

Лекции по предмету Алгоритмы и структуры данных

Группа лектория ФКН ПМИ 2015-2016

Никита Попов

Тамерлан Таболов

Лёша Хачиянц

2016 год

Содержание

1	Задача о Ханойских башнях 12.01.2016	1
2	Алгоритмы сортировки 14.01.2016	4

1 Задача о Ханойских башнях 12.01.2016

Оргмоменты

Одна контрольная — контекст на реализацию какого-то алгоритма. Пользоваться своим кодом запрещено.

Задача — привести алгоритм, провести теоретический анализ (доказать его корректность, оценить время работы) и запрограммировать. Сначала сдаётся теория, потом практика.

Экзамен устный.

$$O_{\text{итоговая}} = 0.7 \cdot O_{\text{накопленная}} + 0.3 \cdot O_{\text{экзамен}}$$

$$O_{\text{накопленная}} = 0.2 \cdot O_{\text{КР}} + 0.12 \sum_{i=1}^5 O_{\text{ДЗ } i} + 0.2 \cdot O_{\text{семинары}}$$

Списывание, как обычно, *не поощряется*. ДЗ предполагается не обсуждать.

[Здесь](#) (ссылка слева) можно найти ссылки на ДЗ и краткое содержание лекций.

Автоматов *пока* не предусмотрено.

Литература:

- Кормен Т., Лейзерсон Ч., Ривест Р., Штайн К. — «Алгоритмы. Построение и анализ»
- Дасгупта С., Пападимитриу Х., Вазирани У. — «Алгоритмы»

Лекция

Ханойские башни

Есть три стержня. На первый стержень нанизано 64 диска, от самого большого к самому маленькому. Задача: переложить все диски на второй стержень. Ограничения:

- Диски можно переносить только по одному.

- Нельзя класть диск большего диаметра на диск меньшего диаметра.

Какой может быть алгоритм?

Варианты из аудитории:

1. Полный перебор
2. Рекурсивный алгоритм.

Рассмотрим такой рекурсивный алгоритм:

1. Переложим все диски, кроме n -ного, на третий стержень;
2. Переложим n -ный диск с первого на второй стержень;
3. Переложим все остальные с третьего стержня на второй.

Запишем этот алгоритм с помощью псевдокода:

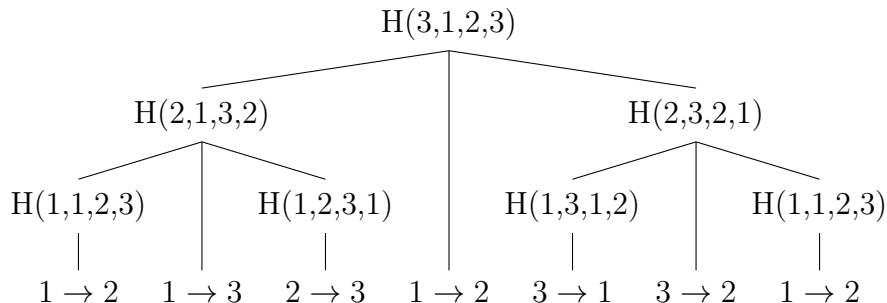
Algorithm 1 Рекурсивный алгоритм решения задачи о Ханойской башне

```

1: function HANOI3( $n, i, j, k$ )            $\triangleright n$  — количество дисков,  $i, j, k$  — номера стержней
2:   if  $n > 0$  then
3:     HANOI3( $n - 1, i, k, j$ )
4:     move  $i \rightarrow j$ 
5:     HANOI3( $n - 1, k, j, i$ )

```

Нарисуем дерево операций для $n = 3$:



Алгоритм, по сути, обходит это дерево в глубину и при этом слева направо, выполняя все перемещения, что встретятся.

Это дерево можно рассматривать, как полное бинарное дерево глубины n , если перемещения учитывать не в отдельных листьях, а в родительских узлах. Тогда в каждом узле мы выполняем одно действие, а в полном бинарном дереве $2^n - 1$ узлов. Следовательно, выполнится $2^n - 1$ перемещение.

Пусть число перемещений для n дисков равно $f(n)$. Тогда верно следующее:

$$f(n) = \begin{cases} 0, & n = 0 \\ 2f(n-1) + 1, & n > 0 \end{cases}$$

Свойство: $f(n) = 2^n - 1$

Доказательство. Докажем это по индукции. База верна, так как $f(0) = 0 = 2^0 - 1$. Теперь пусть предположение верно для $n - 1$, то есть $f(n - 1) = 2^{n-1} - 1$. Тогда $f(n) = 2f(n - 1) + 1 = 2(2^{n-1} - 1) + 1 = 2^n - 2 + 1 = 2^n - 1$, что и требовалось доказать. \square

Можно ли улучшить время работы? Оказывается, что нет.

Рассмотрим некоторый алгоритм. Он рано или поздно должен переложить наибольший диск на второй стержень. Для этого ничего не должно быть на нём и на втором, т.е. все на третьем. А как получить эту конфигурацию? Оптимальным алгоритмом на $n - 1$ шаг, что приводит к нашим вычислениям и уже полученному минимальному результату в $2^n - 1$.

Утверждение. *Задачу о Ханойских башнях нельзя решить за меньшее число шагов, причём решение с таким числом шагов ровно одно.*

Доказательство. По индукции.

База ($n = 0$): очевидно, решить быстрее, чем за 0 шагов нельзя и последовательность такая ровно одна.

Переход ($n - 1 \rightarrow n$): предположим, что мы доказали это утверждение для $n - 1$. Рассмотрим утверждение для n . Рано или поздно алгоритму понадобится освободить первые два стержня, чтобы переложить первый диск на второй стержень. Необходимо сделать это одно перекладывание и после вернуть все оптимальным алгоритмом. Итого, опираясь на предположение индукции шагов в оптимальном и единственном решении для $n - 1$, нам понадобится $2(2^{n-1} - 1) + 1$ шаг, что и равно $2^n - 1$. \square

Изменим задачу:

Четыре стержня

Условие в остальном ровно то же. Стала ли задача проще?

Сложнее она точно не стала, т.к. четвёртым можно просто не пользоваться.

Рассмотрев переход от двух к трём, кажется, что должно быть проще; как можно воспользоваться четвёртым?

Предложение: переложить предпоследний отдельно на четвёртый и сэкономить на перекладывании башни из $n - 1$, перекладывая вместо неё башню из $n - 2$

Algorithm 2 Рекурсивный алгоритм решения задачи о Ханойской башне на 4-х стержнях, версия 1

```
1: function HANOI4( $n, i, j, k, l$ )            $\triangleright n$  — количество дисков,  $i, j, k, l$  — номера стержней
2:   if  $n > 0$  then
3:     HANOI4( $n - 1, i, l, k, j$ )
4:     move  $i \rightarrow k$ 
5:     move  $i \rightarrow j$ 
6:     move  $k \rightarrow j$ 
7:     HANOI4( $n - 1, l, j, i, k$ )
```

Построив аналогичное дерево, получим, что в каждом узле три перемещения, а узлов $2^{\lfloor \frac{n}{2} \rfloor} - 1$ — экономия, но не очень большая.

Построим другой алгоритм:

Algorithm 3 Рекурсивный алгоритм решения задачи о Ханойской башне на 4-х стержнях, версия 2

```
1: function HANOI4( $n, i, j, k, l$ )            $\triangleright n$  — количество дисков,  $i, j, k, l$  — номера стержней
2:   if  $n > 0$  then
3:     HANOI4( $n - m, i, l, k, j$ )
4:     HANOI3( $m, i, j, k$ )
5:     HANOI4( $n - m, l, j, i, k$ )
```

Заметим, что число шагов зависит от m . Пусть $n_m = \frac{m(m+1)}{2}$ (если n другое, то на первом шаге выберем такой m , чтобы $n - m$ было таким, а дальше на вход будет поступать число такого вида).

Построим дерево алгоритма. Оно также будет полным бинарным деревом. Заметим, что в нём m уровней, так как при каждом шаге m уменьшается на единицу. Это связано с тем, что $n_m - m = \frac{m(m+1)}{2} - m = \frac{(m-1)m}{2} = n_{m-1}$. При этом в узле на i -м уровне (нумерация с нуля) проводится $2^{m-i} - 1$ операция, так как мы пользуемся доказанным ранее алгоритмом для трёх стержней. Тогда на каждом уровне выполняется $2^m - 2^i$ операций. Тогда всего выполняется $\sum_{i=0}^{m-1} (2^m - 2^i) = m2^m - \sum_{i=0}^{m-1} 2^i = (m-1)2^m + 1$ операций.

Пусть число перемещений для n дисков равно $g(n)$. Тогда

$$g(n_m) = \begin{cases} 0, & m = 0 \\ 2g(n_{m-1}) + 2^m - 1, & m > 0 \end{cases}$$

Предложение: $g(n_m) = (m-1)2^m + 1$

Доказательство. По индукции. База верна, так как $g(n_0) = 0 = (0-1)2^0 + 1 = -1 + 1$. Теперь допустим, что предположение верно для n_{m-1} , то есть $g(n_{m-1}) = (m-2)2^{m-1} + 1$. Тогда $g(n_m) = 2g(n_{m-1}) + 2^m - 1 = (m-2)2^m + 2 + 2^m - 1 = (m-1)2^m + 1$. \square

Заметим, что при достаточно больших n верно, что $m \approx \sqrt{2n}$. Тогда $g(n) \approx \sqrt{2n} \cdot 2^{\sqrt{2n}}$. Тогда $g(n) = \Theta(\sqrt{n} \cdot 2^{\sqrt{2n}})$. Θ означает (грубо говоря), что функция растёт примерно так же.

Попробуем обобщить этот алгоритм для любого числа стержней:

Algorithm 4 Рекурсивный алгоритм решения задачи о Ханойской башне, общий случай

```

1: function HANOI( $n, i, j, P$ )                                 $\triangleright n$  — количество дисков,  $i, j$ , — основные стержни
2:                                                          $\triangleright P$  — множество вспомогательных стержней
3:   if  $n > 0$  then
4:     choose  $p \in P$ 
5:      $R := P \setminus p$ 
6:     if  $R = \emptyset$  then
7:       HANOI3( $n, i, j, p$ )
8:     else
9:       HANOI( $n - m, i, p, R \cup \{j\}$ )
10:      HANOI( $m, i, j, R$ )
11:      HANOI( $n - m, p, j, R \cup \{i\}$ )

```

Пусть для перемещения n дисков с помощью алгоритма на k стержнях ($k \geq 3$) нужно $h(n, k)$ операций. Тогда верно следующее:

$$h(n_m, k) = \begin{cases} 0 & n = 0 \\ 2^{n_m} - 1 & n > 0, k = 3 \\ 2h(n_{m-1}, k) + h(m, k-1) & n > 0, k > 3 \end{cases}$$

2 Алгоритмы сортировки 14.01.2016

Задача сортировки

Вход: последовательность чисел (строго говоря, может быть что угодно с полным порядком) (a_1, a_2, \dots, a_n) .

Algorithm 5 Неэффективный алгоритм сортировки

```
1: function RECURSIVE_SORT( $a$ )  $\triangleright a = (a_1, a_2, \dots, a_n)$ 
2:    $n := |a|$ 
3:   if  $n > 1$  then
4:     RECURSIVE_SORT( $a[1 : n - 1]$ )
5:      $k := a_n$ 
6:     for  $i := n - 1$  downto 1 do
7:       if  $a_i > k$  then
8:          $a_{i+1} := a_i$ 
9:       else
10:        break
11:       $a_{i+1} := k$ 
```

Выход: $(a_{i_1}, a_{i_2}, \dots, a_{i_n})$, где $a_{i_k} \leq a_{i_{k+1}}$. Другими словами, на выходе получается отсортированная по возрастанию последовательность.

Рассмотрим неэффективный алгоритм:

$[6, 8, 3, 4] \rightarrow [3, 6, 8, 4] \xrightarrow{8>4} [3, 6, _, 8] \xrightarrow{6>4} [3, _, 6, 8] \xrightarrow{3<4} [3, 4, 6, 8]$

По сути, мы идём слева направо и каждому элементу находим место среди прошлых уже отсортированных элементов.

Теперь рассмотрим алгоритм *сортировки вставками*.

Algorithm 6 Алгоритм сортировки вставками

```
1: function RECURSIVE_SORT( $a$ )  $\triangleright a = (a_1, a_2, \dots, a_n)$ 
2:    $n := |a|$ 
3:   for  $j := 2$  to  $n$  do
4:      $k := a_j$ 
5:     for  $i := j - 1$  downto 1 do
6:       if  $a_i > k$  then
7:          $a_{i+1} = a_i$ 
8:       else
9:        break
10:       $a_{i+1} := k$ 
```

Докажем корректность алгоритма формально. Для этого найдём *инвариант*.

Инвариант: в начале каждой итерации цикла по j массив с 1 по $j - 1$ индекс уже отсортирован. При этом он состоит из тех же элементов, что и раньше.

Если это условие выполняется, то после выполнения алгоритма, весь массив (с 1-го по n -ый индексы) будет отсортирован.

Доказательство. По индукции:

База: $j = 2 - a[1 : 1]$ отсортирован

Переход Всё до j -го отсортировано; Поставим a_j на нужное место. Тогда полученный массив также будет отсортирован. \square

Насколько эффективно он работает? Понятно, что это зависит от входных данных. Ясно, что чем больше элементов, тем дольше он работает. Понятно также, что если массив уже отсортирован, то работать он будет быстрее.

$T(n)$ — время работы на входе длины n в худшем случае. (1)

в среднем случае. (2)

в лучшем случае. (3)

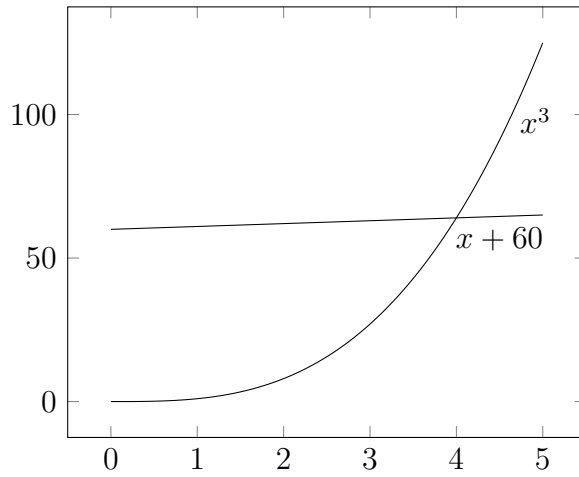
Однако оценка в лучшем случае, вообще говоря, бесполезна. Ведь любой алгоритм можно модифицировать так, чтобы в каком-то случае он работал очень быстро.

Асимптотический анализ: как меняется $T(n)$ при $n \rightarrow \infty$? Для исследования этого обычно применяют O -нотацию или Θ -нотацию.

$$\Theta(g(n)) = \{f(n) \mid \exists c_1, c_2 > 0 : \forall n \geq n_0 \implies 0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n)\}$$

Например, пусть задана функция $f(n) = 3n^2 + 2n - 6$. Тогда $f(n) \in \Theta(n^2)$.

Асимптотика — это хорошо, но на константы тоже стоит обращать внимание: для маленьких n вполне может быть, что n^3 работает быстрее, чем n .



Оценим *худший случай* нашего алгоритма (когда на каждом шагу приходится совершать максимальное число перемещений): $T(n) = \sum_{j=2}^n \sum_{i=j-1}^1 \Theta(1) = \sum_{j=2}^n \Theta(j) = \Theta(n^2)$

Средний случай: Предположим, что все входы равновероятны. Тогда будет выполняться примерно половина сравнений и $T(n) = \sum \Theta(\frac{j}{2}) = \Theta(n^2)$

Лучший случай — это случай, когда массив уже отсортирован. Тогда $T(n) = \Theta(n)$.

Рассмотрим другой алгоритм — *сортировку слиянием*.

Algorithm 7 Алгоритм сортировки слиянием

```

1: function MERGE_SORT( $a$ )  $\triangleright a = (a_1, a_2, \dots, a_n)$ 
2:    $n := |a|$ 
3:   if  $n > 1$  then
4:      $b_1 := \text{MERGE\_SORT}(a[1 : \frac{n}{2}])$ 
5:      $b_2 := \text{MERGE\_SORT}(a[\frac{n}{2} + 1 : n])$ 
6:      $a := \text{MERGE}(b_1, b_2)$ 
7:   return  $a$ 

```

Рассмотрим, как может работать merge.

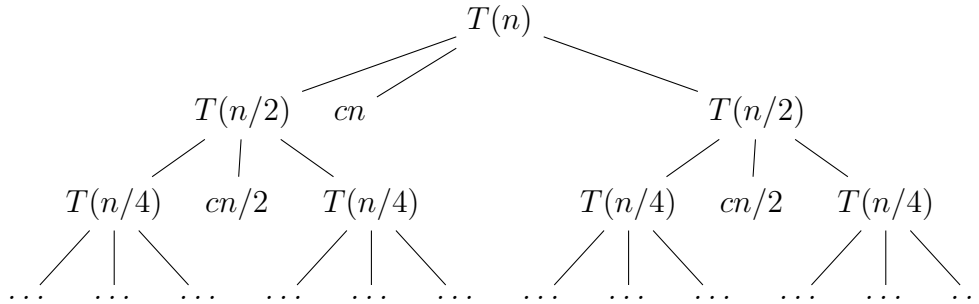
$[2, 5, 6, 8]; [1, 3, 7, 9]$

Будем сливать элементы из массивов в результирующий массив, сравнивая поочерёдно минимальные элементы, которые ещё не вошли в результирующий массив. $2 > 1$ — первый элемент — 1; $2 < 3$ — второй элемент — 2. . . Очевидно, что алгоритм корректен, а его сложность — линейная, так как мы один раз проходим по массивам, то есть $\Theta(n)$. // ЯННП. Тамерлан, прошу поправить. Короткая правка от окаеринасаи

Пусть худшее время для MERGE_SORT — $T(n)$. Тогда

$$T(n) = \begin{cases} \Theta(1), n = 1 \\ 2T(n/2) + \Theta(n) \end{cases}$$

Построим дерево рекурсии:



На каждом уровне cn работы, а высота дерева — $\log_2 n$. Общее время работы — $n\Theta(1) + cn \log n = \Theta(n \log n)$.