

Бинарный поиск

04/02/2023

Задача

- Есть массив из 0 и 1 вида 0000000011111111....
- Нужно найти место разделения 0 / 1

Модификации задачи

Если заменить 0 и 1 на значение `f(idx) -> bool`, то можно решать мощные задачи

Например:

```
def f(idx):  
    return a[idx] >= K
```

поможет найти в массиве первый элемент, больше или равный K

```
def f(idx):  
    return sum(a[0:idx]) >= K
```

поможет найти место, где **префиксная сумма** больше или равна K

**Функцию можно не писать явно*

Проверка

Какая из функций с прошлого слайда требует отсортированность массива?

Тривиальный подход

Идем слева направо по массиву, и ждем, когда найдем 1

```
for i in range(len(a)):  
    if f(i):  
        print(i)  
        break
```

Тривиальный подход

Идем слева направо по массиву, и ждем, когда найдем 1

```
for i in range(len(a)):  
    if f(i):  
        print(i)  
        break
```

Time Limit

Тривиальный подход: оптимизация

Идем слева направо и справа налево по массиву одновременно, и ждем, когда найдем замену 0 на 1

```
for i in range(len(a)):
    if f(i) or not f(len(a) - i - 1):
        print(i)
        break
```

Тривиальный подход: оптимизация

Идем слева направо и справа налево по массиву, и ждем, когда найдем замену 0 на 1

```
for i in range(len(a)):
    if f(i) or not f(len(a) - i - 1):
        print(i)
        break
```

Мы вроде бы и ускорили алгоритм, но это все еще **Time Limit**

Алгоритм: ключевая идея

Представьте себе, что вы ищете слово **Тинькофф** в словаре.

Вы открыли словарь на случайной странице и нашли там слово **Компьютер**.

"К" < "Т"

Значит, искать нужно точно во второй половине.

Если повторить много раз и делить примерно пополам, то мы быстро окажемся в ситуации, где разделение между 0 и 1 находится в очень маленьком промежутке.

<https://youtu.be/DSffdCT5Cx4>

кстати, зацените cs50

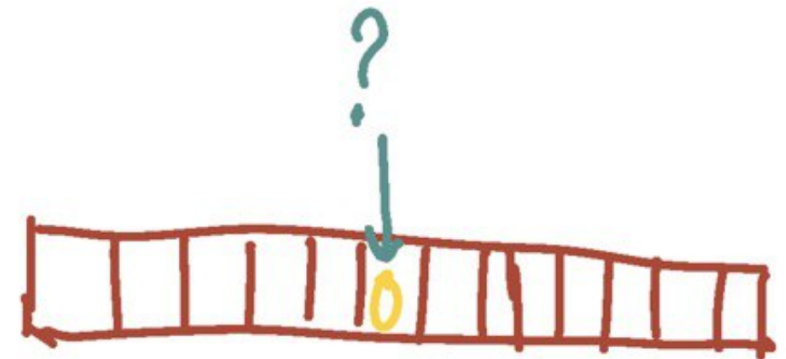
Пример

Изначальный массив



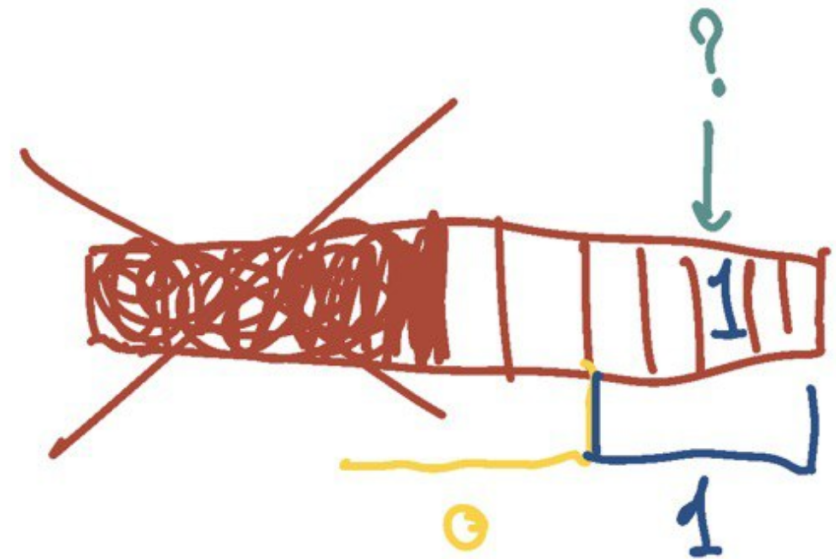
Пример

Смотрим на значение в точке



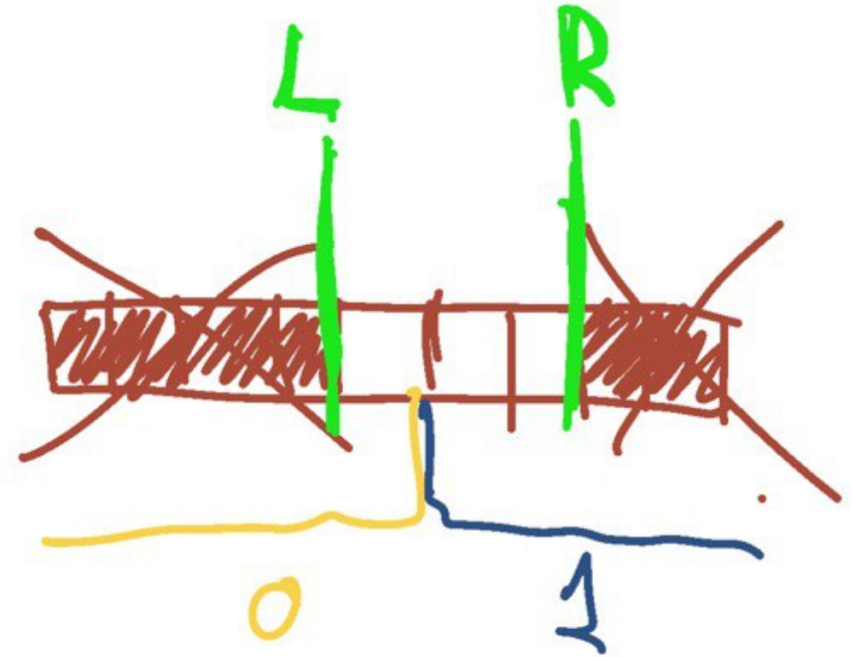
Пример

Отсекаем все, что нам больше не интересно, делаем следующий "запрос"



Пример

Получаем новую ограниченную область - ответ лежит между L и R



Псевдокод

```
l = 0
r = len(arr)
while l + 1 < r:
    m = (l + r) // 2
    if arr[m] == 0:
        l = m
    else:
        r = m

print(f'First "1" has index {r}')
```

Сравнение алгоритмов

Наверное, нам хочется сказать, что алгоритм бинарного поиска лучше тривиального подхода (*линейного поиска*).

Но что это значит? И в какой ситуации лучше?

Асимптотика

Если алгоритм имеет *асимптотическую сложность* $O(f(n))$, это значит, что его время работы имеет ограничение $T(n) \leq f(n) \cdot c$ при любых входных данных, достаточно больших n и **фиксированной** c .

Проверка

Какую асимптотику имеет тривиальный поиск?

1. $O(n)$
2. $O(2 \cdot n)$
3. $O(n^2)$

Проверка

Какую асимптотику имеет тривиальный поиск?

1. $O(n)$
2. $O(2 \cdot n)$
3. $O(n^2)$

Все вышеперечисленные подходят под определение асимптотической сложности

Обычно стараются давать *разумную* асимптотики. За этим есть немного теории, но идеальный вариант - это когда нет константы (ее можно спрятать в c) и функцию нельзя "уменьшить".

Бинарный поиск: асимптотика

Выпишем количество шагов алгоритма явно:

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

Иначе говоря, мы на каждом шаге сужали пространство поиска в два раза. Если в конце мы дошли до размера 1, то за $T(n)$ шагов мы получили

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

Отсюда

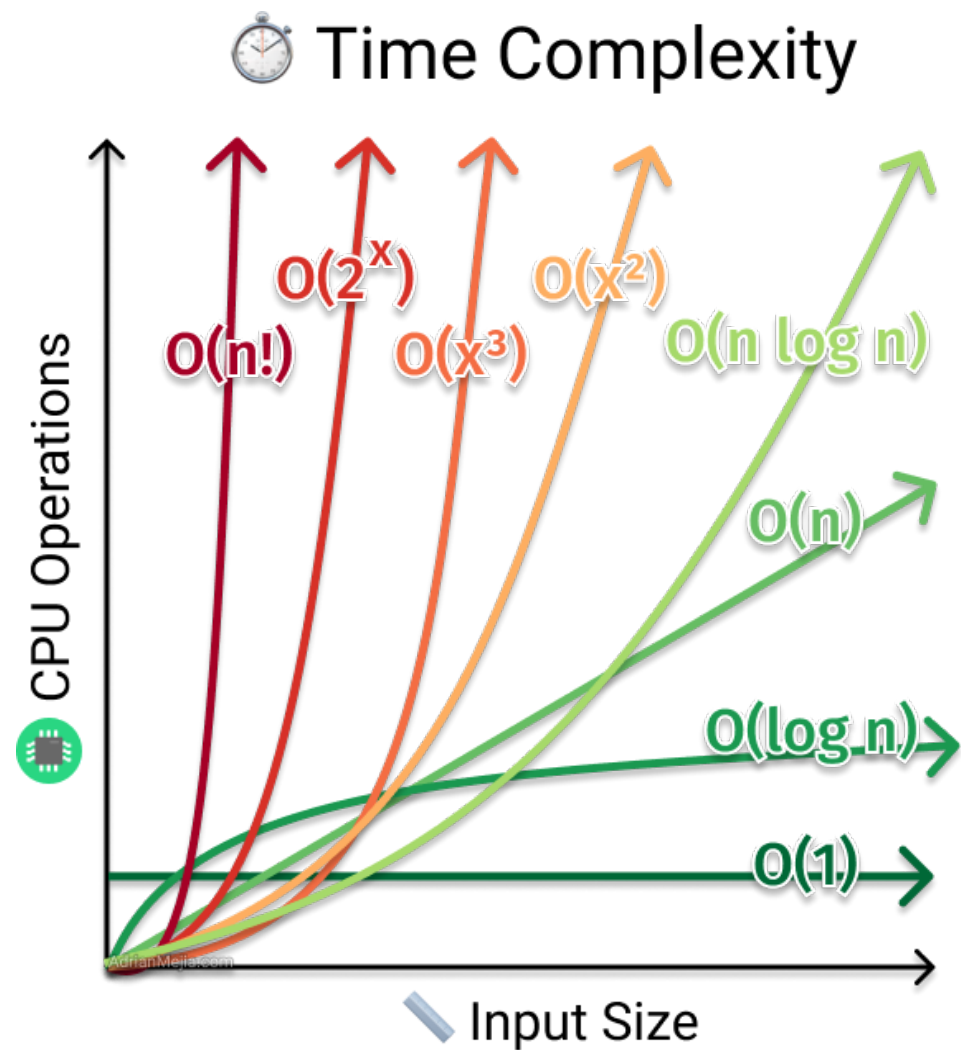
$$T(n) \sim \log_2(n)$$

Значит, бинарный поиск работает за $O(\log n)$

Сравнение асимптотик

На глаз некоторые асимптотики отличить сложно, формально можно проверить $\frac{T(n)}{f(n)} \leq c$

Надо понимать, что это математический подход к анализу алгоритма, с инженерной точки зрения два алгоритма с разной асимптотикой имеют разную производительность.



Мемы!



Pranay Pathole
@PPathole

2 hours ago

Alternative Big O notations:

$O(1) = O(\text{yeah})$

$O(\log n) = O(\text{nice})$

$O(n \log n) = O(\text{k-ish})$

$O(n) = O(\text{ok})$

$O(n^2) = O(\text{my})$

$O(2^n) = O(\text{no})$

$O(n^n) = O(\text{fuck})$

$O(n!) = O(\text{mg!})$

tag yourself



(1)

- never changes
- v efficient
- kind of boring
- but you know
- you're supposed to
- like them



(N)

- straightforward
- tells it like it is
- increases when
- input increases
- does what it's
- supposed to



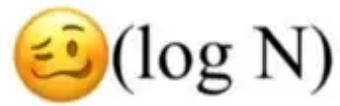
(N²)

- peer pressurer
- brings everyone
- else into their mess
- takes each
- element and does
- something with
- each other element
- probably the
- source of mono



(2^N)

- doubles every
- time input is
- added
- kind of dramatic



(log N)

- v efficient
- peaked in high school,
- then plateaued
- doesn't care about your
- input

Therapist: O-notations aren't real, they can't hurt you

O-notations:

$$\begin{aligned} O(1) < O(\log N) < O(N) < O(N \log N) \\ < O(N^2) < O(N^3) < O(2^N) \\ < O(10^N) < O(N!) \end{aligned}$$

Идея для пет-проекта

Программа/библиотека, которая делает вывод об асимптотике алгоритма на основе случайных тестовых данных и нескольких размерах выборок.

Если узнать средние замеры на $n = 10, n = 100, \dots, n = 5000$, то по набору замеров можно сделать предположение - рост линейный, меньше линейного или больше линейного.

