Дисциплина

«РАЗРАБОТКА ЭФФЕКТИВНЫХ АЛГОРИТМОВ»

Лекция 16

ТЕОРЕМА О РЕКУРРЕНТНЫХ СООТНОШЕНИЯХ, ПОРОЖДЕННЫХ МЕТОДОМ ДЕКОМПОЗИЦИИ

Лектор: Михаил Васильевич Ульянов,

muljanov@mail.ru, **8 916 589 94 04**

1. Рекуррентное соотношение на трудоемкость в методе декомпозиции

Пусть в результате применения метода декомпозиции к некоторой задаче (допускающей применение этого метода!) при разбиении задачи на $b > 1$ частей, мы получаем $a \geq 1$ подзадач размерности n/b. При этом некоторое число элементарных операций затрачивается на разбиение и последующее (после возврата из рекурсивных вызовов) объединение решений. Пусть функция $g(n)$ описывает сумму этих затрат. При останове рекурсии при размерности $n = n_0$ задача данной размерности требует $f(n_0) = c_0$ элементарных операций.

Тогда мы получаем следующее рекуррентное соотношение на трудоемкость

$$\begin{cases} f(n_0) = c_0 \\ f(n) = af\left(\left[\frac{n}{b}\right]\right) + g(n) \end{cases}.$$

2. Основная теорема о рекуррентных соотношениях

Непосредственное решение этого рекуррентного соотношения связано с трудностями, вызванными округлениями в «пол» и «потолок».

Если наша задача состоит в том, чтобы получить только асимптотическую оценку трудоемкости — вычислительную сложность, то в нашем распоряжении следующая теорема, доказанная в 1980 г. Дж. Бентли, Д. Хакен и Дж Саксом. Полученный авторами результат является достаточно мощным средством асимптотической оценки рекурсивно заданных функций данного вида, и применим к анализу трудоемкости рекурсивных алгоритмов, основанных на методе декомпозиции.

Теорема. (J.L. Bentley, Dorothea Haken, J.B. Saxe, 1980)

Пусть $a \geq 1$ и $b > 1$ — константы, $g(n)$ — известная функция, $f(n)$ определено при положительных значениях n формулой

$$\begin{cases} f(n_0) = c_0 \\ f(n) = af\left(\left[\frac{n}{b}\right]\right) + g(n) \end{cases},$$

тогда:

1) Если $g(n) = O\left(n^{\log_b a - \varepsilon}\right)$ для некоторого $\varepsilon > 0$, то $f(n) = \Theta\left(n^{\log_b a}\right)$;

2) Если $g(n) = \Theta\left(n^{\log_b a}\right)$, то $f(n) = \Theta\left(n^{\log_b a} \log_b n\right)$;

3) Если найдутся $c > 0$ и $\varepsilon > 0$, такие, что при достаточно больших n выполнено условие:

$$g(n) > cn^{\log_b a + \varepsilon},$$

и найдется положительная константа $0 < d < 1$ такая, что при достаточно больших n выполнено условие (названное авторами теоремы условием регулярности):

$$ag\left(\frac{n}{b}\right) \leq dg(n), \text{ то } \quad f(n) = \Theta(g(n)).$$

3. Примеры применения теоремы

Пример 1 (абстрактный). Предположим, что метод декомпозиции приводит к разделению задачи на четыре части и возникновению восьми подзадач меньшей размерности, причем деление и объединение решений имеют линейную трудоемкость. В этом случае функция $f(n)$, очевидно, имеет вид

$$f(n) = 8f\left(\left[\frac{n}{4}\right]\right) + cn.$$

В обозначениях теоремы $a = 8$, $b = 4$, $g(n) = cn$.

При этом $n^{log_b a} = n^{log_4 8} = \Theta(n^{1,5})$. Поскольку $g(n) = O(n^{1,5-\varepsilon})$ для $\varepsilon = 0,5$, то, применяя первое утверждение теоремы, делаем вывод, что

$$f(n) = \Theta(n^{1,5}).$$

Пример 2 (абстрактный). Предположим, что метод декомпозиции приводит к такому разбиению задачи, что функция $f(n)$ имеет вид

$$f(n) = f\left(\left[\frac{3n}{4}\right]\right) + 5.$$

Для получения асимптотической оценки функции $f(n)$ выпишем коэффициенты в обозначениях теоремы — $a = 1$, $b = 4/3$, $g(n) = 5$, а $n^{log_b a} = n^{log_{4/3} 1} = n^0 = \Theta(1)$. На этом основании воспользуемся вторым случаем теоремы. Поскольку

$g(n) = 5 = \Theta\left(n^{log_b a}\right) = \Theta(1)$, то получаем, что $f(n) = \Theta\left(log_{4/3} n\right)$.

Пример 3 (абстрактный). Пусть метод декомпозиции при разработке алгоритма решения задачи приводит к разделению задачи на четыре части и возникновению двух подзадач меньшей размерности, а деление и объединение решений имеет следующую трудоемкость — $g(n) = cn\, log_4 n$.

Для этого примера функция $f(n)$ может быть записана в виде

$$f(n) = 2f\left(\left[\frac{n}{4}\right]\right) + cn\, log_4 n.$$

Имеем (в обозначениях теоремы) $a = 2$, $b = 4$, $g(n) = cn\, log_4 n$, а $n^{log_b a} = n^{log_4 2} = n^{0,5} = \Theta(\sqrt{n})$ и, при больших n и $\varepsilon = 0,5$

$$g(n) = cn\, log_4 n \geq n^{0,5+\varepsilon}.$$

Это третий случай теоремы и остается проверить условие регулярности для третьего случая. Для достаточно больших значений n имеем

$$2g\left(\frac{n}{4}\right) = 2\frac{n}{4} log_4 \frac{n}{4} \leq \frac{1}{2} n\, log_4 n = dg(n), \text{ где } d = \frac{1}{2} < 1,$$

и, тогда

$$f(n) = \Theta(n\, log_4 n).$$

Пример 4. Рассмотрим известный ***метод половинного деления***, применяемый для решения уравнений $f(x) = 0$ с заданной точностью h.

Предполагаем, что известно, что корень лежит на отрезке $[a; b]$. Кроме того, известно, что на этом отрезке лежит только один корень данного уравнения. Следовательно, на концах этого отрезка значения функции имеют разные знаки.

Для определенности будем считать, что $f(a) < 0$, $f(b) > 0$, и что корень определен с нужной степенью точности, если мы определим такой отрезок $[c;\ d]$, что $f(\text{c}) < 0$, $f(d) > 0$ и при этом $d - c \leq h$ (точность по аргументу).

Обозначим $a_1 = a$, $b_1 = b$. Мы можем утверждать, поскольку $f(a_1) < 0$, $f(b_1) > 0$, а функция, непрерывная на отрезке, принимает все промежуточные значения, что неизвестный корень уравнения $f(x) = 0$ лежит на отрезке $[a_1; b_1]$. Идея алгоритма состоит в построении такой последовательности отрезков $[a_i; b_i]$, что $f(a_i) < 0$, $f(b_i) > 0$, и при этом $b_{i+1} - a_{i+1} < b_i - a_i$, то есть на каждом шаге алгоритма длина отрезка, на котором лежит корень уравнения, уменьшается. Возьмем значение c в середине отрезка $c = \frac{a_i + b_i}{2}$. Вычислим значение $f(c)$. Возможны два случая $f(c)$ <0 или $f(c)$ >0. (Случай $f(c)$ =0 в виду его слишком малой вероятности для реальных уравнений не рассматриваем). В случае $f(c)$ <0 полагаем $a_{i+1} = c$, $b_{i+1} = b_i$. В противном случае, когда $f(c)$ >0, полагаем $a_{i+1} = a_i$, $b_{i+1} = c$. Далее вычисления повторяются. Условием останова алгоритма является выполнение условия $b_{i+1} - a_{i+1} < h$.

Рекурсивный характер данного алгоритма очевиден, но может возникнуть вопрос, что в этом алгоритме играет роль параметра n. Параметр n, несмотря на то, что он явно не проявляется, тем не менее, присутствует в данном алгоритме. Роль параметра n играет то, сколько раз значение требуемой точности h «укладывается» на отрезке:

$$n = \left[\frac{b-a}{h}\right].$$

Для асимптотической оценки количества необходимых базовых операций этого алгоритма получаем соотношение вида (при условии, что вычисление значения функции требует фиксированного числа операций!)

$$f(n) = 1 \cdot f\left(\left[\frac{n}{2}\right]\right) + c,$$

где значение c определяется количеством операций, требуемых для вычисления функции и соответствующих операции сравнения и присваивания новых значений концам текущего отрезка.

Для получения асимптотической оценки функции $f(n)$ выпишем коэффициенты в обозначениях теоремы — $a = 1$, $b = 2$, $g(n) = \text{с}$, а $n^{log_b a} = n^{log_2 1} = n^0 = \Theta(1)$. На этом основании воспользуемся вторым случаем теоремы. Поскольку

$$g(n) = \text{с} = \Theta\left(n^{log_b a}\right) = \Theta(1), \text{ то получаем}$$

асимптотическую оценку сложности

$$f(n) = \Theta(log_2 n).$$

4. Сортировка слиянием

Поскольку на шаге разделения задачи мы организуем деление на две подзадачи — массив разделяется на два подмассива, содержащих $\lfloor n/2 \rfloor$ и $\lceil n/2 \rceil$ элементов. Получаем $a = 2, b = 2$.

На шаге объединения решений выполняется слияние двух отсортированных фрагментов массивов, по которому получил название и сам алгоритм сортировки. Этот шаг может быть выполнен с линейной трудоемкостью, т.е. функция $g(n) = c_1 \cdot n + c_2 = \Theta(n)$.

Имеем $n^{log_b a} = n^{log_2 2} = n^1 = \Theta(n)$, и второй случай теоремы, из которого следует асимптотическая оценка трудоемкости (сложность) $\Theta(n \cdot ln\, n)$.

Более детально мы получаем $f(n) = \Theta(c_1 \cdot n \cdot log_2 n)$.

5. Умножение матриц

Мы организуем деление пополам линейного размера матрицы. Принимаем, что значение n является степенью двойки (как это ни странно, выгоднее добавить размерность до полной степени двойки, чем разбираться с матрицами, размер которых отличается на единицу), тогда на любом шаге деления значение $n/2$ является целым числом. Это приводит к необходимости решения восьми подзадач размерностью $n/2$.

По цепочке рекурсивных возвратов мы получаем восемь результирующих матриц, имеющих размер $n/2 \times n/2$ относительно размера текущей матрицы.

Основная формула (разделения и объединения решений) имеет вид

$$\begin{pmatrix} a & b \\ c & d \end{pmatrix} \times \begin{pmatrix} e & f \\ g & h \end{pmatrix} = \begin{pmatrix} r & s \\ t & u \end{pmatrix},$$

где подматрицы результата задаются уравнениями

$$r = a \times e + b \times g;$$

$$s = a \times f + b \times h;$$

$$t = c \times e + d \times g;$$

$$u = c \times f + d \times h.$$

В соответствии с этой формулой умножения нам необходимо выполнить сложение полученных матриц и заполнение соответствующих позиций матрицы результата.

Элементарный анализ этих шагов позволяет получить рекуррентное соотношение, задающее суммарное количество выполненных алгоритмом элементарных операций (умножения и сложения над числами — элементами матриц)

$$f(n) = 8 \cdot f(n/2) + \Theta(n^2).$$

Асимптотическая оценка решения этого рекуррентного соотношения.

$$n^{log_b a} = n^{log_2 8} = n^3 = \Theta(n^3),$$

и, в соответствии с первым случаем теоремы, мы получаем $f(n) = \Theta(n^3)$, что соответствует обычному алгоритму умножения матриц. Накладные затраты на рекурсию делают этот алгоритм неэффективным!

W. Strassen в 1969 предложил формулы, позволяющие получить результирующую матрицу за **7** умножений и 18 сложений. (https://ru.wikipedia.org/wiki/%D0%90%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC_%D0%A8%D1%82%D1%80%D0%B0%D1%81%D1%81%D0%B5%D0%BD%D0%B0). В соответствии с теоремой мы моментально получаем асимптотику этого алгоритма, лучшую, чем у классического алгоритма — $\Theta\left(n^{log_2 7}\right)$. Это был первый алгоритм умножения матриц, работающий быстрее куба. Отметим, что большая мультипликативная константа делает его неэффективным для матриц малой размерности!

6. Литература

1. Кормен Т., Лейзерсон Ч., Ривест Р., Штайн К. Алгоритмы: построение и анализ, 2-ое издание: Пер. с англ. — М.: Издательский дом «Вильямс», 2005. — 1296 с.
2. Jon L. Bentley, Dorothea Haken, and James B. Saxe. A general method for solving divide-and-conquer recurrences. SIGAT News, 12(3):36-44, 1980.