Алгоритмы. ДЗ на неделю 4.

Прохоров Юрий, 771

Задача 6 (с семинара)

Алгоритм

Итеративный алгоритм MergeSort. Пусть на вход подается массив

$$A = [a_1, a_2, \dots, a_L] \,.$$

Пусть процедура MERGE(A, m, n) объединяет два подмассива A' и A'' массива A

$$A' = [a_m, \dots, a_{n-1}] \,, \qquad A'' = [a_n, \dots, a_{m+n-1}]$$

по тому же принципу, что и в рекурсивном алгоритме MergeSort. На работу этой операции требуется $c(n-m)$ элементарных операций (сравнения, присваивания), которые будем считать за $O(1)$.

```
for i = 1; i < L; i = i × 2 do
    for j = 1; j ≤ L; j = j + 2i do
        MERGE(A, j, j+i);
    end
end
```

Корректность

Докажем корректность по индукции. Покажем, что после каждого i-го шага алгоритма (внешнего цикла) массив содержит некоторое число отсортированных блоков длины 2^i.

На первом шаге каждый $2k$-ый и $(2k+1)$-ый элементы массива будут упорядочены по возрастанию, поэтому массив будет состоять из пар отсортированных элементов.

Пусть $(k-1)$-го шага массив разбит на отсортированные блоки длины 2^{k-1}. На k-ой итерации мы будем процедурой MERGE сливать соседние блоки в отсортированные длиной $2 \cdot 2^{k-1} = 2^k$.

Оценка по времени

Внешний цикл выполняет $\lceil \log_2 L \rceil$ операций, и на i-м шаге внешнего цикла выполняется $\left\lceil \frac{L}{2i} \right\rceil$ процедур MERGE, который требует $c(j+i-j) = ci$ единиц времени. Тогда каждая итерация внешнего цикла занимает $c \cdot \left\lceil \frac{L}{2} \right\rceil = O(L)$ времени. Весь алгоритм тогда работает за $O(L \log L)$.

Задача 1 (ДЗ)

1. $F(3,5) = 3^5$.

2. $F(x,m) = x^m$.

3. Корректность. Заметим, что если бы процедура $SX(m)$ не отбрасывала первую слева букву S, то S всегда стояла бы на первой позиции из-за того, что двоичная запись любого положительного целого числа (так сказано в условии) начинается с единицы. Поэтому в таком случае на первом шаге алгоритма число $y = 1$ будет просто возводиться в квадрат, что на конечный результат не повлияет. Пусть SX_1 - такое преобразование.

Число $y = 1$ всегда умножается либо на себя, либо на постоянное число x, поэтому y всегда будет степенью числа x. Для нахождения возвращаемого значения достаточно найти конечный показатель степени числа $y = x^s$.

Преобразуем данный алгоритм в эквивалентный:

```
Function F(x, m) :
    a = SX_1(m);
    s = 0;
    for i = 1 to a.size do
        if a[i] == X then
            s = s + 1;
        else
            s = 2s;
        end
    end
    Output: x^s
end
```

При такой реализации каждый бит двоичной записи числа m преобразуется либо в SX, если это 1, либо в S. Тогда алгоритм можно сократить еще больше:

```
Function F(x, m) :
    b = bin(m);
    s = 0;
    for i = 1 to b.size do
        s = 2s;
        if b[i] == 1 then
            s = s + 1;
    end
    Output: x^s
end
```

В такой записи алгоритма видно, что число s строится по двоичной записи числа m, поэтому в итоге $s = m$, и на выходе будет x^m.

4. Оценка по времени. Будем считать длиной входа n длину двоичной записи числа m. Процедура SX в худшем случае преобразует его в строку из $2n-1$ символов. На каждом из $2n-1$ шагов происходит элементарная арифметическая операция, поэтому общая сложность алгоритма $T(n) = c(2n-1) = \Theta(n)$.

Задача 2 (ДЗ)

Раз округлений не дано, будем считать, что количество вложенных отрезков, подаваемых на вход, является степенью числа 3.

Алгоритм и корректность

Допустим, мы отсортировали, например, левые концы отрезков по возрастанию. Тогда правые концы автоматически будут отсортированы по убыванию. Мы получим следующий массив строго вложенных отрезков:

$$\left\{[l_{(1)}, r_{(1)}], [l_{(2)}, r_{(2)}], \ldots, [l_{(n)}, r_{(n)}]\right\},$$

в котором каждый следующий отрезок содержится в предыдущем.

Пусть некоторая точка покрыта отрезком $[l_{(k)}, r_{(k)}]$. Тогда она покрыта всеми отрезками $[l_{(i)}, r_{(i)}], i \in \overline{[1, k]}$, то есть хотя бы k отрезками. Аналогично, пусть эта же точка не покрыта отрезком $[l_{(k+1)}, r_{(k+1)}]$, тогда она покрыта ровно k отрезками. Верно и обратное: если точка покрыта ровно k отрезками, то она покрыта отрезком $[l_{(k)}, r_{(k)}]$, но не покрыта отрезком $[l_{(k+1)}, r_{(k+1)}]$.

Таким образом, точка покрыта ровно $\frac{2n}{3}$ отрезками тогда и только тогда, когда она покрыта отрезком $[l_{(s)}, r_{(s)}]$, но не покрыта отрезком $[l_{(s+1)}, r_{(s+1)}]$, где $s = \frac{2n}{3}$. Следовательно, задача сводится к нахождению s-ой и $(s+1)$-ой порядковой статистике в массиве левых концов отрезков. Множество точек Φ, покрытое ровно s отрезкам

$$\Phi = \left[l_{(s)}, l_{(s+1)} - 1\right] \cup \left[r_{(s+1)} + 1, r_{(s)}\right].$$

Пусть L — массив левых концов отрезков, R — массив правых концов, $kStat(k, A)$ — алгоритм поиска k-ой порядковой статистики в массиве A, возвращающий номер найденного элемента в данном массиве.

Input: L, R
$p = kStat\left(\frac{2n}{3}, L\right)$;
$q = kStat\left(\frac{2n}{3} + 1, L\right)$;
Output: $[l_p, l_q - 1] \cup [r_q + 1, r_p]$

Оценка по времени

Алгоритм $kStat$ поиска порядковой статистики является линейным, и он выполняется в данной задаче дважды, поэтому общее время работы данного алгоритма линейно: $T(n) = \Theta(n)$.

Задача 3 (ДЗ)

Опустим подробное описание алгоритма поиска порядковой статистики, так как он уже был разобран и на лекции, и на семинаре.

Пусть мы делим подаваемый на вход массив на группы по 7 элементов, а не по 5. Отсортируем эти $\left\lceil \frac{n}{7} \right\rceil$ групп и выделим в них медианы. В массиве медиан рекурсивным вызовом найдем медиану, то есть серединную порядковую статистику d. Перебирая все элементы в исходном массиве, будем процедурой $PARTITION$ левее d ставить все элементы, меньшие d, а правее — все остальные.

Слева от d окажется $\left\lfloor \frac{1}{2} \left\lceil \frac{n}{7} \right\rceil \right\rfloor$ медиан, а вместе с каждой из этих медиан еще по 3 элемента из той же группы. Поэтому левее d будет хотя бы $4\left\lfloor \frac{1}{2} \left\lceil \frac{n}{7} \right\rceil \right\rfloor$ элементов. Аналогичное количество будет и справа. Затем алгоритм рекурсивно вызывается от подмассива, в котором содержится исходная порядковая статистика. Длина этого подмассива составляет не больше $n - 4\left\lfloor \frac{1}{2} \left\lceil \frac{n}{7} \right\rceil \right\rfloor$. Поэтому время работы алгоритма задается формулой

$$T(n) = \underbrace{T\left(n - 4\left\lfloor \frac{1}{2} \left\lceil \frac{n}{7} \right\rceil \right\rfloor\right)}_{\substack{\text{рекурсивный} \\ \text{вызов}}} + \underbrace{T\left(\left\lceil \frac{n}{7} \right\rceil\right)}_{\substack{\text{поиск} \\ \text{медианы}}} + \underbrace{c_1 n + c_2 n + c_3 n}_{\substack{\text{разбиение на группы,} \\ \text{сортировка групп,} \\ PARTITION}}.$$

Докажем, что $T(n) = \Theta(n)$ по индукции. Допустим, что $\exists c : T(n) < cn$. Для малых n это верно.

При достаточно больших n

$$n - 4\left\lfloor \frac{1}{2} \left\lceil \frac{n}{7} \right\rceil \right\rfloor \le n - 2\left\lceil \frac{n}{7} \right\rceil + 2 \le n - \frac{2n}{7} + 2 = \frac{5n}{7} + 2 \le \frac{5,1n}{7},$$

$$\left\lceil \frac{n}{7} \right\rceil \le \frac{1,1n}{7}.$$

Тогда

$$T(n) \le T\left(\frac{5,1n}{7}\right) + T\left(\frac{1,1n}{7}\right) + c'n < c\frac{5,1n}{7} + c\frac{1,1n}{7} + c'n < cn\left(\frac{6,2}{7} + \frac{c'}{c}\right),$$

$$\frac{6,2}{7} + \frac{c'}{c} < 1 \quad \Longrightarrow \quad c > \frac{7c'}{0,8}.$$

Таким образом, $T(n) < cn$. Нижняя линейная оценка ясна из-за наличия явных линейных членов в $T(n)$. Следовательно, при делении на группы по 7:

$$T(n) = \Theta(n).$$

Задача 4 (ДЗ)

Алгоритм и корректность

Для решения будем использовать сортировку *Counting Sort*. Пройдемся по всему массиву A и посчитаем, сколько раз встретилось число 0. Затем в начало массива запишем это количество нулей, а во все остальные ячейки — единицы.

Input: A
$count = 0$;
for $i = 0; i < A.Length; i = i + 1$ **do**
    if $A[i] == 0$ **then**
        $count = count + 1$;
end
for $i = 0; i < A.Length; i = i + 1$ **do**
    if $i < count$ **then**
        $A[i] = 0$;
    else
        $A[i] = 1$;
    end
end
Output: A

Оценка по времени

Алгоритм требует двух проходов по массиву, то есть совершения $c_1 n$ сравнений и $c_2 n$ присваиваний. Хотя это не самый оптимальный алгоритм по числу элементарных операций, он имеет линейную асимптотику $\Theta(n)$.

Задача 5 (ДЗ)

Алгоритм и корректность

Любое решение x сравнения

$$a \cdot x + b \equiv 0 \pmod M$$

при некотором целом y является решением диофантового уравнения

$$-ax + My = b.$$

Вместо этого уравнения с помощью расширенного алгоритма Евклида (*ExtendedEuclid*) решим уравнение

$$ax + My = b. \tag{1}$$

Алгоритм вернет некоторое частное решение $\begin{pmatrix} x_0 \\ y_0 \end{pmatrix}$ и период $\begin{pmatrix} x_1 \\ y_1 \end{pmatrix}$, то есть решением являются пары

$$\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} x_0 \\ y_0 \end{pmatrix} + k \begin{pmatrix} x_1 \\ y_1 \end{pmatrix}. \tag{2}$$

Решением исходного уравнения будут числа

$$\begin{pmatrix} \tilde{x} \\ \tilde{y} \end{pmatrix} = \begin{pmatrix} -x_0 \\ y_0 \end{pmatrix} + k \begin{pmatrix} -x_1 \\ y_1 \end{pmatrix}.$$

Для решения сравнения достаточно ограничиться только значением переменной $\tilde{x}$:

$$\tilde{x} = -x_0 + cx_1.$$

Корректность напрямую следует из корректности работы расширенного алгоритма Евклида. Пусть функция $ExtendedEuclid(a, b, c)$ возвращает решение вида (2) диофантового уравнения вида (1) с коэффициентами a, b, c в виде четырех чисел (x_0, y_0, x_1, y_1).

Input: a, b, M
$(x_0, y_0, x_1, y_1) = ExtendedEuclid(a, M, b)$;
Output: $(-x_0, x_1)$

Оценка по времени

Время работы расширенного алгоритма Евклида состоит из $\Theta(n)$ рекурсивных вызовов, на каждом из которых выполняются операции умножения и деления, требующие времени $\Theta(n^2)$, где n - длина битовой цепочки входных чисел. Асимптотика данного алгоритма совпадает с асимптотикой расширенного алгоритма Евклида и равна $\Theta(n^3)$.

Задача 6 (ДЗ)

1. Будем рассматривать вариацию быстрой сортировки, в которой в качестве опорного элемента берется последний элемент массива. Пусть входной массив длины n уже отсортирован. Тогда

$$T(n) = T(n-1) + cn \quad \Longrightarrow \quad T(n) = \Theta(n^2)$$

и будет произведено $\Theta(n)$ рекурсивных вызовов. Это же число является глубиной стека или высотой дерева рекурсии. Больше сделать не получится, потому что на каждом шаге алгоритма задача должна разбиваться на более простые, а сумма аргументов на каждом k-м уровне дерева рекурсии должна быть $n - k$ (так устроен алгоритм $QSort$). В данном случае на каждом уровне дерева только один узел.

$$\begin{array}{rc}
Level\ 0: & T(n) \\
 & | \\
Level\ 1: & T(n-1) \\
 & | \\
\dots & \dots \\
 & | \\
Level\ (n-1): & T(1)
\end{array}$$

Худшая глубина стека — $\Theta(n)$.

2. Чтобы улучшить асимптотику глубины стека будем вместо последнего элемента массива в качестве опорного элемента брать медиану массива, которую за линейное время можно найти с помощью алгоритма поиска порядковой статистики.

В таком алгоритме

$$T(n) = 2T\left(\frac{n}{2}\right) + \Theta(n).$$

Глубина стека или высота дерева рекурсии будет равна $\lceil \log_2 n \rceil = \Theta(\log n)$. Дерево рекурсии выглядит следующим образом:

$$
\begin{array}{lccccc}
\textit{Level } 0: & & & T(n) & & \\
\textit{Level } 1: & & T\left(\frac{n}{2}\right) & & T\left(\frac{n}{2}\right) & \\
\dots & & & & & \\
\textit{Level } \lceil \log_2 n \rceil: & \underbrace{T(1) \quad \cdots \quad \cdots \quad \cdots \quad T(1)}_{2^{\log_2 n} = n} & & & &
\end{array}
$$

Задача 7(a) (ДЗ №3)

Начнем заполнять массив с конца. Для начала посчитаем $\mathrm{invfac}[n] = (n!)^{-1} \pmod p$. Нужно найти такое число x:

$$n! \times x \equiv 1 \pmod p, \qquad 0 < x < p.$$

Другими словами, нужно решить диофантово уравнение

$$n!x + py = 1$$

относительно переменной x. Оно имеет решение, так как число p является простым, а $n < p$, вследствие чего $\gcd(n!, p) = 1$. С помощью расширенного алгоритма Евклида за $\Theta(log_2(n!)) = \Theta(n \log n)$ или при больших p за $\Theta(\log p)$ можно найти его решение $x : 0 < x < p$.

Покажем, что

$$(k-1)!^{-1} \equiv k!^{-1} \times k \pmod p.$$

Пусть

$$k!^{-1} \pmod p = m \qquad \Longleftrightarrow \qquad k! \times m \equiv 1 \pmod p. \tag{3}$$

Тогда

$$(k-1)!^{-1} \pmod p = x \qquad \Longleftrightarrow \qquad (k-1)!^{-1} \times x \equiv 1 \pmod p.$$

Вследствие равенства (3), последнее сравнение является верным при

$$x \equiv km \pmod p$$

Input: n, p
$(x_0, y_0, x_1, y_1) = \mathit{ExtendedEuclid}(n!, p, 1)$;
$\mathrm{invfac}[n] = x_0 \mod p$;
for $i = n-1; i > 0; i = i-1$ **do**
  $\mathrm{invfac}[i] = \mathrm{invfac}[i+1] \times i \mod p$;
end
Output: invfac

Время работы алгоритма — $\Theta(n \log n)$ или $\Theta(n + \log p)$ из-за вычисления алгоритма Евклида.

Есть и другой вариант решения: по теореме Вильсона

$$(p-1)! \equiv -1 \equiv p-1 \pmod p \qquad \Longrightarrow \qquad (p-1)^{-1} \pmod p = p-1.$$

Далее аналогичным методом мы вычисляем все предыдущие обратные остатки. Такой метод требует $\Theta(p)$ времени.