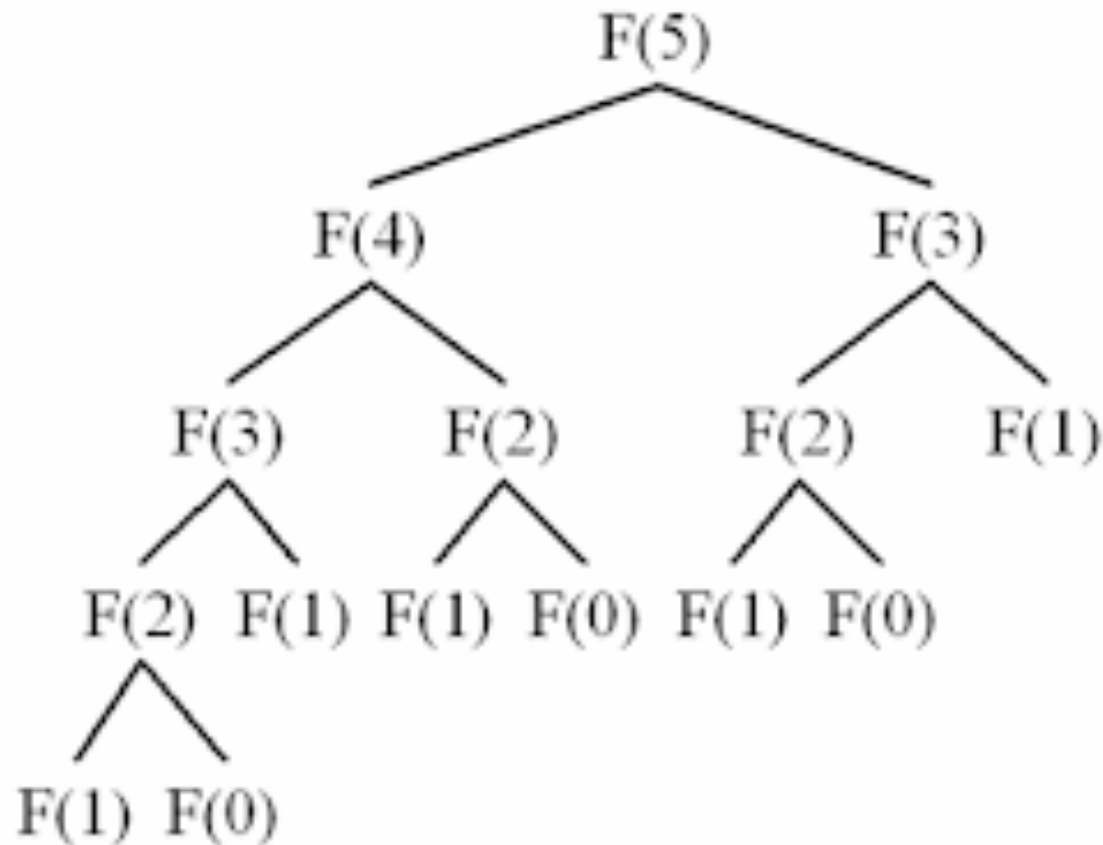
- * Числа Фибоначчи – это элементы числовой последовательности, вычисленные по формуле


$$F_0 = 0, \quad F_1 = 1, \quad F_n = F_{n-1} + F_{n-2}, \quad n \geq 2, \quad n \in \mathbb{Z}.$$

- * 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, 987, 1597, 2584, 4181, 6765, 10946, 17711

```
public int fib(int n) {  
    if (n <= 2) {  
        return 1;  
    }  
    return fib(n - 2) + fib(n - 1);  
}
```

Графическое представление

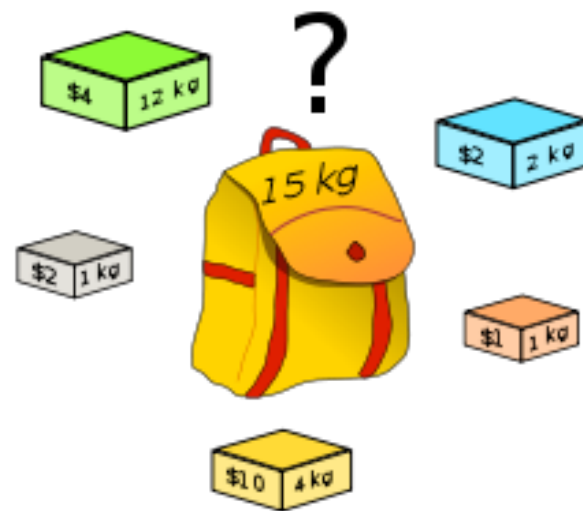




Дополнительный материал

Задача о рюкзаке

- * Уложить как можно большее число ценных вещей в рюкзак при условии, что вместимость рюкзака ограничена
- * Из заданного множества предметов со свойствами «стоимость» и «вес» требуется отобрать подмножество с максимальной полной стоимостью, соблюдая при этом ограничение на суммарный вес



Рюкзак 0-1

- * Существует несколько видов задач о рюкзаке:
 - * Рюкзак 0-1
 - * Ограниченный рюкзак
 - * Неограниченный рюкзак
 - * Рюкзак с мультивыбором
 - * Множественный рюкзак
 - * Многомерный рюкзак
- * Рассмотрим простейшую из них: Рюкзак 0-1 (0-1 Knapsack Problem)
- * Дано N предметов, n_i предмет имеет массу $w_i > 0$ и стоимость $p_i > 0$. Необходимо выбрать из этих предметов такой набор, чтобы суммарная масса не превосходила заданной величины W (вместимость рюкзака), а суммарная стоимость была максимальна.

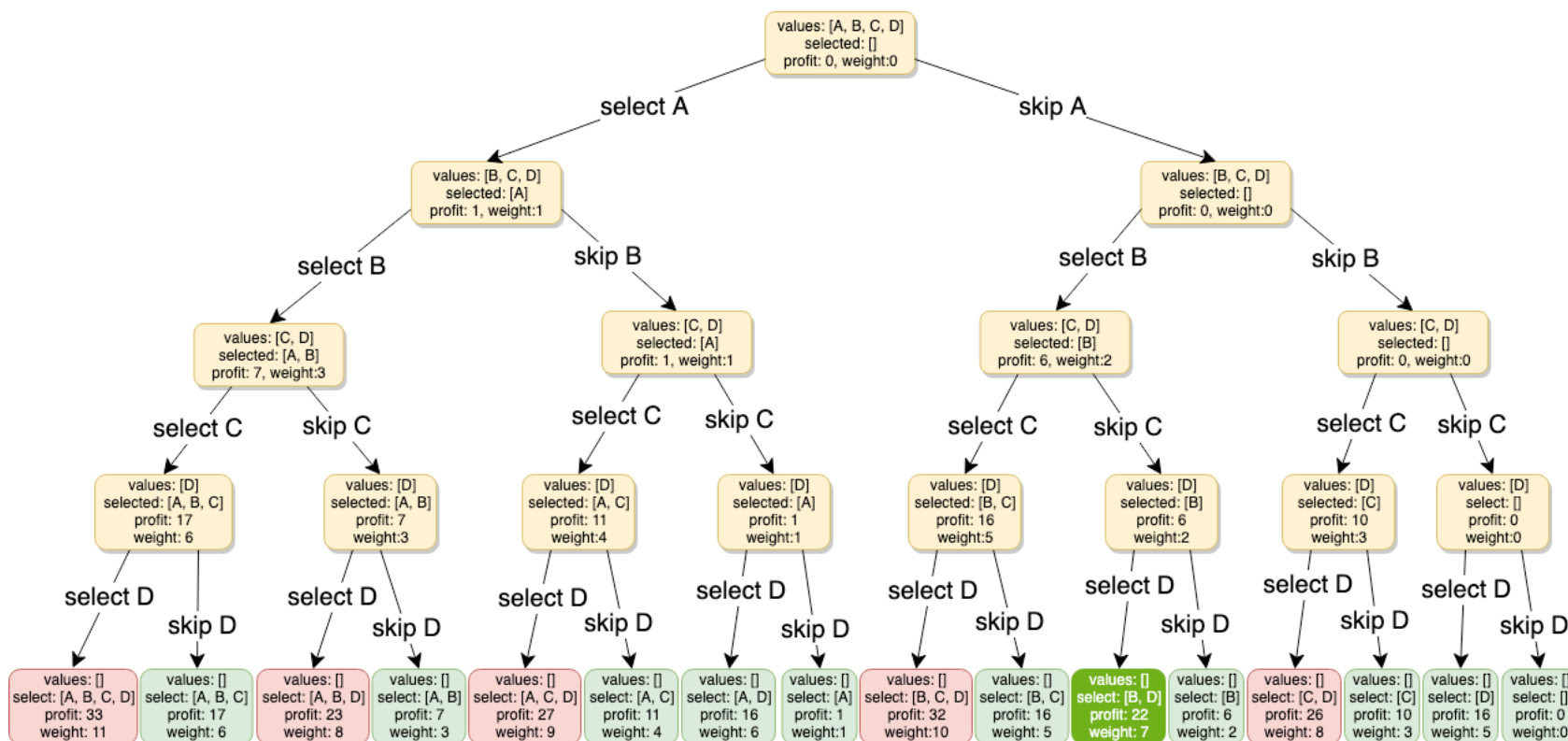
Полный перебор

- * Допустим, имеется N предметов, которые можно укладывать в рюкзак. Нужно определить максимальную стоимость груза, вес которого не превышает W
- * Берём 4 объекта у которых есть вес и стоимость
- * Для каждого предмета существует 2 варианта: предмет либо кладётся в рюкзак, либо нет

Графическое изображение

items	A	B	C	D
profit	1	6	10	16
weight	1	2	3	5

Capacity: 7



```

public class Knapsack {
    /**
     * @param profits стоимость предметов
     * @param weights веса предметов
     * @param n количество предметов
     * @param capacity объём рюкзака
     * @return максимальная стоимость предметов
     */
    public static int knapSack(int[] profits, int[] weights, int n, int capacity) {
        // основной случай: отрицательный объём
        if (capacity < 0) {
            return Integer.MIN_VALUE;
        }

        // основной случай: больше нет предметов или нулевой объём
        if (n < 0 || capacity == 0) {
            return 0;
        }

        // Case 1. Включить текущий элемент n в рюкзак(profits[n]) и рекурсивный вызов для
        // оставшихся элементов (n - 1) с уменьшенным объёмом(capacity - weights[n])
        int include = profits[n] + knapSack(profits, weights, n - 1, capacity - weights[n]);

        // Case 2. Иключить текущий элемент n из рюкзака и рекурсивный вызов для оставшихся элементов (n - 1)
        int exclude = knapSack(profits, weights, n - 1, capacity);

        // выбираем наибольшее значения из двух случаев - включая текущий предмет или не включая
        return Math.max(include, exclude);
    }

    // 0-1 Knapsack problem
    public static void main(String[] args) {
        // задаём наши предметы с помощью двух массивов, в первом цена, во втором вес
        int[] profit = {1, 6, 10, 16};
        int[] weights = {1, 2, 3, 5};
        // Объём рюкзака
        int capacity = 7;
        System.out.println("Knapsack value is " + knapSack(profit, weights, profit.length - 1, capacity));
    }
}

```

Метод ветвей и границ

- * Идея метода состоит в том, что бы отсортировать предметы по их удельной стоимости (отношению ценности к весу) и строить дерево полного перебора.
- * Его улучшение заключается в том, что в процессе построения дерева, для каждого узла мы оцениваем верхнюю границу ценности решения, и продолжаем строить дерево только для узла с максимальной оценкой.
- * Когда максимальная верхняя граница оказывается в листе дерева, алгоритм заканчивает свою работу.

Динамическое программирование

- * *Динамическое программирование* — метод решения задачи путём её разбиения на несколько одинаковых подзадач, рекуррентно связанных между собой
- * Основная суть – нахождение одинаковых задач, решение одной из них только один раз и затем использование этого решения для всех

Рекурсия N-го порядка

- * Рекурсией N-го порядка называют рекурсивный метод в котором находятся N рекурсивных ВЫЗОВОВ

```
public void recursionN(int n) {  
    for (int i = 0; i < n; i++) {  
        recursionN(n);  
    }  
}
```

Домашнее задание

- * Написать рекурсивный метод, который выводит последовательность чисел от N до K
- * Написать рекурсивный метод, который возвращает наибольшее значение в массиве
- * Написать рекурсивный метод, который определяет, является ли переданная в метод строка палиндромом
- * Написать рекурсивный метод, который возвращает сумму цифр, переданного в метод числа