Алгоритмы. Бинарный поиск.

Сведение о алгоритме

Алгоритм бинарного поиска.

Сложность по времени в наихудшем случае	O(ln(n))
Затраты памяти	O(n)

Принцип работы алгоритма

1) Сортируется последовательность в которой будет проводится поиск. Если последовательность уже отсортирована то этот шаг можно пропустить.

2) Определяется значение элемента в середине последовательности. Полученный элемент сравнивается с искомым элементом. Различают случаи :
   a) Средний элемент равен искомому. Заканчиваем алгоритм. Поиск успешен.
   b) Средний элемент больше искомого. Рассматриваем левую от среднего элемента часть последовательности.
   c) Средний элемент меньше искомого. Рассматриваем правую от среднего элемента часть последовательности.
3) Повторяем пункт 2 до тех пор, пока не будет найден искомый элемент или не станет пустым интервал для поиска.

Графическая иллюстрация работы алгоритма

l r

[-2, 0, 3, 5, 7, 9, 11, 15, 18]

1) m

l r

[-2, 0, 3, 5, 7, 9, 11, 15, 18]

2) m

l r

[-2, 0, 3, 5, 7, 9, 11, 15, 18]

3) m

Работа алгоритма продемонстрирована в предположении, что искомым элементом является **5**.

Графическая иллюстрация работы алгоритма

l r

[-2, 0, 3, 5, 7, 9, 11, 15, 18]

1) m

На первом шаге, в качестве левой границы выбирается первый элемент последовательности в качестве правой границы последний элемент последовательности. Средний элемент (в случае индексируемых последовательностей) вычисляется как элемент индекс которого равен:

$$index_m = \frac{index_l + index_r}{2} = \frac{0+8}{2} = 4$$

$index_m$ индекс середины последовательности

$index_l$ индекс левой границы

$index_r$ индекс правой границы

Графическая иллюстрация работы алгоритма

l r

[-2, 0, 3, 5, 7, 9, 11, 15, 18]

1) m

Так как в середине последовательности стоит элемент больше искомого (5 < 7) то сдвигаем правую границу.

Графическая иллюстрация работы алгоритма

l r

[-2, 0, 3, 5, 7, 9, 11, 15, 18]

2) m

$$index_m = \frac{index_l + index_r}{2} = \frac{0+4}{2} = 2$$

Установив правую границу на место найденного на прошлом шаге среднего элемента повторяем вычисление среднего элемента. Это элемент с индексом 2. Его значение равно 3. И его значение меньше искомого (3 < 5). Следовательно нужно сдвинуть левую границу.

Графическая иллюстрация работы алгоритма

l r

[-2, 0, 3, 5, 7, 9, 11, 15, 18]

3) m

$$index_m = \frac{index_l + index_r}{2} = \frac{2+4}{2} = 3$$

Установив левую границу на место найденного на прошлом шаге среднего элемента повторяем вычисление среднего элемента. Это элемент с индексом 3 и его значение равно 5. Он равен искомому элементу (5 == 5). Поиск окончен.

В каком случае стоит использовать бинарный поиск

Предположим что существует не отсортированная последовательность размером N элементов. В ней нужно выполнить K поисков (0 < K). При каком значении K стоит выполнить обычный линейный поиск (не сортируя массив)? При каком значении стоит отсортировать массив и использовать бинарный поиск ?

Вспомогательные данные:

Сложность линейного поиска $O(N)$

Сложность оптимальной сортировки (например TimSort) $O(N \cdot \ln(N))$

Сложность бинарного поиска $O(\ln(N))$

В каком случае стоит использовать бинарный поиск

Линейный поиск

$$C_1 \cdot K \cdot N$$

Сортировка + бинарный поиск

$$C_2 \cdot N \cdot \ln(N) + C_3 \cdot K \cdot \ln(N)$$

Условие перехода

$$C_1 \cdot K \cdot N \geqslant C_2 \cdot N \cdot \ln(N) + C_3 \cdot K \cdot \ln(N)$$

$$C_1 \cdot K \cdot N = C_2 \cdot N \cdot \ln(N) + C_3 \cdot K \cdot \ln(N) \Rightarrow K \cdot (C_1 \cdot N - C_3 \cdot \ln(N)) = C_2 \cdot N \cdot \ln(N) \Rightarrow K = \frac{C_2}{C_1} \cdot \frac{N \cdot \ln(N)}{N - \frac{C_3}{C_1} \ln(N)}$$

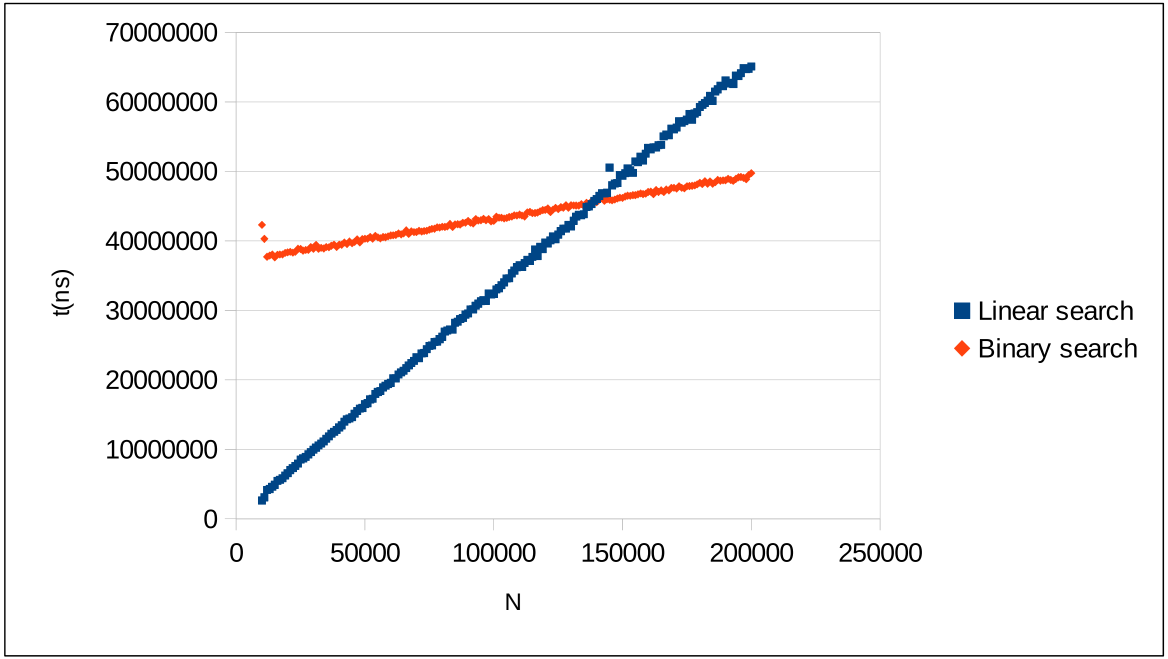
Где :

$C_{1,} C_{2,} C_3$ - Константы зависящие от ПК

$$\lim_{N \to \infty} \frac{C_2}{C_1} \cdot \frac{N \cdot \ln(N)}{N - \frac{C_3}{C_1} \ln(N)} = \frac{C_2}{C_1} \cdot \ln(N)$$

Экспериментальная проверка

Для проверки математического утверждения был проведен вычислительный эксперимент. Для одномерного массива из 1 000 000 элементов построена зависимость времени поиска от количества искомых элементов. Рассмотрен как линейный так и бинарный поиск (время сортировки массива также учитывалось). Для корректности каждый замер был повторен 100 раз и было взято усредненное время.



Как можно видеть из графика с ростом числа операций поиска в массиве наступает момент когда целесообразнее выполнить сортировку массива и выполнять бинарный поиск, чем использовать линейный поиск.

Реализация алгоритма на Python

Реализация алгоритма на Python

```
sequence = [-2, 0, 3, 5, 7, 9, 11, 15, 18]


def find_element(sequence, required_element):
    l = 0
    r = len(sequence) - 1
    while l <= r:
        m = (l + r)//2
        element = sequence[m]
        if element == required_element:
            return m
        if element < required_element:
            l = m + 1
        else:
            r = m - 1
    return -1
```

Отсортированная последовательность

Определение левой и правой границы

Нахождение индекса среднего элемента

Определение успеха поиска

Определение левой и правой границы

Неудачный поиск

Реализация алгоритма на Java

Реализация алгоритма на Java

```
public static int binarySearch(int[] array, int requiredElement) {

    int l = 0;
    int r = array.length - 1;

    for (; l <= r;) {
        int m = l + (r - l) / 2;
        int element = array[m];
        if (requiredElement == element) {
            return m;
        }
        if (element < requiredElement) {
            l = m + 1;
        } else {
            r = m - 1;
        }
    }
return -1;
}
```

Определение левой и правой границы

Нахождение индекса среднего элемента

Определение успеха поиска

Определение левой и правой границы

Неудачный поиск

Список литературы

1) Ананий Левитин. Алгоритмы: введение в разработку и анализ. : Пер. с англ. — М. : Издательский дом "Вильяме", 2006. — 576 с. : ил. — Парал. тит. Англ. ISBN 5-8459-0987-2. Стр. [180-183]
2) Стивенсон Род. Алгоритмы. Теория и практическое применение — М: Издательство «Э», 2016 — 544. ISBN 978-5-699-81729-0. Стр. [163-165]