Государственное образовательное учреждение высшего профессионального образования
Санкт-Петербургский национальный исследовательский университет
Информационных Технологий, Механики и Оптики
Факультет инфокоммуникационных технологий

ОТЧЕТ
ПО ЛАБОРАТОРНОЙ РАБОТЕ № 4

По алгоритмам и структурам данных
На тему: «Поиск»

Проверил(а):

_____

Дата: «___»_____201__г.

Оценка: _____

Работу выполнил(а):

Никончук А.П.
Студент(ка) группы К3242
Дневного отделения

_____

2018 г.

Цель:
Изучить алгоритмы поиска.
Задание:

1. Освоить синтаксис нового языка программирования;
2. Написать алгоритмы поиска, не упомянутые в работах ранее

Задание согласно варианту:
Напишите программу, которая создает связное упорядоченное дерево и совершает по нему симметричный и обратный симметричный обходы. Побочно решается задача создания дерева и отрисовка.
Реализация:

1. Описание принципа работы алгоритма

1.1. Бинарный поиск
Применим при поиске некоторого элемента, когда необходимо осуществить в упорядоченной по возрастанию или убыванию последовательности. Метод использует стратегию “разделяй и властвуй”: заданная последовательность делится на две равное части и поиск осуществляется в одной из этих частой, которая потом также делится надвое, до тех пор, пока не обнаружится наличие искомого элемента или устанавливаемые барьеры массива не сойдутся.
Полагаем, что массив будет упорядочен в порядке возрастания. Зоной поиска является весь массив. На первом шаге его границы – последний и начальный (нулевой) элементы. Затем при использовании инварианта происходит расширение границ +1 к левой и -1 к правой.
Инвариант цикла – это соотношение, которое истинно перед циклом, истинно в процессе выполнения цикла и истинно при выходе из цикла. (“Дисциплина программирования” Дейкстры, “Наука программирования” Гриса)
В данном случае присутствуют условия проверки границ на совпадение с искомым элементом, тем самым указанная метрика исключается. В любом случае передвижение левой и правой границ устанавливает рамки работы алгоритма, и тем самым предотвращает появление бесконейного цикла.

1.2. Поиск барьером
Напоминает линейный поиск, но в отличии от него в качестве последнего элемента массива устанавливается искомый, если последний элемент отличается от него. Добавление барера позволяет избавиться от сравнения внутри цикла, что позволяет алгоритму работать быстрее.

2. Код программы на 2 языках программирования

2.1. JavaScript

```
function Binary(arr, f){
   console.log('binary')
   barier1 = 0;
   barier2 = arr.length
   console.log(barier1 + ' ' + barier2);
   if (f>barier1 and barier2>f){
      while(barier1<=barier2){
         currentElem = barier1 + Math.floor((barier2-barier1)/2);
         console.log(arr[barier1] + ' ' + arr[currentElem] + ' ' + arr[barier2-1]);
         if arr[barier1]==f{
            return barier1
         }
         if arr[barier2-1]==f{
            return barier2
         }
         if arr[currentElem]!=f{
            if arr[currentElem]<f{
               barier1 = currentElem
            }
```

```
            if arr[currentElem]>f{
                barier2 = currentElem
            }
        }else{
            return currentElem
        }
    }
  }
  return 'not found'
}

function barier(arr, f)
{
            console.log('barier');
            var position = 0;
            if (arr[arr.length - 1] != f)
            {
                    console.log("in if");
                    arr[arr.length - 1] = f;
                    for (; arr[position] != f; position++);
            } else {
                    console.log("else");
                    return arr.length;
            }
            return position < arr.length-1 ? position : 0;
}
```

2.2. Python

```
import os
import sys
from matplotlib import pyplot as plt
import numpy as np

def binary(arr, f):
  print('binary')
  barier1 = 0
  barier2 = len(arr)
  print(barier1, ' ', barier2);
  if f>barier1 and barier2>f:
    while(barier1<=barier2):
    #for i in range(barier2):
      currentElem = barier1 + (barier2-barier1)//2;
      print(arr[barier1], ' ', arr[currentElem], ' ', arr[barier2-1]);
      if arr[barier1]==f:
        return barier1
      if arr[barier2-1]==f:
        return barier2
      if arr[currentElem]!=f:
        print('ce dont equal f')
        if arr[currentElem]<f:
          barier1 = currentElem
        if arr[currentElem]>f:
          barier2 = currentElem
      else:
        return currentElem
  return 'not found'
```

```
def barier(array, find_elem):
    position = 0
    if (array[len(array)-1] != find_elem):
        print("in if")
        array.append(find_elem)
        while (array[position] != find_elem):
            position+=1
    else:
        print("else")
        return len(array)-1
    return position if position < len(array)-1 else 0
```

3. Оценка сложности алгоритма

На каждом шаге алгоритм бинарного поиска уменьшает зону поиска вдвое. Если всего элементов N, то после прохождения O(logN) шагов в части массива, где может располагаться искомый элемент, останется только одно значение. Т.о. будет найден индекс искомого элемента, либо обнаружится, что он отсутсвует в массиве.
Вычислительная сложность поиска с барьером меньше, чем у линейного поиска, но имеет тот же порядок, в котором при равномерном распределении элементов в массиве среднее время поиска обычно пропор-ционально величине n/2, в худшем n.

4. Пример входных данных & примеры результатов работы

4.1. Python

```
array = [0, 1,2,3,4,5,6,7,8,9,10,11,12]
numforfind = 4
print(binary(array, numforfind))
```

```
binary
0   13
0   6   12
ce dont equal f
0   3   5
ce dont equal f
3   4   5
4
```

```
array = [0, 1,2,3,4,5,6,7,8,9,10,11,12,13]
numforfind = 6
print(binary(array, numforfind))
```

```
binary
0   14
0   7   13
ce dont equal f
0   3   6
7
```

```
array = [0, 1,2,3,4,5,6,7,8,9,10,11,12,13]
numforfind = 13
print(binary(array, numforfind))
```

```
binary
0   14
0   7   13
14
```

```
array = [0, 1,2,3,4,5,6,7,8,9,10,11,12]
numforfind = 11
print(binary(array, numforfind))
```

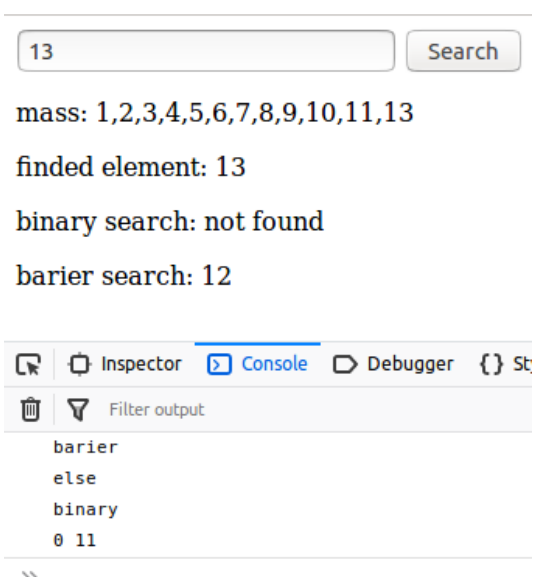
```
binary
0   13
0   6   12
ce dont equal f
6   9   12
ce dont equal f
9   11   12
11
```

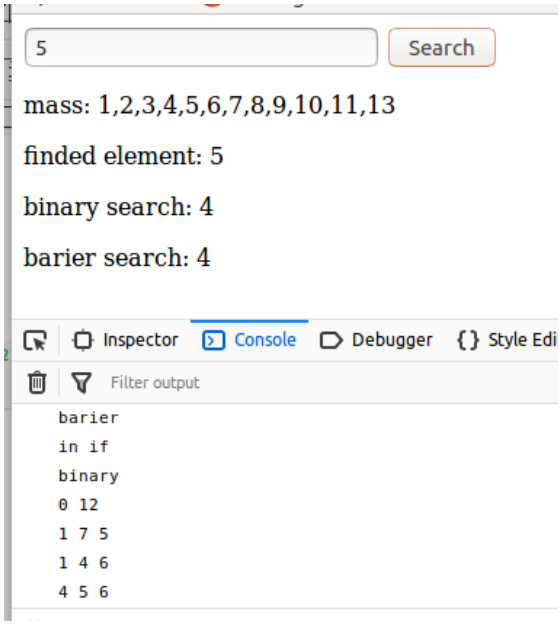
```
In [48]: array = [1, 2, 3, 4, 6, 7, 9, 12, 28]
         barier(array, 28)

         else

Out[48]: 8

In [50]: barier(array, 3)

         in if

Out[50]: 2

In [51]: barier(array, 5)

         in if

Out[51]: 0
```

```
array = [0, 1,2,3,4,5,6,7,8,9,10,11,12]
numforfind = 14
print(binary(array, numforfind))
```

```
binary
0   13
not found
```

4.2. JavaScript





Выводы:

В результате выполенииния работы были разобраны базовые алгоритмы сортировок с барьером(ами), сложность выполенииния бинарного поиска, а также особенности работы с различными структурами дынных на примере двух языков программировая Python и JavaScript.