**프로그래밍 실습 #5**

201501489 최영진

다음과 같은 탐색 알고리즘을 파이썬으로 구현하고, N 값의 변화에 따른 실행

시간을 측정한 후 실행 시간을 비교해 본다.

1. 이진 트리 탐색

class node:  
 def \_\_init\_\_(self, key=None, left=None, right=None):  
 self.key = key  
 self.left = left  
 self.right = right  
  
class Dict:  
 x = p = node  
 z = node(key=0, left=0, right=0)  
 z.left = z  
 z.right = z  
 head = node(key = 0, left = 0, right = z)  
  
 def search(self, search\_key):  
 x = self.head.right  
 while x != self.z:  
 if x.key == search\_key:  
 return x.key  
 if x.key > search\_key:  
 x = x.left  
 else:  
 x = x.right  
 return -1  
  
 def insert(self, v):  
 x = p = self.head  
 while (x != self.z):  
 p = x  
 if x.key == v:  
 return  
 if x.key > v:  
 x = x.left  
 else:  
 x = x.right  
 x = node(key=v, left=self.z, right=self.z)  
 if p.key > v:  
 p.left = x  
 else:  
 p.right = x  
  
import random, time  
N = 10000  
key = list(range(1, N+1))  
s\_key = list(range(1, N+1))  
random.shuffle(key)  
d = Dict()  
for i in range(N):  
 d.insert(key[i])  
start\_time = time.time()  
for i in range(N):  
 result = d.search(s\_key[i])  
 if result == -1 or result != s\_key[i]:  
 print('탐색 오류')  
end\_time = time.time() - start\_time  
print('이진 트리 탐색의 실행 시간 (N = %d) : %0.3f'%(N, end\_time))  
print('탐색 완료')

**2) 성능 비교**

**- 난수 발생 시**

N = 100000 : 0.693, 0.705, 0.771 / **평균 :** **0.723sec**

N = 500000: 4.034, 3.669, 4.044 / **평균 : 3.916sec**

N = 1000000 : 7.482, 8.045, 8.831 / **평균 : 8.119sec**

**- 정렬 시** – random.shuffle(key)를 사용하지 않음.

N = 1000 : 0.174, 0.127, 0.157 / **평균 : 0.153sec**

N = 5000 : 3.313, 3.169, 3.598 / **평균 : 3.36sec**

N = 10000 : 15.526, 13.634, 14.708 / **평균 : 14.623sec**

**- 역순 정렬 시** – random.shuffle(key)를 사용하지 않고, key.reverse() 함수 사용

N = 1000 : 0.214, 0.138, 0.144 / **평균 : 0.165sec**

N = 5000 : 3.809, 3.710, 3.567 / **평균 : 3.695sec**

N = 10000 : 32.387, 32.537, 33.226 / **평균 : 32.717sec**

(2) 레드-블랙 트리 탐색

black = 0  
red = 1  
  
class node:  
 def \_\_init\_\_(self, color, key=None, left=None, right=None):  
 self.color = color  
 self.key = key  
 self.left = left  
 self.right = right  
  
class Dict:  
 x = p = q = gg = node  
  
 z = node(color=black, key=0, left=0, right=0) # 단말노드  
 z.left = z  
 z.right = z  
 head = node(color=black, key=0, left=0, right=z) # 헤드노드, z가 현재 루트  
  
 def check(self, check\_key):  
 p = self.head.right # p는 현재 루트 노드  
 x = self.head.right # x는 현재 루트 노드  
 while x != self.z:  
 if x.key == check\_key:  
 if x.color == 0:  
 print('key :', x.key, ', parents :', p.key, ', color : black')  
 else:  
 print('key :', x.key, ', parents :', p.key, ', color : red')  
 return  
 if x.key > check\_key:  
 if p != x:  
 p = x  
 x = x.left  
 else:  
 if p != x:  
 p = x  
 x = x.right  
  
 def search(self, search\_key):  
 x = self.head.right  
 while x != self.z:  
 if x.key == search\_key:  
 return x.key  
 if x.key > search\_key:  
 x = x.left  
 else:  
 x = x.right  
 return -1  
  
 def insert(self, v):  
 x = p = g = self.head  
 while x != self.z:  
 gg = g  
 g = p  
 p = x  
 if x.key == v:  
 return  
 if x.key > v:  
 x = x.left  
 else:  
 x = x.right  
 if x.left.color and x.right.color:  
 self.split(x, p, g, gg, v)  
 x = node(color=red, key=v, left=self.z, right=self.z)  
 if p.key > v:  
 p.left = x  
 else:  
 p.right = x  
 self.split(x, p, g, gg, v)  
 self.head.right.color = black  
  
 def split(self, x, p, g, gg, v):  
 x.color = red  
 x.left.color = black  
 x.right.color = black  
 if p.color:  
 g.color = red  
 if (g.key > v) != (p.key > v):  
 p = self.rotate(v, g)  
 x = self.rotate(v, gg)  
 x.color = black  
  
 def rotate(self, v, y):  
 gc = c = node  
 if y.key > v:  
 c = y.left  
 else:  
 c = y.right  
 if c.key > v:  
 gc = c.left  
 c.left = gc.right  
 gc.right = c  
 else:  
 gc = c.right  
 c.right = gc.left  
 gc.left = c  
 if y.key > v:  
 y.left = gc  
 else:  
 y.right = gc  
 return gc  
  
import random, time  
  
N = 9  
key = [2, 1, 8, 9, 7, 3, 6, 4, 5]  
key\_s = list(range(1, 10))  
  
# N = 10000  
# key = list(range(1, N+1))  
# key\_s = list(range(1, N+1))  
# random.shuffle(key)  
  
  
d = Dict()  
i = 0  
while i < len(key):  
 d.insert(key[i])  
 i += 1  
  
i = 0  
start\_time = time.time()  
while i < len(key):  
 result = d.search(key\_s[i])  
 if result == -1 or result != key\_s[i]:  
 print("탐색 오류")  
 i += 1  
end\_time = time.time() - start\_time  
  
  
i = 0  
while i < len(key):  
 d.check(key\_s[i])  
 i += 1  
print('레드 블랙 트리 탐색의 실행 시간 (N = %d) : %0.3f'%(N, end\_time))  
print("탐색 완료")

**2) 성능비교**

**- 난수 발생 시**

N = 100000 : 1.232, 1.432, 1.600 / **평균 :** **1.421sec**

N = 200000 : 3.128, 2.820, 3.082 / **평균 : 3.01sec**

N = 500000 : 9.095, 8.751, 7.728 / **평균 : 8.525sec**

**- 정렬 시** – random.shuffle(key)를 사용하지 않음.

N = 100000 : 0.952, 1.044, 1.290 / **평균 : 1.095sec**

N = 200000 : 3.144, 2.718, 3.178 / **평균 : 3.013sec**

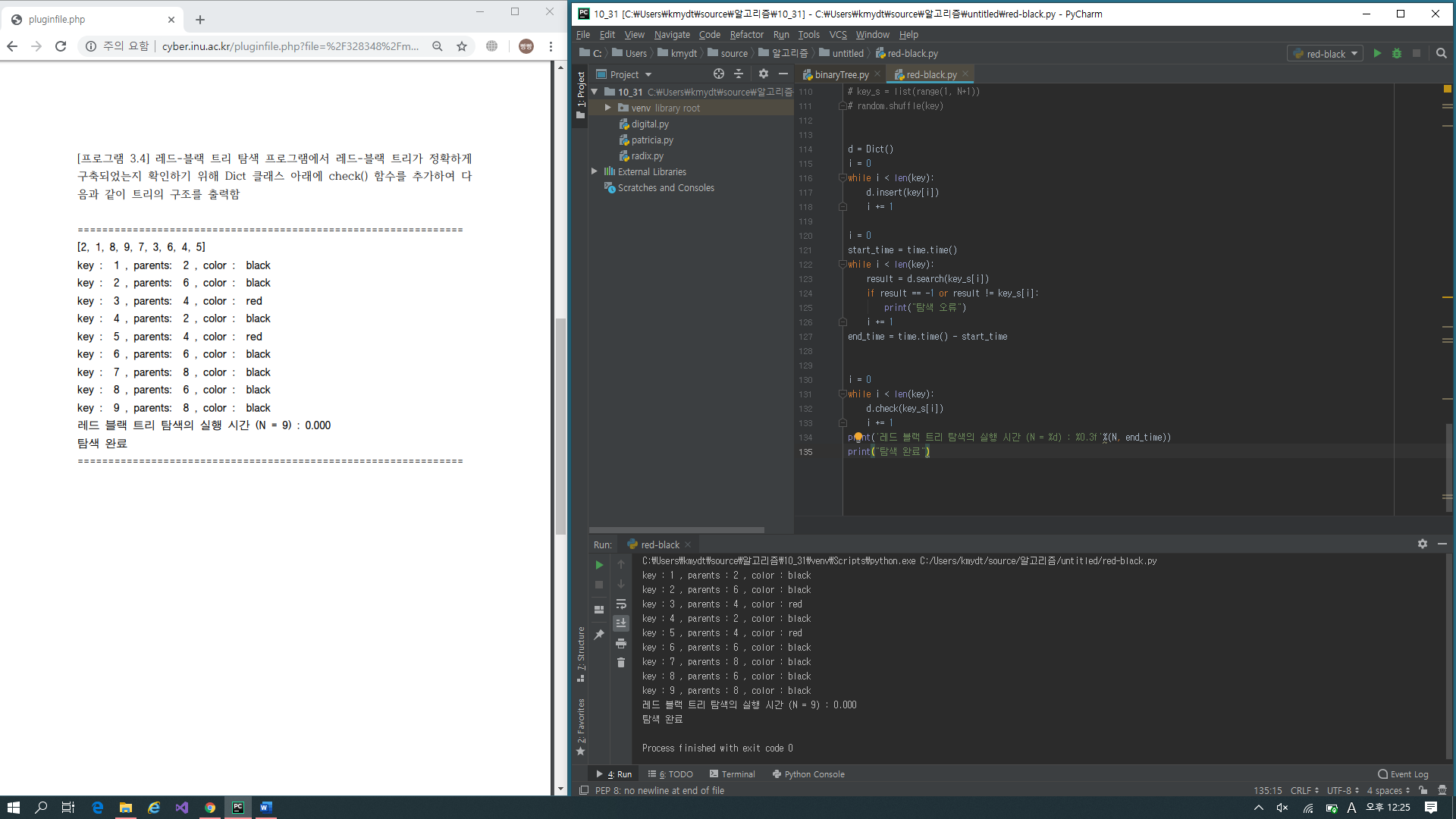
N = 500000 : 7.090, 7.160, 7.014 / **평균 : 7.088sec**

**- 역순 정렬 시** – random.shuffle(key)를 사용하지 않고, key.reverse() 함수 사용

N = 100000 : 1.383, 1.316, 1.606 / **평균 : 1.435sec**

N = 200000 : 3.261, 2.497, 2.287 / **평균 : 2.682sec**

N = 500000 : 7.312, 7.201, 7.206 / **평균 : 7.240sec**



(3) AVL 트리 탐색

class node:  
 def \_\_init\_\_(self, key):  
 self.key = key  
 self.left = None  
 self.right = None  
  
class Dict:  
 def \_\_init\_\_(self):  
 self.node = None  
 self.height = 0  
 self.balance = 0  
  
  
 def check(self, check\_key):  
 x = self.node  
 p = self.node  
 while x is not None:  
 if x.key == check\_key:  
 print('key :', x.key, ', parents :', p.key)  
 return  
 if x.key > check\_key:  
 if p != x:  
 p = x  
 x = x.left.node  
 else:  
 if p != x:  
 p = x  
 x = x.right.node  
  
 def search(self, search\_key):  
 x = self.node  
 while x is not None:  
 if x.key == search\_key:  
 return x.key  
 if x.key > search\_key:  
 x = x.left.node  
 else:  
 x = x.right.node  
 return -1  
  
 def insert(self, v):  
 x = self.node  
 if x is None:  
 self.node = node(v)  
 self.node.left = Dict()  
 self.node.right = Dict()  
 elif x.key > v:  
 self.node.left.insert(v)  
 else:  
 self.node.right.insert(v)  
 self.check\_balance()  
  
 def check\_balance(self):  
 self.update\_heights(False)  
 self.update\_balances(False)  
  
 while self.balance < -1 or self.balance > 1:  
 if self.balance > 1:  
 if self.node.left.balance < 0:  
 self.node.left.rotate\_left()  
 self.rotate\_right()  
 else:  
 if self.node.right.balance > 0:  
 self.node.right.rotate\_right()  
 self.rotate\_left()  
 self.update\_heights()  
 self.update\_balances()  
  
 def rotate\_right(self):  
 g = self.node  
 p = g.left.node  
 x = p.right.node  
  
 self.node = p  
 p.right.node = g  
 g.left.node = x  
  
 def rotate\_left(self):  
 g = self.node  
 p = g.right.node  
 x = p.left.node  
  
 self.node = p  
 p.left.node = g  
 g.right.node = x  
  
 def update\_heights(self, recurse=True):  
 if self.node is not None:  
 if recurse:  
 if self.node.left is not None:  
 self.node.left.update\_heights()  
 if self.node.right is not None:  
 self.node.right.update\_heights()  
 self.height = max(self.node.left.height, self.node.right.height) + 1  
 else:  
 self.height = 0  
  
 def update\_balances(self, recurse=True):  
 if self.node is not None:  
 if recurse:  
 if self.node.left is not None:  
 self.node.left.update\_balances()  
 if self.node.right is not None:  
 self.node.right.update\_balances()  
 self.balance = self.node.left.height - self.node.right.height  
 else:  
 self.balance = 0  
  
import random, time  
  
N = 9  
key = [2, 1, 8, 9, 7, 3, 6, 4, 5]  
s\_key = list(range(1, 10))  
  
# N = 10000  
# key = list(range(1, N+1))  
# s\_key = list(range(1, N+1))  
# random.shuffle(key)  
  
d = Dict()  
for i in range(0, N):  
 d.insert(key[i])  
  
start\_time = time.time()  
for i in range(N) :  
 result = d.search(s\_key[i])  
 if result == -1 or result != s\_key[i]:  
 print("탐색 오류")  
end\_time = time.time() - start\_time  
  
for i in range(N):  
 d.check(s\_key[i])  
 i += 1  
  
print('AVL 트리 탐색의 실행 시간 (N = %d) : %0.3f'%(N, end\_time))  
print('탐색 완료')

**2) 성능 비교**

**- 난수 발생 시**

N = 100000 : 1.451, 1.272, 1.063 / **평균 : 1.262sec**

N = 200000 : 3.139, 2.721, 2.669 / **평균 : 2.843sec**

N = 500000 : 8.080, 7.680, 7.551 / **평균 : 7.770sec**

**- 정렬 시** – random.shuffle(key)를 사용하지 않음.

N = 100000 : 1.221, 1.403, 1.117 / **평균 : 1.247sec**

N = 200000 : 2.971, 2.181, 2.534 / **평균 : 2.562sec**

N = 500000 : 3.101, 3.178, 3.140 / **평균 : 3.140sec**

**- 역순 정렬 시** – random.shuffle(key)를 사용하지 않고, key.reverse() 함수 사용

N = 100000 : 0.520, 0.581, 0.574 / **평균 : 0.558sec**

N = 200000 : 1.289, 1.089, 1.038 / **평균 : 1.138sec**

N = 500000 : 3.971, 3.397, 2.827 / **평균 : 3.398sec**

