Алгоритми та складність

I семестр

Лекція 4

Рекурентні співвідношення

- *Рекурентне співвідношення* – рівняння або нерівність, яка описує функцію із використанням самої себе, але тільки з меншими аргументами:

$$x(n) = x(n-1) + n, \text{ при } n > 0,$$
$$x(0) = 0.$$

- Граничні (початкові) умови можуть вказуватися для значень *n* відмінних від 0 і їх може бути декілька (наприклад, для чисел Фібоначчі).
- Розв'язати рекурентне співвідношення означає вказати явну залежність послідовності від *n* з врахуванням початкових умов або ж довести, що її не існує.

Рекурентні співвідношення

- Наприклад, розв'язок розглянутого рекурентного співвідношення з заданою початковою умовою:

$$x(n) = \frac{n(n+1)}{2} \text{ для } n \geq 0.$$

- Безпосередньо підставивши його у вихідне співвідношення і початкову умову, можна переконатися: воно виконується для всіх $n > 0$:

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

а для початкової умови вірно $x(0) = 0$:

$$\frac{0(0+1)}{2} = 0.$$

Рекурентні співвідношення

- Строго кажучи, рекурентне співвідношення зазвичай має безліч розв'язків (*загальний розв'язок*). Врахувавши граничні умови, отримуємо *частковий розв'язок*.
- Загальний розв'язок містить одну або декілька констант, присвоюючи значення яким ми отримаємо всі розв'язки співвідношення (і тільки їх).
- Наприклад, загальний розв'язок розглянутого рекурентного співвідношення має вигляд

$$x(n) = c + \frac{n(n+1)}{2}.$$

Рекурентні співвідношення

- В нашому випадку під розв’язком рекурентного співвідношення вважатимемо або явний вигляд залежності послідовності від *n*, або асимптотичні оцінки розв’язку.
- Найчастіше припускають, що всі *n* – цілочисельні (або натуральні).
- Вважають, що для малих *n* (граничні умови) час виконання Θ(1).
- Не існує універсального методу розв’язання довільних рекурентних співвідношень (ніби можливо розв’язати набагато простіше рівняння загального вигляду $f(x) = 0$!).

Загальний план аналізу ефективності рекурсивних алгоритмів

1. Виберіть параметр (або параметри), за яким буде оцінюватися розмір вхідних даних алгоритму.

2. Визначте основну операцію алгоритму.

3. Перевірте, чи залежить число виконуваних основних операцій лише від розміру вхідних даних. За необхідності розгляньте, як змінюється ефективність алгоритму для найгіршого, середнього і найкращого випадків.

4. Складіть рекурентне рівняння, що виражає кількість виконуваних основних операцій алгоритму і вкажіть відповідні початкові умови.

5. Розв’яжіть отримане рекурентне співвідношення або, якщо це неможливо, визначте хоча б його порядок зростання.

Розв’язання рекурентних співвідношень. Метод підстановок

- Робиться здогад про вигляд розв’язку.
- За матіндукцією визначаються константи і показується правильність розв’язку.
- Назва означає, що розв’язок-припущення *підставляється* в рекурентне співвідношення.
- Основний недолік методу – необхідно передбачити розв’язок, що далеко не завжди можливо.
- Методом можна визначати верхню або нижню границі рекурентного співвідношення.

Метод підстановок

<u>Приклад</u>. Знайдемо верхню границю рекурентного співвідношення

$$T(n) = 2T(\lfloor n/2 \rfloor) + n$$

Припустимо, розв’язок має вигляд $T(n) = O(n \lg n)$.

Доведемо, що при деякій константі *c>0* виконується $T(n) \leqslant cn \lg n.$ Нехай вона виконується і для [*n/2*], тобто $T(\lfloor n/2 \rfloor) \leqslant c \lfloor n/2 \rfloor \lg(\lfloor n/2 \rfloor)$.

Підставимо вираз у рекурентне співвідношення:

$$\begin{aligned} T(n) &\leqslant 2(c\lfloor n/2 \rfloor \lg(\lfloor n/2 \rfloor)) + n \leqslant \\ &\leqslant cn \lg(n/2) + n = \\ &= cn \lg n - cn \lg 2 + n = \\ &= cn \lg n - cn + n \leqslant \\ &\leqslant cn \lg n, \end{aligned}$$

останнє виконується при *c≥1*.

Метод підстановок

Тепер слід показати, що розв’язок справедливий для граничних умов, тобто показати існування константи *c* такої, щоб співвідношення $T(n) \leqslant cn \lg n$ виконувалося і для граничних умов.

Припустимо, маємо граничну умову $T(1)=1$, однак наше співвідношення при $n=1$ не виконується.

Але при використанні асимптотичних позначень співвідношення має виконуватися для всіх $n \geq n_0$, тому можна вибрати потрібне n_0.

Виберемо $n_0=2$. Тоді базою індукції будуть $T(2)=4$ та $T(3)=5$.

Для нерівностей $T(2) \leqslant 2c \lg 2$ та $T(3) \leqslant 3c \lg 3$.

можна взяти $c \geq 2$.

Заміна змінних

Приклад. Розглянемо співвідношення

$$T(n) = 2T\left(\lfloor\sqrt{n}\rfloor\right) + \lg n$$

Введемо заміну $m = \lg n$:

$$T(2^m) = 2T\left(2^{m/2}\right) + m.$$

Візьмемо $S(m) = T(2^m)$ і отримаємо нове рекурентне співвідношення

$$S(m) = 2S(m/2) + m,$$

Воно має порядок $S(m) = O(m \lg m)$.

Після зворотної заміни отримуємо:

$$T(n) = T(2^m) = S(m) = O(m \lg m) = O(\lg n \lg \lg n).$$

Зворотна підстановка

- Використовуючи задане рекурентне співвідношення *T(n)*, виражаємо *T(n–1)* як функцію від *T(n–2)*. Результат підставляємо в початкове рівняння. Отримали *T(n)* як функцію від *T(n–2)*.
- Повторюємо до деякого *i*, виражаючи *T(n)* як функцію від *T(n– i)* і виявляючи залежності.
- Вибравши *i* так, щоб *(n– i)* досягало граничної умови і використавши одну із формул сумування, може вдатися отримати залежність у явному вигляді.
- Метод на диво добре працює для великої кількості нескладних рекурентних рівнянь.

Зворотна підстановка

Приклад. Розглянемо алгоритм

```
АЛГОРИТМ  BinRec (n)
// Вхідні дані:          ціле додатне число n
// Вихідні дані:         кількість розрядів в двійковому
//                       представленні числа n
if  n  = 1
  return  1
else
  return  BinRec(⌊n/2⌋) + 1
```

Запишемо рекурентне співвідношення для обчислення кількості операції додавання.

$$A(n) = A(\lfloor n/2 \rfloor) + 1 \text{ при } n > 1,$$

$$A(1) = 0.$$

Будемо шукати розв'язки тільки для $n = 2^k$. Зроблена оцінка буде з високою степінню точності відповідати всім великим значенням n.

Зворотна підстановка

$$A\left(2^k\right) = A\left(2^{k-1}\right) + 1 \quad \text{при } k > 0,$$
$$A\left(2^0\right) = 0.$$

За методом зворотної підстановки:

$$A\left(2^k\right) = A\left(2^{k-1}\right) + 1 =$$ підставляємо $A\left(2^{k-1}\right) = A\left(2^{k-2}\right) + 1$

$$= \left[A\left(2^{k-2}\right) + 1\right] + 1 =$$

$$= A\left(2^{k-2}\right) + 2 =$$ підставляємо $A\left(2^{k-2}\right) = A\left(2^{k-3}\right) + 1$

$$= \left[A\left(2^{k-3}\right) + 1\right] + 2 =$$

$$= A\left(2^{k-3}\right) + 3 =$$ ...

$$\dots$$

$$= A\left(2^{k-i}\right) + i =$$

$$\dots$$

$$= A\left(2^{k-k}\right) + k.$$

Зворотна підстановка

В результаті отримуємо

$$A\left(2^k\right) = A(1) + k = k.$$

Оскільки $n = 2^k$, то $k = \log_2 n$, тому

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

(Точним розв'язком буде $A(n) = \lfloor \log_2 n \rfloor$.)

Лінійні рекурентні співвідношення 2-го порядку зі сталими коефіцієнтами

- $ax(n) + bx(n-1) + cx(n-2) = f(n)$, де a, b, c – дійсні числа, причому $a \neq 0$.
- Клас рекурентних співвідношень, що не може бути розв'язаний через підстановки.
- Однорідне рекурентне співвідношення:

$$ax(n) + bx(n-1) + cx(n-2) = 0$$

- **Теорема**. Загальний розв'язок неоднорідного рівняння можна отримати як суму загального розв'язку відповідного однорідного рівняння і часткового розв'язку неоднорідного.

Лінійні рекурентні співвідношення 2-го порядку зі сталими коефіцієнтами

- Розглянемо однорідний випадок
$$ax(n) + bx(n-1) + cx(n-2) = 0$$
- Його характеристичне рівняння $ar^2 + br + c = 0$ нехай має корені r_1 та r_2.

<u>Випадок 1</u>. Корені r_1 та r_2 дійсні та різні. Тоді загальний розв’язок рекурентного співвідношення

$x(n) = \alpha {r_1}^n + \beta {r_2}^n,$ де $\propto, \beta$ – дійсні константи.

<u>Випадок 2</u>. Корені r_1 та r_2 однакові. Тоді загальний розв’язок рекурентного співвідношення

$x(n) = \alpha r^n + \beta n r^n,$ де $r = r_1 = r_2,$ а $\propto, \beta$ – дійсні.

Лінійні рекурентні співвідношення 2-го порядку зі сталими коефіцієнтами

Випадок 3. Якщо $r_{1,2} = u \pm iv$ – два різних комплексних кореня, то загальний розв'язок рекурентного співвідношення

$x(n) = \gamma^n(\alpha \cos n\theta + \beta \sin n\theta)$, де $\propto$, β – довільні дійсні константи, $\gamma = \sqrt{u^2 + v^2}$, $\theta = \mathrm{arctg}(v/u)$.

Для прикладу розв'яжемо рекурентне співвідношення

$$x(n) - 6x(n-1) + 9x(n-2) = 0$$

Лінійні рекурентні співвідношення 2-го порядку зі сталими коефіцієнтами

Приклад. $x(n) - 6x(n-1) + 9x(n-2) = 0.$

Характеристичне рівняння $r^2 - 6r + 9 = 0$ має однакові корені $r_1 = r_2 = 3.$

Маємо випадок 2 – загальний розв'язок буде

$$x(n) = \alpha 3^n + \beta n 3^n$$

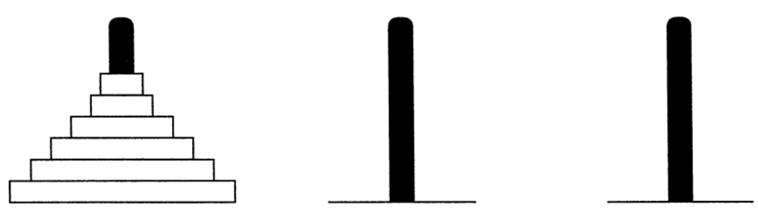
Нехай дані початкові умови $x(0) = 0, x(1) = 3.$

Підставляємо $n = 0$ та $n = 1$ в загальний розв'язок і отримуємо систему двох лінійних рівнянь з двома невідомими. Для наших початкових умов маємо $\alpha = 0$ та $\beta = 1$, отже шуканий частковий розв'язок

$$x(n) = n3^n$$

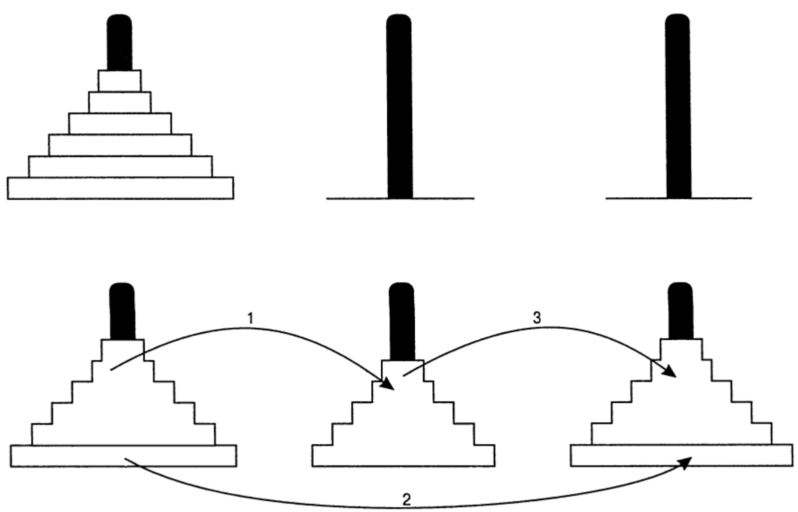
Задача про ханойські вежі

- Є *n* дисків різного діаметру та 3 кілочки. Спочатку всі диски нанизані на перший кілок і впорядковані за діаметром від найбільшого (знизу) до найменшого (зверху).
- Треба перенести всі диски на третій кілок, використовуючи другий як допоміжний.
- За один раз переміщується один диск.
- Не можна класти більший диск на менший.



Задача про ханойські вежі

- Щоб перенести *n>1* дисків з першого кілочка на третій (другий допоміжний), треба спочатку рекурсивно перенести (*n–1*) диск на кілок 2 (третій допоміжний).
- Потім перемістити найбільший диск з кілка 1 на кілок 3.
- Нарешті рекурсивно перенести (*n–1*) диск з другого на третій кілок (перший допоміжний).
- При *n=1* безпосередньо переносимо диск з кілка 1 на кілок 3.



Ілюстрація рекурсивного розв’язку задачі

Задача про ханойські вежі

Тоді кількість переносів дисків для *n>1* можна визначити так:

$$M(n) = M(n-1) + 1 + M(n-1).$$

З додаванням граничної умови *M(1)=1* отримуємо рекурентне співвідношення

$$M(n) = 2M(n-1) + 1 \quad \text{для } n > 1,$$
$$M(1) = 1.$$

Використаємо метод зворотних підстановок:

$$\begin{aligned} M(n) &= 2M(n-1) + 1 = \\ &= 2[2M(n-2) + 1] + 1 = \\ &= 2^2 M(n-2) + 2 + 1 = \\ &= 2^2 [2M(n-3) + 1] + 2 + 1 = \\ &= 2^3 M(n-3) + 2^2 + 2 + 1 \end{aligned}$$

підставимо

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

підставимо

$$M(n-2) = 2M(n-3) + 1$$

Задача про ханойські вежі

Для *i* отримаємо

$$M(n) = 2^i M(n-i) + 2^{i-1} + 2^{i-2} + \cdots + 2 + 1 = 2^i M(n-i) + 2^i - 1.$$

Підставимо $i = n - 1$ (початкові умови) і отримаємо розв'язок рекурентного співвідношення:

$$M(n) = 2^{n-1} M(n-(n-1)) + 2^{n-1} - 1 =$$
$$= 2^{n-1} M(1) + 2^{n-1} - 1 = 2^{n-1} + 2^{n-1} - 1 = 2^n - 1.$$

Отже, алгоритм працює за експоненціальний час.

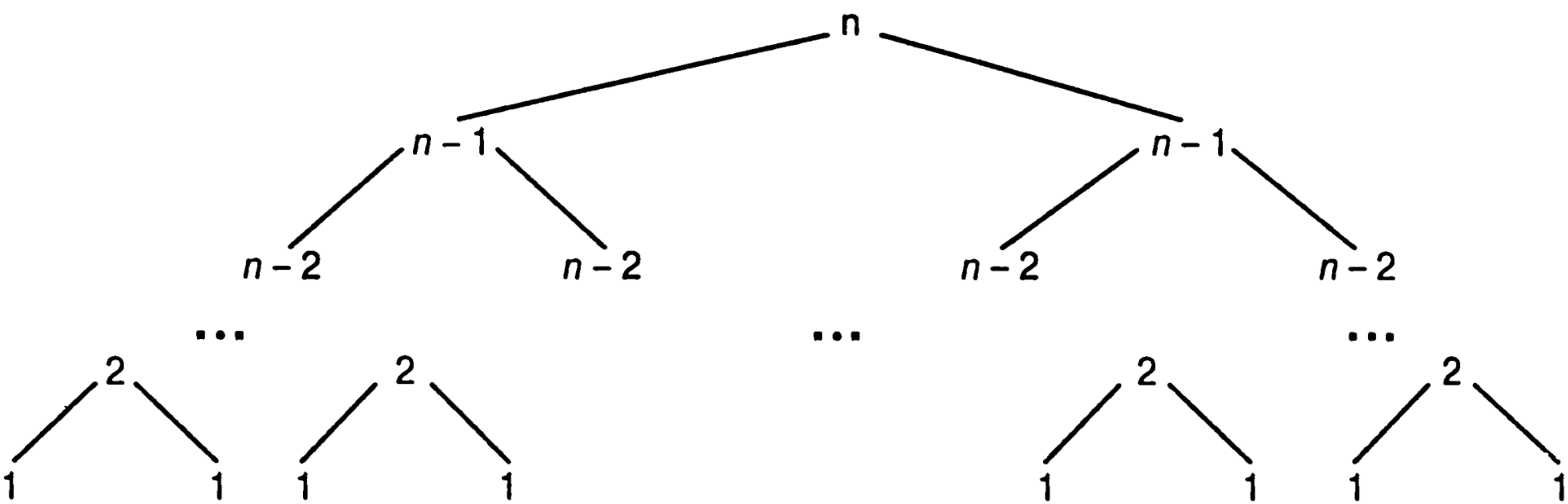
Незважаючи на це, розглянутий алгоритм буде найефективнішим з усіх можливих – складність лежить у самій задачі.

Метод дерев рекурсії

- Якщо в рекурсивному алгоритмі відбувається більше одного рекурсивного виклику, то в процесі аналізу є сенс побудувати дерево його рекурсивних викликів.
- Вузли такого дерева відповідатимуть рекурсивним викликам. Кожен вузол представляє час, необхідний для виконання окремо взятої підзадачі, яка розв'язується при одному з рекурсивних викликів функцій.
- Далі значення часу роботи окремих етапів підсумовуються в межах кожного рівня, а потім – по всіх рівнях дерева, в результаті чого отримуємо повний час роботи алгоритму.
- Дерева рекурсії часто використовуються при розгляді рекурентних співвідношень, що описують час роботи алгоритмів на основі підходу «розділяй та владарюй».

Метод дерев рекурсії

Дерево рекурсивних викликів для задачі про ханойські вежі:



Загальна кількість вузлів (викликів):

$$C(n) = \sum_{l=0}^{n-1} 2^l = 2^n - 1$$

де l – номер рівня в дереві.

Метод дерев рекурсії

- Дерева рекурсії використовуються як безпосередньо для доведення коректності розв'язку, так і для його прикидки з наступним застосуванням методу підстановок.

Приклад. Спробуємо припустити, як виглядає розв'язок рекурентного співвідношення $T(n) = 3T(\lfloor n/4 \rfloor) + \Theta(n^2)$.

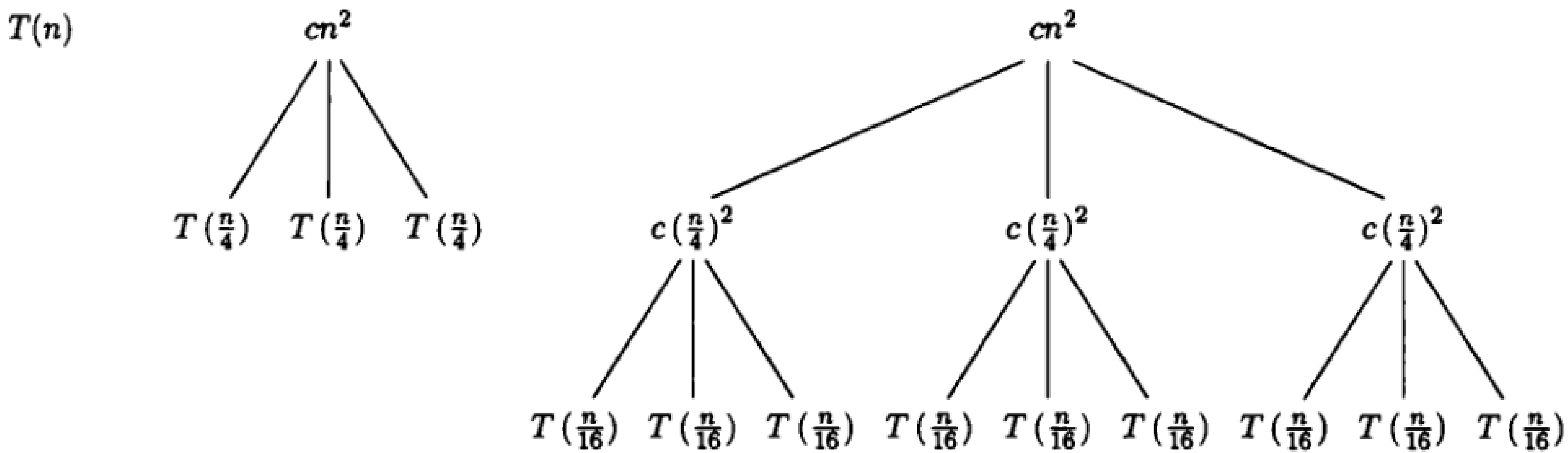
Шукаємо верхню границю розв'язку.

Можна опустити взяття цілої частини як (зазвичай) несуттєву і будувати дерево для

$$T(n) = 3T(n/4) + cn^2, \text{ при } c > 0.$$

Для зручності також припустимо, що n – степінь 4, так що розміри всіх підзадач цілі.

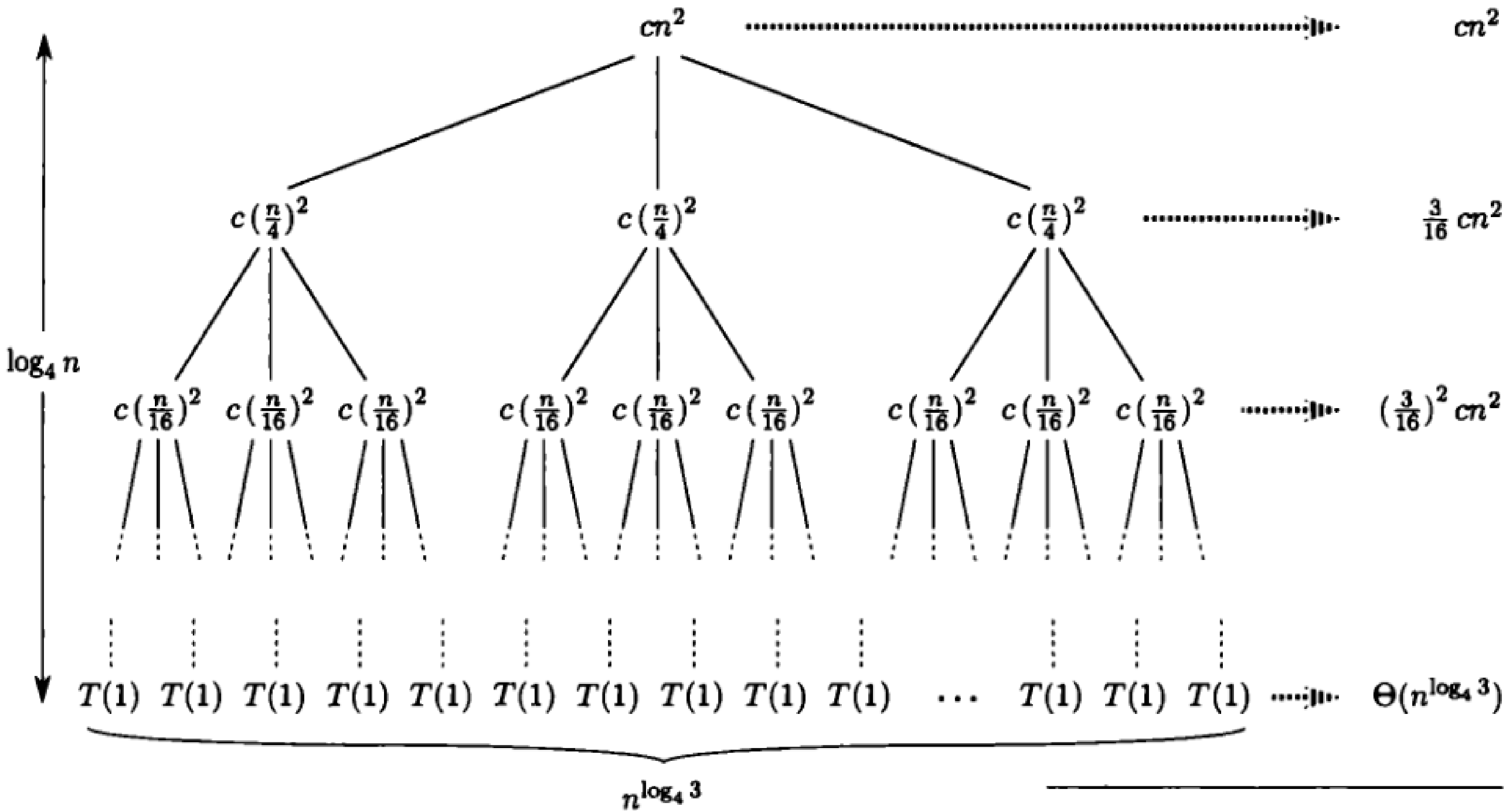
Метод дерев рекурсії



Поступовий процес побудови дерева рекурсії для

$$T(n) = 3T(n/4) + cn^2$$

Метод дерев рекурсії



Усього: $O(n^2)$

Дерево рекурсії для $T(n) = 3T(n/4) + cn^2$

Метод дерев рекурсії

Покажемо, що загальний час роботи дорівнює $O(n^2)$:

$$
\begin{aligned}
T(n) &= cn^2 + \frac{3}{16}cn^2 + \left(\frac{3}{16}\right)^2 cn^2 + \cdots + \left(\frac{3}{16}\right)^{\log_4 n - 1} cn^2 + \Theta(n^{\log_4 3}) \\
&= \sum_{i=0}^{\log_4 n - 1} \left(\frac{3}{16}\right)^i cn^2 + \Theta(n^{\log_4 3}) \\
&< \sum_{i=0}^{\infty} \left(\frac{3}{16}\right)^i cn^2 + \Theta(n^{\log_4 3}) \\
&= \frac{1}{1-(3/16)} cn^2 + \Theta(n^{\log_4 3}) \\
&= \frac{16}{13} cn^2 + \Theta(n^{\log_4 3}) \\
&= O(n^2)\ .
\end{aligned}
$$

Отже, припущення щодо розв'язку початкового співвідношення $T(n) = 3T(\lfloor n/4 \rfloor) + \Theta(n^2)$:

$$T(n) = O(n^2).$$

Метод дерев рекурсії

Переконаємося, що наше припущення вірне за допомогою методу підстановок.

Покажемо, що $T(n) \le dn^2$ для константи $d > 0$. Взявши з попередньої побудови те саме $c > 0$, маємо

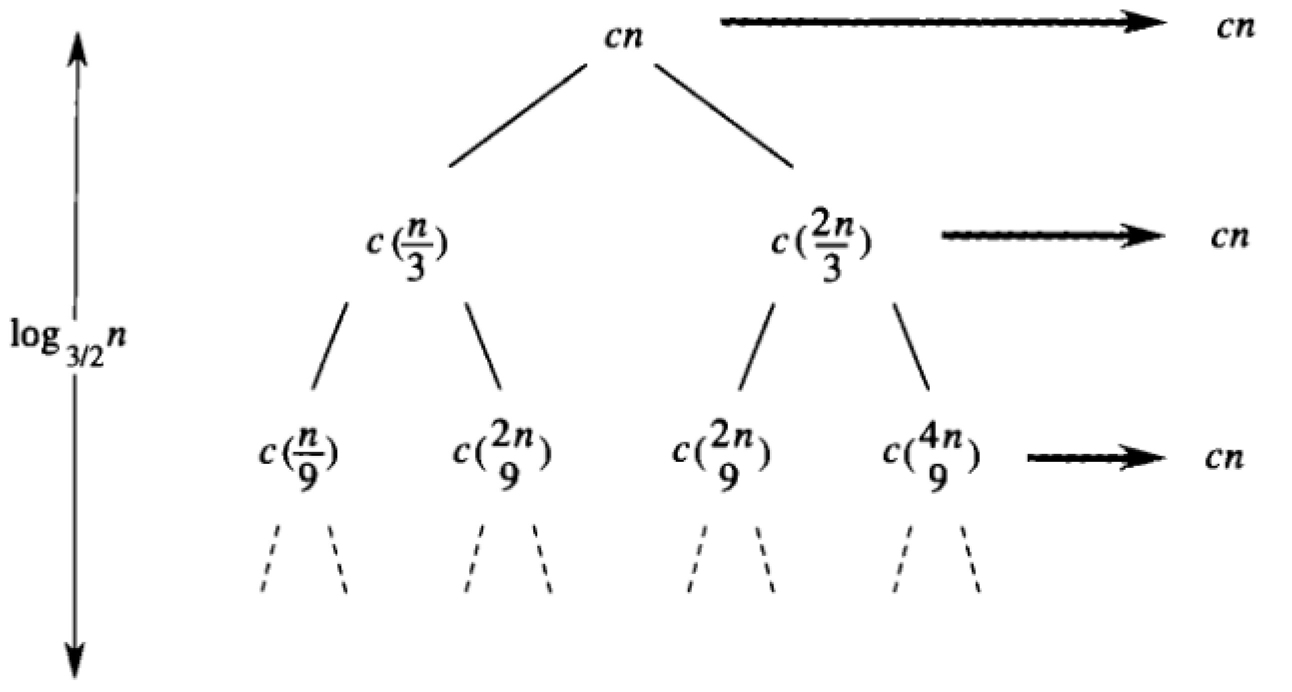
$$
\begin{aligned}
T(n) &\le 3T(\lfloor n/4 \rfloor) + cn^2 \\
&\le 3d\lfloor n/4 \rfloor^2 + cn^2 \\
&\le 3d(n/4)^2 + cn^2 \\
&= \frac{3}{16}dn^2 + cn^2 \\
&\le dn^2 ,
\end{aligned}
$$

де останнє справджується при $d \ge (16/13)c$.

Насправді $O(n^2)$ буде асимптотично точною оцінкою для розглянутого рекурентного співвідношення.

Метод дерев рекурсії

<u>Приклад</u>. Нехай будуємо дерево рекурсії для співвідношення $T(n) = T(n/3) + T(2n/3) + O(n)$.



Усього: $O(n \log n)$

Дерево рекурсії для $T(n) = T(n/3) + T(2n/3) + cn$

Метод дерев рекурсії

Висота дерева дорівнює $\log_{3/2} n$.

Розв'язок співвідношення не перевищить

$$O(cn \log_{3/2} n) = O(n \lg n).$$

Однак це бінарне дерево не буде повним, і чим далі від кореня, тим більша кількість внутрішніх вузлів буде відсутня.

Значить, не всі рівні дерева даватимуть вклад *cn*, час роботи на менших рівнях буде зменшуватись.

Метод дерев рекурсії

Але спробуємо не робити більше підрахунків і перевіримо методом підстановок розв'язок вигляду $O(n \lg n)$.

Покажемо: $T(n) \leq dn \lg n$ для константи $d > 0$.

$$\begin{aligned}
T(n) &\leq T(n/3) + T(2n/3) + cn \\
&\leq d(n/3)\lg(n/3) + d(2n/3)\lg(2n/3) + cn \\
&= (d(n/3)\lg n - d(n/3)\lg 3) \\
&\quad + (d(2n/3)\lg n - d(2n/3)\lg(3/2)) + cn \\
&= dn\lg n - d((n/3)\lg 3 + (2n/3)\lg(3/2)) + cn \\
&= dn\lg n - d((n/3)\lg 3 + (2n/3)\lg 3 - (2n/3)\lg 2) + cn \\
&= dn\lg n - dn(\lg 3 - 2/3) + cn \\
&\leq dn\lg n
\end{aligned}$$

при $d \geq c/(\lg 3 - (2/3))$.

Як бачимо, більш грубої оцінки по дереву виявилось достатньо.

Типові рекурентні співвідношення при аналізі алгоритмів. Зменшення на одиницю

- Алгоритм розв’язує задачу на основі співвідношення між даним екземпляром розміром *n* та меншим екземпляром розміром (*n*–1).
- Приклади: рекурсивне обчислення *n*!, сортування вставками.
- $T(n) = T(n-1) + f(n)$, де $f(n)$ враховує час для переходу до меншого екземпляру задачі і повернення від розв’язання меншого екземпляру до розв’язання більшого.

Типові рекурентні співвідношення при аналізі алгоритмів. Зменшення на одиницю

Зворотна підстановка дає

$$T(n) = T(0) + \sum_{j=1}^{n} f(j)$$

В конкретних випадках сума або обчислюється точно, або шукається порядок її зростання.

Зокрема,

- при $f(n) = 1$, тоді $\sum_{j=1}^{n} f(j) = n$;
- при $f(n) = \log n$, тоді $\sum_{j=1}^{n} f(j) \in \Theta(n \log n)$;
- при $f(n) = n^k$, тоді $\sum_{j=1}^{n} f(j) \in \Theta(n^{k+1})$.

Типові рекурентні співвідношення при аналізі алгоритмів. Зменшення на постійний множник

- Алгоритм розв’язує задачу приводячи її екземпляр розміром *n* до екземпляру розміром *n*/*b* (найчастіше *b* = 2). Розв’язується менший екземпляр, і на основі цього отримується розв’язок більшого.
- Приклад: бінарний пошук.
- $T(n) = T(n/b) + f(n)$, де $b > 1$, а $f(n)$ враховує час для переходу до меншого екземпляру задачі і повернення від розв’язання меншого екземпляру до розв’язання більшого.

Типові рекурентні співвідношення при аналізі алгоритмів. Зменшення на постійний множник

- Строго кажучи, рекурентне рівняння буде коректним тільки для $n = b^k$, $k = 0,1, \dots$. Для інших значень застосовується округлення.
- Традиційно співвідношення розв'язується спочатку для $n = b^k$, а потім розв'язок певним чином розповсюджується на всі *n*.
- Загальний вигляд розв'язку буде

$$T\left(b^k\right) = T(1) + \sum_{j=1}^{k} f(b^j)$$

Типові рекурентні співвідношення при аналізі алгоритмів. Декомпозиція

- Алгоритм розв’язує задачу шляхом поділу даного екземпляру на декілька менших, рекурсивного їх розв’язання і комбінації отриманих розв’язків в розв’язок початкового екземпляра.
- Припускаємо, що всі менші екземпляри мають однаковий розмір *n*/*b* і їх розв’язується *a* штук.
- $T(n) = aT(n/b) + f(n)$, де $a > 0$, $b > 1$, а $f(n)$ враховує час для розбиття задачі на менші екземпляри і комбінування їх розв’язків.
- Приклади: алгоритми на основі підходу «розділяй та владарюй».

Типові рекурентні співвідношення при аналізі алгоритмів. Декомпозиція

Приклад. Отримаємо рекурентне співвідношення для верхньої оцінки часу *T(n)* виконання алгоритму сортування злитттям Merge_Sort:

Поділ: шукається середина підмасиву за фіксований час Θ(1)

Підкорення: рекурсивно розв’язуються обидві підзадачі розміру *n/2* кожна за час *2T(n/2)*

Комбінування: злиття *n*-елементного підмасиву за час Θ(*n*)

$$T(n) = \begin{cases} \Theta(1), & \text{при } n = 1, \\ 2T(n/2) + \Theta(n), & \text{при } n > 1. \end{cases}$$

При цьому Θ(1)+ Θ(*n*)= Θ(*n*)

Типові рекурентні співвідношення при аналізі алгоритмів. Декомпозиція

- Підхід зменшення задачі на постійний множник є частковим випадком декомпозиції.
- Рекурентне співвідношення

$$T(n) = aT(n/b) + f(n)$$

називають *загальним рекурентним співвідношенням декомпозиції*.

- Воно буде справедливим для $n = b^k,\ k = 0,1, \ldots$

Типові рекурентні співвідношення при аналізі алгоритмів. Декомпозиція

- Розв’язок рекурентного співвідношення матиме вигляд

$$T(n) = n^{\log_b a}\left(T(1) + \sum_{j=1}^{\log_b n} f(b^j)/a^j\right)$$

- Порядок зростання розв’язку $T(n)$ залежить від значень констант a, b та порядку зростання $f(n)$. Знання певних властивостей $f(n)$ дозволяє точно визначити порядок зростання $T(n)$.

Правило гладкості

- При розгляді часової ефективності алгоритмів для значень $n = b^k,\ k = 0,1,\ldots$ виникає питання, яким чином можна розповсюдити отриманий результат на всі n.
- Невід’ємну функцію $f(n)$ на множині ***N*** назвемо *неспадною в кінцевому рахунку*, якщо існує ціле n_0 таке, що $f(n)$ буде неспадною на $[n_0,\ \infty)$.
- Наприклад, функція $(n-100)^2$ буде неспадною в кінцевому рахунку, хоч і спадає на відрізку $[1,\ 100]$. В той же час функція $\sin^2 \pi n/2$ не є неспадною в кінцевому рахунку.
- Більшість функцій, які з’являються при аналізі алгоритмів, мають цю властивість.

Правило гладкості

- Невід’ємну функцію $f(n)$ на множині ***N*** назвемо *гладкою*, якщо вона неспадна в кінцевому рахунку та $f(2n) = \Theta\big(f(n)\big)$.
- Функції, що ростуть не дуже швидко, є гладкими, зокрема $\log n$, n, $n \log n$ та n^{α} ($\alpha \geq 0$). Наприклад:
$$f(2n) = 2n\log 2n = 2n(\log 2 + \log n) =$$
$$= (2\log 2)n + 2n\log n = \Theta(n\log n).$$
- Функції, які ростуть швидко, такі як a^n ($a > 1$) та $n!$, не є гладкими. Зокрема:
$$f(2n) = 2^{2n} = 4^n \neq \Theta(2^n).$$

Правило гладкості

- Нехай функція $f(n)$ гладка. Тоді для довільного цілого $b \geq 2$ справджується $f(bn) = \Theta\big(f(n)\big)$.
- <u>**Правило гладкості.**</u> Нехай $\mathrm{T}(n)$ − неспадна в кінцевому рахунку функція, а $f(n)$ − гладка функція. Тоді якщо $T(n) = \Theta\big(f(n)\big)$ для всіх $n = b^k$, $k = 0,1,\dots$ для деякого $b \geq 2$, то $T(n) = \Theta\big(f(n)\big)$ для всіх n.
- Таким чином, правило дозволяє розповсюдити інформацію про порядок зростання функції, отриману для степенів b, на всю її область визначення.

Основний метод (Master theorem)

Розв’язується рекурентне співвідношення загального вигляду

$$T\left(n\right)=aT\left(n/b\right)+f\left(n\right)$$

Тут $a \geq 1$, $b > 1$ – константи, а $f(n)$ – асимптотично додатна функція.

<u>Основна теорема</u>. Асимптотична поведінка функції *T(n)* може бути описана наступним чином:

1. Якщо $f(n) = O(n^{\log_b a - \varepsilon})$ для деякої константи $\varepsilon > 0$, то $T(n) = \Theta\left(n^{\log_b a}\right)$.
2. Якщо $f(n) = \Theta\left(n^{\log_b a}\right)$, то $T(n) = (n^{\log_b a} \cdot \lg n)$.
3. Якщо $f(n) = \Omega(n^{\log_b a + \varepsilon})$ для деякої константи $\varepsilon > 0$ і якщо $af(n/b) \leq cf(n)$ для деякої константи $c < 1$, то $T(n) = \Theta(f(n))$.

Основний метод

- В першому випадку необхідно, щоб функція $f(n)$ була *поліноміально меншою* за функцію $n^{\log_b a}$, тобто асимптотично менша в n^{ε} разів, де ε – деяка додатна константа.
- Так само у третьому випадку необхідно, щоб функція $f(n)$ була *поліноміально більшою* за функцію $n^{\log_b a}$ і крім того задовольняти умову регулярності $af(n/b) \leq cf(n)$.
- Однак функція $f(n)$ може виявитися меншою (більшою) за функцію $n^{\log_b a}$, але не поліноміально, або для третього випадку не виконуватиметься умова регулярності – тоді основний метод буде незастосовний.

Основний метод

<u>Приклад 1</u>. Розглянемо рекурентне співвідношення

$$T(n) = 9T(n/3) + n$$

В цьому випадку $a = 9, b = 3, f(n) = n$,

тому $n^{\log_b a} = n^{\log_3 9} = \Theta(n^2)$.

Оскільки $f(n) = O(n^{\log_3 9 - \varepsilon})$, де $\varepsilon = 1$, маємо випадок 1 теореми, тому $T(n) = O(n^2)$.

<u>Приклад 2</u>. Розглянемо рекурентне співвідношення

$$T(n) = T(2n/3) + 1,$$

Тут $a = 1, b = 3/2, f(n) = 1$, а $n^{\log_b a} = n^{\log_{3/2} 1} = n^0 = 1$.

Отримали випадок 2, оскільки $f(n) = \Theta(n^{\log_b a}) = \Theta(1)$.

Тому $T(n) = \Theta(\lg n)$.

Основний метод

Приклад 3. Розглянемо рекурентне співвідношення

$$T(n) = 3T(n/4) + n \lg n$$

Маємо $a = 3, b = 4, f(n) = n \lg n$,

тут $n^{\log_b a} = n^{\log_4 3} = \mathrm{O}(n^{0.793})$.

Оскільки $f(n) = \Omega(n^{\log_4 3+\varepsilon})$, де $\varepsilon \approx$ 0.2, перевіряємо випадок 3. Умова регулярності

$$af(n/b) = 3(n/4)\lg(n/4) \leq (3/4)\, n \lg n = cf(n)$$

виконується при $c = 3/4$. Тому $T(n) = \Theta(n \lg n)$.

Основний метод

<u>Приклад 4</u>. Розглянемо рекурентне співвідношення

$$T(n) = 2T(n/2) + n \lg n$$

Тут $a = 2, b = 2, f(n) = n \lg n$ та $n^{\log_b a} = n$.

Однак співвідношення $f(n)/n^{\log_b a} = n \lg n / n = \lg n$ асимптотично менше функції n^{ε} для довільної додатної константи ε. Тому розглянуте рекурентне співвідношення знаходиться «між» випадками 2 і 3, і основна теорема тут незастосовна.

Запитання і завдання

- Розв’яжіть рекурентне співвідношення

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

через заміну змінних. Розв’язок має бути асимптотично точним.

- За допомогою дерева рекурсії визначте асимптотичну верхню границю рекурентного співвідношення

$$T(n) = 3T(\lfloor n/2 \rfloor) + n.$$

Перевірте відповідь методом підстановок.

- За допомогою дерева рекурсії доведіть, що розв’язок рекурентного співвідношення

$$T(n) = T(n/3) + T(2n/3) + cn,$$

де *c* – константа, поводить себе як $\Omega(n \lg n)$.

Запитання і завдання

- За допомогою основної теореми знайдіть точні асимптотичні оцінки рекурентних співвідношень:

$$\text{а) } T(n) = 4T(n/2) + n.$$

$$\text{б) } T(n) = 4T(n/2) + n^2.$$

$$\text{в) } T(n) = 4T(n/2) + n^3.$$

- Побудуйте рекурентне співвідношення для кількості викликів функції при рекурсивному обчисленні $n!$ і знайдіть його розв'язок.
- Розробіть рекурсивний алгоритм обчислення 2^n для довільного невід'ємного n на основі формули

$$2^n = 2^{n-1} + 2^{n-1}$$

Побудуйте рекурентне співвідношення для кількості операцій додавання, що виконуються в алгоритмі і розв'яжіть його. Зобразіть дерево рекурсивних викликів для алгоритму і підрахуйте кількість викликів рекурсивної функції.

Запитання і завдання

- Розглянемо рекурсивний алгоритм обчислення суми кубів перших *n* цілих чисел $S(n) = 1^3 + 2^3 + \cdots + n^3$:

```
АЛГОРИТМ  S (n)
// Вхідні дані:        ціле додатне число n
// Вихідні дані:       сума кубів перших n цілих чисел
if  n  = 1
  return  1
else
  return  S(n − 1) + n * n * n
```

Побудуйте рекурентне співвідношення для кількості виконання основної операції алгоритму і знайдіть його розв’язок. Порівняйте цей алгоритм з простим нерекурсивним алгоритмом.