1 Знайомство з алгоритмами

У цьому розділі

- Ти опановуєш основу для розуміння решти книжки.
- Пишеш свій перший алгоритм пошуку (бінарний пошук).
- Розбираєшся у часі виконання алгоритму (нотація O-велике).
- Знайомишся з основною технікою проєктування алгоритмів (рекурсією).

Введення

Алгоритм — це набір інструкцій для розв'язання задачі. Кожну ділянку коду можна назвати алгоритмом, але ця книжка розкриває найцікавіші його частини. Я обирав алгоритми, які або швидкі, або їх цікаво вирішувати, або й те, й інше. Ось деякі моменти:

- У Розділі 1 описано алгоритм бінарного пошуку й пояснено, як він може пришвидшити твій код. На одному прикладі ми бачимо, як необхідну кількість дій може бути зменшено з 4 мільярдів до 32!
- Пристрій GPS використовує алгоритми з теорії графів (вивчиш у розділі 6, 7 та 8), щоб розрахувати найкоротший шлях до місця твого призначення.

- Щоб написати ШІ алгоритм гри у шашки, можеш залучити динамічне програмування (розбираємо у розділі 9).

У кожному випадку я буду описувати алгоритм і наводити приклади.

Далі будуть пояснення про час виконання цього алгоритму в контексті нотації О-велике.

І, нарешті, дізнаємося, які ще задачі може бути вирішено завдяки наведеному алгоритму.

Що ти дізнаєшся про ефективність програмування

Гарні новини полягають у тому, що реалізація кожного алгоритму з цієї книжки напевно доступна твоєю рідною мовою, тож тобі не доведеться прописувати кожен алгоритм самостійно. Але ці реалізації даремні, якщо ти не розумієш суті оптимізації та компромісних рішень.

Із цією книжкою ти навчишся порівнювати ефективність різних алгоритмів: доцільніше застосувати алгоритми злиття — чи швидкого сортування? Краще використати масив — чи зв'язаний список? Саме лише використання різних структур сортування інформації може стати вирішальним.

Що ти дізнаєшся про розв'язання задач

Ти опануєш техніки розв'язування задач, які донині могли бути поза межами твого розуміння. Наприклад:

- Якщо тобі подобається розробляти комп'ютерні ігри, напиши ШІ програму, що буде слідувати за гравцем, використовуючи алгоритми з теорії графів.
- Ти навчишся писати систему рекомендацій на основі методу k-найближчих сусідів (k-nearest neighbour або алгоритм KNN).
- Певні задачі неможливо вирішити в контексті часу! У тій частині книжки, де описано NP-повні задачі,

пояснюється, як визначити такі задачі та як обрати оптимально наближений алгоритм.

Кажучи загалом, коли дочитаєш книжку, ти вже будеш знати деякі з найрозповсюдженіших алгоритмів. Надалі ти зможеш застосувати ці знання для глибшого вивчення конкретних алгоритмів для роботи з ШІ, базами даних і таке інше. Або ти зможеш випробувати себе у розв'язанні складніших задач на роботі.

Що тобі потрібно знати

На початку тобі знадобиться шкільна алгебра. А саме — розглянь ось цю функцію: $f(x) = x \times 2$. Скільки буде $f(5)$? Якщо твоя відповідь 10, із тобою все ок.

Крім того, цей розділ (як і всю книжку) буде легше розуміти, якщо ти володієш бодай одною мовою програмування. Усі приклади в книжці написано на Python. Якщо ти взагалі не знайомий із програмуванням, але хотів би навчитися, тоді обирай Python — чудова мова для початківців. Якщо знаєш іншу мову, скажімо Ruby, то теж не пропадеш.

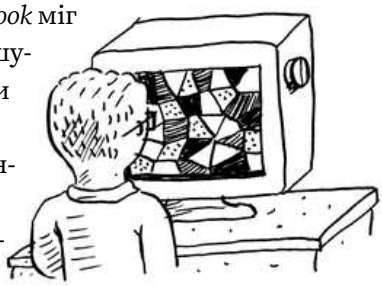
Бінарний пошук

Припустімо, ти шукаєш людину в телефонній книзі (яке старомодне речення вийшло!). Її ім'я починається на літеру К. Ти можеш почати з першої сторінки й гортати, поки не дістанешся до К-прізвищ. Та скоріше за все ти розгорнеш книжку ближче до середини, бо знаєш, що прізвища на «К» за порядком будуть десь посередині телефонної книги.

Чи, уявімо, ти шукаєш слово у словнику, а воно починається на літеру О. Знову ж таки, ти почнеш із середини.

А тепер спробуй авторизуватися на *Facebook*. Коли ти авторизуєшся, *Facebook* думає, що ти маєш акаунт на сайті. Тож йому треба знайти вказане ім'я у своїй базі даних.

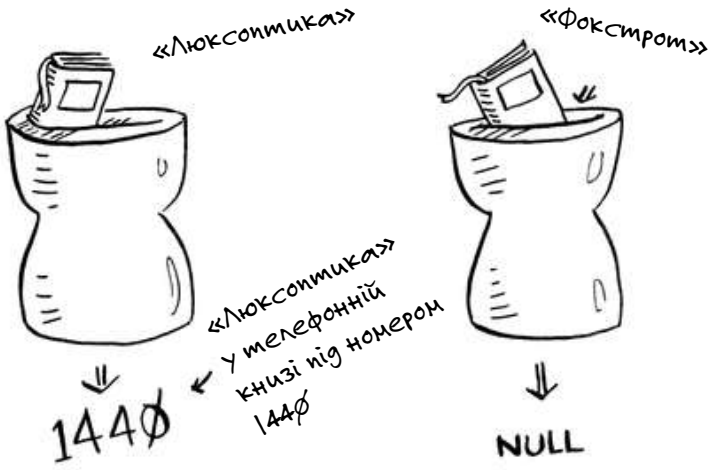
Впевнений, твоє ім'я karlmageddon. *Facebook* міг би почати з імен на літеру A і вже далі шукати твоє ім'я — але логічніше розпочати саме з середини.



Це і є проблема пошуку. І для розв'язання цієї задачі в усіх наведених прикладах використовується один і той самий алгоритм: бінарний пошук.

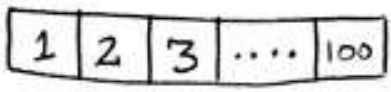
Бінарний пошук — це алгоритм; вхідними даними для цього алгоритму є сортовані елементи (поясню згодом, чому їх має бути посортовано). Якщо елемент, який ти шукаєш, є в цьому списку, бінарний пошук у відповідь на твій запит повертає його розташування. Якщо ж ні — пошуковий алгоритм поверне `null`.

Наприклад:



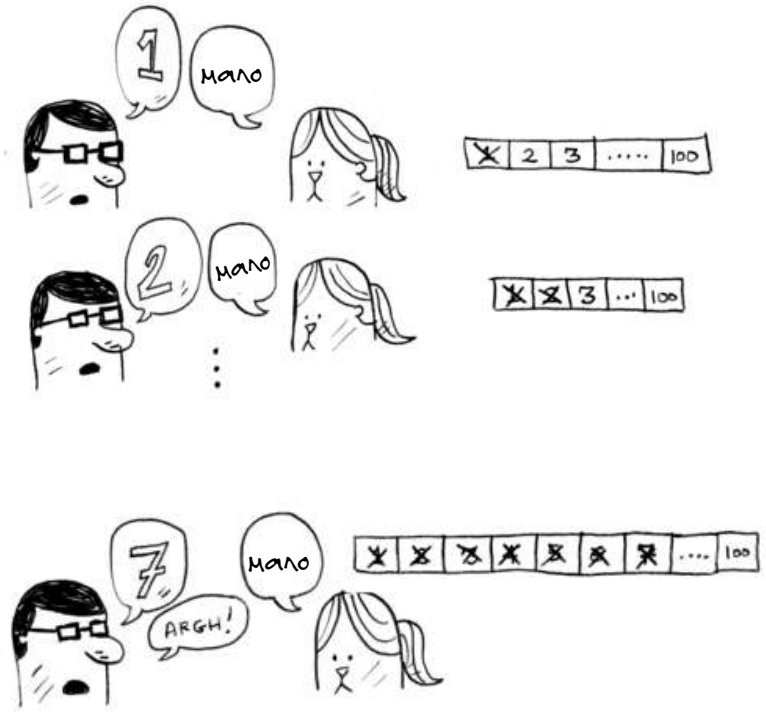
Шукаємо компанії в телефонній книзі за допомогою бінарного пошуку.

Розгляньмо докладніше, як працює алгоритм бінарного пошуку. Я загадую будь-яке число між 1 та 100.



А тобі треба вгадати це число за найменшу кількість спроб. Із кожним разом я буду спрямовувати тебе: припущення замале, завелике, або ж — ти вгадав число.

Скажімо, ти почав із такого варіанту: 1, 2, 3, 4... Ось як це виглядатиме.



Не найкращий спосіб вгадувати число.

Це називається *простим пошуком* (хоча «дурний пошук» звучало б доцільніше). Із кожною спробою ти закреслюєш лише одне число. Якби я загадав число 99, тобі знадобилося б 99 спроб, щоб його вгадати!

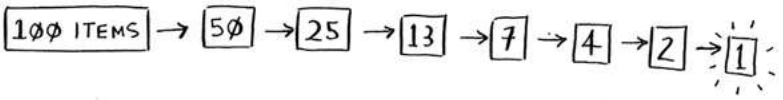
Кращий спосіб пошуку

Ось тобі кращий метод пошуку. Починаємо з числа 50.

Замало, але ти тільки що відкинув *половину* чисел! Тепер ти знаєш, що числа від 1 до 50 занадто малі й не треба кожне з них окремо перевіряти. Наступна спроба: 75.

Тепер цифра завелика, але ти знову позбувся половини чисел, що залишилися! *У бінарному пошуку ти перевіряєш серединне число з тих, що лишаються, і кожного разу відкидаєш одразу половину*. Тоді наступне число — 63 (половина між 50 та 75).

Ось тобі й бінарний пошук. Ти щойно вивчив свій перший алгоритм! Тільки подивися, скільки чисел ми можемо викреслити кожного наступного разу.

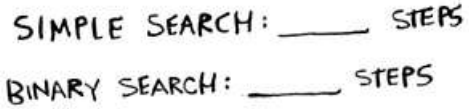


7 кроків

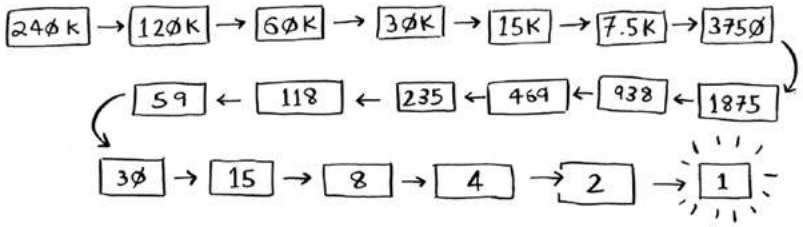
У бінарному пошуку прибираємо половину чисел кожного разу.

Яке б число я не загадав від 1 до 100, ти зможеш відгадати його щонайбільше за 7 спроб — саме тому, що кожного разу відкидаєш так багато чисел!

Уяви, що шукаєш якесь слово в словнику, в якому 240 000 слів. Як ти вважаєш, *у найгіршому випадку*, скільки знадобиться спроб для простого та бінарного пошуків?



У простому пошуку знадобиться 240 000 спроб, якщо слово, яке ти шукаєш, розташовано останнім. Зате з кожним кроком бінарного пошуку ти відкидаєш половину слів, поки не залишиться одне слово.

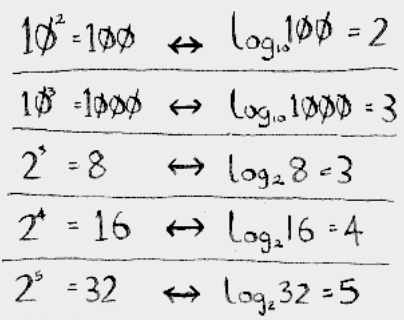


18 кроків

Виходить, що в бінарному пошуку тобі знадобиться лише 18 кроків, — просто уяви, це ж величезна різниця! Загалом, для будь-якого списку з кількістю елементів n бінарний пошук буде виконуватися за $\log_2 n$ кроків у найгіршому випадку, а простий пошук — n кроків.

Логарифми

Ти можеш не пам'ятати, що таке логарифми, але точно знаєш, що таке ступінь числа. Наприклад, $\log_{10}100$ — це, по суті, пошук ступеня, до якого треба піднести число-основу 10, щоб отримати 100. Відповідь: 2. Тобто, $10^2 = 10 \times 10$. Логарифм — це ніби обернена дія до піднесення в ступінь.



Логарифми — це знаходження ступеня числа.

У цій книжці, коли я говорю про час виконання у нотації О-велике (поясню трохи пізніше), log завжди значить $\log_2$ (тобто, число-основа логарифма це завжди 2). Коли ти шукаєш елемент, використовуючи простий пошук, то у найгіршому випадку доведеться переглянути кожен із них. Тобто, для списку з 8 елементів перевірити треба максимум 8 елементів. Для бінарного пошуку необхідно перевірити log *n* елементів, і це за найгіршого сценарію виконання алгоритму. Тому для нашого списку з 8 елементів, log 8 = 3, бо $2^3 = 8$. Виходить, що для списку з 8 чисел тобі треба перевірити не більше ніж 3. Для списку з 1024 елементів log 1024 = = 10, тому що $2^{10} = 1024$. Отже, для списку з 1024 чисел, треба перевірити не більше ніж 10 чисел.

Примітка

Я буду говорити про логарифмічний час дуже багато, тому важливо, щоб ти зрозумів його суть уже зараз. Якщо ж ні — на Khan Academy (khanacademy.org) є гарне відео, що допоможе тобі в цьому.

Примітка

Бінарний пошук працює лише тоді, коли список посортований. Наприклад, імена в телефонній книзі впорядковано в алфавітному порядку, тому ти можеш застосувати бінарний пошук, щоб знайти ім'я. Але що трапиться, коли імена будуть несортованими?

Погляньмо, як написати алгоритм бінарного пошуку на Python. У прикладі коду тут використовується масив. Якщо ти не знаєш, як саме вони працюють, — не хвилюйся, ми розберемося з ними у наступному розділі. Зараз ти повинен запам'ятати, що впорядковані елементи, розташовані у послідовних комірках, називаються масивом. Нумерація комірок починається з 0: перша комірка — це позиція #0, друга — #1, третя — #2 і так далі.

Функція binary_search приймає відсортований масив і число. Якщо число присутнє в масиві, то функція поверне його розташування. Ти будеш слідкувати, у якій саме частині масиву треба шукати. На початку пошук включає весь масив:

```
low = 0
high = len(list)-1
```

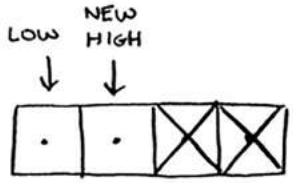
Кожного разу ти перевіряєш серединний елемент:

```
mid = (low + high) // 2
guess = list[mid]
```

mid — це середина, яку Python округлює автоматично, коли значення (low + high) непарне

Якщо припущення занизьке, значення `low` оновлюється відповідно:

```
if guess < item:
low = mid + 1
```



Але якщо припущення завелике, ти оновлюєш `high`. Ось повний код:

```
def binary_search(list, item):
    low = 0
    high = len(list)-1

    while low <= high:
        mid = (low + high) / 2
        guess = list[mid]
        if guess == item:
            return mid
        if guess > item:
            high = mid-1
        else:
            low = mid + 1
    return None

my_list = [1, 3, 5, 7, 9]

print binary_search(my_list, 3) # => 1
print binary_search(my_list,-1) # => None
```

значення low & high обмежують ту частину списку у якій ти будеш шукати

Поки ти не зменшив до одного елемента...

... перевір серединний елемент

Знайшов елемент

Припущення було завелике

Припущення було замале

Елемент не існує

Проведімо тест!

Пам'ятай, список починається з 0. Друга позиція має індекс 1

«None» — це nil у Python, тобто «нічого». Це значення показує, що елемент не знайдено

Вправи

1.1 Уявімо, що ти маєш відсортований список зі 128 імен і перебираєш його за допомогою бінарного пошуку. Якою буде максимальна кількість спроб?

1.2 А тепер збільшимо список удвічі. Яка максимальна кількість спроб тепер?