**Федеральное агентство связи  
Ордена Трудового Красного Знамени федеральное государственное бюджетное образовательное учреждение высшего образования   
«Московский технический университет связи и информатики»**

Кафедра «Теории вероятностей и прикладной математики»

**Лабораторная работа №2**

По дисциплине «Структуры и алгоритмы обработки данных»   
по теме «Методы поиска»  
Вариант №6

Выполнила: Гердт Таисия

Группа: БСТ1955

Руководитель: Кутейников Иван Алексеевич

## Задание

Реализовать заданный метод поиска в соответствии с индивидуальным заданием. Организовать генерацию начального набора случайных данных. Для всех вариантов добавить реализацию добавления, поиска и удаления элементов. Оценить время работы каждого алгоритма поиска и сравнить его со временем работы стандартной функции поиска, используемой в выбранном языке программирования.

В качестве языка для реализации алгоритмов был выбран язык Python.

Код программы:

#класс – узел дерева, хранит в себе значение текущего элемента и указание на левый(меньший) и правый(больший) элемент

class Node():

def \_\_init\_\_(self, key):

self.left = None

self.right = None

self.val = key

#метод поиска значение в узлах дерева

def search(root, key):

if root is None or root.val == key:

return root

if root.val < key:

return search(root.right,key)

return search(root.left,key)

#метод поиска узла дерева с минимальным значением

def min(root):

if root.left is None: return root

else: return min(root.left)

#метод удаления узла из дерева

def delete(root, key):

if root is None:

return root

if key < root.val:

root.left = delete(root.left, key)

elif(key > root.val):

root.right = delete(root.right, key)

else:

if root.left is None:

temp = root.right

root = None

return temp

elif root.right is None:

temp = root.left

root = None

return temp

temp = min(root.right)

root.val = temp.val

root.right = delete(root.right, temp.val)

return root

#метод вставки значения в дерево

def insert(root, key):

if root is None:

return Node(key)

else:

if root.val == key:

return root

elif root.val < key:

root.right = insert(root.right, key)

else:

root.left = insert(root.left, key)

return root

#метод – вывод узлов дерева в порядке возрастания

def inorder(root):

if root:

inorder(root.left)

print(root.val)

inorder(root.right)

import random, time

a = random.sample(range(-2147483648, 2147483647), 100000)

#print(f"Generated array:")

#for i in a: print(i)

b = Node(a[0]) #первый сгенерированного массива задаем корнем дерева

for i in range (1, len(a)-1):

b = insert(b, a[i]) #каждый следующий элемент массива вставляем в дерево

#print(f"\nBinary tree:")

#inorder(b)

k = a[-1]

t = time.time()

search(b, k)

t = (time.time() - t)\*1000

print(f"Binary search tree execution time: {t\*1000} milliseconds")

t = time.time()

a.index(k)

t = (time.time() - t)\*1000

print(f"Python search execution time: {t\*1000} milliseconds")

Выполнение программы:

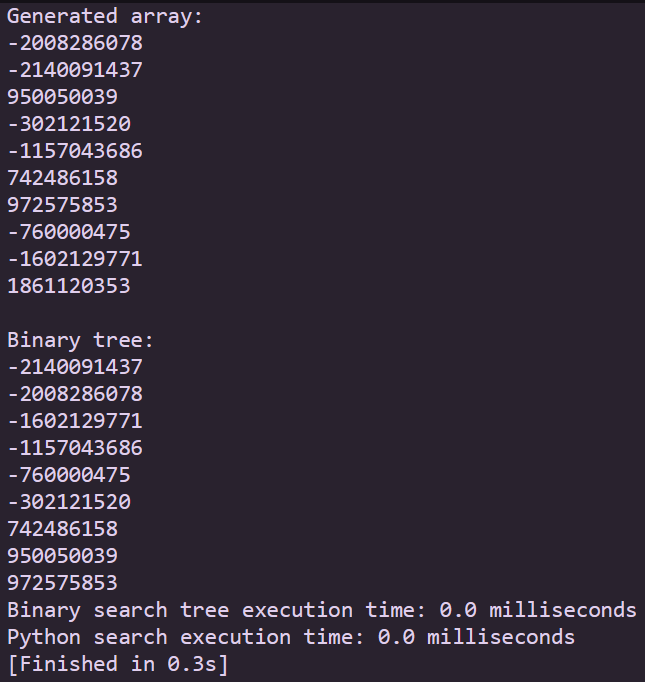


Рисунок 1. Результат выполнения программы с массивом размера 10

Составим таблицу времени выполнения поиска:

Таблица 1. Время выполнения поиска

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Сортировка\размер** | **1000** | **10000** | **100000** | **100000** |
| **BinarySort** | 0.0000 | 0.0000 | 0.0000 | 0.0000 |
| 0.0000 | 0.0000 | 0.0000 | 997.5433 |
| 0.0000 | 0.0000 | 0.0000 | 0.0000 |
| **Среднее** | 0.0000 | 0.0000 | 0.0000 | 332.5144 |
| **Index** | 997.5433 | 997.5433 | 2993.3453 | 19947.5288 |
| 0.0000 | 996.8281 | 2993.1068 | 18949.7471 |
| 0.0000 | 998.0202 | 2989.7690 | 20946.0258 |
| **Среднее** | 332.5144 | 997.4639 | 2992.0737 | 19947.7673 |

Так как для генерации бинарного дерева требуется большое количество времени, выполнение самой программы заметно увеличивается с увеличением размера массива:

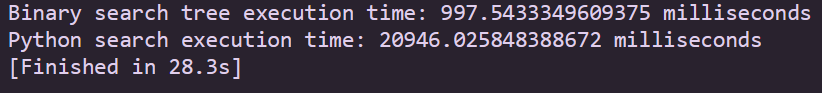


Рисунок 2. Сравнение времени выполнения поиска и основной программы

В данном случае, программа выполнялась без вывода массивов на экран, потому временем, затраченным на выполнение остальных команд, можно пренебречь: для генерации дерева потребовалось около 7 секунд, для поиска элемента в нем – менее секунды, для поиска в массиве – около 21 секунды.

**Вывод:** Поиск в бинарном дереве показал значительное преобладание в скорости над стандартной функцией поиска особенно с увеличением количества данных в массиве, но требует время на генерацию бинарного дерева.