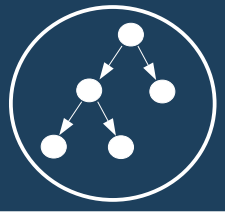


Data Structures and Algorithms

Алгоритмы.
Экспоненциальный поиск

Сведение о алгоритме

Сложность по времени в наихудшем случае $O(\ln(k))$

k — Позиция искомого элемента в последовательности

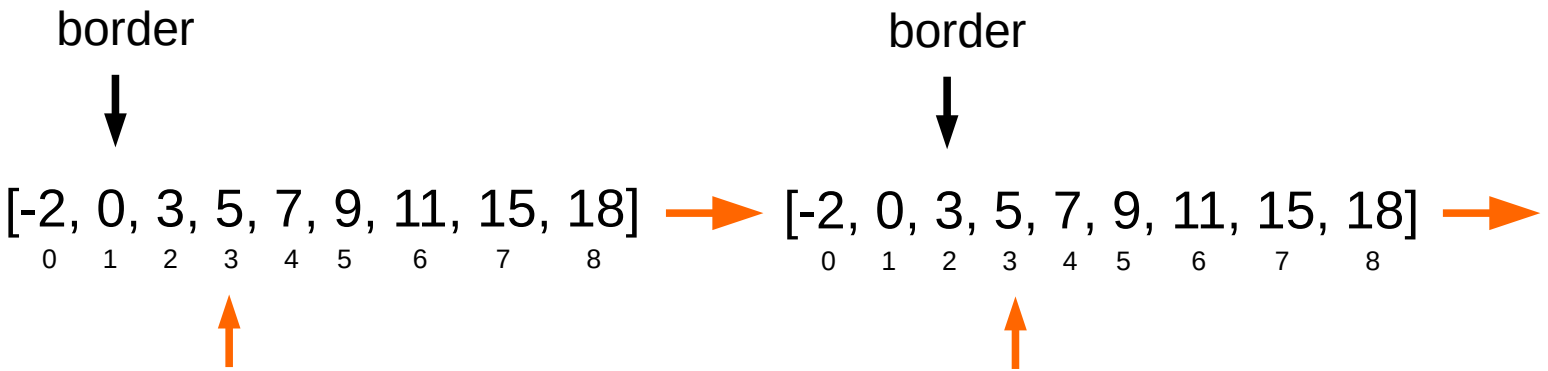
Краткие сведение о авторах и принципе алгоритма

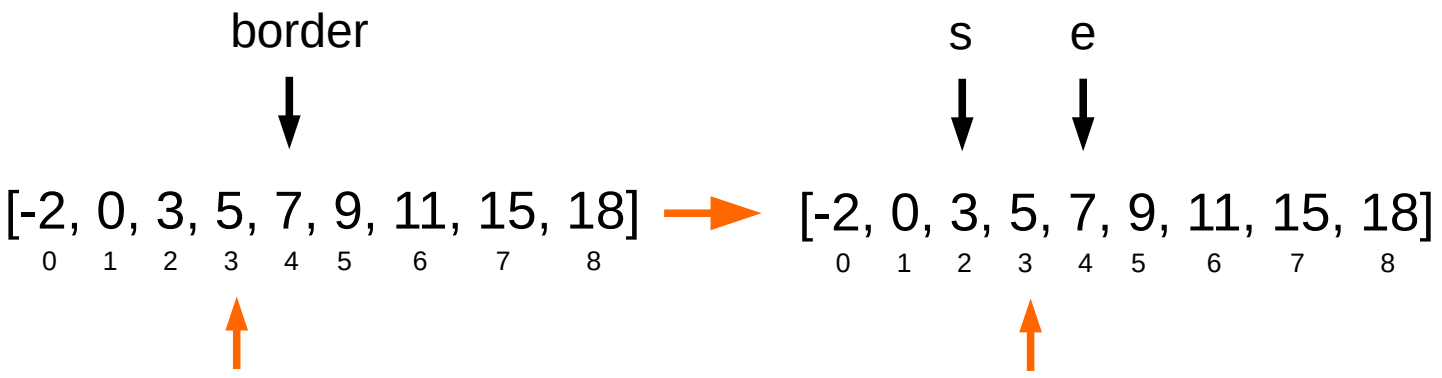
Алгоритм был разработан Джоном Бентли и Эндрю Чи-Чи Яо в 1976 году. Этот алгоритм используется для быстрого определение диапазона поиска, с последующим поиском в указанном диапазоне с помощью бинарного поиска.

Описание алгоритма

1) Поиск проводим в отсортированной последовательности. Для определения границы объявляется дополнительная переменная (в дальнейшем border) ее значение устанавливается равной единице. Переходим к пункту **2**.
2) Выполняется проверка: если значение border больше чем длинна последовательности то в таком случае выполняем бинарный поиск нужного элемента в промежутке от border/2 до размера последовательности. **Заканчиваем поиск**. В противном случае переходим к пункту **3**.
3) Выполняем проверку: если значение на индексе border больше искомого элемента то выполняем бинарный поиск нужного элемента в промежутке от border/2 до border. **Заканчиваем поиск**. В противном случае переходим к пункту **4**.
4) Увеличиваем значение border в два раза. Переходим к пункту **2**.

Графическое пояснение алгоритма





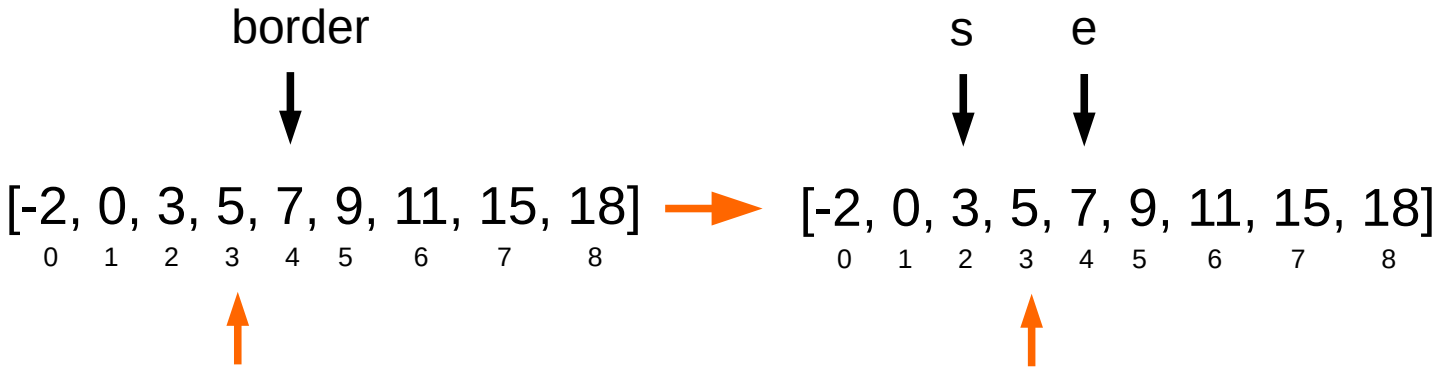
Графическое пояснение алгоритма

Значение border = 1. Это меньше чем длинна последовательности. Значение на 1 индексе меньше искомого. Увеличиваем значение border в два раза и переходим к следующему шагу.

Графическое пояснение алгоритма

Значение border = 2. Это меньше чем длинна последовательности. Значение на 2 индексе меньше искомого. Увеличиваем значение border в два раза и переходим к следующему шагу.

Графическое пояснение алгоритма



Значение border = 4. Это меньше чем длинна последовательности. Значение на 4 индексе больше искомого. Проводим бинарный поиск в диапазоне от 2 (border/2) до 4(border). Заканчиваем алгоритм.

Python

Реализация алгоритма на Python

Бинарный поиск в указанных границах

```
def binary_search(sequence, required_element, start, end):
    while start <= end:
        m = (start + end)//2
        element = sequence[m]
        if element == required_element:
            return m
        if element < required_element:
            start = m + 1
        else:
            end = m - 1
    return None
```

Экспоненциальный поиск

```
def exponential_search(sequence, required_element):
    border = 1
    while border < len(sequence)-1 and sequence[border] < required_element:
        border = border * 2
    if border > len(sequence)-1:
        border = len(sequence)-1
    return binary_search(sequence, required_element, border//2, border)
```

Java

Реализация алгоритма на Java

Бинарный поиск в заданном диапазоне

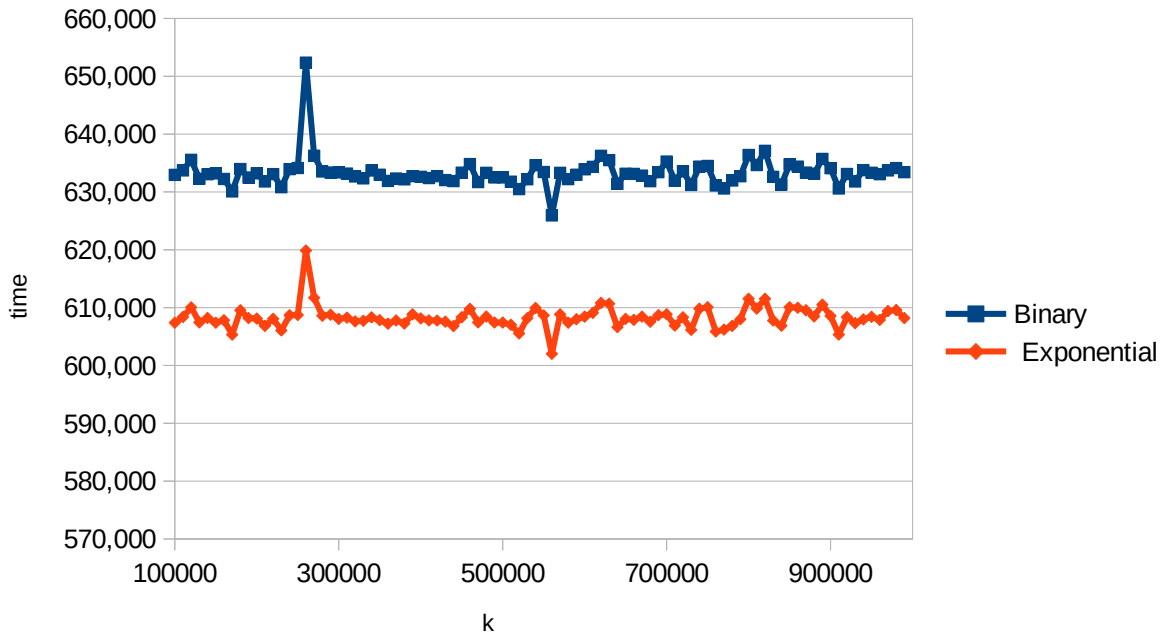
```
public static int binarySearch(int[] array, int requiredElement, int l, int r){
    for (; l <= r;) {
        int m = l + (r - l) / 2;
        int element = array[m];
        if (requiredElement == element) {
            return m;
        }
        if (element < requiredElement) {
            l =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

Экспоненциальный поиск

```
public static int exponentialSearch(int[] array, int requiredElement) {
    long border = 1;
    for (; border < array.length && array[(int) border] < requiredElement;) {
        border *= 2;
    }
    int l = (int) (border / 2);
    int r;
    if (border > array.length - 1) {
        r = array.length - 1;
    } else {
        r = (int) border;
    }
    return binarySearch(array, requiredElement, l, r);
}
```

Вычислительный эксперимент

Для проверки ускорения от применения экспоненциального поиска. Был выполнен вычислительный эксперимент. В массиве размером 100_000_000 элементов было произведено k поисков. На графике представлена зависимость времени на k поисков от значения k. Искомые значения равномерно распределены по диапазону значений в массиве.



Как видно из графика экспоненциальный поиск оказывается незначительно оптимальнее бинарного поиска.

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.