

Построение и анализ алгоритмов – Лекция 3.  
Парадигма “Разделяй и властвуй”. Быстрое  
возведение в степень. Числа Фибоначчи.  
Алгоритм Евклида

Олег Смирнов  
oleg.smirnov@gmail.com

13 октября 2011 г.

## Содержание

1	Стратегия “Разделяй и властвуй”	2
2	Сортировка слиянием	2
3	Основная теорема	4
4	Бинарный поиск	4
5	Возведение в степень	4
6	Числа Фибоначчи	5
7	Умножение матриц	5
8	Алгоритм Евклида	7

## Цель лекции

- Метод “Разделяй и властвуй” на примере алгоритмов бинарного поиска, быстрого возведения в степень и вычисления чисел Фибоначчи
- Алгоритм Евклида для вычисления наибольшего общего делителя

## 1 Стратегия “Разделяй и властвуй”

Идея метода “Разделяй и властвуй” (англ. “divide and conquer”) состоит из нескольких шагов

1. Разделение задачи на подзадачи, как правило меньшего размера
2. Решение каждой из подзадач (напрямую, если они достаточно небольшого объёма – иначе рекурсивно, разбивая на меньшие части)
3. Объединение полученных решений подзадач

В силу специфики этого метода оценка времени выполнения всегда будет представляться в виде рекуррентности, которую удобно решать с помощью основной теоремы, то есть рекуррентности вида:

$$T(n) = aT(n/b) + f(n)$$

где  $a$  – количество подзадач,  $n/b$  – размер подзадачи, а  $f(n)$  – работа, которая тратится на разделение и объединение.

## 2 Сортировка слиянием

Рассмотрим пример: сортировку слиянием.

1. Разделение тривиально, просто выбираем элемент в центре массива и считаем что мы разделили массив
2. Решение: рекурсивно сортируем два получившихся подмассива
3. Объединение: слияние за линейное время  $\Theta(n)$

Псевдокод процедуры объединения:

```

MERGE( $A, p, q, r$ )
1   $n_1 \leftarrow q - p + 1$  //длина левой части
2   $n_2 \leftarrow r - q$  //длина правой части
3  //создаём массивы  $L$  и  $R$ 
4  for  $i \leftarrow 1$  to  $n_1$ 
5      do  $L[i] \leftarrow A[p + i - 1]$ 
6  for  $j \leftarrow 1$  to  $n_2$ 
7      do  $R[j] \leftarrow A[q + 1 + j - 1]$ 
8   $L[n_1 + 1] \leftarrow \infty$ 
9   $R[n_2 + 1] \leftarrow \infty$ 
10  $i \leftarrow 1$ 
11  $j \leftarrow 1$ 
12 for  $k \leftarrow p$  to  $r$ 
13     do if  $L[i] \leq R[j]$ 
14         then  $A[k] \leftarrow L[i]$ 
15              $i \leftarrow i + 1$ 
16         else  $A[k] \leftarrow R[j]$ 
17              $j \leftarrow j + 1$ 

```

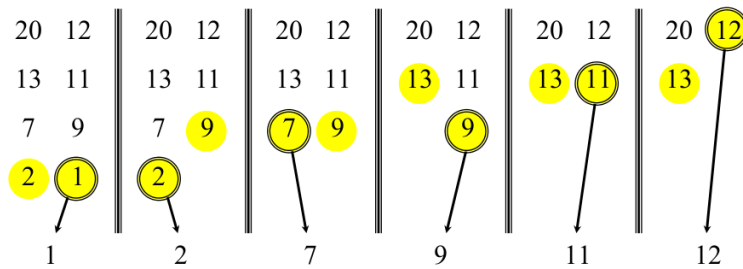


Рис. 1: Процедура объединения

Псевдокод сортировки:

```

MERGE_SORT( $A, p, r$ )
1  if  $p < r$ 
2      then  $q \leftarrow \lfloor (p + r) / 2 \rfloor$ 
3           $Merge\_Sort(A, p, q)$ 
4           $Merge\_Sort(A, q + 1, r)$ 
5           $Merge(A, p, q, r)$ 

```

Анализируя, получаем рекуррентность  $T(n) = 2T(n/2) + \Theta(n)$ . Её решением, согласно второму случаю основной теоремы, будет  $\Theta(n \log(n))$

### 3 Основная теорема

При использовании основного метода функция  $f(n)$  сравнивается с  $n^{\log_b a}$  и рассматриваются три случая. Интуитивно понятно, что асимптотическое поведение решения рекуррентного соотношения определяется большей из двух функций.

1. Если  $f(n) = O(n^{\log_b a - \epsilon})$ , для некоторой константы  $\epsilon > 0$ , т.е.  $f(n)$  растёт полиномиально медленней, чем  $n^{\log_b a}$  в  $n^\epsilon$  раз.  
Тогда  $T(n) = \Theta(n^{\log_b a})$
2. Если  $f(n) = \Theta(n^{\log_b a} \lg^k n)$ , т.е.  $f(n)$  и  $n^{\log_b a}$  растут с одинаковой скоростью с точностью до множителя  $\lg^k n$ , для константы  $k \geq 0$ .  
Тогда  $T(n) = \Theta(n^{\log_b a} \lg^{k+1} n)$
3. Если  $f(n) = \Omega(n^{\log_b a + \epsilon})$ , для некоторой константы  $\epsilon > 0$ , т.е.  $f(n)$  растёт полиномиально быстрее, чем  $n^{\log_b a}$  в  $n^\epsilon$  раз  
и  $f(n)$  удовлетворяет неравенству  $af(n/b) \leq cf(n)$  для некоторого  $c < 1$   
Тогда  $T(n) = \Theta(f(n))$

В сортировке слиянием  $a = 2, b = 2, f(n) = \Theta(n)$ , то есть нам нужно сравнить  $\Theta(n)$  и  $n^{\log_2 2} = n^1 = n$ . Они асимптотически равны (случай 2 теоремы,  $k = 0$ ), таким образом, ответом будет  $T(n) = \Theta(n \lg n)$

### 4 Бинарный поиск

Другим примером метода “разделяй и властвуй” является бинарный поиск.

Оценка времени выполнения:  $T(n) = T(n/2) + \Theta(1)$  (одно подзадача в два раза меньше основной задачи и константное время для объединения), снова второй случай основной теоремы и решение  $T(n) = \Theta(\lg n)$

### 5 Возведение в степень

Задача: найти целую степень числа, то есть найти  $a^n$ , где  $n \in \mathbb{N}$ . Наивный алгоритм решает задачу за  $\Theta(n)$ . Алгоритм “разделяй и властвуй” даёт лучшее решение:

$$a(n) = \begin{cases} a^{n/2} a^{n/2}, & \text{если } n - \text{чётное} \\ a^{(n-1)/2} a^{(n-1)/2} a, & \text{если } n - \text{нечётное} \end{cases}$$

Время работы:  $T(n) = T(n/2) + \Theta(1) \Rightarrow T(n) = \Theta(\lg n)$

## 6 Числа Фибоначчи

Рекурсивное определение:

$$F_n = \begin{cases} 0, & \text{если } n = 0 \\ 1, & \text{если } n = 1 \\ F_{n-1} + F_{n-2}, & \text{если } n > 1 \end{cases}$$

Наивный рекурсивный алгоритм:  $\Omega(\phi^n)$ , где  $\phi$  – золотое сечение. Экспоненциальное время работы – очень плохо.

Итеративный алгоритм: последовательно считать  $F_0, F_1, F_2, \dots$  для получения нового числа складывать два предыдущих – на каждое число одно сложение, значит время выполнения  $T(n) = \Theta(n)$

Наивное возведение в степень: мы можем использовать известную формулу  $F_n = \phi^n / \sqrt{5}$ , округлённое в сторону ближайшего целого. Используя оценку из предыдущего раздела, получим время выполнения  $\Theta(\lg n)$ . Однако этот метод не слишком надёжный, т.к. из-за ошибок округления вещественных чисел можем получить неверный результат.

Теорема:

$$\begin{bmatrix} F_{n+1} & F_n \\ F_n & F_{n-1} \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}^n$$

Алгоритм: рекурсивное возведение в квадрат (то же, что применялось для возведения в степень). Его оценка, как известно  $\Theta(\lg n)$ . Возведение матрицы 2 на 2 в квадрат асимптотически ничем не отличается от возведения числа в квадрат, просто вместо одного умножения придётся делать восемь и ещё четыре сложения, но все равно константное время  $\Theta(1)$

Доказательство теоремы (индукция по  $n$ ): Базис ( $n = 1$ )

$$\begin{bmatrix} F_2 & F_1 \\ F_1 & F_0 \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}^1$$

Индукция (для  $n > 1$ ):

$$\begin{bmatrix} F_{n+1} & F_n \\ F_n & F_{n-1} \end{bmatrix} = \begin{bmatrix} F_n & F_{n-1} \\ F_{n-1} & F_{n-2} \end{bmatrix} \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}^{n-1} \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}^n$$

## 7 Умножение матриц

Вход:  $A = [a_{ij}], B = [b_{ij}]$

Выход:  $C = [c_{ij}] = A \cdot B, i \in 1 \dots n$

$$\begin{pmatrix} c_{11} & c_{12} & \dots & c_{1n} \\ c_{21} & c_{22} & \dots & c_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ c_{n1} & c_{n2} & \dots & c_{nn} \end{pmatrix} = \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{pmatrix} \cdot \begin{pmatrix} b_{11} & b_{12} & \dots & b_{1n} \\ b_{21} & b_{22} & \dots & b_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ b_{n1} & b_{n2} & \dots & b_{nn} \end{pmatrix}$$

Формула для нахождения элементов матрицы произведения:

$$c_{ij} = \sum_{k=1}^n a_{ik} b_{kj}$$

Наивный алгоритм на псевдокоде:

```
1 for i ← 1 to n
2   do for j ← 1 to n
3     do cij ← 0
4       for k ← 1 to n
5         do cij ← cij + aikbkj
```

Три вложенных цикла, т.е. сложность алгоритма будет  $\Theta(n^3)$ . Рассмотрим алгоритм в стиле “разделяй и властвуй”. Идея: представим, что матрица  $n \times n$  – это  $2 \times 2$ -матрица  $(n/2) \times (n/2)$  подматриц:

$$\begin{bmatrix} r & s \\ t & u \end{bmatrix} = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \cdot \begin{bmatrix} e & f \\ g & h \end{bmatrix}$$
$$C = A \cdot B$$

$$8 \text{ рекурсивных умножений} + 4 \text{ сложения матриц } 2 \times 2 \begin{cases} r = ae + bg \\ s = af + bh \\ t = ce + dg \\ u = cf + dh \end{cases}$$

Получим рекуррентность:  $T(n) = 8T(n/2) + \Theta(n)$

Нужно сравнить  $n^{\log_2 8}$  и  $\Theta(n)$ . Первое полиномиально больше, получаем первый случай основной теоремы и, стало быть, решение:  $T(n) = \Theta(n^3)$

Решение оказалось точно таким же, как и оригинальное.

## 8 Алгоритм Евклида

Наибольший общий делитель (НОД) двух чисел  $m$  и  $n$  обычно обозначается как

$$\gcd(m, n)$$

Если  $\gcd(m, n) = 1$ , то числа  $m$  и  $n$  называются *взаимно простыми*. Кроме того,  $\gcd(n, 0) = n$  для любого  $n > 0$ .

Первый из известных алгоритмов нахождения НОД описан Евклидом в книге “Начала” (около 300 г. до н.э.). Он записывается рекурсивной формулой:

```
EUCLID( $m, n$ )
1  if  $n = 0$ 
2     then return  $m$ 
3     else return Euclid( $n, m \bmod n$ )
```

Идея заключается в том, что НОД двух чисел не изменится, если из большего числа вычесть меньшее. Т.е. если 21 – НОД 252 и 105, то  $205 - 105 = 147$  и 21 по-прежнему является НОД 147 и 105. Остаток от последовательного вычитания второго аргумента из первого и выражается формулой  $m \bmod n$ .

Работа этого рекурсивного алгоритма не может продолжаться до бесконечности, т.к. второй аргумент функции всегда строго убывает, и он всегда неотрицательный. Таким образом, результатом всегда будет правильный ответ.

Существует другой метод поиска НОД, где операции умножения и деления заменены операциями бинарного сдвига. Этот метод обычно работает быстрее, т.к. сдвиг намного дешевле других операций. Алгоритм бинарного НОД был известен ещё в Китае в первом веке н.э., но опубликован был лишь в 1967 году израильским программистом Джозефом Стайном. Метод использует следующие свойства НОД:

1.  $\gcd(0, n) = n$ ;  $\gcd(m, 0) = m$ ;  $\gcd(m, m) = m$  – по определению gcd
2. Если  $m, n$  чётные, то  $\gcd(m, n) = 2 * \gcd(m/2, n/2)$ , т.к. 2 – общий делитель
3. Если  $m$  чётное,  $n$  нечётное, то  $\gcd(m, n) = \gcd(m/2, n)$ , т.к. 2 – не общий делитель
4. Если  $n$  чётное,  $m$  нечётное, то  $\gcd(m, n) = \gcd(m, n/2)$

5. Если  $m, n$  нечётные и  $n > m$ , то  $\gcd(m, n) = \gcd((n - m)/2, m)$  – комбинация из одного шага простого алгоритма Евклида (вычитания) и шагов 3-4 бинарного алгоритма
6. Если  $m, n$  нечётные и  $n < m$ , то  $\gcd(m, n) = \gcd((m - n)/2, n)$

Шаги повторяются, пока не будет  $m = n$ , а затем ещё раз, пока  $n = 0$ . Определение алгоритма рекурсивно, но это хвостовая рекурсия, а значит её можно заменить итерацией.

Время работы алгоритма  $O(\log^2 mn)$ , т.е. пропорционально квадрату количества бит  $m$  и  $n$  вместе.

Существуют расширенные версии обоих вариантов алгоритмов, которые параллельно с НОД вычисляют полезные константы  $a$  и  $b$ , где

$$am + bn = \gcd(a, b)$$

Наибольший общий делитель двух чисел имеет ряд важных применений в криптографии с открытым ключём, в решении Диофантовых уравнений и в ряде других задач теории чисел.

## Заключение

- “Разделяй и властвуй” – одна из нескольких мощных техник для разработки алгоритмов
- Оценка времени выполнения этих алгоритмов сводится к рекуррентностям, которые легко решаются основной теоремой
- Стратегия “Разделяй и властвуй” часто позволяет получить весьма эффективные алгоритмы