Европейски съюз

ОП “Развитие на човешките ресурси”
Договор BG051PO001-4.3.04-0018
Разработване на програми за електронни форми на дистанционно обучение във Факултета по математика и информатика

Европейски социален фонд

Четвърта лекция по ДАА на втори поток КН
Рекурсивни Алгоритми, алгоритмична схема Разделяй-и-Владей и Рекурентни Отношения

24 март 2014

Абстракт

Въвеждаме понятията рекурсивен алгоритъм и рекурентно отношение. Разискваме най-обща класификация на рекурентните отношения. Разглеждаме пет начина за решаване на рекурентни отношения: чрез развиване, чрез дървото на рекурсията, по индукция, чрез Мастър теоремата и чрез характеристични уравнения.

Съдържание

1 Рекурсивни алгоритми

Думата *рекурсия* идва от латинския глагол *curro*[†], което означава “тичам”, и префикса *re-*, който означава “назад” или “отново”. *Recurro* означава “тичам назад”, откъдето *рекурсивен* буквално означава “тичащ назад”.

Рекурсивен алгоритъм е алгоритъм, който вика себе си. Очевидно, ако детерминиран алгоритъм извика себе си с точно същия вход, с който е бил извикан, той ще неизбежно ще вика себе си отново и отново със същия вход до безкрай, така че викане със същия вход е недопустимо. По правило рекурсивните алгоритми са реализация на *алгоритмичната схема* Разделяй-и-Владей. Алгоритмична схема е най-общ шаблон, по който са конструирани множество алгоритми за множество задачи. В този курс ще разгледаме три алгоритмични схеми: Разделяй-и-Владей, Алчна схема и Динамично Програмиране. Засега ще се спрем на схемата Разделяй-и-Владей. При нея алгоритъмът се състои от три фази:

[†]Откъдето идва, примерно, и думата “куриер”.

- **Разделяй** – входът се разделя[†] на части, всяка от които (очевидно) е по-малка от целия вход.
- **Владей** – решаваме задачата върху всяка от по-малките части и получаваме резултат за нея.
- **Комбинирай** – използваме тези резултати, за да генерираме решение за задачата върху първоначалния вход.

Очевидно първата фаза на разделянето става само докато размерът на входа не стане достатъчно малък, защото няма как да разделяме дискретен вход на по-малки части неограничено, без да зациклим. Следователно, по-правилно е да се каже, че разделянето се случва ако и само ако размерът на входът надхвърля еди-колко; в противен случай, генерираме резултат по друг начин, примерно с пълно изчерпване[‡]. Терминът "Разделяй-и-Владей" (на английски, Divide and Conquer) идва от латинската фраза *Divide et impera*. Според наложилия се фолклор, тази максима е била ръководен принцип на древния Рим (и републиката, и империята), който се е справял с даден силен противник, като първо е създавал вътрешни разделения в него, а после е побеждавал всеки от получените фрагменти последователно. Както ще видим в многобройни алгоритмични примери, някои изчислителни задачи са податливи на ефикасно алгоритмично решение, което първо разделя входа на части, стига той да е достатъчно голям, решава задачата върху всяка от частите, и накрая комбинира решенията. За съжаление, този подход съвсем не е универсален.

> *Както ще видим в следваща лекция, алгоритмичната схема Динамично Програмиране също е свързана с рекурсия, но при онези алгоритми акцентът е върху ефикасното пресмятане, което означава умело използване на вече получени резултати на общи подподзадачи.*

Ако разглеждаме произволен рекурсивен алгоритъм като решение, изградено по схемата Разделяй-и-Владей, е ясно, че

- рекурсивният алгоритъм вика себе си винаги върху **входове с по-малък размер** от първоначалния, ако първоначалният размер е достатъчно голям, но
- ако размерът на входа е по-малък от някаква гранична стойност, алгоритъмът не вика себе си, а изчислява изхода директно и го връща.

Типичен пример за рекурсивен алгоритъм е следният алгоритъм, пресмятащ факториел на число.

```
FACTORIAL(n ∈ ℕ)
1  if n = 0
2      return 1
3  else
4      return n × FACTORIAL(n − 1)
```

Да разгледаме работата му върху някакъв малък вход, примерно 5.

- FACTORIAL(5) проверява условието на ред 1 и понеже то е ЛЪЖА, на ред 4 изпълнява FACTORIAL(4).
- FACTORIAL(4) проверява условието на ред 1 и понеже то е ЛЪЖА, на ред 4 изпълнява FACTORIAL(3).
- FACTORIAL(3) проверява условието на ред 1 и понеже то е ЛЪЖА, на ред 4 изпълнява FACTORIAL(2).
- FACTORIAL(2) проверява условието на ред 1 и понеже то е ЛЪЖА, на ред 4 изпълнява FACTORIAL(1).
- FACTORIAL(1) проверява условието на ред 1 и понеже то е ЛЪЖА, на ред 4 изпълнява FACTORIAL(0).
- FACTORIAL(0) проверява условието на ред 1 и понеже то е ИСТИНА, на ред 2 връща стойност 1 на FACTORIAL(1).
- FACTORIAL(1) получава стойност 1 от викането на FACTORIAL(0), след което умножава 1×1 и връща резултата, който е 1, на FACTORIAL(2).

[†]Терминът "разбива" не е подходящ, защото общоприетата дефиниция на "разбива" означава, на части, които не се припокриват. При алгоритмите Разделяй-и-Владей не е задължително частите на входа да не се припокриват, просто всяка трябва да е по-малка от целия вход.

[‡]Още известен като "брутална сила".

- Factorial(2) получава стойност 1 от викането на Factorial(1), след което умножава 2×1 и връща резултата, който е 2, на Factorial(3).
- Factorial(3) получава стойност 2 от викането на Factorial(2), след което умножава 3×2 и връща резултата, който е 6, на Factorial(4).
- Factorial(4) получава стойност 6 от викането на Factorial(3), след което умножава 4×6 и връща резултата, който е 24, на Factorial(5).
- Factorial(5) получава стойност 24 от викането на Factorial(4), след което умножава 5×4 и връща 120.

Работата на Factorial(5) се състои от два етапа. Първият етап можем да наречем, *движение надолу*. През него входът намалява, докато не бъде достигната стойността 0, определена от ред 1. Тази стойност можем да наречем, *спирачката на рекурсията*. Вторият етап да наречем, *движение нагоре*. Същинското изчисляване на факториела става във втория етап. По отношение на софтуерна имплементация на алгоритъма, първият етап е само попълване на подходяща структура данни, а именно стек, с подходящи стойности. Името "рекурсия", което, както казахме, буквално означава "тичане назад", без съмнение се дължи на първия етап – движението надолу.

2 Рекурентни отношения

2.1 Рекурентни отношения по принцип

2.1.1 Определение на рекурентно отношение

Рекурентно отношение е уравнение, в което вляво от знака за равенство има символ за функция, примерно $T(n)$ или a_k, а вдясно има математически израз, който включва една или повече появи на същия символ за функция, но с по-малки стойности на променливата. Алтернативна дефиниция е, уравнение, което изразява общия член на някаква редица от числа чрез някои или всички предишни членове. Освен това, за една или повече *начални стойности* на променливата, има други уравнения, наречени *начални условия*, които определят стойността на въпросната функция (върху началните стойности). Типичен пример е рекурентното отношение на числата на Фибоначи:

$$\begin{aligned} F_0 &= 0 \\ F_1 &= 1 \\ F_n &= F_{n-1} + F_{n-2} \quad \text{за } n \geqslant 2 \end{aligned} \tag{1}$$

Еквивалентен запис е:

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

Добре дефинирано рекурентно отношение ще наричаме рекурентно отношение, което определя коректен (рекурсивен) алгоритъм, който пресмята безкрайната числова редица[†] $a_0, a_1, a_2, \ldots$. На читателя остава да съобрази, че следното **не е** добре дефинирано рекурентно отношение заради "недостиг" на начални условия:

$$\begin{aligned} T(0) &= 1 \\ T(n) &= 2T(n-3) \quad \text{за } n \geqslant 1 \end{aligned}$$

Въпросът дали дадено рекурентно отношение е добре дефинирано, дали началните условия са достатъчни и смислени, може да е нетривиален. Следното рекурентно отношение не е добре дефинирано

$$\begin{aligned} T(1) &= 1 \\ T(n) &= 2T\left(\left\lfloor \frac{n}{2} + 17 \right\rfloor\right) + n \end{aligned}$$

[†] Допускаме, че функционалният символ е "a" и че най-малката начална стойност на променливата е 0.

поради това, че трансформацията $n \mapsto \left\lfloor \frac{n}{2} + 17 \right\rfloor$[†], започвайки от произволно естествено число n, се "установява" на $n = 34$,[‡] така че началната стойност 1 не е достижима от никое число освен самото 1. С други думи, началното условие $T(1) = 1$ в случая е безполезно; ако напишем съответната програма като реален софтуер, тя ще "гърми"[§] за всички естествени числа без 1.

И така, всяко добре дефинирано рекурентно отношение определя еднозначно числова редица. Отношението между рекурентни отношения и числови редици не е биективно: една и съща числова редица може да се определя от много рекурентни отношения. Рекурентни отношения, които определят една и съща числова редица, ще наричаме *еквивалентни*. Убедете се сами, че рекурентното отношение (2) е еквивалентно на (1), защото и на двете съответства редицата на Фибоначи 0, 1, 1, 2, 3, 5, 8, ...

$$\begin{aligned} T(0) &= 0 \\ T(1) &= 1 \\ T(2) &= 1 \\ T(n) &= 2T(n-1) - T(n-3) \qquad \text{за } n \geqslant 3 \end{aligned} \tag{2}$$

Подчертаваме, че не трябва да се бъркат двете понятия

- рекурентно отношение, което е понятие от синтактичното ниво
- функцията, която се реализира от някакво рекурентно отношение – това е понятие от семантичното ниво. Числовата редица, която съответства на рекурентното отношение, е практически същото нещо като функцията, която то реализира.

2.1.2 Обобщения на рекурентно отношение

Има различни обощения на рекурентно отношение спрямо изложението, което правим тук. Рекурентните отношения могат да бъдат на повече от една променлива, примерно биномният коефициент е рекурентно отношение на две променливи:

$$\binom{n}{0} = 1, \quad \binom{n}{n} = 1, \quad \binom{n}{k} = \binom{n-1}{k} + \binom{n-1}{k-1}$$

Може да бъдат дадени няколко взаимно зависими рекурентни отношения – това е проявление на *взаимна рекурсия*. Пример за такива рекурентни отношения има в [KL12, стр. 9] (за краткост изпускаме началните условия):

$$\begin{aligned} T(n) &= T\left(\left\lceil \frac{n}{2} \right\rceil\right) + T\left(\left\lfloor \frac{n}{2} \right\rfloor + 1\right) + 2T_1\left(\left\lceil \frac{n}{2} \right\rceil - 1\right) + 2T_1\left(\left\lfloor \frac{n}{2} \right\rfloor\right) + 2 \\ T_1(n) &= T\left(\left\lfloor \frac{n}{2} + 1 \right\rfloor\right) + T_1\left(\left\lceil \frac{n}{2} \right\rceil\right) + 2T_1\left(\left\lfloor \frac{n}{2} \right\rfloor\right) + 2T_2\left(\left\lceil \frac{n}{2} \right\rceil - 1\right) + 2 \\ T_2(n) &= T_1\left(\left\lceil \frac{n}{2} \right\rceil\right) + T_1\left(\left\lfloor \frac{n}{2} \right\rfloor + 1\right) + 2T_2\left(\left\lceil \frac{n}{2} \right\rceil - 1\right) + 2T_2\left(\left\lfloor \frac{n}{2} \right\rfloor\right) + 2 \end{aligned}$$

Друг пример за взаимна рекурентност има на тази страница на специализирано списания за числа на Фибоначи, където е дадена задачата, да се докаже, че H_{n-1} и G_n са последователни числа на Фибоначи $\forall n \geqslant 1$, ако F_n са числата на Фибоначи:

$$\begin{aligned} G_1 &= 1 \\ H_0 &= 0 \\ G_n &= F_{n+1}G_{n-1} + F_nH_{n-2}, \quad n \geqslant 2 \\ H_n &= F_{n+1}G_n + F_nH_{n-1}, \quad n \geqslant 1 \end{aligned}$$

[†] Чете се, "n се изобразява в в $\left\lfloor \frac{n}{2} + 17 \right\rfloor$".
[‡] Казваме, че 34 е *фиксирана точка* за тази трансформация.
[§] Става дума за препълване на стека, който операционната система осигурява на съответния процес.

Може да се дефинират рекурентни отношения върху рационални или реални числа. Досега допускахме, че стойността на функцията е естествено число, но това не е задължително. Примерно (вж. [SF96, стр. 56]), рекурентното отношение

$$a_0 = 1$$
$$a_n = \frac{1}{1 + a_{n-1}}$$

определя редицата от дроби $1, \frac{1}{2}, \frac{2}{3}, \frac{3}{5}, \frac{5}{8}, \ldots$, която очевидно има връзка с числата на Фибоначи. Пак в книгата на Flajolet и Sedgewick се посочва (вж. [SF96, стр. 46]), че методът на Newton за пресмятане на квадратен корен от дадено положително реално β е рекурентно отношение с $a_0 = 1$:

$$a_n = \frac{1}{2}\left(a_{n-1} + \frac{\beta}{a_{n-1}}\right), \quad n > 0$$

2.1.3 Рекурентни отношения и рекурсивни алгоритми

Отношението между рекурентните отношения и рекурсивните алгоритми е двояко. От една страна, всяко рекурентно отношение **е** рекурсивен алгоритъм, който за всеки вход-число връща изход-число след пресмятане, което включва рекурсия (викане на себе си върху по-малко число). От друга страна, сложността на всеки[†] рекурсивен алгоритъм се описва от рекурентно отношение. Примерно, сложността на MERGESORT, който ще разгледаме в следваща лекция, се описва от рекурентното отношение (както ще видим нататък, началните условия се пропускат, защото ни интересува само асимптотиката на решението):

$$T(n) = T\left(\left\lfloor \frac{n}{2} \right\rfloor\right) + T\left(\left\lceil \frac{n}{2} \right\rceil\right) + n \tag{3}$$

При изследването на сложността на рекурсивни алгоритми чрез рекурентни отношения е важно да се прави разлика между алгоритъма, чиято сложност изследваме, и рекурентното отношение, което описва тази сложност. На свой ред то също е рекурсивен алгоритъм, но е **съвършено различен обект** от първоначалния алгоритъм. Като пример да разгледаме рекурентното отношение на факториела

$$T(n) = \begin{cases} 1, & \text{ако } n = 0 \\ nT(n-1), & \text{в противен случай} \end{cases} \tag{4}$$

където $n \in \mathbb{N}$. Рекурентно отношение (4) е практически същото нещо като алгоритъм FACTORIAL на стр. 2. Но (4) е съвършено различно нещо от алгоритъм, да го наречем X, чиято сложност по време се описва от (4). X е рекурсивен алгоритъм, който при вход с големина n, за достатъчно голямо n, прави n на брой рекурсивни викания, всяко от които е върху вход с големина $n - 1$.[‡] Сложността на X е съизмерима с $n!$, така че това е един изключително бавен алгоритъм. Докато сложността на (4) е линейна като функция на n.[§]

Да повторим: рекурентното отношение също е рекурсивен алгоритъм, но той е съвършено различен от алгоритъма, чиято сложност описва.

2.1.4 Класифициране на рекурентните отношения

Тук ще игнорираме началните условия и ще дискутираме "същинското" рекурентно отношение. Рекурентните отношения може да се класифицират по много признаци.

[†] "Всеки" е твърде силно казано, но поне по отношение на простите рекурсивни алгоритми, които ще разгледаме, сложността се описва от рекурентни отношения.

[‡] Говорейки педантично, няма алгоритъм, чиято сложност се описва от (4), защото освен тези рекурсивни викания алгоритъмът трябва да върши поне константна допълнителна работа, така че реалистичен израз за сложността по време всъщност е

$$T(n) = \begin{cases} 1, & \text{ако } n = 0 \\ nT(n-1) + 1, & \text{в противен случай} \end{cases} \tag{5}$$

Но това $+1$ в (5) не се отразява на асимптотиката на $T(n)$. И в (5), и в (4), $T(n) \asymp n!$.

[§] Каква е сложността на (4) като **функция на големината на входа**? Както казахме в предна лекция, това зависи от дефиницията на "големина на входа". Ако приемем, че всяко n има големина единица, сложността на (4) не е определена.

- **Брой на появите на функционалния символ вдясно.** Функционалният символ може да се появява точно един път вдясно, примерно

$$T(n) = 2T(n-1) + 1 \tag{6}$$

а може да се появява много пъти с различни стойности на аргумента си, примерно

$$T(n) = T(n-1) + T(n-3) + T(n-5) \tag{7}$$

или

$$T(n) = T(n-1) + T(n-2) + \ldots + T(1) + T(0) \tag{8}$$

Забележка: в следното рекурентно отношение, появите вдясно са една, а не две:

$$T(n) = T(n-1) + T(n-1) + 1$$

Тривиално е да се съобрази, че това рекурентно отношение всъщност е (6). Следователно, броят на появите вдясно е по отношение на различни аргументи, както е в (7).

Класификацията по брой появи вдясно не е от особена важност, но обикновено рекурентните отношения с една поява вдясно се решават по-лесно в сравнение с тези с няколко появи. Мастър теоремата, за която ще говорим след малко, е приложима само за рекурентни отношения с една поява вдясно.

- **Каква е функцията на намаляването** За всяка поява на функционалния символ вдясно, *функцията на намаляването*, неформално казано, е тази математическа операция, която е записана там. Примерно, в (4), в (6), в (7) и в (8), функцията на намаляването е изваждане. Функцията на намаляването може да е деление:

$$T(n) = 2T\left(\left\lfloor \frac{n}{3} \right\rfloor\right) + 4T\left(\left\lfloor \frac{n}{7} \right\rfloor\right) + 1$$

или коренуване:

$$T(n) = nT(\lfloor \sqrt{n} \rfloor) + 1$$

или логаритмуване

$$T(n) = T(\lfloor \lg n \rfloor) + 1$$

Възможностите са неограничени. Може да има и "смесен" вид намаляване, примерно:

$$T(n) = T(n-2) + T\left(\left\lfloor \frac{n}{3} \right\rfloor\right) + 3T(\lfloor \sqrt{n} \rfloor) + 50T(\lfloor \lg n \rfloor) - 10$$

При изследването на сложността на "естествени" алгоритми, смесено намаляване не се среща: функцията на намаляването е или изваждане, или деление. При алгоритмите, изградени по схемата Разделяй-и-Владей, сложността се описва с рекурентни отношения, при които функцията на намаляването е деление. Намаляване чрез изваждане се среща често (макар и не само там) в рекурентните отношения, описващи сложността на алгоритми, изградени на принципа на пълното изчерпване на възможностите.

Следните определения и класификации са в сила **само за рекурентни отношения, при които функцията на намаляването е изваждане**.

- *Ред* (на английски, *order*) на рекурентното отношение е най-голямото число, което се изважда от n вдясно. В следните три примера, числото, което определя реда на отношението, е в червено. И така, това рекурентно отношение

$$T(n) = 2T(n-1) + 1$$

е от първи ред, това

$$T(n) = 3T(n-1) + 4T(n-2) + n$$

е от втори ред, а това

$$T(n) = T(n-1) + T(n-2) + \ldots + T(n-k) + n^2$$

е от k-ти ред. Прието е рекурентно отношение от k-ти ред, в което k е константа, да се нарича *рекурентно отношение с крайна история*. Рекурентно отношение, в чиято дясна страна функциите имат аргументи от $n-1$ чак до началната стойност, примерно

$$T(n) = T(n-1) + T(n-2) + \ldots + T(1) + T(0)$$

се нарича *рекурентно отношение с пълна история*[†].

Подчертаваме, че редът на рекурентното отношение **не е** същото като броят на появите на функционалния символ вдясно. Примерно, следното рекурентно отношение

$$T(n) = T(n-2) + 2T(n-5) + 1$$

е от пети ред, мака да има само две появи вдясно[‡].

- *Линейно рекурентно отношение* е рекурентно отношение, в което дясната страна е линейна функция на функцията на отношението. Примерно, следните три рекурентни отношения са линейни:

$$A(n) = A(n-1) + 1$$
$$B(n) = nB(n-2) + n^2B(n-4) + \left\lfloor \frac{n \sin n}{\sqrt{n^3 + \lg n}} \right\rfloor$$
$$C(n) = \sum_{k=0}^{n-1} 2^k C(k)$$

докато тези четири **не са** линейни:

$$A(n) = A(n-1)A(n-2) + 1$$
$$B(n) = \left\lceil \frac{1}{1 + B(n-1)} \right\rceil$$
$$C(n) = \lfloor \sqrt{C(n-2)} \rfloor$$
$$D(n) = \lceil \lg (D(n-1) + D(n-2)) \rceil$$

Във всички примери, появите на съответните функции на отношението вдясно са оцветени в червено.

- Различаваме рекурентни отношение с константни коефициенти, примерно

$$P(n) = 1P(n-1)P(n-2) + 3P(n-3)$$
$$S(n) = \left\lfloor \sum_{k=1}^{\infty} \frac{1}{k^2} \right\rfloor S(n-2) + n$$
$$T(n) = 2T(n-1) + 5T(n-3)$$

от рекурентни отношения с променливи коефициенти, примерно

$$P(n) = \lfloor \lg n \rfloor P(n-1)$$
$$S(n) = nS(n-1)$$
$$T(n) = \left\lceil \frac{2 + n^2}{3 + n} \right\rceil T(n-2)$$

В тези примери, в червено са коефициентите.

[†] Ако допуснем k да не е константа, а някаква нарастваща функция на n, то за всяко рекурентно отношение може да твърдим, че функцията на намаляването е изваждане. Примерно, $T(n) = T\left(\left\lfloor \frac{n}{2} \right\rfloor\right) + 1$ може да се запише като $T(n) = T\left(n - \left\lceil \frac{n}{2} \right\rceil\right) + 1$; $S(n) = S(\lfloor \sqrt{n} \rfloor) + 1$ може да се запише като $S(n) = S(n - (n - \lfloor \sqrt{n} \rfloor)) + 1$, и така нататък. От това съображение следва, че класифицирането на рекурентните отношения по функцията на намаляването е—говорейки теоретично—безсмислено, защото, в някакъв смисъл, тя винаги е изваждане. Но на практика това класифициране е доста удобно. И така, рекурентните отношения, в които функцията на намаляването е изваждане, са или с константен ред, или с пълна история.

[‡] Можем да мислим, че $T(n) = 0T(n-1) + 1T(n-2) + 0T(n-3) + 0T(n-4) + 2T(n-5) + 1$

- Различаваме *хомогенни* и *нехомогенни* линейни рекурентни отношения с крайна история и константни коефициенти. Хомогенните са от вида

$$T(n) = c_1T(n-1) + c_2T(n-2) + \ldots + c_kT(n-k) \tag{9}$$

където k и $c_1, \ldots, c_k$ са константи. Нехомогенните са от вида

$$T(n) = c_1T(n-1) + c_2T(n-2) + \ldots + c_kT(n-k) + f(n) \tag{10}$$

където k и $c_1, \ldots, c_k$ са константи, където $f(n)$ е функция, която зависи от n и не зависи от $T(n)$. Казваме, че $f(n)$ е *нехомогенната част*.

2.1.5 Решение на рекурентно отношение

Нека е дадено някакво рекурентно отношение, в което функционалният символ е $T(n)$. Както стана ясно, то реализира някаква функция; иначе казано, определя някаква числова редица. *Да решим* това рекурентно отношение означава да намерим друг, еквивалентен израз (формула) за същата функция, който обаче е затворена формула. Примерно, ако рекурентното отношение е

$$\begin{aligned} T(0) &= 0 \\ T(n) &= T(n-1) + n \text{ за } n > 0 \end{aligned} \tag{11}$$

то *решение* е $T(n) = \frac{n(n+1)}{2}$. Какво точно е затворена формула зависи от това, какви операции са допустими. Решението от примера е валидно, ако можем да ползваме събиране с единица, умножение и деление на две. Да разгледаме отново рекурентното отношение (4). Както е добре известно, решението е $T(n) = n!$, но това решение е допустимо само ако можем да ползваме факториела – ако са разрешени само събиране, изваждане, умножение, деление, коренуване и степенуване, то (4) няма решение[†].

Тук ще допускаме, че всички "нормални" функции като изброените, плюс логаритмуването и факториелът, са допустими и че решение, което ги ползва, е валидно. Също така, решението може да съдържа биномни коефициенти, числа на Стирлинг от втори род, суми и произведения. Важното е да няма рекурсия – функционалният символ не трябва да се среща вдясно.

Ползите от това, да решим дадено рекурентно отношение, са поне две.

- По правило решението е по-бърз начин за изчисляване[‡] на функцията, която съответства на рекурентното отношение. Примерно, $\frac{n(n+1)}{2}$ е по-бърз алгоритъм за решаване на функцията, която рекурентното отношение (11) реализира в сравнение със самото (11). Ако $n = 1\,000\,000\,000$, директното изпълнение на (11) ще се извърши в около $2\,000\,000\,000$ стъпки[§], докато $\frac{1\,000\,000\,000(1\,000\,000\,000+1)}{2}$ се изчислява с една инкрементация, едно умножение и едно деление на две.

- Ако решението е достатъчно просто, много бързо можем да преценим каква е асимптотиката на нарастването на $T(n)$. Примерно, в (11) имаме $T(n) = \Theta(n^2)$, което се вижда веднага от решението $T(n) = \frac{n(n+1)}{2}$. По отношение на числата на Фибоначи, от (1) е трудно да се прецени каква е асимптотиката на F_n, докато от добре известното решение

$$F_n = \frac{1}{\sqrt{5}}\left(\frac{1+\sqrt{5}}{2}\right)^n - \frac{1}{\sqrt{5}}\left(\frac{1-\sqrt{5}}{2}\right)^n \tag{12}$$

веднага се вижда, че $F_n \asymp \left(\frac{1+\sqrt{5}}{2}\right)^n$, тоест, горе-долу, $F_n \approx 1.6^n$, понеже $\left(\frac{1-\sqrt{5}}{2}\right)$ е по-малко от единица по абсолютна стойност.

Методите за решаване на рекурентни отношения ще разискваме в детайли нататък. Тук само ще спомена, че универсален **формален** метод за решаване на рекурентни отношения няма.

[†]При изброените ограничения, $T(n) = \prod_{i=1}^{n} i$ или $T(n) = n \cdot (n-1) \cdots 2 \cdot 1$ не са решения, защото не са затворени формули. Забележете, че тези два израза използват нотации, касаещи умножение, но не са затворени формули, ако символът "$\prod$" или многоточието не са разрешени.

[‡]С други думи, по-бърз алгоритъм. Както казахме, самото рекурентно отношение е алгоритъм, който пресмята число. Решението е алгоритъм, който изчислява същата задача, но по-бързо.

[§]Един милиард за движението надолу до достигане на началното условие и още един милиард за движението нагоре, от началното условие до $T(1\,000\,000\,000)$.

2.2 Рекурентни отношения в изследването на алгоритми

Рекурентните отношения се използват широко в теорията на алгоритмите, защото сложността на даден рекурсивен алгоритъм се описва с подходящо рекурентно отношение. Правилото за съставяне на подходящо рекурентно отношение е просто: лявата страна е, да речем, $T(n)$ и изразява работата на алгоритъма върху вход с размер n, а дясната страна има по едно $T(\,)$ за всяко рекурсивно викане, като аргументът е функцията, с която намалява входа при даденото рекурсивно викане, плюс израз, който отразява работата на алгоритъма извън рекурсивните викания. Ще видим достатъчно примери за конструиране на рекурентни отношения, които описват сложността на рекурсивни алгоритми.

При изследването на алгоритмите не се интересуваме от точен израз за сложността – както се каза в минала лекция, това не е нито възможно, нито е необходимо. Интересува ни асимптотиката. Съответно няма да се опитваме да решаваме рекурентните отношения точно, а само ще търсим асимптотиката на решението. Основно наблюдение, което оставяме без доказателство е, че асимптотиката на решението не зависи от началните условия. Разбира се, има примери за обратното, примерно асимптотиката на

$$T(n) = T(n-1)^{T(n-2)}$$

зависи от началните стойности. Алгоритмите, които ще разглеждаме, са такива, че съотвентите им рекурентни отношения **не зависят** от началните стойности; в смисъл, че асимптотиката е една и съща, каквито и да са началните стойности. Поради това отсега нататък ще изпускаме началните условия от описанията на рекурентните отношения. Под "рекурентно отношение" ще разбираме само "същината"[†] на рекурентното отношение.

Да разгледаме отново рекурентното отношение (3), което описва сложността по време на рекурсивен алгоритъм, който разделя входа на две равни части и прави по едно рекурсивно викане върху всяка от тях. Използването на нотациите за точна долна и горна граница е формално коректно, защото, ако n е нечетно, няма как да разделим входа на две **точно равни** части – най-близкото до разделяне на две равни части е разделянето на една част с $\lfloor \frac{n}{2} \rfloor$ елемента и друга част с $\lceil \frac{n}{2} \rceil$ елемента. Но нотациите за точна горна и долна граница правят израза по-труден за четене, без да променят асимптотиката на функцията. И така, в изложението нататък ще правим още една опростяване: в рекурентните отношения, в които функцията на намаляването не е непременно целочислена, няма да използваме нотация за точна горна или точна долна граница. Примерно, наместо (3), пишем:

$$T(n) = 2T\left(\frac{n}{2}\right) + n \tag{13}$$

И още едно опростяване, което не променя асимптотиката на функцията, което ще покажем с пример. Ако последното рекурентно отношение описва работата на алгоритъм, който прави две рекурсивни викания и извършва освен това линейна работа, събираемото, които отразява тази линейна работа, би трябвало да е "$\Theta(n)$", така че рекурентното отношение би трябвало да е:

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

Но асимптотиката на решението не се променя, ако наместо "$\Theta(n)$" напишем просто "n", тоест най-простия за записване представител на класа от функции $\Theta(n)$. И така, записът, който ще ползваме, е (13).

3 Методи за решаване на рекурентни отношения

3.1 Чрез развиване

Средството, което можем да опитаме винаги, за да решим дадено рекурентно отношение, е да започнем да го развиваме, търсейки някаква закономерност. Примерно, ако рекурентното отношение е съвсем простото

$$T(0) = 0$$
$$T(n) = T(n-1) + n \text{ за } n > 0$$

[†]Примерно, $F_n = F_{n-1} + F_{n-2}$ е същината на (1).

можем да разсъждаваме така. Нека n е достатъчно голямо. Тогава $T(n-1) = T((n-1)-1) + (n-1)$, тоест $T(n-1) = T(n-2) + n - 1$. Ако заместим дясната страна в (11), получаваме

$$T(n) = T(n-2) + (n-1) + n \tag{14}$$

Но за достатъчно големи n, $T(n-2) = T((n-2)-1) + (n-2)$, тоест $T(n-2) = T(n-3) + (n-2)$. Ако сега заместим дясната страна в (14), получаваме

$$T(n) = T(n-3) + (n-2) + (n-1) + n \tag{15}$$

По правило, ако след 3-4 такива последователни развивания не видим такава закономерност, няма смисъл да опитваме повече. В случая рекурентното отношение е достатъчно просто, така че закономерност се появи. Очевидно е, че ако продължим по същия начин, след краен брой развивания ще достигнем до

$$T(n) = T(0) + 1 + 2 + \ldots + (n-2) + (n-1) + n \tag{16}$$

В случая това наистина е очевидно, защото аргументът намалява с единица и рано или късно ще стане нула, а събираемите вдясно, ако се пишат систематично, започват от числото, с единица по-голямо от аргумента на $T(\,)$ вдясно и завършват с n, като нарасват с единица.

Сега прилагаме два известни факта. Първо, че $T(0) = 0$, което знаем от началното условие, и второ, че $1 + 2 + \ldots n = \frac{n(n+1)}{2}$. Замествайки в (16), веднага получаваме

$$T(n) = \frac{n(n+1)}{2}$$

Методът с развиването (на английски, *unfolding*) не е формален, защото можем да развием началния израз не повече от константен брой пъти и неизбежно използваме интуиция, за да напишем уравнение, в което участват началните условия. Примерно, при прехода от (15) към (16) ползваме интуиция. Разбира се, винаги можем да докажем по индукция нашата интуиция (стига да е вярна), но това е друго нещо. Преди да ползваме индукция, трябва да знаем какво да докажем по индукция.

Методът с развиването освен това не е универсално приложим. Опитайте да решите рекурентното отношение на числата на Фибоначи (1) с развиване. Практически е невъзможно да получите (12), ако използвате само здрав разум без никаква теория.

Ето един значително по-сложен пример за прилагане на метода с развиването. Дадено е следното рекурентно отношение:

$$T(n) = n^2 T\left(\frac{n}{2}\right) + 1$$

Ще го решим с развиване. И така,

$$\begin{aligned}
T(n) &= n^2 T\left(\frac{n}{2}\right) + 1 \\
&= n^2\left(\frac{n^2}{4}T\left(\frac{n}{4}\right) + 1\right) + 1 \\
&= \frac{n^4}{2^2}T\left(\frac{n}{2^2}\right) + n^2 + 1 \\
&= \frac{n^4}{2^2}\left(\frac{n^2}{2^4}T\left(\frac{n}{2^3}\right) + 1\right) + n^2 + 1 \\
&= \frac{n^6}{2^6}T\left(\frac{n}{2^3}\right) + \frac{n^4}{2^2} + n^2 + 1 \\
&= \frac{n^6}{2^6}\left(\frac{n^2}{2^6}T\left(\frac{n}{2^4}\right) + 1\right) + \frac{n^4}{2^2} + n^2 + 1 \\
&= \frac{n^8}{2^{12}}T\left(\frac{n}{2^4}\right) + \frac{n^6}{2^6} + \frac{n^4}{2^2} + n^2 + 1 \\
&= \frac{n^8}{2^{12}}\left(\frac{n^2}{2^8}T\left(\frac{n}{2^5}\right) + 1\right) + \frac{n^6}{2^6} + \frac{n^4}{2^2} + n^2 + 1 \\
&= \frac{n^{10}}{2^{20}}T\left(\frac{n}{2^5}\right) + \frac{n^8}{2^{12}} + \frac{n^6}{2^6} + \frac{n^4}{2^2} + n^2 + 1 \\
&= \frac{n^{10}}{2^{20}}T\left(\frac{n}{2^5}\right) + \frac{n^8}{2^{12}} + \frac{n^6}{2^6} + \frac{n^4}{2^2} + \frac{n^2}{2^0} + \frac{n^0}{2^0} \\
&= \frac{n^{10}}{2^{5.4}}T\left(\frac{n}{2^5}\right) + \frac{n^8}{2^{4.3}} + \frac{n^6}{2^{3.2}} + \frac{n^4}{2^{2.1}} + \frac{n^2}{2^{1.0}} + \frac{n^0}{2^{0.(-1)}} \\
&\dots \\
&= \underbrace{\frac{n^{2i}}{2^{i(i-1)}}T\left(\frac{n}{2^i}\right)}_{A} + \underbrace{\frac{n^{2(i-1)}}{2^{(i-1)(i-2)}} + \frac{n^{2(i-2)}}{2^{(i-2)(i-3)}} \dots + \frac{n^8}{2^{4.3}} + \frac{n^6}{2^{3.2}} + \frac{n^4}{2^{2.1}} + \frac{n^2}{2^{1.0}} + \frac{n^0}{2^{0.(-1)}}}_{B}
\end{aligned}$$

Полученият израз е общият вид на дясната страна в процеса на развиването; с други думи, дясната страна след i развивания. Максималната стойност на i е $i_{max} = \lg n$, защото можем итеративно да прилагаме деление на две върху n логаритъм-от-n на брой пъти, преди текущото число да достигне началната стойност. Изразът вдясно е сума от два израза, които означаваме с A и B. Първо ще пресметнем A с $\lg n$ наместо i, като смятаме, че началното условие е, че $T(1)$ е някаква константа.

$$A = \frac{n^{2\lg n}}{2^{\lg^2 n - \lg n}}T(1) = \frac{T(1).2^{\lg n}n^{2\lg n}}{2^{\lg^2 n}} = \frac{T(1).n.n^{2\lg n}}{2^{\lg^2 n}}$$

Но

$$2^{\lg^2 n} = 2^{\lg n.\lg n} = 2^{\lg\left(n^{\lg n}\right)} = n^{\lg n} \tag{17}$$

Тогава

$$A = \frac{T(1).n.n^{2\lg n}}{n^{\lg n}} = \Theta(n^{1+\lg n}) \tag{18}$$

Сега да разгледаме B. Очевидно, можем да го представим като сума по следния начин:

$$\begin{aligned}
B &= \sum_{j=1}^{\lg n}\frac{n^{2((\lg n)-j)}}{2^{((\lg n)-j)((\lg n)-j-1)}} \\
&= \sum_{j=1}^{\lg n}\frac{1}{n^{2j}}\frac{n^{2\lg n}}{2^{(\lg^2 n - j\lg n - j\lg n + j^2 - \lg n + j)}} \\
&= \sum_{j=1}^{\lg n}\frac{1}{n^{2j}}\frac{\left(n^{2\lg n}\right)\left(2^{2j\lg n}\right)\left(2^{\lg n}\right)}{\left(2^{\lg^2 n}\right)\left(2^{j^2+j}\right)}
\end{aligned} \tag{19}$$

Но

$$2^{2j \lg n} = 2^{\lg\left(n^{2j}\right)} = n^{2j} \tag{20}$$

и

$$2^{\lg n} = n \tag{21}$$

Прилагаме (17), (20) и (21) върху (19) и получаваме

$$\begin{aligned}
B &= \sum_{j=1}^{\lg n} \frac{1}{n^{2j}} \frac{\left(n^{2\lg n}\right)\left(n^{2j}\right)(n)}{\left(n^{\lg n}\right)\left(2^{j^2+j}\right)} \\
&= \sum_{j=1}^{\lg n} \frac{n^{1+\lg n}}{2^{j^2+j}} \\
&= \left(n^{1+\lg n}\right) \sum_{j=1}^{\lg n} \frac{1}{2^{j^2+j}} \\
&\leqslant \left(n^{1+\lg n}\right) \sum_{j=1}^{\infty} \frac{1}{2^{j^2+j}} \\
&\leqslant n^{1+\lg n} \qquad \text{тъй като знаем, че } \sum_{j=1}^{\infty} \frac{1}{2^j} = 1 \\
&= \Theta\left(n^{1+\lg n}\right)
\end{aligned} \tag{22}$$

От (18) и (22) следва, че

$$T(n) = \Theta\left(n^{1+\lg n}\right)$$

3.2 Чрез дървото на рекурсията

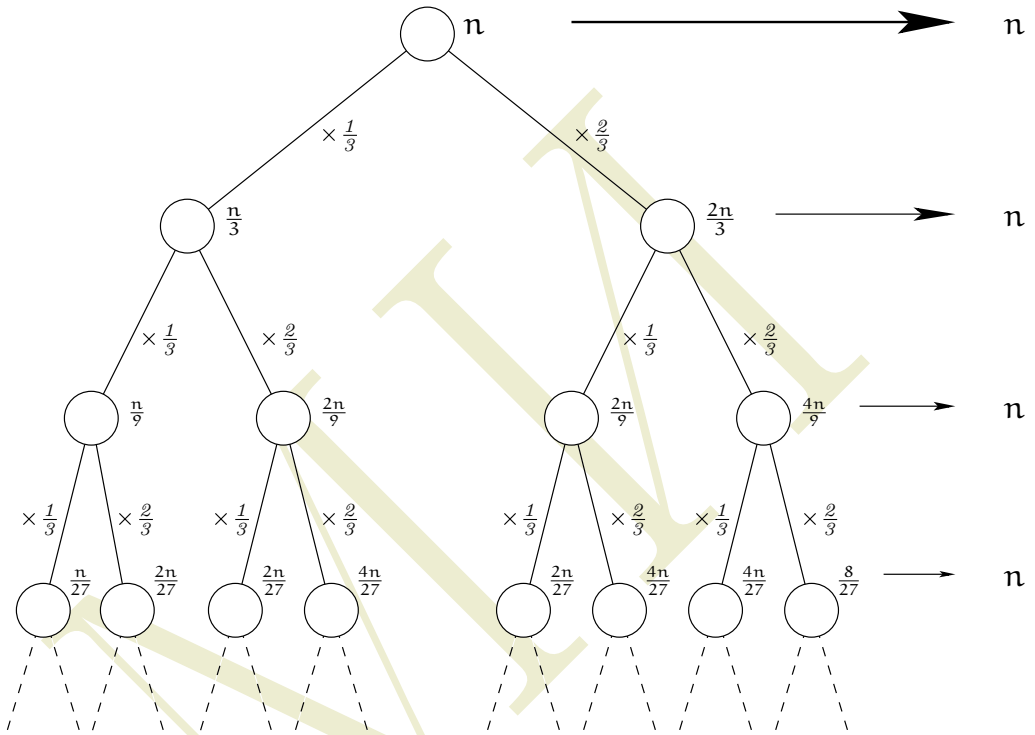
При този метод си представяме работата на рекурсивен алгоритъм, чиято сложност по време се описва от рекурентното отношение, което изследваме. Представяме си дървото на извикванията на рекурсивния алгоритъм, съобразяваме колко работа се извършва във всеки връх от дървото (тоест, във всяко индивидуално изпълнение на рекурсивния алгоритъм), и сумираме тези количества по нива в дървото. След което търсим закономерност в сумите по нива. Виждаме, че метода с дървото е подобен на метода с развиването по това, че не е формален и за да даде резултати, трябва човекът, който го ползва, да прояви достатъчно добра интуиция. Тук ще направим илюстрация с пример. Ще решим рекурентното отношение

$$T(n) = T\left(\frac{n}{3}\right) + T\left(\frac{2n}{3}\right) + n \tag{23}$$

използвайки метода с дървото на рекурсията. Дървото на рекурсията—по-точно, само "горната" му част—е изобразено на Фигура 1. Всеки връх представлява едно индивидуално изпълнение на алгоритъма без рекурсивните викания. Двете ребра към децата му представляват рекурсивните викания, като те (ребрата) са маркирани с множителите, с които входът намалява. До всеки връх е написана работата, която алгоритъмът извършва в това индивидуално изпълнение извън рекурсивните викания. Върховете от едно и също ниво в дървото са нарисувани на едно и също ниво на фигурата.

Коренът е първоначалното изпълнение. В него алгоритъмът извършва работа n извън двете рекурсивни викания, затова вдясно от корена е написано "n". Едното рекурсивно викане се върши върху вход, който е с големина $\frac{1}{3}$ от големината на началния, а другото върху вход, който е с големина $\frac{2}{3}$ от големината на началния. Затова двете ребра, свързващи корена с децата му, са маркирани с $\frac{1}{3}$ и $\frac{2}{3}$. До всяко от двете деца е написана работата, която се извършва от съответното извикване на алгоритъма. Тази работа е същата като големината на входа за съответното викане, затова записаните стойности са $\frac{n}{3}$ и $\frac{2n}{3}$. И така нататък.

Да мислим за дървото като за подредено дърво. Припомняме, че това означава, че различаваме ляво от дясно дете на всеки вътрешен връх. Стойността, асоциирана с лявото дете на даден връх, е стойността, асоциирана



Фигура 1: Дървото на рекурсията, отговарящо на $T(n) = T\left(\frac{n}{3}\right) + T\left(\frac{2n}{3}\right) + n$. За всяко от изобразените нива, сумата от стойностите на върховете в него е n.

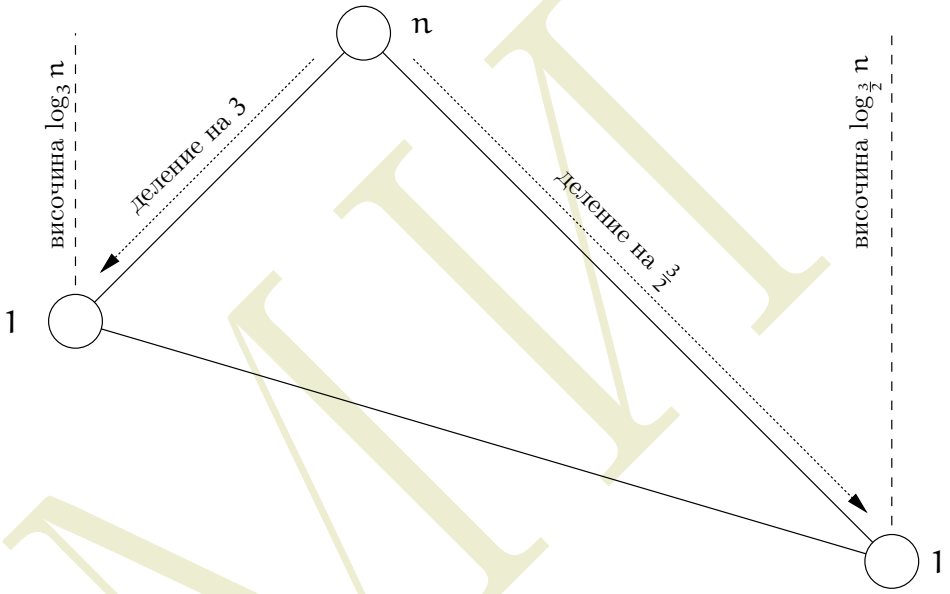
с родителя, умножена по $\frac{1}{3}$. Стойността, асоциирана с дясното дете, е стойността, асоциирана с родителя, умножена по $\frac{2}{3}$. С други думи, наляво има деление на 3, а надясно, деление на $\frac{3}{2}$. Това, че разглеждаме дървото като подредено дърво, има значение единствено за по-лесното решаване на задачата. Листата на дървото са очевидно на различни нива, така че ни трябва някаква систематика в описването на дървото, за да видим закономерност. По начина, по който разглеждаме дървото, най-близкото до корена листо е краят на пътя, който започва от корена и във всеки вътрешен връх продължава наляво, а най-отдалеченото от корена листо е краят на пътя, който започва от корена и във всеки вътрешен връх продължава надясно. Образно казано, дървото има формата, показана на Фигура 2. Допускаме, че началната стойност е 1, така че листата са асоциирани с единици. Дървото е нарисувано по начин, който подсказва, че с деление на 3 достигаме 1 по-бързо, отколкото с деление на $\frac{3}{2}$.

Тривиално е да се докаже, че нива с номер от 0 (коренът) до $\log_3 n$ (най-лявото дете) са пълни[†] и във всяко от тях, сумата от стойностите е n. Фигура 3 илюстрира пълните нива и останалите: пълните нива са в червено, останалите са в зелено. Да намерим асимптотиката на рекурентното отношение е същото като да оценим сумата от асоциираните стойности по всички върхове на дървото. За червеното поддърво (Фигура 3), тази сума е височината, умножена по n, защото по всяко негово ниво, сумата е n. Височината на червеното поддърво е $\log_3 n$, така че сумата на стойностите по неговите върхове е $n \log_3 n$. Това е долна граница за сумата по всички върхове на дървото. Но очевидно $n \log_{\frac{3}{2}} n$ е горна граница за същата сума, понеже $\log_{\frac{3}{2}}$ е височината на цялото дърво и никое ниво няма сума повече от n. Но $n \log_3 n \asymp n \lg n$ и $n \log_{\frac{3}{2}} n \asymp n \lg n$. Щом $T(n) \preceq n \lg n$ и $T(n) \succeq n \lg n$, следва, че $T(n) \asymp n \lg n$.

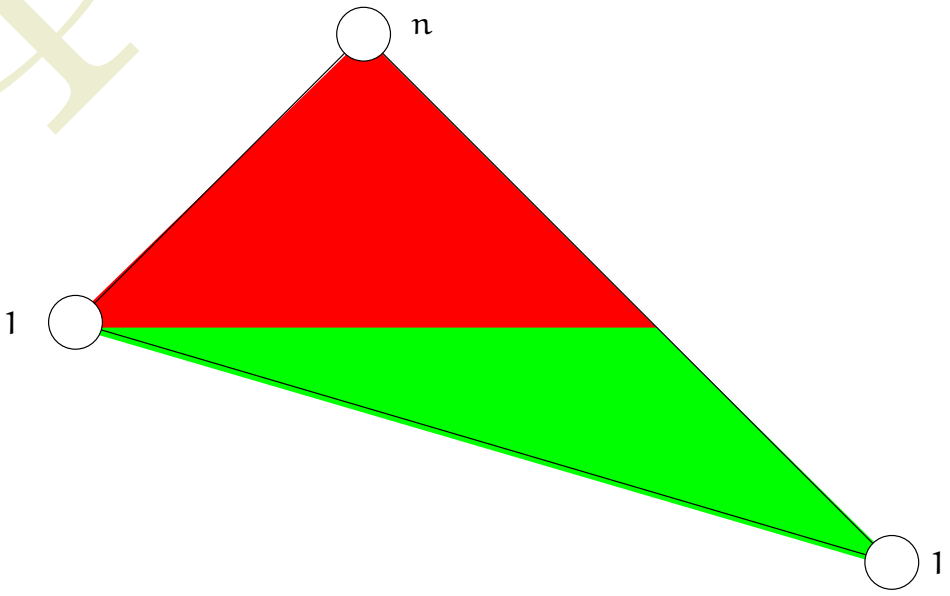
Посоченият пример илюстрира основната характеристика на метода с дървото на рекурсията, а именно, че той изчислява сумата от всички “нехомогенни части” на рекурентното отношение, по всички извиквания. Тук “нехомогенни” е в кавички, защото понятията “хомогенно” и “нехомогенно” по принцип не се прилагат за рекурентно отношение като (23). Става дума за събираемото n: това е работата, която алгоритъмът извършва извън рекурсивните викания. Ако наречем това събираемо, нехомогенната част по аналогия с (10), то методът с дървото на рекурсията действително събира нехомогенните части[‡]. Хомогенно рекурентно отношение като

[†] В смисъл, че в ниво k има 2^k върха.

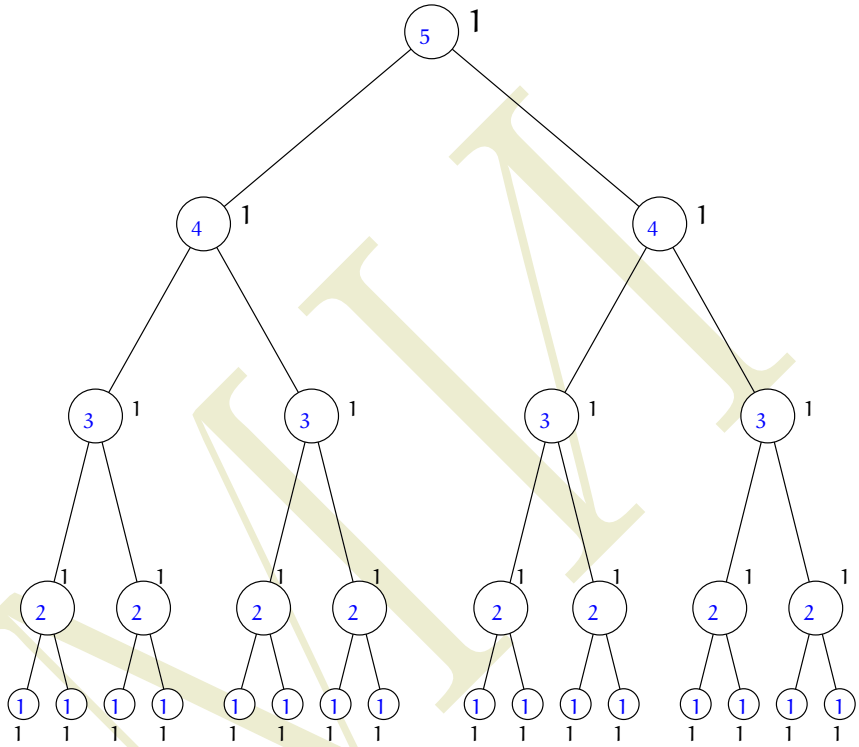
[‡] По-точно, събира нехомогенните части и началните условия, но тях ние игнорираме при асимптотичния анализ.



Фигура 2: Най-лявото дете е най-близо до корена, а най-дясното дете е най-отдалечено от корена. Височината на най-лявото дете е $\log_3 n$, а на най-дясното е $\log_{\frac{3}{2}} n$.



Фигура 3: Червеното поддърво е съвършено, понеже нивата му са пълни. Следващите нива, нарисувани в зелено, са непълни.



Фигура 4: Дървото на рекурсията на $T(1) = 1,\ T(n) = 2T(n-1) + 1$ при $n = 5$. Във върховете, в син цвят са написани съответните стойности на n.

(1) не подлежи на решаване с този метод†. От друга страна, очевидно е, че всяко рекурентно отношение, което описва сложност на алгоритъм, има ненулева нехомогенна част, защото алгоритъмът няма как да не върши поне константна работа извън рекурсивните викания. Така че, по отношение на изследването на алгоритми, методът с дървото е приложим, стига да успеем да открием закономерност в сумите по нивата в дървото. Подобно на развиването, този метод също е неформален и приложимостта му зависи от интуицията на човека, който го прилага. Ако човек не открие някаква закономерност в сумите, методът не е приложим.

Сега ще дадем пример за това, колко е важно да открием закономерност в сумите при този метод. Да разгледаме рекурентното отношение

$$\begin{aligned} T(1) &= 1 \\ T(n) &= 2T(n-1) + 1,\ n \geqslant 2 \end{aligned} \tag{24}$$

и съответното дърво на рекурсията на Фигура 4) при $n = 5$. На ниво 0 сумата е 1, на ниво 1 сумата е 2, и така нататък, на ниво 4 сумата е 16. Сумата от стойностите, асоциирани с върховете, е 31. Сега да преобразуваме (24) ето така:

$$T(n) = 2T(n-1) + 1 \ \leftrightarrow\ T(n) = T(n-1) + T(n-1) + 1$$

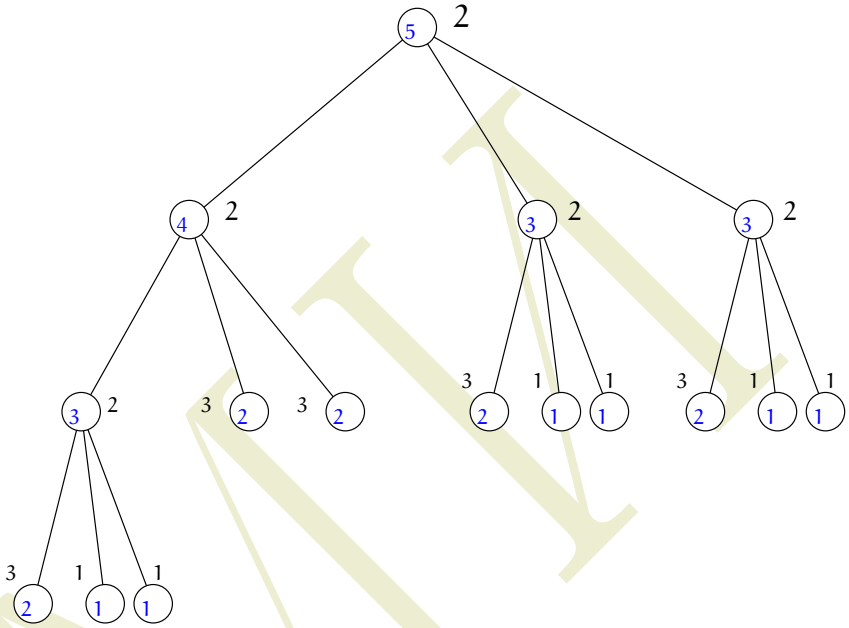
Очевидно $T(n-1) = 2T(n-2) + 1$

Тогава $T(n) = T(n-1) + 2T(n-2) + 1 + 1 = T(n-1) + 2T(n-2) + 2$

Тъй като в случая искаме точно рекурентно отношение с началните условия, налага се да добавим и начално условия $T(2) = 3$. Рекурентното отношение става:

$$\begin{aligned} T(1) &= 1 \\ T(2) &= 3 \\ T(n) &= T(n-1) + 2T(n-2) + 2,\ n \geqslant 3 \end{aligned} \tag{25}$$

†Хомогенно рекурентно отношение като (1) има дърво на рекурсията, в което всеки вътрешен връх е асоцииран с нула. Решението за дадено n идва само от сумиране на началните условия. Тъй като в асимптотичния анализ игнорираме началните условия, то няма какво да сумираме, ако се опитаме решение с метода с дървото.



Фигура 5: Дървото на рекурсията на $T(1) = 1,\ T(2) = 3,\ T(n) = T(n-1) + 2T(n-2) + 2$. Във върховете, в син цвят са написани съответните стойности на n.

Рекурентни отношения (24) и (25) са еквивалентни. Но техните съответни дървета на рекурсията нито са еднакви, нито са изоморфни. Фигура 5 показва дървото на рекурсията, съответстващо на (25), за $n = 5$. Дървото от Фигура 5 е неподходящо за откриване на закономерност както за прецизното решение $T(n) = 2^n - 1$, така и за асимптотичното решение $T(n) \asymp 2^n$.

3.3 По индукция

Ще демонстрираме доказването на асимптотиката на нарастването на функцията от дадено рекурентно отношение с пример. Да разгледаме:

$$T(n) = 2T\left(\left\lceil \frac{n}{2} \right\rceil\right) + n \tag{26}$$

Ще докажем, че $T(n) = \Theta(n \lg n)$ по индукция по n. За да постигнем това, ще докажем поотделно, че $T(n) = O(n \lg n)$ и $T(n) = \Omega(n \lg n)$. В тези доказателство база няма, тъй като игнорираме началните условия.

Част i: Доказателство, че $T(n) = O(n \lg n)$. По дефиниция, е същото като да докажем, че съществуват положителна константа c и положително n_0, такова че за всяко $n \geqslant n_0$:

$$T(n) \leqslant cn \lg n \tag{27}$$

Индуктивното предположение е, че

$$T\left(\left\lceil \frac{n}{2} \right\rceil\right) \leqslant c \left\lceil \frac{n}{2} \right\rceil \lg \left\lceil \frac{n}{2} \right\rceil \tag{28}$$

От (26) и (28):

$$\begin{aligned}
T(n) &\leqslant 2.c. \left\lceil \frac{n}{2} \right\rceil \lg \left\lceil \frac{n}{2} \right\rceil + n \\
&\leqslant 2.c. \left(\frac{n}{2} + 1\right) \lg \left\lceil \frac{n}{2} \right\rceil + n && (29) \\
&\leqslant 2.c. \left(\frac{n}{2} + 1\right) \lg \left(\frac{3n}{4}\right) + n && \text{защото } \forall n \geqslant 2 \left(\frac{3n}{4} \geqslant \left\lceil \frac{n}{2} \right\rceil\right) \quad (30) \\
&= c(n+2)(\lg n + \lg 3 - 2) + n \\
&= cn \lg n + cn(\lg 3 - 2) + 2c \lg n + 2c(\lg 3 - 2) + n \\
&\leqslant cn \lg n && \text{ако } cn(\lg 3 - 2) + 2c \lg n + 2c(\lg 3 - 2) + n \leqslant 0
\end{aligned}$$

Да разгледаме

$$cn(\lg 3 - 2) + 2c\lg n + 2c(\lg 3 - 2) + n = (c(\lg 3 - 2) + 1)n + 2c\lg n + 2c(\lg 3 - 2)$$

Асимптотиката на нарастването се определя от линейното събираемо. Ако константата $c(\lg 3 - 2) + 1$ е отрицателна, тогава целият израз е положителен за всички достатъчно големи стойности на n. С други думи, за по-кратко решение, няма да определяме n_0, бидейки убедени, че такава начална стойност има. За да бъде изпълнено $c(\lg 3 - 2) + 1 < 0$, трябва да е вярно, че $c > \frac{1}{2-\lg 3}$. Следователно, всяко c, такова че $c > \frac{1}{2-\lg 3}$, "върши работа" за нашето доказателство.

> ***ВНИМАНИЕ!*** *В (29) ние заместваме $\lceil \frac{n}{2} \rceil$ с $\frac{3n}{4}$. Бихме могли да използваме всяка друга дроб $\frac{pn}{q}$, стига да е изпълнено $\frac{1}{2} < \frac{p}{q} < 1$. Защо трябва да е изпълнено $\frac{1}{2} < \frac{p}{q}$? Ако това не е изпълнено, нямаме право да твърдим, че неравенството "$\leqslant$" между (29) и (30) е в сила. А защо трябва да е изпълнено $\frac{p}{q} < 1$? Да допуснем, че $\frac{p}{q} = 1$; с други думи, заместваме $\lceil \frac{n}{2} \rceil$ с n. Тогава (30) става:*
>
> $$c(n+2)(\lg n) + n = cn\lg n + 2c\lg n + n$$
>
> *Очевидно, това е по-голямо от $cn\lg n$ за всички достатъчно големи n, така че нямаме доказателство.*

Част ii: Доказателство, че $T(n) = \Omega(n\lg n)$. По дефиниция, това есъщото като да докажем, че съществуват положителна константа d и положително n_1, такова че за всяко $n \geqslant n_1$:

$$T(n) \geqslant dn\lg n \tag{31}$$

Индуктивното предположение е, че

$$T\left(\left\lceil \frac{n}{2} \right\rceil\right) \geqslant d\left\lceil \frac{n}{2} \right\rceil \lg \left\lceil \frac{n}{2} \right\rceil \tag{32}$$

От (26) и (32):

$$\begin{aligned} T(n) &\geqslant 2.d.\left\lceil \frac{n}{2} \right\rceil \lg \left\lceil \frac{n}{2} \right\rceil + n \\ &\geqslant 2.d.\left(\frac{n}{2}\right)\lg\left(\frac{n}{2}\right) + n \\ &= dn(\lg n - 1) + n \\ &= dn\lg n + (1-d)n \\ &\geqslant dn\lg n \qquad \text{при условие, че } (1-d)n \geqslant 0 \end{aligned} \tag{33}$$

Очевидно всяко d, такова че $0 < d \leqslant 1$, "върши работа" за нашето доказателство.

С това доказателството е приключено. Преди да преминем към следващия метод, едно предупреждение за потенциални грешки при използването на индукция за доказване на асимптотиката на рекурентни отношения. Да разгледаме отново следното рекурентно отношение:

$$T(n) = 2T(n-1) + 1$$

Както вече знаем, решението е $T(n) = \Theta(2^n)$. Сега помислете, къде е грешката в следното доказателство, което "доказва", че $T(n) = O(n)$. Ще "докажем", че $T(n) \leqslant cn$ за някаква константа c. Индуктивното предположение е, че $T(n-1) \leqslant c(n-1)$. Тогава $T(n) \leqslant 2c(n-1) + 1$. Тогава $T(n) \leqslant 2cn - 2c + 1$. Но последният израз казва, че $T(n)$ е ограничена отгоре от константа, умножена по n. Тогава $T(n) = O(n)$, което и искахме да докажем.

Къде е грешката? Грешката е в това, че доказателството не се извършва по отношение на константата, с която е започнато. Твърдението, което доказваме е, че **съществува положителна константа c и стойност на аргумента n_0, такива че за всяко $n \geqslant n_0$, $T(n) \leqslant cn$.** Естествено, това твърдение е лъжа, така че няма как да го докажем, но важното е, че **това** е твърдението, което доказваме. "$T(n) = O(n)$" е просто кратък запис за него. Щом избраният символ за константата е c, доказателството трябва да бъде извършено по отношение на него. $2c$ е **друга** константа. Наистина, $2c$ е константа, но за да имаме доказателство, трябва да го направим по отношение на c.

3.4 Чрез Мастър Теоремата

Изложението ще бъде извършено учебника на Cormen и др. [CLRS09, стр. 61]. Гръмкото име на теоремата се дължи на факта, че тя се явява обобщение на множество теореми, даващи решения на рекурентни отношения, в които функцията на намаляване е деление. Но това съвсем не е най-мощният теоретичен апарат за решаване на такива рекурентни отношения. Теоремата на Akra и Bazzi [AB98] е значително по-мощна[†]. От друга страна, теоремата на Akra и Bazzi е значително по-сложна, така че ще се задоволим с Мастър теоремата.

Теорема 1 (Мастър теорема, [CLRS09], стр. 62)**.** *Нека* $a \geqslant 1$ *и* $b > 1$ *са константи и нека* $f(n)$ *е положителна функция. Нека*

$$T(n) = aT\left(\frac{n}{b}\right) + f(n) \tag{34}$$

където $\frac{n}{b}$ *има смисъл или на* $\left\lfloor \frac{n}{b} \right\rfloor$*, или на* $\left\lceil \frac{n}{b} \right\rceil$*. Тогава асимптотиката на* $T(n)$ *е следната*

Случай 1 *Ако* $f(n) = O\left(n^{\log_b a - \epsilon}\right)$ *за някоя положителна константа* ϵ*, тогава* $T(n) = \Theta(n^k)$.

Случай 2 *Ако* $f(n) = \Theta(n^{\log_b a})$*, тогава* $T(n) = \Theta\left(n^{\log_b a}.\lg n\right)$*. С други думи,* $T(n) = f(n).\lg n$.

Case 3 *Ако са изпълнени следните:*

1. $f(n) = \Omega\left(n^{\log_b a + \epsilon}\right)$ *за някоя положителна константа* ϵ*, и*
2. $a.f\left(\frac{n}{b}\right) \leqslant c.f(n)$ *за някоя положителна константа* c*, такава че* $0 < c < 1$ *за всички достатъчно големи* n*,*

то $T(n) = \Theta(f(n))$. □

Условие 3-2 се нарича *условие за регулярност.*

Няма да даваме доказателство на теоремата. Читателят може да го види в [CLRS09]. Тук само ще дадем няколко пояснения.

- И в трите случая на Мастър теоремата (MT), сравняваме големината на дървото на рекурсията с функцията $f(n)$. Да се убедим, че големината на дървото наистина е, приблизително, $n^{\log_b a}$. Нека да мислим, че a и b са цели числа. В [CLRS09] е доказано, че същите съображения остават в сила дори когато a и b не са цели. Ако a и b са цели положителни числа, като $b \geqslant 2$, то (34) отговаря на рекурсивен алгоритъм от схемата Разделяй-и-Владей, който прави a рекурсивни извиквания, всяко върху вход с големина $\frac{n}{b}$, и освен това извършва $f(n)$ работа (като време) по разделянето на входа и комбинирането на резултатите от извикванията. Дървото на този алгоритъм е показано на Фигура 6. Всеки връх на дървото има разклоненост a, и входът намалява с деление на b. Сумите по нивата на дървото са $f(n)$, $af\left(\frac{n}{b}\right)$, $a^2 f\left(\frac{n}{b^2}\right)$, и така нататък. Височината на дървото е $\log_b n$. Броят на върховете от ниво[‡] k очевидно е a^k, така че броят на листата е е $a^{\log_b n}$. Броят на листата е пропорционален на големината на дървото (освен ако a не е единица), така че големината на дървото е пропорционална на $a^{\log_b n}$. Лесно се вижда, че $a^{\log_b n} = n^{\log_a b}$. Така че наистина големината на дървото е $n^{\log_a b}$. В първия случай на MT големината на дървото доминира над $f(n)$ и тя (големината) задава асимптотиката на $T(n)$, във втория случай големината на дървото и $f(n)$ са асимптотично еквивалентни, в третия случай $f(n)$ доминира над големината на дървото и тя ($f(n)$) определя асимптотиката, ако е изпълнено и условието за регулярност[§].

- Трите случай на MT **не са разбиване** на всички възможни случаи дори когато $n^{\log_a b}$ и $f(n)$ са асимптотично сравними. Има примери за a, b и $f(n)$, в които $f(n)$ расте прекалено бързо, за да попаднем в първия случай, но прекалено бавно, за да попаднем във втория.

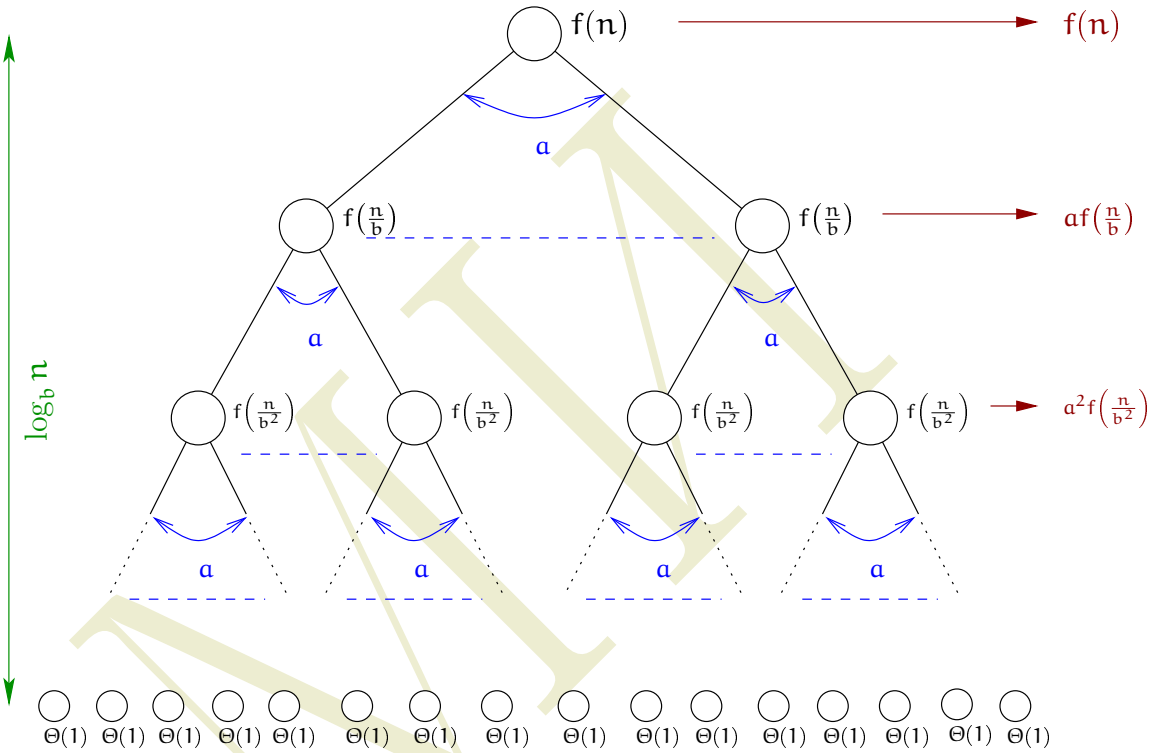
  Примерно, нека $a = b = 2$ и $f(n) = \frac{n}{\lg n}$. Тогава $f(n) \neq O\left(n^{\log_b a - \epsilon}\right)$ за всяка положителна константа ϵ, така че рекурентното отношение не попада в първия случай. От друга страна, $f(n) \neq \Theta(n^{\log_b a})$.

  Аналогично, има примери за a, b и $f(n)$, в които $f(n)$ расте прекалено бързо, за да попаднем във втория случай, но прекалено бавно, за да попаднем в третия.

[†] Виж [Lei96] за подробности

[‡] Върховете от ниво k са върховете на разстояние k от корена.

[§] Сравняването на големината на дървото с функцията-нехомогенна част има смисъл и при други видове рекурентни отношения. Примерно, рекурентното отношение $S(n) = 2S(n-1) + n!$ има решение $S(n) \asymp n!$, защото факториелът доминира над експоненциалната големина на дървото. В някакъв смисъл, това е аналог на третия случай на MT.



Фигура 6: Дървото на рекурсията на алгоритъм, чиято сложност по време се описва от $T(n) = aT\left(\frac{n}{b}\right) + f(n)$, където a и b са цели положителни числа.

- Условието $f(n) = O\left(n^{\log_b a-\epsilon}\right)$ е по-силно от $f(n) = o\left(n^{\log_b a-\epsilon}\right)$, също както $f(n) = \Omega\left(n^{\log_b a+\epsilon}\right)$ е по-силно от $f(n) = \omega\left(n^{\log_b a+\epsilon}\right)$.

  $f(n) = O\left(n^{\log_b a-\epsilon}\right)$, където ϵ е положителна константа, казва, че между $f(n)$ и $n^{\log_b a}$ може да бъде “вкарано” n^ϵ, с други думи, че $f(n)$ изостава от $n^{\log_b a}$ “на един полином”[†].

  Аналогично, $f(n) = \Omega\left(n^{\log_b a+\epsilon}\right)$ казва, че $f(n)$ изпреварва $n^{\log_b a}$ “с един полином”.

- Условието за регулярност в третия случай е по-силно от условието $f(n) = \Omega\left(n^{\log_b a+\epsilon}\right)$ в смисъл, че

$$\exists c > 0\left(a.f\left(\frac{n}{b}\right) \leqslant c.f(n)\right) \rightarrow \exists \epsilon\left(f(n) = \Omega\left(n^{\log_b a+\epsilon}\right)\right)$$
$$\exists \epsilon\left(f(n) = \Omega\left(n^{\log_b a+\epsilon}\right)\right) \nrightarrow \exists c > 0\left(a.f\left(\frac{n}{b}\right) \leqslant c.f(n)\right)$$

  Доказателство на първото твърдение има в сборника с решени задачи по алгоритми. Пример в подкрепа на второто има в статията в Wikipedia за MT: да вземем $f(n) = n(2 - \cos n)$.

  Следователно, третият случай на MT би могъл да бъде формулиран само с условието за регулярност; условието $(n) = \Omega\left(n^{\log_b a+\epsilon}\right)$ е излишно. Дадената формулировка е избрана, без съмнение, за да има външна аналогия с формулировките на първите два случая.

Ще дадем само един пример за прилагането на MT теоремата. Чрез нея ще решим

$$T(n) = 3T\left(\frac{n}{4}\right) + n \lg n$$

Използвайки терминологията на MT, a е 3, b е 4, така че $\log_b a$ е $\log_4 3$, което е приблизително 0.79, и $f(n)$ е $n \lg n$. Със сигурност е вярно, че $n \lg n = \Omega(n^{\log_4 3+\epsilon})$ за някое $\epsilon > 0$, примерно $\epsilon = 0.1$. Но трябва да проверим

[†] n^ϵ за целите на асимптотичния анализ се нарича полином, ако $\epsilon > 0$

и дали условието за регулярност е в сила, за да видим дали може да приложим третия случай на МТ. Условието за регулярност е:

$$\exists c \text{ такова че } 0 < c < 1 \text{ и } 3\frac{n}{4}\lg\frac{n}{4} \leqslant cn\lg n \text{ за всички достатъчно големи } n$$

Очевидно това е вярно за, да речем, $c = \frac{3}{4}$, така че третият случай на МТ е приложим. Съгласно него, $T(n) = \Theta(n\lg n)$.

3.5 Чрез метода с характеристичното уравнение

Ще изложим следната теорема без доказателство съгласно учебника Увод в Дисткретната Математика на Кр. Манев [Man12, стр. 60]. Аргументация на твърдение има, примерно, в книгата на Flajolet и Sedgewick [SF96].

Theorem 1. *Нека редицата* $\tilde{a} = (a_0, a_1, a_2, \ldots)$ *е зададена с линейното рекурентно отношение*

$$a_n = c_1 a_{n-1} + c_2 a_{n-2} + \ldots + c_{r-1} a_{n-(r-1)} + c_r a_{n-r},\ n \geqslant r \tag{35}$$

Нека α_1, α_2, ..., α_s *са различните комплексни корени на* характеристичното уравнение

$$x^r - c_1 x^{r-1} - c_2 x^{r-2} - \ldots - c_{r-1}x - c_r = 0 \tag{36}$$

като α_i *е с кратност* k_i *for* $1 \leqslant i \leqslant s$[†]. *Тогава*

$$a_n = P_1(n)\alpha_1^n + P_2(n)\alpha_2^n + \ldots + P_s(n)\alpha_s^n \tag{37}$$

където $P_i(n)$ *е полином на* n *от степен* $< k_i$. *Полиномите* $P_1(n)$, $P_2(n)$, ..., $P_s(n)$ *имат общо* r *коефициента, които се определят еднозначно от първите* r *члена на редицата* $\tilde{a}$. □

В текущата терминология, (35) изглежда така:

$$T(n) = c_1 T(n-1) + c_2 T(n-2) + \ldots + c_{r-1} T(n-(r-1)) + c_r T(n-r) \tag{38}$$

Както вече казахме, това е общ пример за линейно хомогенно рекурентно отношение с константни коефициенти и крайна история. Такова рекурентно отношение не може да описва сложността на рекурсивен алгоритъм, защото алгоритъмът не може да не извършва поне константна работа извън рекурсивните викания. Понеже се интересуваме от анализ на алгоритми, налага се да разгледаме по-общ вид рекурентни отношения, а именно нехомогенни. Ще разглеждаме нехомогенни рекурентни отношения от този вид:

$$\begin{aligned} T(n) = {} & c_1 T(n-1) + c_2 T(n-2) + \ldots + c_{r-1} T(n-(r-1)) + c_r T(n-r) \\ & + b_1^n Q_1(n) + b_2^n Q_2(n) + \ldots + b_m^n Q_m(n) \end{aligned} \tag{39}$$

$b_1, b_2, \ldots, b_m$ са различни положителни константи. $Q_1(n), Q_2(n), \ldots, Q_m(n)$ са полиноми от съответни степени $d_1, d_2, \ldots, d_m$.

Ще означаваме мултимножествата със следната нотация: "$\{\quad\}_M$" примерно $\{1,1,2,3,3,3\}_M$. За всеки елемент a по отношение на някакво мултимножество A, с $\#(a, A)$ означаваме броя на появите на a в A. $\#(a, A) = 0$, ако a не се среща в A. Примерно, $\#(1, \{1,1,2,3,3,3\}) = 2$. Обединението на две мултимножества A и B е:

$$A \cup B = \{x \mid (x \in A \text{ или } x \in B) \text{ и } (\#(x, A \cup B) = \#(x, A) + \#(x, B))\}_M$$

Кардиналност на мултимножеството A е сумата от броевете на появите на елементите му и се означава с $|A|$. Примерно, $|\{1,1,2,3,3,3\}_M| = 6$.

Решението на (39) се получава по следния начин. Нека мултимножеството от корените на характеристичното уравниение е A. Очевидно, $|A| = r$. Нека $B = \{b_i \mid \#(b, B) = d_i + 1\}_M$. Нека $Y = A \cup B$. Очевидно, $|Y| = r + \sum_{i=1}^{m}(d_i + 1)$. Нека преименуваме различните елементи на Y като y_1, y_2, ..., y_t и да дефинираме, че $\#(y_i, Y) = z_i$, за $1 \leqslant i \leqslant t$. Тогава

$$\begin{aligned} T(n) = {} & \beta_{1,1}\, y_1^n + \beta_{1,2}\, n\, y_1^n + \ldots + \beta_{1,z_1}\, n^{z_1-1}\, y_1^n + \\ & \beta_{2,1}\, y_2^n + \beta_{2,2}\, n\, y_2^n + \ldots + \beta_{2,z_2}\, n^{z_2-1}\, y_2^n + \\ & \ldots \\ & \beta_{t,1}\, y_1^n + \beta_{t,2}\, n\, y_t^n + \ldots + \beta_{t,z_t}\, n^{z_t-1}\, y_1^n \end{aligned} \tag{40}$$

[†] Очевидно, $k_i \geqslant 1$ за $1 \leqslant i \leqslant s$ и освен това, $k_1 + k_2 + \ldots + k_s = r$.

Индексираните β-и са константи, общо $|Y|$ на брой. Понеже се интересуваме само от асимптотичното нарастване на $T(n)$, точните стойности на константите са без значение. Асимптотичното нарастване се определя от точно едно събираемо, а именно, най-голямото[†] y_i, умножено по най-голямата възможна степен на n.

Ще решим две рекурентни отношения с с метода с характеристичното уравнение. Първото е това:

$$T(n) = T(n-1) + 1$$

Преписваме рекурентното отношение така: $T(n) = T(n-1) + 1^n n^0$, за да сме сигурни, че фо́рмата му удовлетворява (39). Характеристичното уравнение е $x - 1 = 0$ с единствен корен $x_1 = 1$. Тогава мултимножеството от корените на характеристичното уравнение е $\{1\}_M$. Използвайки конвенцията за именуване на (39), $m = 1$, $b_1 = 1$, и $d_1 = 0$. Така че към мултимножеството $\{1\}_M$ добавяме $b_1 = 1$ с кратност $d_1 + 1 = 1$, получавайки $\{1,1\}_M$. Тогава $T(n) = A\,1^n + B\,n\,1^n$ за някакви константи A и B, следователно $T(n) = \Theta(n)$.

Второто е това:

$$T(n) = 4T(n-3) + 1$$

Характеристичното уравнение е

$$x^3 - 4 = 0$$

Корените му са

$$x_1 = \sqrt[3]{4}$$
$$x_2 = \sqrt[3]{4}\, e^{i\frac{2\pi}{3}}$$
$$x_3 = \sqrt[3]{4}\, e^{i\frac{-2\pi}{3}}$$

Ако A, B, C и D са комплексни константи, решението е:

$$\begin{aligned}
T(n) &= A\left(\sqrt[3]{4}\right)^n + B\left(\sqrt[3]{4}\right)^n e^{\frac{2n\pi i}{3}} + C\left(\sqrt[3]{4}\right)^n e^{\frac{-2n\pi i}{3}} + D1^n = \\
&= A\left(\sqrt[3]{4}\right)^n + B\left(\sqrt[3]{4}\right)^n \left(\cos\left(\frac{2n\pi}{3}\right) + i\sin\left(\frac{2n\pi}{3}\right)\right) + \\
&\quad C\left(\sqrt[3]{4}\right)^n \left(\cos\left(\frac{-2n\pi}{3}\right) + i\sin\left(\frac{-2n\pi}{3}\right)\right) + D \\
&= A\left(\sqrt[3]{4}\right)^n + \left(\sqrt[3]{4}\right)^n \cos\left(\frac{2n\pi}{3}\right)(B+C) + \\
&\quad \left(\sqrt[3]{4}\right)^n \sin\left(\frac{2n\pi}{3}\right)(B-C)i + D
\end{aligned}$$

Ако вземем $B = C = \frac{1}{2}$, получаваме едно решение:

$$T_1(n) = A\left(\sqrt[3]{4}\right)^n + \left(\sqrt[3]{4}\right)^n \cos\left(\frac{2n\pi}{3}\right) + D$$

Ако вземем $B = -\frac{1}{2}i$ and $C = \frac{1}{2}i$, получаваме друго решение:

$$T_2(n) = A\left(\sqrt[3]{4}\right)^n + \left(\sqrt[3]{4}\right)^n \sin\left(\frac{2n\pi}{3}\right) + D$$

Съгласно принципа на суперпозицията[‡], имаме общо решение

$$T(n) = A_1\left(\sqrt[3]{4}\right)^n + A_2\left(\sqrt[3]{4}\right)^n \cos\left(\frac{2n\pi}{3}\right) + A_3\left(\sqrt[3]{4}\right)^n \sin\left(\frac{2n\pi}{3}\right) + A_4$$

за някакви константи A_1, A_2, A_3 и A_4. Асимптотиката на решението е $T(n) = \Theta\left(\left(\sqrt[3]{4}\right)^n\right)$.

[†] Ако $T(n)$ описва сложността на алгоритъм, най-голямото по абсолютна стойност y_i задължително е положително.

[‡] Принципът на суперпозицията казва, че ако имаме линейно рекурентно отношение и знаем, че някакви функции $g_i()$, $1 \leqslant i \leqslant k$ са решения, тогава всяка тяхна линейна комбинация също е решение. Виж [SF96] или [Bal91, стр. 97].

Литература

[AB98] Mohamad Akra and Louay Bazzi. On the solution of linear recurrence equations. *Computational Optimization and Applications*, 10(2):195–210, 1998.

[Bal91] V. K. Balakrishnan. *Introductory Discrete Mathematics*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 1st edition, 1991. Available online at http://books.google.com/books?id=pOBXUoVZ9EEC&printsec=frontcover&dq=Introductory+discrete+mathematics++By+V.+K.+Balakrishnan&source=bl&ots=1lYLvMpVfY&sig=Jwklfma4Zf3EIvNCOUH-fmI5JPA&hl=en&ei=1MCkTf2II4_1sgaROISBBw&sa=X&oi=book_result&ct=result&resnum=1&ved=0CBcQ6AEwAA#v=onepage&q&f=false.

[CLRS09] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms, Third Edition*. The MIT Press, 3rd edition, 2009.

[KL12] Mark Korenblit and Vadim E. Levit. A one-vertex decomposition algorithm for generating algebraic expressions of square rhomboids. *CoRR*, abs/1211.1661, 2012.

[Lei96] Leighton. Note on Better Master Theorems for Divide-and-Conquer Recurrences, 1996. Available online at http://courses.csail.mit.edu/6.046/spring04/handouts/akrabazzi.pdf.

[Man12] Krasimir Manev. *Uvod v Diskretnata Matematika*. KLMN – Krasimir Manev, fifth edition, 2012.

[SF96] Robert Sedgewick and Philippe Flajolet. *An introduction to the analysis of algorithms*. Addison-Wesley-Longman, 1996.