Алгоритмы поиска

education

Содержание

education

Определение поиска

- Нахождение заданного элемента(-ов) во множестве (в нашем случае это будет массив), причём искомые элементы должны обладать определённым свойством.
- Иногда задача может звучать как «найти первое или последнее вхождение заданного числа».

Свойство может быть как абсолютным, так и относительным

Относительное — максимум или минимум во множестве

Абсолютное — эквивалентное искомому значение

Критерии поиска

1

Рефлексивность
(A ~ A)

2

Симметричность
(A ~ B <=> B ~ A)

3

Транзитивность
(A ~ B, B ~ C <=> A ~ C)

Линейный (последовательный) поиск

1

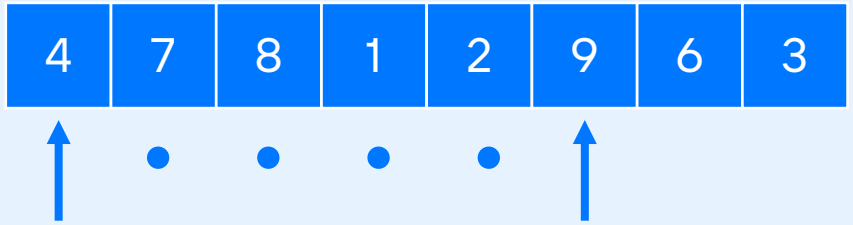
Сложность по времени
в наихудшем случае O(n)

2

Затраты памяти O(1)

education

- Начиная с первого элемента, последовательно просматриваем весь массив и сравниваем каждое значение с заданным. Если значения равны, то возвращаем его номер.
- Недостатки следуют из самого описания — нам необходимо пройтись по всему массиву.



Для нахождения искомого элемента проверить надо будет каждый

Когда применим алгоритм

- Если данные неотсортированные, то найти элемент можно только путём последовательного перебора всех элементов.
- Если речь идёт о поиске максимума или минимума в массиве.

```
for (i = 0; i < n; i++) {
        if (A[i] == B) {
                return i
        }
}
```

education

Бинарный поиск

1

Сложность по времени
в наихудшем случае O(log(n))

2

Затраты памяти O(1)

education

Аналогия из жизни

Игра «Угадай число»
(в пример приводят почти все)

Поиск слова в словаре
или человека в записной книжке

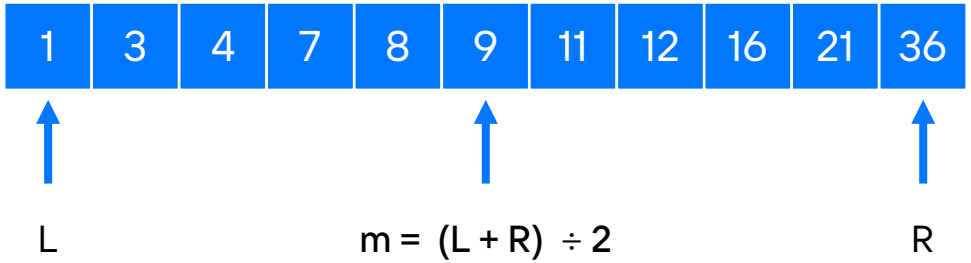
education

Алгоритм работы поиска

- Массив должен быть отсортирован. Определяем левую границу в качестве первого элемента массива и правую в качестве последнего элемента.
- Делим всю последовательность пополам и находим элемент, находящийся в середине. Сравниваем его с искомым значением.
- Если значения равны, то возвращаем индекс элемента.

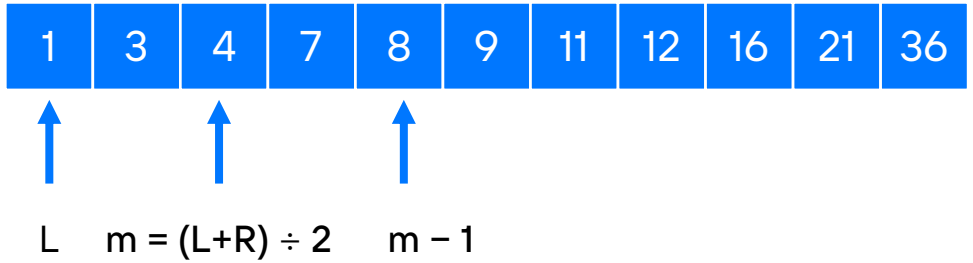
- В случае, если элемент, стоящий в середине, больше искомого, то обрабатываем левую сторону. В противном случае — наоборот.
- Повторяем алгоритм, начиная со второго пункта, пока не найдём необходимый элемент или не удостоверимся, что он отсутствует.

Бинарный поиск наглядно



Допустим, нам нужно найти число 7.
Устанавливаем границы отрезка
и вычисляем середину

L = 0; R = 10; m = (0 +10) ÷ 2 = 5



Устанавливаем правую границу в значении L = m − 1

Почему − 1?

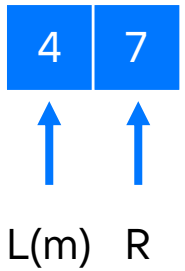
Значение правой границы
мы уже проверили

Сдвиг вправо даст + 1

education

Бинарный поиск наглядно

- Сдвигаемся вправо.
- Остаётся последний элемент.
- Сравниваем его.
- Возвращаем его индекс, если он эквивалентен искомому.



$4 < 7$

Рекурсивный подход

Преимущества:

- Код становится более компактным и читабельным.

Недостатки:

- Требует больше памяти, возможно переполнение стека. При каждом рекурсивном вызове функция добавляется в стек.

```
func binarySearch(data, l, r, needle) {
   if l>r {
      return -1
   }
   m = (l+r)/2
   if (data[mid] == needle) {
      return mid
   }
   if (data[mid] > needle) {
      //ищем в левой стороне
       //правая граница смещается до mid-1 включительно
      return binarySearch(data, l, mid-1, needle)
   } else {
      //ищем в правой стороне
       //левая граница смещается до mid+1 включительно
      return binarySearch(data, mid+1, r, needle)
   }
}
```

Итеративный подход

При каждом рекурсивном вызове мы меняем только левый и правый индексы.

Это значит, что в зависимости от расположения элемента относительно середины нам надо в рамках **текущей итерации** обновить либо левый, либо правый индекс.

```
func binarySearch(data, needle) {
   l = 0
   r = len(data)

   // корнер кейсы
   if r == 0 || needle < data[0] || needle > data[r-1] {
      return -1
   }

   for l <= r {
      mid = (l + r)/2
      if (needle == data[mid]) {
         return mid
      }
      if (needle < data[mid]) {
         //ищем в левой стороне
          //правая граница смещается до mid-1 включительно
         r = mid - 1
      } else {
         //ищем в правой стороне
          //левая граница смещается до mid+1 включительно
         l = mid + 1
      }
   }

   return -1
}
```

education

Определение сложности

- Количество элементов перед выполнением поиска = n
- После первой итерации = n ÷ 2
- После второй = n ÷ 4
- Итого на i-ом проходе получаем: n ÷ 2^i
- На последнем проходе: 1

- В итоге получаем формулу

$$1 = n \div 2^i$$

- Это равносильно записи

$$I = \log(n)$$

Где n — это размер массива

education

Сравнение количества итераций

Возьмём для простоты массив из 64 элементов

64 → 32 → 16 → 8 → 4 → 2 → 1

Итого 6 раз, что как раз эквивалентно логарифму 64

Значения в наихудшем случае

Количество элементов в массиве	Линейный поиск	Бинарный поиск
100	100	7
10 000	10 000	14
1 000 000	1 000 000	20
1 000 000 000	1 000 000 000	30

Одно НО

Для бинарного поиска необходима сортировка, поэтому необходимо подсчитать, когда двоичный поиск точно будет выгоден.

n — количество элементов

k — количество операций поиска

n × log(n) — сложность сортировки

log(n) — сложность поиска

Получаем выражение

$$n \times k >= n \times \log(n) + k \times \log(n)$$

Левый бинарный поиск

Поиск первого вхождения

Правила написания

- Цикл продолжается, пока не останется два элемента (вместо одного как раньше), то есть
  for (l + 1 < r).
- Двигаем правую и левую границы строго на середины без плюс минус единицы.

- Если мы ищем первое вхождение, то есть искомый элемент находится слева, то вначале проверяем левый индекс, а только потом — правый.
- Если мы ищем последнее вхождение, то вначале проверяем правый индекс и только потом — левый.

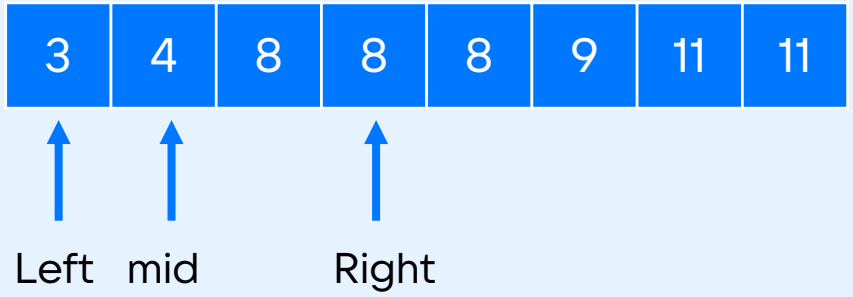
Алгоритм работы

- Нам задан массив, и в нём необходимо найти число 8.
- Находим середину, и в случае, если найденное значение меньше, то сдвигаем правую границу строго на индекс mid!
- Продолжаем поиск, пока не останутся только левая и правая границы.

education

Алгоритм работы

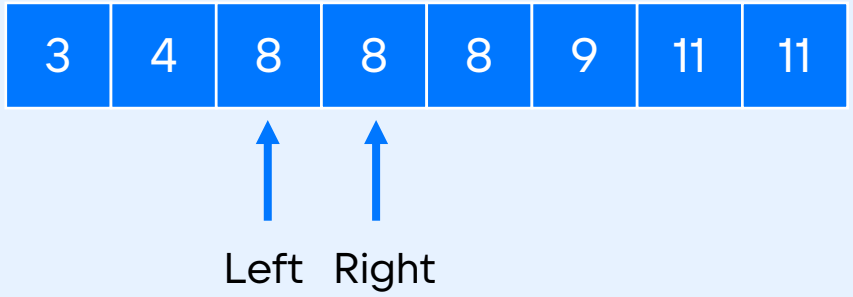
- Нам задан массив, и в нём необходимо найти число 8.
- Находим середину, и в случае, если найденное значение меньше, то сдвигаем правую границу строго на индекс mid!
- Продолжаем поиск, пока не останутся только левая и правая границы.



education

Алгоритм работы

- Нам задан массив, и в нём необходимо найти число 8.
- Находим середину, и в случае, если найденное значение меньше, то сдвигаем правую границу строго на индекс mid!
- Продолжаем поиск, пока не останутся только левая и правая границы.



education

Левый бинарный поиск

```
func leftBinarySearch(needle int, nums []int) int {
  low := 0
  high := len(nums) - 1

  for low+1 < high {
    median := (low + high) / 2
    if nums[median] < needle {
      low = median
    } else {
      high = median
    }
  }

  if nums[low] == needle {
    return low
  }
  if nums[high] == needle {
    return high
  }
  return -1
}
```

education

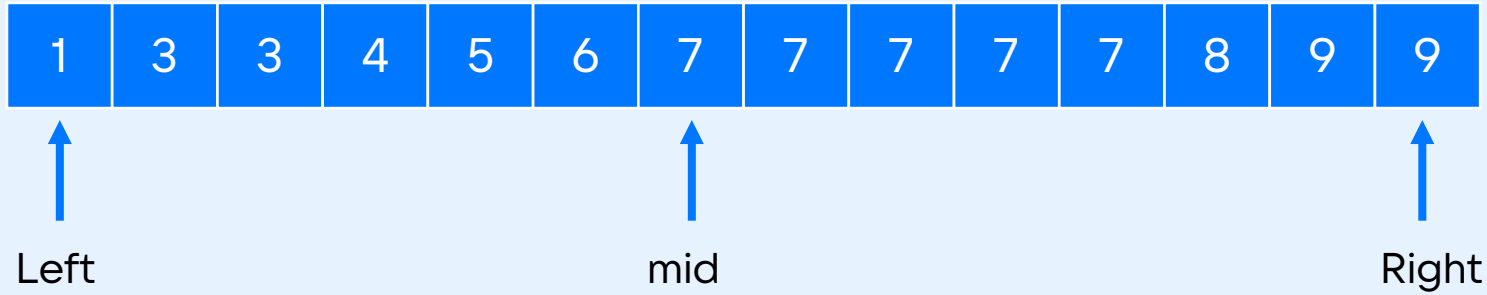
Правый бинарный поиск

```
func rightBinarySearch(needle int, nums []int) int {
  low := 0
  high := len(nums) - 1

  for low+1 < high {
    median := (low + high) / 2
    if nums[median] <= needle {
      low = median
    } else {
      high = median
    }
  }

  if nums[high] == needle {
    return high
  }
  if nums[low] == needle {
    return low
  }

  return -1
}
```

education

Иллюстрация работы

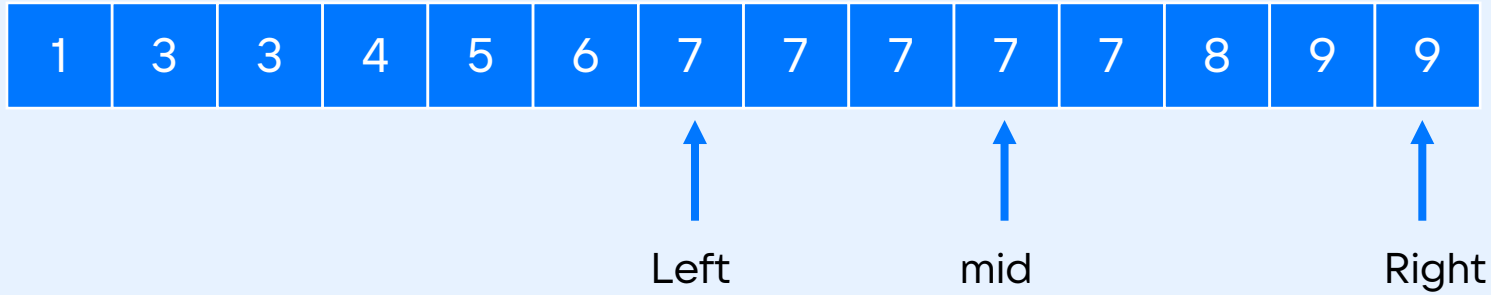
- Дан массив, и нам необходимо найти число 7, а точнее — последнее вхождение.
- Здесь, в случае, если data[mid] == 7, мы не заканчиваем поиск, а лишь сдвигаем левую границу (low = median в коде).



education

Иллюстрация работы

- Дан массив, и нам необходимо найти число 7, а точнее — последнее вхождение.
- Здесь, в случае, если data[mid] == 7, мы не заканчиваем поиск, а лишь сдвигаем левую границу (low = median в коде).



education

Иллюстрация работы

- Дан массив, и нам необходимо найти число 7, а точнее — последнее вхождение.
- Здесь, в случае, если data[mid] == 7, мы не заканчиваем поиск, а лишь сдвигаем левую границу (low = median в коде).

education

Иллюстрация работы

- Дан массив, и нам необходимо найти число 7, а точнее — последнее вхождение.
- Здесь, в случае, если data[mid] == 7, мы не заканчиваем поиск, а лишь сдвигаем левую границу (low = median в коде).

education

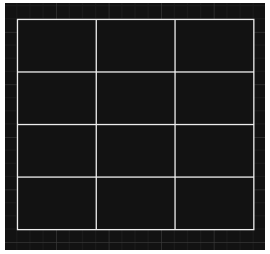
Иллюстрация работы

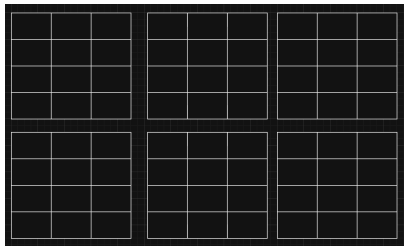
- Дан массив, и нам необходимо найти число 7, а точнее — последнее вхождение.
- Здесь, в случае, если data[mid] == 7, мы не заканчиваем поиск, а лишь сдвигаем левую границу (low = median в коде).

education

Бинарный поиск по ответу

- Итак, наступает кульминация нашего изучения бинарного поиска, и мы с вами подходим к ключевой главе — бинарный поиск по ответу.
- Разберём его на примере культовой задачи про дипломы.
- Есть ещё одна не менее культовая задача про коров и стойла, но её вы разберёте самостоятельно.





Задача

Петя активный малый и участвует во всех олимпиадах по математике и физике. Накопил кучу дипломов, которые лежали в столе, и он не знал, что с ними делать. И вот он решил: чтобы они перестали пылиться в столе, надо их повесить пылиться на стену.

Ему хотелось их разместить на квадратной доске. Итак, есть 9 прямоугольных дипломов (3 × 4), которые надо разместить на квадратной поверхности. Необходимо найти минимальную сторону квадрата для размещения всех дипломов.

Посмотрим на код

- Определяем минимальное и максимальное значения.
- Определяем цикл, в котором будем искать ближайший возможный вариант.
- Очевидно, что если у нас получился квадрат, в котором не помещается заданное число дипломов, то и все меньшие квадраты нам не подойдут. И надо будет расширять границу поиска.

education

```
def binary_search(w, h, n):
  # определяем минимальное значение как самую длинную сторону одного диплома
  # очевидно что квадрат с меньшей стороной точно не подойдет
  left = max(w, h)
  # определяем максимальное значение
  right = left * n
  # сужаем поиск
  while right - left > 1:
    # как и раньше определяем середину
    mid = (right + left) // 2
    # на каждой итерации мы будем подсчитывать,
    # количество дипломов в высоту, назовем это строки
    # количество дипломов в ширину, назовем это столбцами
    # (mid // h) - количество возможных строк
    # (mid // w) - количество возможных столбцов
    # перемножая эти два значения мы получим возможное количество дипломов
    res = (mid // w) * (mid // h)
    if res < n:
      left = mid
    else:
      right = mid

  return right
```

Будем ВКонтакте!

Виды поиска

Линейный поиск

Бинарный поиск

Разновидности бинарного поиска:

- Тернарный поиск
- Экспоненциальный поиск
- Поиск Фибоначчи
- Интерполяционный поиск