**프로그래밍 실습 #5**

2023년 10월 2주차

□ 다음 탐색 알고리즘을 ADL로 작성하고 파이썬으로 구현해 보라.

(1) 순차 탐색 알고리즘

N 값의 변화에 따라 시간복잡도가 O(N)이 되는지 확인해 보라.

= Data 개수에 비례하여 탐색 시간이 증가

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

sequentialSearch(a[], search\_key, n)

i ← 0;

|  |
| --- |
| While (i < n and a[i].key != search\_key) do  i <- i + 1;  if (i = n) then return -1;  Else retrun i; |

end sequentialSearch()

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

순차 탐색 알고리즘을 위한 파이썬 코드는 다음과 같다. 탐색 키와 비교할 때는 Dict.a[i].key를 사용하라.

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

class node:

def \_\_init\_\_(self, key=None):

self.key = key

class Dict:

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

Dict.a = []

def search(self, search\_key, n):

# 탐색 알고리즘

def insert(self, v):

Dict.a.append(node(v))

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], N)

if result == -1 or key[result] != s\_key[i]:

print('탐색 오류')

end\_time = time.time() - start\_time

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

print('탐색 완료')

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

(2) 이진 탐색 알고리즘

N 값의 변화에 따라 시간복잡도가 O(log N)이 되는지 확인해 보라.

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

binarySearch(a[], search\_key, n)

left ← 0; right ← n - 1;

|  |
| --- |
|  |

return -1;

end binarySearch()

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

이진 탐색 알고리즘을 위한 파이썬 코드는 다음과 같다.

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

class node:

def \_\_init\_\_(self, key=None):

self.key = key

class Dict:

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

Dict.a = []

def search(self, search\_key, n):

# 탐색 알고리즘

def insert(self, v):

Dict.a.append(node(v))

import random, time

N = 10000

key = list(range(1, N + 1))

s\_key = list(range(1, N + 1))

random.shuffle(s\_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], N)

if result == -1 or key[result] != s\_key[i]:

print('탐색 오류')

end\_time = time.time() - start\_time

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

print('탐색 완료')

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

(3) 이진 트리 탐색 알고리즘

N 값의 변화에 따라 난수 배열의 경우 시간복잡도가 O(log N)이 되는지, 또한 최악의 경우 O(N)이 되는지 확인해 보라.

**기존 Data 순서의 최악, 최선 실험하기**

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

binaryTreeSearch(T, search\_key)

x ← T;

while (x ≠ null) do {

|  |
| --- |
|  |

}

return -1;

end binaryTreeSearch()

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

이진 트리 탐색 알고리즘을 위한 파이썬 코드는 다음과 같다.

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

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:

# 탐색 알고리즘

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')

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

(4) 이진 탐색 트리에 대한 정확성 검사

Dict 클래스에 check(self, search\_key) 함수를 추가하여 이진 탐색 트리가 제대로 생성되었는지 검사한다.

[실행 예]

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

키 리스트: [2, 1, 7, 8, 6, 3, 5, 4]

key : 1 , parents : 2

key : 2 , parents : 2

key : 3 , parents : 6

key : 4 , parents : 5

key : 5 , parents : 3

key : 6 , parents : 7

key : 7 , parents : 2

key : 8 , parents : 7

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

**<코딩 테스트 연습 #3>**

**□ 최소값 찾기**

[문제]

N개의 수 A1, A2, ..., AN과 L이 주어진다.

Di = Ai-L+1 ~ Ai 중의 최소값이라고 할 때, D에 저장된 수를 출력하는 프로그램을 작성하시오. 이때, i ≤ 0 인 Ai는 무시하고 D를 구해야 한다.

[입력]

- 첫째 줄에 N과 L이 주어진다. (1 ≤ L ≤ N ≤ 5,000,000)

- 둘째 줄에는 N개의 수 Ai가 주어진다. (-109 ≤ Ai ≤ 109)

[출력]

첫째 줄에 Di를 공백으로 구분하여 순서대로 출력한다.

[참고]

파이썬 모듈에 있는 히프(heap)나 데크(deque)를 사용해 보라.

- 히프를 사용할 경우

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

import heapq

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

- 데크를 사용할 경우

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

from collections import deque

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

[실행 예]

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

12 3

1 5 2 3 6 4 5 7 3 5 2 6

1 1 1 2 2 3 4 4 3 3 2 2

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

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

10 4

3 5 1 3 2 4 7 5 3 2

3 3 1 1 1 1 2 2 3 2

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