(1) 레드 블랙 트리 탐색 알고리즘

--------------------------------------------------------------------

black ← 0;

red ← 1;

insert(T, v)

x ← T; p ← T; g ← T;

while (x ≠ null) do {

gg ← g; g ← p; p ← x;

if (x.key = v) then return;

if (x.key > v) then x ← x.left;

else x ← x.right;

if ((x.left.color = red) and (x.right.color = red)) then

split(x, p, g, gg, v);

x ← color가 red이고 key 값이 v인 새로운 노드;

if (p.key > v) then p.left ← x;

else p.right ← x;

split(x, p, g, gg, v);

end insert()

split(T, x, p, g, gg, v)

x.color ← red;

x.left.color ← black;

x.right.color ← black;

if (p.color = red) then {

g.color ← red;

if ((g.key > v) ≠ (p.key > v)) then

p ← rotate(v, g);

x ← rotate(v, gg);

x.color ← black;

end split()

rotate(v, y)

// c : y의 자녀 노드, gc : y의 손자녀 노드

if (y.key > v) then c ← y.left;

else c ← y.right;

if (c.key > v) then {

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) then y.left ← gc;

else y.right ← gc;

return gc

end rotate()

--------------------------------------------------------------------

□ 파이썬 소스 코드

--------------------------------------------------------------------

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:

z = node(color=black, key=0, left=0, right=0)

z.left = z

z.right = z

head = node(color=black, 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 = 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):

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 = 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

print('레드 블랙 트리 탐색의 실행 시간 (N = %d) : %0.3f'%(N, end\_time))

print('탐색 완료')

--------------------------------------------------------------------

(2) 레드 블랙 트리에 대한 정확성 검사

□ 파이썬 소스 코드

--------------------------------------------------------------------

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)

z.left = z

z.right = z

head = node(color=black, 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 = 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 check(self, search\_key):

x = p = self.head.right

while (x != self.z):

if x.color == 0:

str\_color = 'black'

else:

str\_color = 'red'

print('key : ', x.key, ', parents: ', p.key, ', color : ', str\_color)

p = x

if x.key > search\_key:

x = x.left

else:

x = x.right

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

d = Dict()

key = int(input('키 : '))

while key != 999:

d.insert(key)

d.check(key)

key = int(input('키 : '))

--------------------------------------------------------------------

(3) AVL 트리에 대한 정확성 검사

□ 파이썬 소스 코드

--------------------------------------------------------------------

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 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

def check(self, search\_key):

x = p = self.node

while x is not None :

print('key : ', x.key, ', parents: ', p.key,)

p = x

if x.key > search\_key:

x = x.left.node

else:

x = x.right.node

import random, time

d = Dict()

key = int(input('키 : '))

while key != 999:

d.insert(key)

d.check(key)

key = int(input('키 : '))

--------------------------------------------------------------------

(5) 선형 탐사법

--------------------------------------------------------------------

insert(a[], v)

x ← hash(v);

while (a[x] ≠ -1) do x ← (x + 1) mod M;

a[x] ← v;

end insert()

search(a[], v)

x ← hash(v);

while (a[x] ≠ -1) do {

if (a[x] = v) then return a[x];

else x ← (x + 1) mod M;

}

return -1;

end search()

hash(v)

return v mod M;

end hash()

--------------------------------------------------------------------

□ 파이썬 소스 코드

--------------------------------------------------------------------

class Dict:

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

Dict.a = [-1] \* M

def search(self, v):

x = self.hash(v)

while Dict.a[x] != -1:

if Dict.a[x] == v:

return Dict.a[x]

else:

x = (x + 1) % M

return -1

def insert(self, v):

x = self.hash(v)

while Dict.a[x] != -1:

x = (x + 1) % M

Dict.a[x] = v

def hash(self, v):

return v % M

import random, time

N = 10000

M = 10391

key = []

s\_key = []

for i in range(N):

r = random.randint(1, 3 \* N)

key.append(r)

s\_key.append(r)

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('선형 탐사법의 실행 시간 (N = %d) : %0.3f'%(N, end\_time))

print('탐색 완료')

--------------------------------------------------------------------

(6) 이중 해싱

--------------------------------------------------------------------

insert(a[], v)

x ← hash(v);

u ← hash2(v);

while (a[x] ≠ -1) do x ← (x + u) mod M;

a[x] ← v;

end insert()

search(a[], v)

x ← hash(v);

u ← hash2(v);

while (a[x] ≠ -1) do {

if (a[x] = v) then return a[x];

else x ← (x + u) mod M;

}

return -1;

end search()

hash(v)

return v mod M;

end hash()

hash2(v)

return 64 - (v mod 64);

end hash()

--------------------------------------------------------------------

□ 파이썬 소스 코드

--------------------------------------------------------------------

class Dict:

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

Dict.a = [-1] \* M

def search(self, v):

x = self.hash(v)

u = self.hash2(v)

while Dict.a[x] != -1:

if v == Dict.a[x]:

return Dict.a[x]

else:

x = (x + u) % M

return -1

def insert(self, v):

x = self.hash(v)

u = self.hash2(v)

while Dict.a[x] != -1:

x = (x + u) % M

Dict.a[x] = v

def hash(self, v):

return v % M

def hash2(self, v):

return 64 - (v % 64)

import random, time

N = 10000

M = 10391

key = []

s\_key = []

for i in range(N):

r = random.randint(1, 3 \* N)

key.append(r)

s\_key.append(r)

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('이중 해싱의 실행 시간 (N = %d) : %0.3f'%(N, end\_time))

print('탐색 완료')

--------------------------------------------------------------------