МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «ЛЬВІВСЬКА ПОЛІТЕХНІКА»

Звіт
до лабораторної роботи з дисципліни
«Теорія алгоритмів та структур даних»

Виконав: Пфайфер В.В.
Група: ТР - 35
Прийняв: Андрущак В.С.

Львів 2021

Мета роботи: дослідження однозвязних списків та дерев в середовищі Jupyter

Хід роботи:

1. Здійснимо ініціалізацію однозв’язного списку на базі згенерованого масиву даних з 10 000 елементів.
2. Використаємо метод вставки для кількості елементів списку – 50, 100, 500, 1000, 2000, 5000, 10 000 – n. Повторимо дану операцію 10 000 разів. Виведемо середній час опрацювання та к-сть необхідної пам'яті для даного методу.
3. Побудуємо графік за допомогою бібліотеки Matplotlib

Однозвязний список:

```
t1 = 0
x = random.randint(1000000, size=(10000)) #creating list
process = psutil.Process(os.getpid())

#A single node of a singly linked list
class Node:
  # constructor
  def __init__(self, data = None, next=None):
    self.data = data
    self.next = next

# A Linked List class with a single head node
class LinkedList:
  def __init__(self):
    self.head = None

  # insertion method for the linked list
  def insert(self, data):
    newNode = Node(data)
    if(self.head):
      current = self.head
      while(current.next):
        current = current.next
      current.next = newNode
    else:
      self.head = newNode

  def deleteHead(self): #sam write
    if(self.head):
      current = self.head
      self.head = current.next
    else:
       return
```

```
  # print method for the linked list
  def printLL(self):
    current = self.head
    while(current):
      print(current.data)
      current = current.next

# Singly Linked List with insertion and print methods
LL=LinkedList()
for i in x:
    LL.insert(i)

for k in range(1,10001,1):
    t1_start = process_time()
    LL.insert(random.randint(0,50000))
    t1_stop = process_time()
    t=t1_start+t1_stop
    t1=t1+t
    LL.deleteHead()
    av_t=t1/10000
print("average t for 10000 elements is: ", av_t)
print("Memory Usage:", (process.memory_info().rss))  # in Mbits
```

Табл.1 Складність методу insert для однозв’язного списку

к-сть елементів	50	100	500	1000	2000	5000	10 000
час	2.84 с	2.77 с	3.45 с	3.92 с	4.82 с	9.57 с	18.9 с
пам’ять	787 мб	785 мб	786 мб	787 мб	792 мб	797 мб	806 мб

```
average t for 50 elements is:  2.8370703125
Memory Usage: 78704640

average t for 100 elements is:  2.7619984375
Memory Usage: 78512128

average t for 500 elements is:  3.453709375
Memory Usage: 78557184

average t for 1000 elements is:  3.919290625
Memory Usage: 78684160

average t for 2000 elements is:  4.816315625
Memory Usage: 79163392

average t for 5000 elements is:  9.5633265625
Memory Usage: 79659008
```

```
average t for 10000 elements is:  18.8898671875
Memory Usage: 80588800
```

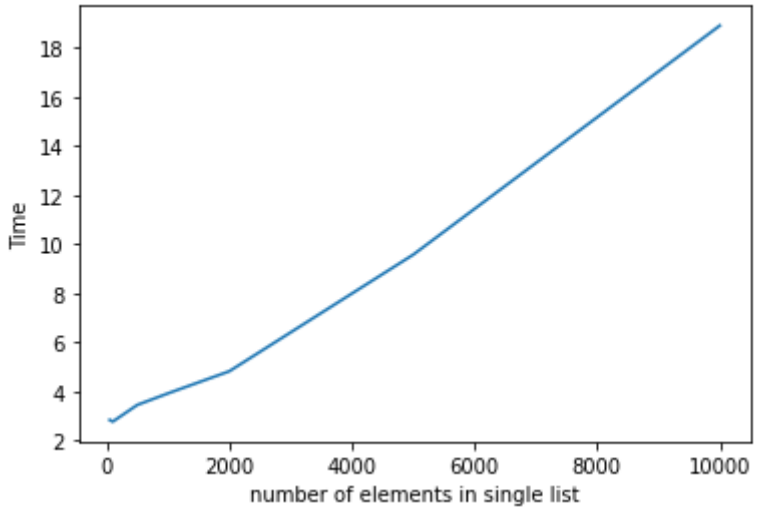

Рис.1 Складність методу вставки елемента в однозв’язний список (Час)

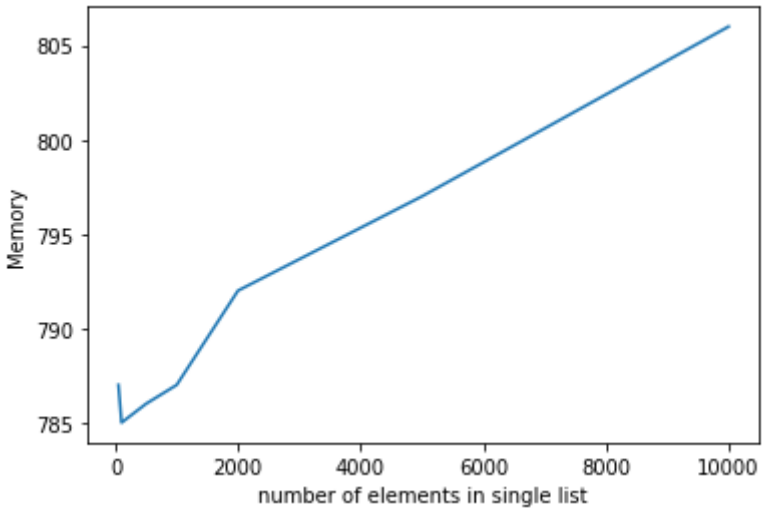

Рис.2 Складність методу вставки елемента в однозв’язний список (Пам’ять)

Дерево:

```
tree = Tree()
tree.create_node("Harry", "harry")  # root node
tree.create_node("Jane", "jane", parent="harry")
tree.create_node("Bill", "bill", parent="harry")
for x in range(2,3,1): # For change A
    tree.create_node(random.choice(string.ascii_letters), x, parent="jane")
tree.create_node("hopkins", "hopkins", parent="jane")
for k in range(2510,2610,1): # For change B
    tree.create_node(random.choice(string.ascii_letters), k, parent="hopkins")
for d in range(5004,5405,1): # For change C
```

```
    tree.create_node(random.choice(string.ascii_letters), d, parent="bill")
tree.create_node("welles", "welles", parent="bill")
for z in range(7510,180000,1): # For change D
    tree.create_node(random.choice(string.ascii_letters), z, parent="welles")
#time processing
for j in range(1,10001,1):
    t1_start = process_time()
    tree.create_node("Time_Node", 1212212, parent="jane")
    t1_stop = process_time()
    t=t1_start+t1_stop
    t1=t1+t
    tree.remove_node(1212212)
    av_t=t1/10000
print("average t for 10000 elements is: ", av_t)
print("Memory Usage:", (process.memory_info().rss))  # in Mbits
```

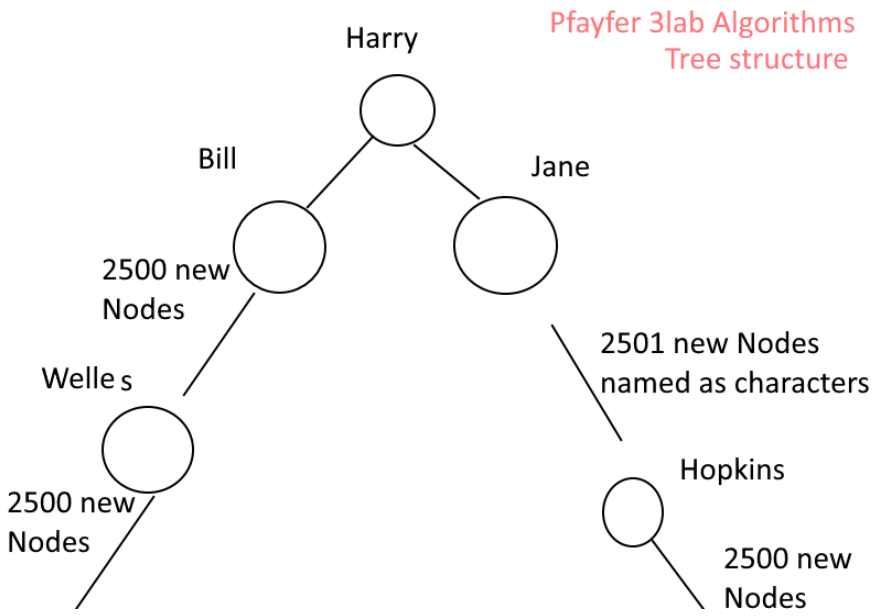

Рис.3 Структура дерева

Табл.2 Складність методу вставки для дерева

к-сть елементів	50	100	500	1000	2000	5000	10 000
час	2.9 с	3.1 с	2.93 с	2.75 с	3 с	5.73 с	7.4 с
пам’ять	79.7 мб	79.7 мб	79.4 мб	79.8 мб	80.1 мб	168 мб	217 мб

```
average t for 50 elements is:  2.868715625
Memory Usage: 79728640
Harry
├── Bill
│   ├── V
│   └── welles
│       └── x
└── Jane
    ├── O
    └── hopkins
        ├── B
```

```
average t for 100 elements is:  3.115959375
Memory Usage: 79691776
```

```
average t for 500 elements is:  2.9269265625
Memory Usage: 79437824
```

```
average t for 1000 elements is:  2.74643125
Memory Usage: 79843328
```

```
average t for 2000 elements is:  3.0138171875
Memory Usage: 80752640
```

```
average t for 5000 elements is:  5.7307953125
Memory Usage: 168005632
```

```
average t for 10000 elements is:  7.383690625
Memory Usage: 217419776
```

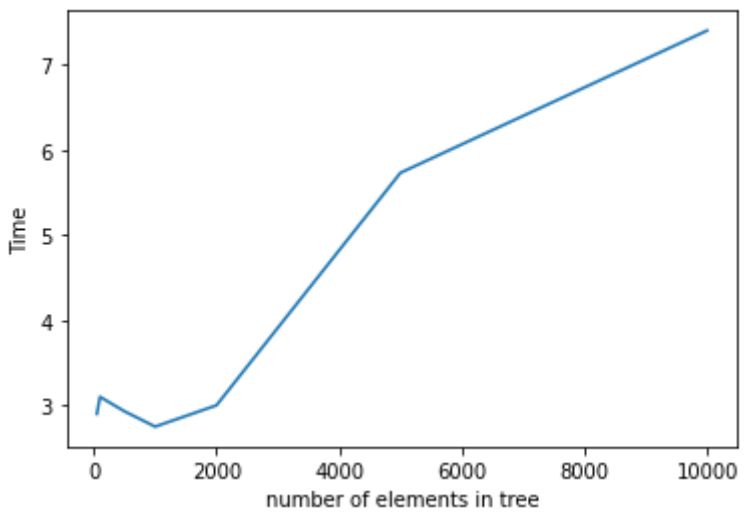

Рис.4 Складність методу вставки елемента в дерево (Час)

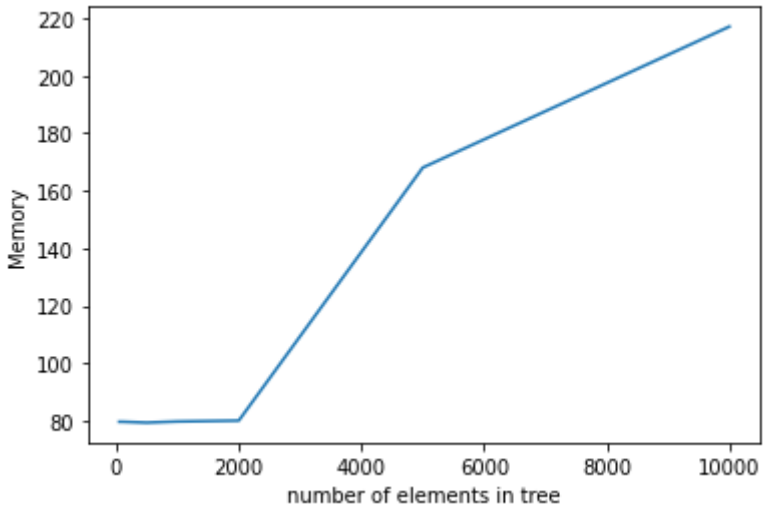

Рис.5 Складність методу вставки елемента в дерево (Пам’ять)

Висновок: на даній лабораторній роботі досліджено складність алгоритмів вставки у однозв’язному списку та дереві. При створенні однозв’язного списку використовувався самописний мало оптимізований варіант з джерела, яке вказаного перед функціональним кодом. Складність даного методу вставки наближена до O(n). Для побудови дерева використано методи бібліотеки TreeLib. Наближена структура побудованого дерева зображена на Рис.3. Із рисунків (Рис.4 та Рис.5) помітно, що метод реалізації вставки є мало оптимізованим, функція складності наближена до O(n). Отже, дані методи вставки (у O.C та Дерево) потребують оптимізації, яку слід проводити із глибшим розумінням теорії алгоритмів, та функціональних структур відповідних бібліотек.