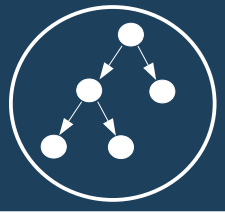


Data Structures and Algorithms

Алгоритмы. Тернарный поиск.

Сведения о алгоритме

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

Описание алгоритма

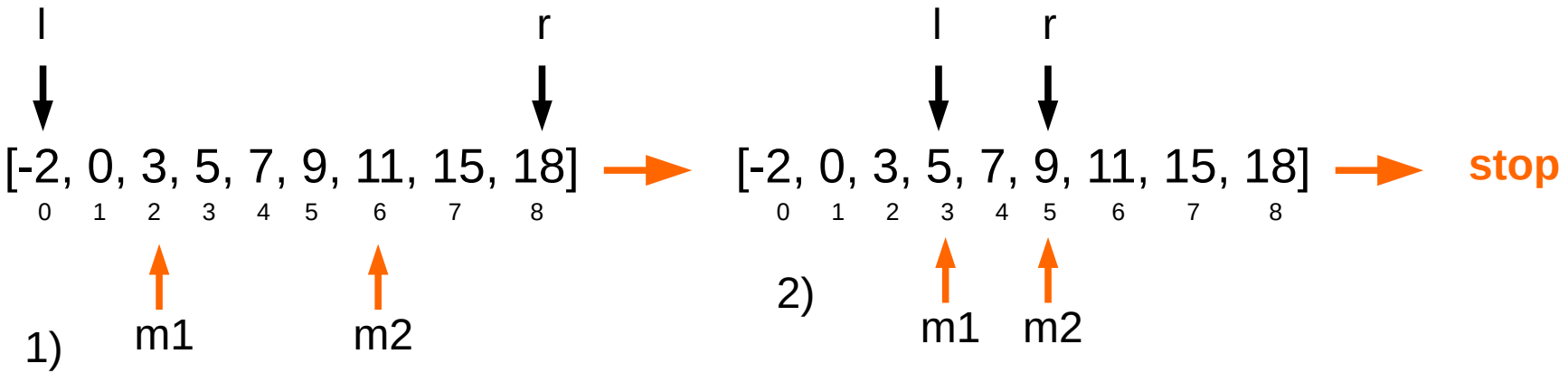
1) Поиск выполняется по отсортированной последовательности. Объявить ряд вспомогательных переменных. В дальнейшем будем использовать такие имена l — индекс первого элемента последовательности, r — индекс последнего элемента последовательности, m1 — индекс вычисляемый как m1 = l + (r - l)/3 , m2 — индекс вычисляемый как m2 = r - (r - l)/3. Перейти к пункту **2**.
2) Выполнить ряд проверок:
   1) Если значение на индексе m1 равно искомому то закончить алгоритм вернуть m1.
   2) Если значение на индексе m2 равно искомому то закончить алгоритм вернуть m2.
   3) Если l>r закончить алгоритм вернуть отрицательный результат поиска.

   Если не одно из условий не выполнено перейти к пункту **3**.

3) Выполнить ряд проверок:
   1) Если значение стоящее на индексе m1 меньше искомого, а значение стоящее на индексе m2 больше . Установить l = m1+1, r = m2-1
   2) Если значение стоящее на индексе m1 больше искомого. Установить r = m1 - 1.
   3) Если значение стоящее на индексе m2 меньше искомого. Установить l = m2 + 1 .

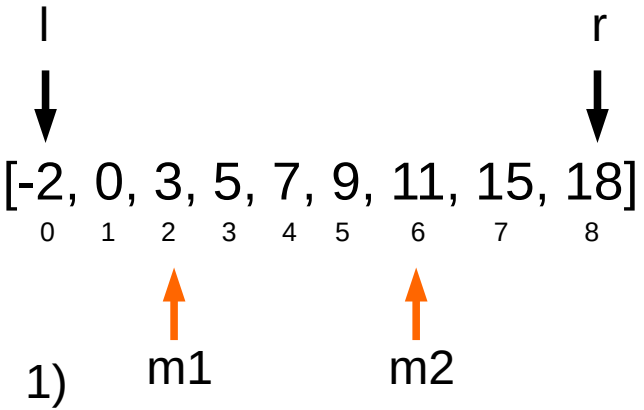
   Установить m1 = l + (r - l)/3, m2 = r - (r - l)/3перейти к пункту **2**.

Графическая иллюстрация работы алгоритма



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

Графическая иллюстрация работы алгоритма

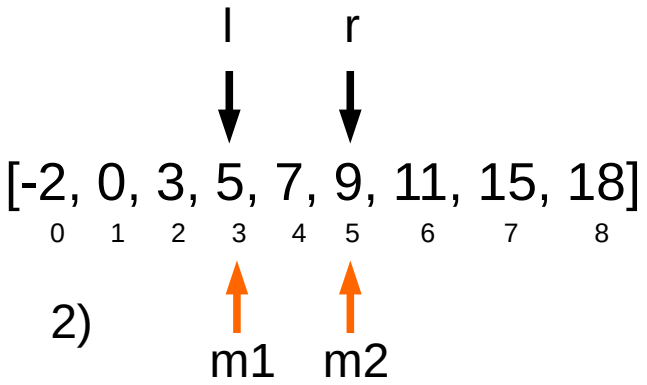


$$m1 = l + \frac{(r-l)}{3} = 0 + \frac{(8-0)}{3} = 0 + 2 = 2$$

$$m2 = r - \frac{(r-l)}{3} = 8 - \frac{(8-0)}{3} = 8 - 2 = 6$$

Выполнили проверку на s[2] и s[6] не стоит элемент равный искомому. Так как s[m1]<5<s[m2] то устанавливаем значение l = m1+1, r = m2-1. Переходим к следующему шагу.

Графическая иллюстрация работы алгоритма



$$m1 = l + \frac{(r-l)}{3} = 3 + \frac{(5-3)}{3} = 3 + 0 = 3$$

$$m2 = r - \frac{(r-l)}{3} = 5 - \frac{(5-3)}{3} = 5 - 0 = 5$$

Выполнили проверку на s[3] стоит искомый элемент. Закончили алгоритм.

Python

Реализация алгоритма на Python

Python

Реализация алгоритма на Python

```
def ternary_search(sequince, element):
    l = 0
    r = len(sequince)-1
    while l <= r:
        h = (r-l)//3
        m1 = l+h
        m2 = r-h
        if sequince[m1] == element:
            return m1
        if sequince[m2] == element:
            return m2
        if sequince[m1] < element < sequince[m2]:
            l = m1+1
            r = m2-1
        elif element < sequince[m1]:
            r = m1-1
        else:
            l = m2+1
    return None
```

Java

Реализация алгоритма на Java

Java

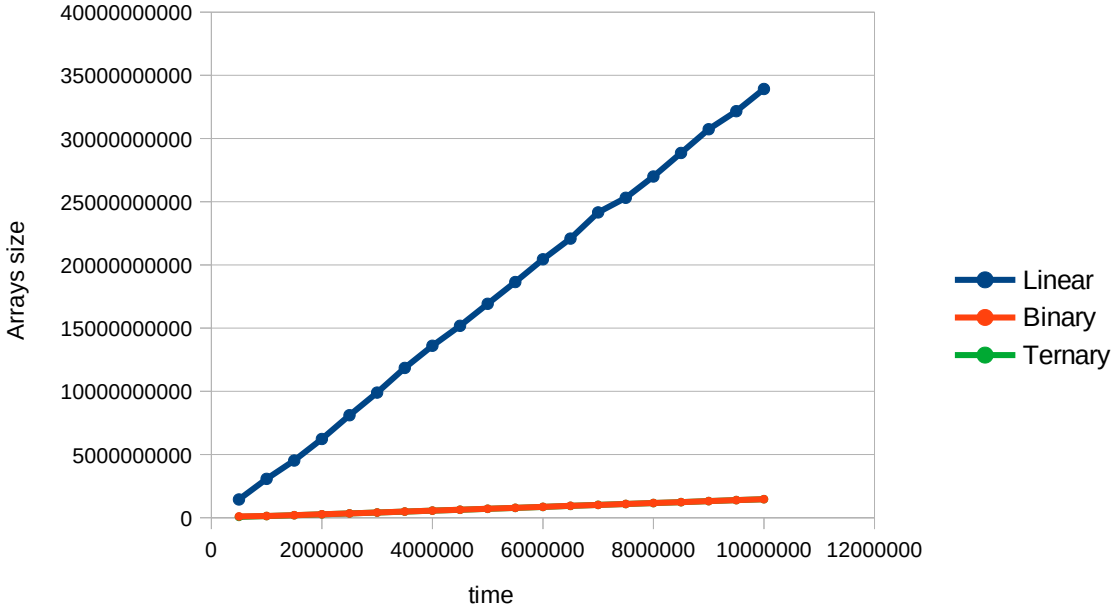
Реализация алгоритма на Java

```java
public static int ternarySearch(int[] array, int element) {
     int r = array.length - 1;
     int l = 0;
     for (; l <= r;) {
          int h = (r - l) / 3;
          int m1 = l + h;
          int m2 = r - h;

          if (array[m1] == element) {
               return m1;
          }
          if (array[m2] == element) {
               return m2;
          }
          if (array[m1] < element && element < array[m2]) {
               l = m1 + 1;
               r = m2 - 1;
          } else if (element < array[m1]) {
               r = m1 - 1;
          } else {
               l = m2 + 1;
          }
     }
     return -1;
}
```

Вычислительный эксперимент

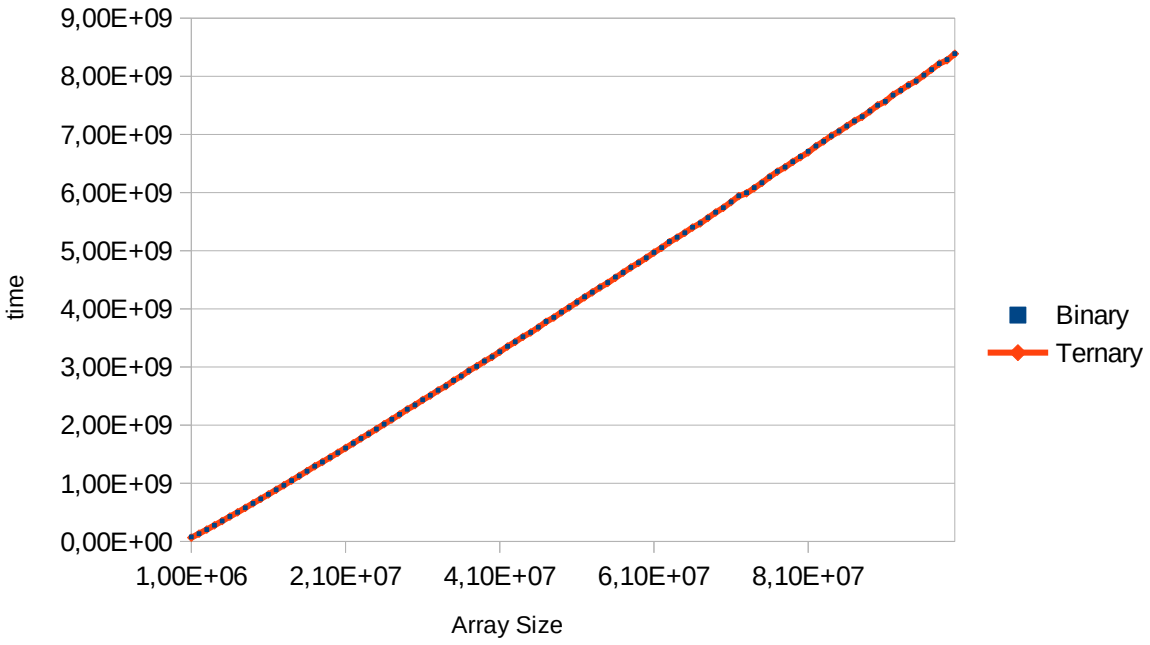
Для оценки асимптотического поведения реализации этого алгоритма был проведен следующий вычислительный эксперимент.
Для одномерных массивов разных размеров построена зависимость времени поиска 4000 элементов от размера массива. Рассмотрен линейный, бинарный и тернарный поиск(для двух последних время сортировки массива также учитывалось). Для корректности каждый замер был повторен несколько раз и было взято усредненное время.



Предсказуемым результатом является то, что бинарный и тернарный поиск даже с учетом времени на сортировку выполняются быстрее линейного. Однако график не позволяет выполнить сравнение бинарного и тернарного поиска. Поэтому их сравнение выполнено отдельно.

Вычислительный эксперимент

На графике приведена зависимость времени поиска 4000 элементов в массиве от размера массива. Как видно у бинарного и тернарного поиска наблюдается паритет. И хотя в тернарном поиске скорость сужения окна поиска пропорциональна степени тройки, но увеличение количества проверок на каждом посещении элемента массива нивелирует это преимущество.



Список литературы

1) Дональд Кнут — Искусство программирования. Том 3. Сортировка и поиск. / Knuth D.E. — The Art of Computer Programming. Vol. 3. Sorting and Searching.