

### **3. Подход заключается в использовании общих решений определенных рекуррентных соотношений.**

Для них доказаны теоремы, позволяющие проводить анализ сложности двух наиболее типичных принципов организации рекурсии: рекурсия, организованная по принципу «разделяй и властвуй» и аддитивное уменьшение размерности задачи на некоторую константу. Использование этих теорем позволяет избежать утомительных расчетов и выбрать наименее трудоемкую схему организации рекурсии.

# Рекуррентные соотношения, характерные для двух основных принципов организации рекурсии

В общем виде значение функции сложности рекурсивного алгоритма вычисляется по формуле:

$$t(n) = \begin{cases} c, & \text{если } 0 \leq n \leq n_0, \\ t_s + t_r + t_u, & \text{если } n > n_0 \geq 0, \end{cases} \quad (1)$$

где  $n$  — параметр рекурсии;  $n_0$  — размер задачи, при котором время работы алгоритма не зависит от  $n$ ;  $c \geq 0$  — трудоемкость нерекурсивной ветви (некоторая постоянная величина);  $t_s$  — время сведения задачи к подзадачам;  $t_r$  — трудоемкость рекурсивной ветви (время выполнения подзадач);  $t_u$  — время объединения решений, полученных подзадачами.

Идея метода состоит в разделении задачи на части меньшей размерности  $n/k$ , где  $k > 1$ , получении решений для частей и объединение решений.

$$t(n) = \begin{cases} c, & \text{если } 0 \leq n \leq n_0, \\ A(n)t(n/k) + B(n), & \text{если } n > n_0 \geq 0. \end{cases} \quad (2)$$

Здесь  $A(n)$ ,  $B(n)$  — неотрицательные, монотонно возрастающие, вещественнозначные функции от  $n \in \mathbb{Z}_+$ , характеризующие затраты на рекурсивный переход. Поскольку  $k$  — постоянная величина, то преобразование  $g(n) = n/k$  линейно относительно  $n$ . Очевидно, что соотношение (2) однозначно определяет функцию  $t(n)$  только при  $n = 0$  и  $n = k^m$ ,  $m \in \mathbb{Z}_+$ .

- **Аддитивное уменьшение параметра рекурсии на константу**

$$t(n) = \begin{cases} c, & \text{если } 0 \leq n \leq n_0, \\ A(n)t(n-k) + B(n), & \text{если } n > n_0 \geq 0, \end{cases} \quad (3)$$

где  $A(n)$ ,  $B(n)$  имеют аналогичный смысл, что и в соотношении (2). В данном случае функция  $g(n) = n - k$  также задает линейное преобразование размерности исходной задачи в размерность подзадач, а рекуррентное соотношение (3) однозначно определяет функцию  $t(n)$  только при  $n = km$ ,  $m \in \mathbb{Z}_+$ .

Решить соотношения (2), (3) в общем виде не представляется возможным – слишком общий вид они имеют. Между тем в частном случае, когда  $A(n) = a$  и  $B(n) = bn^\tau$ , где  $a$ ,  $b$ ,  $\tau$  – положительные константы, вид решений соотношений (2), (3) определяют две основные теоремы. При этом константа  $a$  задает число подзадач, порождаемых рекурсивной ветвью алгоритма, а степенная функция  $bn^\tau$  определяет трудоемкость рекурсивного перехода.

**Теорема 1.** Пусть дано рекуррентное соотношение

$$t(n) = \begin{cases} c, & \text{если } n = 1, \\ at(n/k) + bn^\tau, & \text{если } n > 1, \end{cases}$$

где  $a > 0$ ,  $k > 1$  — целые константы,  $b \geq 0$ ,  $c \geq 0$ ,  $\tau \geq 0$  — вещественные константы. Тогда при  $n = k^m$ ,  $m \in \mathbb{Z}_+$  решением заданного соотношения является функция

$$t(k^m) = \begin{cases} a^m c + bk^{m\tau} m, & \text{если } a = k^\tau, \\ a^m c + bk^{m\tau} \frac{(a/k^\tau)^m - 1}{(a/k^\tau) - 1}, & \text{если } a \neq k^\tau. \end{cases}$$

**Следствие 1.** В предположениях теоремы 1 при больших значениях  $n$  и любых  $b > 0$  и  $c \geq 0$  справедливы оценки

$$t(n) = \begin{cases} O(n^\tau \log_k n), & \text{если } a = k^\tau, \\ O(n^\tau), & \text{если } a < k^\tau, \\ O(n^{\log_k a}), & \text{если } a > k^\tau. \end{cases}$$

Если  $b = 0$  и  $c > 0$ , то всегда

$$t(n) = O(a^{\log_k n}) = O(n^{\log_k a}).$$

Первая основная теорема выявляет ряд особенностей. Все оценки следствия 1 при любых значениях  $t(1) = c \geq 0$  дают полиномиальный порядок роста функции сложности  $t(n)$ . Стало быть, рекурсия, организованная по принципу "разделяй и властвуй", всегда приводит к полиномиальным алгоритмам. Имеет место явная зависимость сложности вычислений от числа подзадач и от их размера: чем более сбалансированным является разбиение задачи на подзадачи, тем лучшую оценку сложности имеет рекурсивный алгоритм.



**Теорема 2.** Пусть дано рекуррентное соотношение

$$t(n) = \begin{cases} c, & \text{если } 0 \leq n \leq k-1, \\ at(n-k) + bn^\tau, & \text{если } n \geq k, \end{cases} \quad (4)$$

где  $a > 0$ ,  $k \geq 1$  – целые константы,  $b \geq 0$ ,  $c \geq 0$ ,  $\tau \geq 0$  – вещественные константы. Тогда при  $n = kt$ ,  $t \in \mathbb{Z}_+$  верны неравенства

$$c + bk^{\tau-1}n \leq t(n) \leq c + \frac{b}{k}n^{\tau+1}, \quad \text{если } a = 1, \quad (5)$$

$$a^{n/k}c + bk^\tau \frac{a^{n/k} - 1}{a - 1} \leq t(n) \leq a^{n/k}c + bn^\tau \frac{a^{n/k} - 1}{a - 1}, \quad \text{если } a \neq 1. \quad (6)$$

**Следствие 2.** В предположениях теоремы 2 при  $\tau = 0$  решением рекуррентного соотношения (4) является функция

$$t(n) = \begin{cases} c + \frac{b}{k}n, & \text{если } a = 1, \\ a^{n/k}c + b \frac{a^{n/k} - 1}{a - 1}, & \text{если } a \neq 1. \end{cases} \quad (15)$$

**Следствие 3.** При  $\tau = 0$ , любых значениях  $c \geq 0$  и  $n \rightarrow \infty$  для решения рекуррентного соотношения (4) верны асимптотические оценки

$$t(n) = \begin{cases} O(n), & \text{если } a = 1, b > 0, \\ O(a^{n/k}), & \text{если } a \neq 1, b > 0. \end{cases} \quad (16)$$

В частности, при  $\tau = 0$ ,  $a = 1$  и  $b = 0$  всегда  $t(n) = O(1)$ .

**Следствие 4.** Для решения рекуррентного соотношения (4) при  $\tau \geq 0$ ,  $a = 1$ ,  $c \geq 0$ ,  $b > 0$  и  $n \rightarrow \infty$  справедлива асимптотическая оценка

$$t(n) = O(n^{\tau+1}). \quad (17)$$

**Следствие 5.** Для решения рекуррентного соотношения (4) при  $a > 1$ ,  $c \geq 0$ ,  $b > 0$  и целых положительных  $\tau$  справедлива оценка

$$t(n) = O(a^{n/k}). \quad (19)$$

Из (16), (17), (19) следует, что рекурсивные алгоритмы, образованные аддитивным уменьшением размера задачи на некоторую константу, могут быть полиномиальными или экспоненциальными. Так, при  $B(n) = bn^\tau$  (когда трудоемкость рекурсивного перехода описывается степенной функцией) алгоритм будет иметь экспоненциальную сложность всегда, если  $a \neq 1$ ,  $b > 0$  и  $c \geq 0$  (или  $b \geq 0$  и  $c > 0$ ).

Для реальных алгоритмов всегда существуют какие-то затраты на организацию рекурсии, т.е.  $b > 0$  и  $c > 0$ . Если эти затраты не зависят от  $n$ , то  $\tau = 0$ . При  $\tau = 0$ ,  $a = 1$  согласно следствию 3 рекурсивный алгоритм имеет линейную сложность. При  $\tau \geq 0$ ,  $a = 1$  и  $n \rightarrow \infty$  по следствию 4 для  $t(n)$  справедлива оценка  $t(n) = O(n^{\tau+1})$ . Таким образом, всегда при  $a = 1$  рекурсивный алгоритм, организованный путем уменьшения размера задачи на некоторую константу, является полиномиальным. По следствию 5 при  $a \neq 1$  и целых положительных значениях константы  $\tau$  верна оценка  $t(n) = O(a^{n/k})$ .

# Анализ ресурсной эффективности рекурсивных алгоритмов методом подсчета вершин дерева рекурсии

Строится **полное дерево рекурсии** узлами которого являются наборы фактических параметров при всех вызовах функции, начиная с первого обращения к ней, а ветви соединяют узлы, соответствующие взаимным вызовам. Корень полного дерева рекурсивных вызовов соответствует начальному обращению к функции.

*Основной особенностью анализа ресурсной эффективности рекурсивных алгоритмов является необходимость учета дополнительных затрат памяти и трудоемкости, связанных с механизмом организации рекурсии в принятой модели вычислений.*

Трудоёмкость рекурсивных реализаций алгоритмов связана с количеством операций рекурсивных вызовов и возвратов, выполняемых при одном рекурсивном обращении, а также с количеством таких обращений. При вызове функции в стек помещается адрес возврата, состояние необходимых регистров процессора, состояние локальных ячеек вызывающей функции, адреса возвращаемых значений и передаваемые параметры.

Введем следующие обозначения:  $p$  — количество передаваемых фактических параметров,  $r$  — количество сохраняемых в стеке регистров,  $k$  — количество возвращаемых по адресной ссылке значений,  $l$  — количество локальных ячеек процедуры.

Трудоёмкость, связанная с обслуживанием одного вызова и одного возврата, обозначается через  $F_{c/b} = 2 \cdot (p+r+k+l+1)$ . Где дополнительная единица учитывает операции с адресом возврата.



## **Анализ дерева рекурсии.**

- Важной характеристикой рекурсивного алгоритма является **глубина рекурсивных вызовов** – наибольшее одновременное количество рекурсивных обращений функции, определяющее максимальное количество слоев рекурсивного стека, в котором осуществляется хранение отложенных вычислений.
- **Объем рекурсии** - это одна из характеристик сложности рекурсивных вычислений для конкретного набора параметров, представляющая собой количество вершин полного рекурсивного дерева без единицы.

Будем использовать следующие обозначения для конкретного входного параметра  $N$ :

- $R(N)$  – общее число вершин дерева рекурсии,
- $R_V(N)$  – объем рекурсии без листьев (внутренние вершины),
- $R_L(N)$  – количество листьев дерева рекурсии,
- $H_R(N)$  – глубина рекурсии.

Очевидно, что  $R(N) = R_V(N) + R_L(N)$ ,  $H_R(N) \leq R_V(N) + R_L(N)$ .

***Требуемый объем памяти*** в области программного стека определяется не общим количеством вершин порождённого дерева рекурсии, ***а максимальной глубиной его листьев.***

$V(N) = H_R(N) * (p+r+k+l+1) * u_\beta$  – **оценка требуемой памяти**, где  $N$  – вход,  $u_\beta$  – длина слова в байтах.

## Анализ трудоемкости методом подсчета вершин дерева рекурсии.

В отличие от оценки объема памяти, которая зависит от максимальной глубины рекурсивного дерева, для функции трудоемкости количество операций со стеком на один вызов/возврат  $F_{c/b}$  должно быть учтено для всех вершин рекурсивного дерева.

**Метод получения ресурсных функций для рекурсивных алгоритмов на основе анализа порожденного дерева рекурсии** заключается в определении ресурсных затрат в каждой вершине дерева и их суммировании.

Таким образом, основная задача при использовании этого метода состоит в теоретическом построении функций  $R_V(N)$ ,  $R_L(N)$  и  $H_R(N)$  — как функций от характеристик множества входных данных.

Для построения ресурсных функций рекурсивных алгоритмов необходимо учесть ряд особенностей рекурсивной реализации, а именно:

- ресурсные затраты на обслуживание рекурсивных вызовов-возвратов, передачу параметров и возврат значений рекурсивных функций (**ресурсные затраты обслуживания рекурсии**);
- специфику фрагмента останова рекурсии, приводящую к необходимости **отдельного учета ресурсных затрат в листьях дерева рекурсии**.

Трудоёмкость алгоритма  $A$  на конкретном входе  $N$  —  $F_A(N)$  определяется трудоёмкостью обслуживания дерева рекурсии, зависящей от общего количества его вершин, и трудоёмкостью продуктивных вычислений, выполненных во всех вершинах дерева рекурсии.



Обозначим через  $F_R(N)$  — трудоемкость порождения и обслуживания дерева рекурсии,  $F_C(N)$  — трудоемкость продуктивных вычислений алгоритма, тогда трудоемкость всего алгоритма:

$$F_A(N) = F_R(N) + F_C(N) (*)$$

Трудоемкость обслуживания дерева рекурсии:

$$F_R(N) = R(N) * F_{c/b}$$

При подсчете трудоемкости продуктивных вычислений необходимо учесть, что для листьев рекурсивного дерева трудоемкость отлична от трудоемкости во внутренних вершинах.

Пусть  $F_{CV}(N)$  — трудоемкость продуктивных вычислений (обработки данных) во внутренних вершинах,  $F_{CL}(N)$  — трудоемкость вычислений в листьях дерева рекурсии, тогда  $F_C(N) = F_{CV}(N) + F_{CL}(N)$ .

Пусть  $F_{cl}(1)$  трудоемкость алгоритма в одном листе порожденного дерева (как правило выражается фиксированным числом базовых операций). Зная количество листьев порожденного дерева рекурсии, можно определить  $F_{cl}(N) = F_{cl}(1) * R_L(N)$ .

Во внутренних вершинах дерева выполняются некоторые действия, связанные с подготовкой параметров для следующих рекурсивных вызовов и обработкой возвращаемых результатов. Трудоемкость такой обработки может зависеть как от обрабатываемых в этой вершине данных, так и от положения вершины в дереве рекурсии. С целью учета этой зависимости, введем

Число уровней внутренних вершин в дереве на единицу меньше глубины рекурсии. Пусть  $m$  номер уровня вершины  $m = \overline{1, H_R(N) - 1}$ , а  $k$  — номер вершины на данном уровне  $k = \overline{1, K(m)}$ , где  $K(m)$  — количество внутренних вершин на уровне  $m$ .

Неполное дерево на уровне  $k$  может содержать как внутренние вершины, так и листья. С учетом такой нумерации обозначим вершины дерева через  $v_{mk}$

$$\sum_{m=1}^{H_R(N)-1} \sum_{k=1}^{K(m)} v_{km} = R_v(N)$$

Обозначим трудоемкость продуктивных вычислений в вершине  $v_{km}$  через  $F_{CV}(v_{km})$ , тогда формула для трудоемкости продуктивных вычислений во внутренних вершинах дерева рекурсии имеет вид

$$F_{CV}(N) = \sum_{m=1}^{H_R(N)-1} \sum_{k=1}^{K(m)} F_{CV}(v_{km})$$

Заметим, что в случае, когда значения функции  $F_{CV}(v_{km})$  не зависят от номера вершины дерева рекурсии, т. е. трудоемкость продуктивных вычислений в вершинах не зависит от данных, то, обозначая трудоемкость продуктивных вычислений для любой внутренней вершины дерева через  $F_{CV}(1)$ , имеем  $F_{CV}(N) = R_v(N) * F_{CV}(1)$ .

В общем случае, при котором трудоемкость продуктивных вычислений различна во внутренних вершинах дерева:

$$F_A(N) = R(N) * F_{c/b} + F_{CL}(1) * R_L(N) + \sum_{m=1}^{H_R(N)-1} \sum_{k=1}^{K(m)} F_{CV}(v_{km})$$

В частном случае, когда трудоемкость продуктивных вычислений для любой внутренней вершины дерева рекурсии одинакова:

$$F_A(N) = R(N) * F_{c/b} + F_{CL}(1) * R_L(N) + R_v(N) * F_{CV}(1)$$



Для анализа дерева рекурсии дополнительный интерес представляют такие характеристики как отношение количества листьев к общему количеству вершин рекурсивного дерева и доля операций обслуживания дерева рекурсии.

Характеристика относительной «ширины» нижнего уровня (уровня листьев) в дереве рекурсии.

$$B_L(N) = \frac{R_L(N)}{R(N)}, 0 < B_L(N) < 1$$

Значение  $B_L(N)$  будет минимально для цепочки (унарного дерева), и будет возрастать при увеличении числа вершин, порожденных во внутренних вершинах дерева рекурсии.

Доля операций обслуживания дерева рекурсии.

$$B_R(N) = \frac{F_R(N)}{F_A(N)}, 0 < B_R(N) < 1$$

Значение  $B_R(N)$  показывает, насколько трудоемкость обслуживания дерева рекурсии значима в общей трудоемкости рекурсивного алгоритма.



## Этапы анализа трудоемкости рекурсивного алгоритма методом анализа порожденного дерева рекурсии:

1. Анализ порождаемого данным алгоритмом дерева рекурсии с целью получения теоретических зависимостей для характеристик дерева как функций от длины входа  $N$  и/или характеристических особенностей множества входных данных.
2. Определение трудоемкости обслуживания рекурсии на один вызов-возврат  $F_{c/b}$ .
3. Определение трудоемкости алгоритма при останове рекурсии  $F_{cl}(1)$ .
4. Исследование трудоемкости продуктивных вычислений во внутренних вершинах дерева рекурсии .
5. Получение функции трудоемкости рекурсивного