Trees

March 15, 2020

```
In [1]: class Node:
            def __init__(self, key, left=None, right=None):
                self.key = key
                self.left = left
                self.right = right

In [2]: l = Node(10)
        r = Node(20)

In [3]: print(l.key, l.left, l.right)

10 None None

In [4]: m = Node(15, l, r)

In [5]: print(m.key, m.left, m.right)

15 <__main__.Node object at 0x7fc4a815d048> <__main__.Node object at 0x7fc4a815d0f0>

In [6]: print(m.left.key, m.key, m.right.key)

10 15 20

In [7]: def printNode(n: Node):
            if n != None:
                printNode(n.left)
                print(n.key)
                printNode(n.right)

In [8]: printNode(m)

10
15
20
```

In [9]:
```
def findNode(root, key):
    n = root
    while n != None and n.key != key:
        if key < n.key:
            n = n.left
        else:
            n = n.right
    return n
```

In [10]:
```
print(findNode(m, 10))
print(findNode(m, 12))
```

```
<__main__.Node object at 0x7fc4a815d048>
None
```

In [11]:
```
def addNode(root, key):
    if root == None:
        root = Node(key)
    else:
        n = root
        while n != None:
            if key <= n.key:
                if n.left == None:
                    n.left = Node(key)
                    n = None
                else:
                    n = n.left
            else:
                if n.right == None:
                    n.right = Node(key)
                    n = None
                else:
                    n = n.right
    return root
```

In [12]:
```
m = addNode(m,18)
printNode(m)
```

```
10
15
18
20
```

In [13]:
```
def nodeHeight (root):
    if root == None:
        return 0
    else:
```

```
            l = nodeHeight(root.left)
            h = nodeHeight(root.right)
            return 1 + max(l,h)
```

In [14]:
```
nodeHeight(m)
```

Out[14]: 3

In [15]:
```
class Tree:
    def __init__(self):
        self.root = None

    def find(self,key):
        return findNode(self.root, key)

    def has(self,key):
        return self.find(key) != None

    def add(self,key):
        self.root = addNode(self.root, key)

    def height(self):
        return nodeHeight(self.root)

    def print(self):
        printNode(self.root)
```

In [16]:
```
t = Tree()
```

In [17]:
```
print(t.find(10))
```

```
None
```

In [18]:
```
t.root = m
```

In [19]:
```
print(t.has(10))
```

```
True
```

In [20]:
```
t.add(20)
```

In [21]:
```
t.print()
```

```
10
15
18
20
20
```

In [22]:
```python
# tools
import time
import random
import matplotlib.pyplot as plt
# generate random data
def make_random(n):
    return [random.randint(1,100000) for i in range(0,n)]
# generate ascending sequence of data
def ascending(n):
    return [i for i in range(0,n)]
# geenrate descending sequence of data
def descending(n):
    return [n-i for i in range(0,n)]
# check if data x are sorted
def check(x):
    n = len(x)
    for i in range(1,n):
        assert x[i-1] <= x[i]
# measure the time used to apply some sorting algorithm "algo" to data "x"
def measure_time(algo,x):
    start = time.time()
    algo(x)
    end = time.time();
    return end - start
# plot time of some sorting algorithms
def show_time(n,scenario,sorts):
    durations = []
    m = len(sorts)
    plt.figure(figsize=(10,6))
    for k in range(0,m):
        durations.append([measure_time(sorts[k][0],scenario(i)) for i in range(1,n)])
        plt.plot(durations[k], label=sorts[k][1])
    plt.xlabel("Array length")
    plt.ylabel("Runtime (s)")
    plt.legend(bbox_to_anchor=(0.85, 1), loc='lower right', borderaxespad=0.)
    plt.show()
```

In [23]:
```python
t = Tree()
for x in make_random(20):
    t.add(x)
```

In [24]:
```python
t.print()
```

```
8532
10240
21530
29188
30040
```

```
37881
41050
51173
54412
63337
71654
73009
78604
80301
83484
85218
86058
86728
91339
99028
```

In [ ]:

In [25]:
```
# right limit r exclusive
def binary_search_r(x, l, r, b):
    if l==r: # not found
        return l
    assert(l<r)
    m = (l+r) // 2
    if b < x[m]:
        return binary_search_r(x,0,m,b)
    elif b > x[m]:
        return binary_search_r(x,m+1,r,b)
    else: # b == x[m] found
        return m
def binary_search(x,b):
    return binary_search_r(x,0,len(x),b)
def insert_binary(x,b):
    p = binary_search(x,b)
    x.insert(p,b)
    return x
```

In [26]:
```
def merge(x,l,m,r):
    result = []
    i = l
    j = m
    while i < m and j < r:
        if x[i] < x[j]:
            result.append(x[i])
            i = i + 1
        else:
            result.append(x[j])
```

```
                j = j + 1
        while i < m:
            result.append(x[i])
            i = i + 1
        while j < r:
            result.append(x[j])
            j = j + 1
        for i in range(l,r):
            x[i] = result[i-l]

    # recursively sort data in x[l:r], (l included, r excluded!)
    def merge_sort_r(x,l,r):
        assert(0 <= l and l <= r)
        assert(r <= len(x))
        if r - l > 1: # more than one element
            m = (l+r) // 2
            merge_sort_r(x,l,m)
            merge_sort_r(x,m,r)
            merge(x,l,m,r)

    def merge_sort(x):
        merge_sort_r(x,0,len(x))
```

```
In [27]: def testTreeInsertion(data):
             t = Tree()
             for x in data:
                 t.add(x)
         def testArrayInsertion(data):
             a=[]
             for x in data:
                 insert_binary(a,x)

         n = 100000
         print(measure_time(testTreeInsertion,make_random(n)))
         print(measure_time(testArrayInsertion,make_random(n)))
         print(measure_time(merge_sort,make_random(n)))
```

```
0.7085223197937012
2.9137728214263916
0.5461540222167969
```

```
In [28]: t = Tree()
         a=[]
         def testTreeInsertion(data):
             for x in data:
                 t.add(x)
         def testArrayInsertion(data):
```

```
        for x in data:
            insert_binary(a,x)

    n = 200000
    print("insert many points into an initially empty data structure")
    print("tree", measure_time(testTreeInsertion,make_random(n)))
    print("array", measure_time(testArrayInsertion,make_random(n)))
    print("array, post sort", measure_time(merge_sort,make_random(n)))
    print("tree height=", t.height())
    # now measure further insertion into the large data structure
    n = 10000
    print("insert some more points")
    print("tree",measure_time(testTreeInsertion,make_random(n)))
    print("array",measure_time(testArrayInsertion,make_random(n)))
    print("tree height=", t.height())
```

```
insert many points into an initially empty data structure
tree 1.5785188674926758
array 8.626590490341187
array, post sort 1.1173694133758545
tree height= 45
insert some more points
tree 0.07555675506591797
array 0.687213659286499
tree height= 47
```

```
In [32]: # fill a tree and a sorted array wtih 100000 elements each
         t = Tree()
         for x in make_random(500000):
             t.add(x)
         a = make_random(500000)
         a.sort()
         print("tree height=", t.height())

         # now measure insertion into the large data structure
         n = 100000
         print("insert in full tree",measure_time(testTreeInsertion,make_random(n)))
         print("insert in full array",measure_time(testArrayInsertion,make_random(n)))
         print("tree height=", t.height())
```

```
tree height= 49
insert in full tree 1.3614959716796875
insert in full array 14.158589839935303
tree height= 50
```

```
In [33]: # the bad news...
         t = Tree()
```

```
a = []
n = 2000
print("ascending")
print("tree",measure_time(testTreeInsertion,ascending(n)))
print("array",measure_time(testArrayInsertion,ascending(n)))
print("tree height=", t.height())
t = Tree()
a = []
print("descending")
print("tree",measure_time(testTreeInsertion,descending(n)))
print("array",measure_time(testArrayInsertion,descending(n)))
print("tree height=", t.height())
```

```
ascending
tree 0.27419185638427734
array 0.005477190017700195
tree height= 2000
descending
tree 0.2688593864440918
array 0.0052337646484375
tree height= 2000
```

```
In [ ]:
```