Data Structures and Algorithms

Алгоритмы. Поиск Фибоначчи.

Сведение о алгоритме

Алгоритм поиска Фибоначчи.

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

Принцип работы алгоритма

1) Последовательность сортируется в возрастающем порядке. Если последовательность уже отсортирована то этот пункт можно пропустить.
2) Производится начальная инициализация. Нужно найти такое число k, что F(k+1) ≥ N+1. После чего нужно ввести следующие значения. M = F(k+1)-(N+1), i = F(k)-M, p=F(k-1), q=F(k-2).
3) Проверить корректность индекса. Если индекс меньше нуля перейти к 5. Если индекс больше или равен N перейти к 4. Выполнить сравнение $K<K_i$, если да то перейти к 4 . Если $K>K_i$ перейти к 5 . $K=K_i$ вернуть i поиск успешен.
4) Выполнить проверку q = 0. Если да, поиск неудачен закончить выполнение. В противном случае установить:
   i=i-q
   выполнить обмен (p,q) = (q, p-q)
   Перейти к 3.
5) Выполнить проверку p = 1. Если да, поиск неудачен закончить выполнение. В противном случае установить:
   i= i + q
   p=p-q
   q=q-p
   Перейти к 3.

Тут N — число элементов в последовательности. K — искомый элемент. K_i _ элемент последовательности расположенный на i — индексе. F(k) — k — й элемент в последовательности Фибоначчи

Последовательность Фибоначчи

Последовательность Фибоначчи $\{F_n\}$ — последовательность которая задаётся линейным рекуррентным соотношением:

$$F_0 = 0,\ \ F_1 = 1,\ \ F_n = F_{n-1} + F_{n-2},\ \ n \geq 2,\ \ n \in \mathbb{Z}$$

Числа Фибоначчи - элементы числовой последовательности Фибоначчи:

0, 1, 1, 2, 3, 5, 8, 13, 21……

Т.е. нулевое число это 0, первое 1, каждое последующее вычисляется как сумма двух предыдущих.

Графическая иллюстрация работы алгоритма

i

[-2, 0, 3, 5, 7, 9, 11, 15, 18, 21]

0 1 2 3 4 5 6 7 8 9

Проводим начальную инициализацию.

$N=10$

$k+1=7.\ \ T.к.\ F(7)=13>10$

$M=F(k+1)-(N+1)=F(7)-11=13-11=2$

$i=F(k)-M=F(6)-2=8-2=6$

$p=F(k-1)=F(5)=5$

$q=F(k-2)=F(4)=3$

Работа алгоритма продемонстрирована в предположении, что искомым элементом является **7**.

Переходим к 3

Графическая иллюстрация работы алгоритма

i

[-2, 0, 3, 5, 7, 9, 11, 15, 18, 21]

0 1 2 3 4 5 6 7 8 9

Входные данные:

$i=6$

$p=5$

$q=3$

Выполняем проверку:

$7 < K_6 = 11$

Переходим к 4

Графическая иллюстрация работы алгоритма

i i

[-2, 0, 3, 5, 7, 9, 11, 15, 18, 21]

0 1 2 3 4 5 6 7 8 9

Входные данные:

$i=6$

$p=5$

$q=3$

Выполняем проверку:

$q=3\neq 0$

Вычисляем:

$i=i-q=6-3=3$

$p=q=3$

$q=p-q=5-3=2$

Переходим к 3

Графическая иллюстрация работы алгоритма

i

[-2, 0, 3, 5, 7, 9, 11, 15, 18, 21]

0 1 2 3 4 5 6 7 8 9

Входные данные:

$i=3$

$p=3$

$q=2$

Выполняем проверку:

$7 > K_3 = 5$

Переходим к 5

Графическая иллюстрация работы алгоритма

i $\longrightarrow$ i

[-2, 0, 3, 5, 7, 9, 11, 15, 18, 21]

0 1 2 3 4 5 6 7 8 9

Входные данные:

$i=3$

$p=3$

$q=2$

Выполняем проверку:

$p=3\neq1$

Вычисляем:

$i=i+q=3+2=5$

$p=p-q=3-2=1$

$q=q-p=2-1=1$

Переходим к 3

Графическая иллюстрация работы алгоритма

i

[-2, 0, 3, 5, 7, 9, 11, 15, 18, 21]

0 1 2 3 4 5 6 7 8 9

Входные данные:

$i=5$

$p=1$

$q=1$

Выполняем проверку:

$7<K_5=9$

Переходим к 4

Графическая иллюстрация работы алгоритма

i i

[-2, 0, 3, 5, 7, 9, 11, 15, 18, 21]

0 1 2 3 4 5 6 7 8 9

Входные данные:

$i=5$

$p=1$

$q=1$

Выполняем проверку:

$q=1\neq 0$

Вычисляем:

$i=i-q=5-1=4$

$p=q=1$

$q=p-q=1-1=0$

Переходим к 3

Графическая иллюстрация работы алгоритма

i

[-2, 0, 3, 5, 7, 9, 11, 15, 18, 21]

0 1 2 3 4 5 6 7 8 9

Входные данные:

$i = 4$

$p = 1$

$q = 0$

Выполняем проверку:

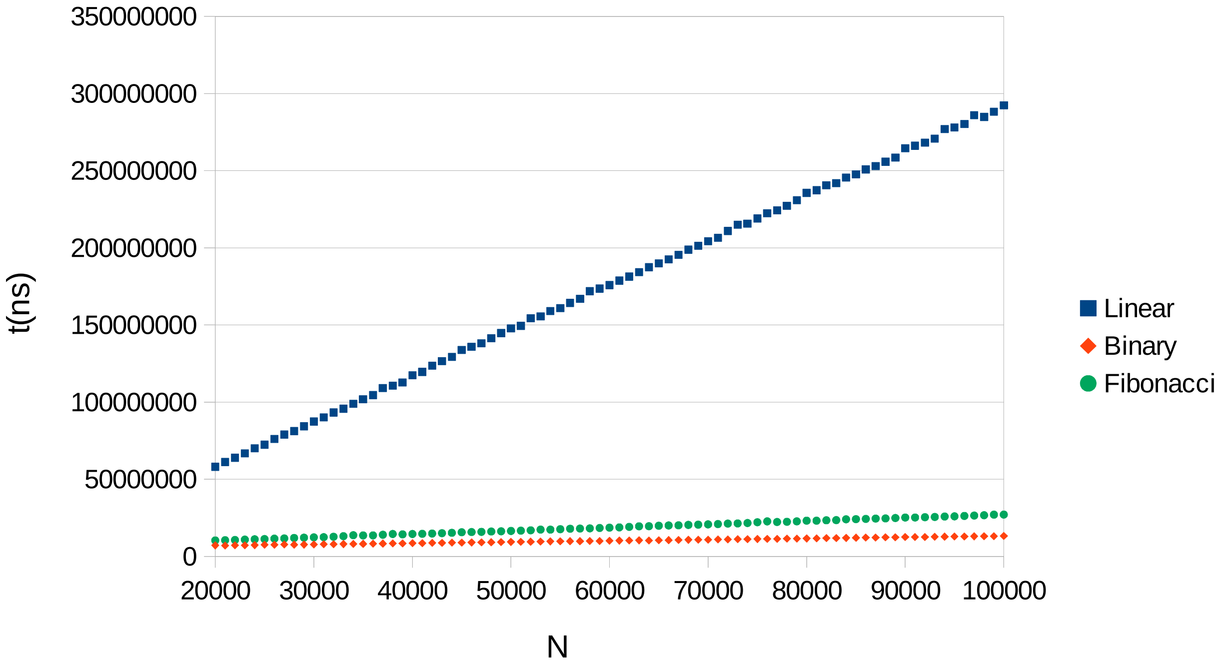
$7 = K_4 = 7$

Поиск успешен. Завершение алгоритма.

Вычислительный эксперимент

Для оценки асимптотического поведения реализации этого алгоритма был проведен следующий вычислительный эксперимент.

Для одномерного массива из 100_000 элементов построена зависимость времени поиска от количества искомых элементов. Рассмотрен как линейный,бинарный и поиск Фибоначчи (для двух последних время сортировки массива также учитывалось). Для корректности каждый замер был повторен 100 раз и было взято усредненное время.



Реализация алгоритма на Python

Реализация алгоритма на Python

Так, как эту задачу проще решить используя сохранение состояния между вызовами методов (пункты 4 и 5) то решение будет на базе класса со следующими полями.

```
class FibonacciSearch:
    def __init__(self):
        self.i = 0
        self.p = 0
        self.q = 0
        self.stop = False
```

Python

Метод для вычисления k-го члена в ряде Фибоначчи.

```
def get_fibonacci_number(self, k):
        first_number = 0
        second_number = 1
        n = 0
        while n < k:
            temp_number = second_number
            second_number = first_number + second_number
            first_number = temp_number
            n = n + 1
        return first_number
```

Python

Метод для начальной инициализации. Реализация пункта 2.

```
def start_init(self, sequince):
        self.stop = False
        k = 0
        n = len(sequince)
        while (self.get_fibonacci_number(k+1) < len(sequince)):
            k = k+1
        m = self.get_fibonacci_number(k+1)-(n+1)
        self.i = self.get_fibonacci_number(k) - m
        self.p = self.get_fibonacci_number(k-1)
        self.q = self.get_fibonacci_number(k-2)
```

Метод для уменьшение индекса. Реализация пункта 4.

```
def down_index(self):
        if self.q == 0:
            self.stop = True
        self.i = self.i-self.q
        temp = self.q
        self.q = self.p - self.q
        self.p = temp
```

Метод для увеличения индекса. Реализация пункта 5.

```
def up_index(self):
        if self.p == 1:
            self.stop = True
        self.i = self.i + self.q
        self.p = self.p - self.q
        self.q = self.q — self.p
```

Python

Метод поиска. Реализация пункта 3.

```
def search(self, sequince, element):
        self.start_init(sequince)
        result_index = -1
        while not self.stop:
            if self.i < 0:
                self.up_index()
            elif self.i >= len(sequince):
                self.down_index()
            elif sequince[self.i] == element:
                result_index = self.i
                break
            elif element < sequince[self.i]:
                self.down_index()
            elif element > sequince[self.i]:
                self.up_index()
        return result_index
```

Реализация алгоритма на Java

Реализация алгоритма на Java

Так, как эту задачу проще решить используя сохранение состояния между вызовами методов (пункты 4 и 5) то решение будет на базе класса со следующими полями.

```
public class FibonacciSearch {
    private int i;
    private int p;
    private int q;
    private boolean stop = false;

    public FibonacciSearch() {

    }
```

Java

Метод для вычисления k-го члена в ряде Фибоначчи.

```java
public long getFibonacciNumber(int k) {
    long firstNumber = 0;
    long secondNumber = 1;
    for (int i = 0; i < k; i++) {
        long temp = secondNumber;
        secondNumber += firstNumber;
        firstNumber = temp;
    }
    return firstNumber;
}
```

Метод для начальной инициализации. Реализация пункта 2.

```java
private void init(int[] sequince) {
    stop = false;
    int k = 0;
    int n = sequince.length;
    for (; getFibonacciNumber(k + 1) < n + 1;) {
        k += 1;
    }
    int m = (int) (getFibonacciNumber(k + 1) - (n + 1));
    i = (int) (getFibonacciNumber(k) - m);
    p = (int) getFibonacciNumber(k - 1);
    q = (int) getFibonacciNumber(k - 2);
}
```

Метод для уменьшение индекса. Реализация пункта 4.

```
private void downIndex() {
    if (q == 0)
        stop = true;
    i = i - q;
    int temp = q;
    q = p - q;
    p = temp;
}
```

Метод для увеличения индекса. Реализация пункта 5.

```
private void upIndex() {
    if (p == 1)
        stop = true;
    i = i + q;
    p = p - q;
    q = q - p;
}
```

Метод поиска. Реализация пункта 3.

```
public int search(int[] sequince, int element) {
    init(sequince);
    int n = sequince.length;
    int resultIndex = -1;
    for (; !stop;) {
        if (i < 0) {
            upIndex();
        } else if (i >= n) {
            downIndex();
        } else if (sequince[i] == element) {
            resultIndex = i;
            break;
        } else if (element < sequince[i]) {
            downIndex();
        } else if (element > sequince[i]) {
            upIndex();
        }
    }
    return resultIndex;
}
```

Список литературы

1) Дональд Кнут. «Искусство программирования, том 3. Сортировка и поиск» 2-е изд. М.: «Вильямс», 2007. С. 824. ISBN 0-201-89685-0. [447 -448]