**Data Structure(SE274): Assignment 1**

201911189 한현영

**A1-1.**

def example1(S):

"""Return the sum of the elements in sequence S."""

n = len(S)

total = 0

for j in range(n): # loop from 0 to n-1

total += S[j]

return total

elements n개를 갖는 for loop를 도는 경우의 시간 복잡도는 n이다. 이를 Big-Oh notation으로 표현할 경우 에 해당한다.

**A1-2.**

def example2(S):

"""Return the sum of the elements with even index in sequence S."""

n = len(S)

total = 0

for j in range(0, n, 2): # note the increment of 2

total += S[j]

return total

range에 따라 for loop 안에서의 시간 복잡도는 이다. 이를 Big-Oh notation으로 나타낼 경우 이다.

**A1-3.**

def example3(S):

"""Return the sum of the prefix sums of sequence S."""

n = len(S)

total = 0

for j in range(n): # loop from 0 to n-1

for k in range(1+j): # loop from 0 to j

total += S[k]

return total

이라 하면, for문 안의 시간 복잡도는 이다. 이를 Big-Oh notation으로 나타내면, 이다.

**A1-4.**

def example4(S):

"""Return the sum of the prefix sums of sequence S."""

n = len(S)

prefix = 0

total = 0

for j in range(n):

prefix += S[j]

total += prefix

return total

for loop 안의 시간 복잡도는 이다. 이를 Big-Oh notation으로 나타내면, 이다.

**A1-5.**

def example5(A, B): # assume that A and B have equal length

"""Return the number of elements in B equal to the sum of prefix sums in A."""

n = len(A)

count = 0

for i in range(n): # loop from 0 to n-1

total = 0

for j in range(n): # loop from 0 to n-1

for k in range(1+j): # loop from 0 to j

total += A[k]

if B[i] == total:

count += 1

return count

for loop 안의 시간 복잡도는 이다. 이를 Big-Oh notation으로 나타내면 이다.

**A2-1. Order the following functions by asymptotic growth rate**

우선 각각의 함수들을 Big-Oh notation 으로 표기해보자.

이를 통해 1차적인 asymptotic growth rate를 판단해보자.

따라서 각 함수들의 asymptotic growth rate는,

이다.

**A2-2. Show that 𝑛2 is Ω(𝑛log𝑛).**

에 대해 을 만족하는 상수가 존재하면 이다. 이라 하고, 양변을 으로 나누어 주면, 이다.

라고 할 때, (l'Hôpital's rule)이므로, 이고, 임을 알 수 있다.

**A2-3. Show that if 𝑝(𝑛) is a polynomial in 𝑛, then log𝑝(𝑛) is 𝑂(log𝑛).**

라 하자. 이고, 라고 할 때, 함수 진행의 dominant factor는 라는 것을 알 수 있다. 따라서 이를 Big-Oh notation으로 나타내면, 이다.

**A2-4. Show that if 𝑑(𝑛) is 𝑂(𝑓(𝑛)), then 𝑎𝑑(𝑛) is 𝑂(𝑓(𝑛)), for any constant 𝑎>0.**

Big-Oh notation에 의해 에 대해 일 것이고,일 때, 양변을 으로 나누면 이다. 이때, 인 상태에서 은 임의의 값에 수렴하고, 에 어떠한 상수가 곱해지더라도 Big-Oh notation을 위한 과정 자체의 c가 조절되기 때문에, 수렴성은 잃지 않는다. 따라서 위는 성립한다.

**A2-5. Show that if 𝑑(𝑛) is 𝑂(𝑓(𝑛)) and 𝑒(𝑛) is 𝑂(𝑔(𝑛)), then 𝑑(𝑛)−𝑒(𝑛) is not necessarily 𝑂(𝑓(𝑛)−𝑔(𝑛)).**

이라 하자. 그러면 이다. 이때 이고, 이고, 이므로, 반례가 성립한다.

**A3-1. Implement a function that reverses a list of elements by pushing them onto a stack in one order, and writing them back to the list in reversed order.**

class Stack: # Stack 클래스 생성

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

self.arr = []

def isEmpty(self):

return len(self.arr) == 0

def push(self, elem):

self.arr.append(elem)

def pop(self):

if not self.isEmpty():

return self.arr.pop(-1)

def reverse(list):

arr = Stack()

reversed\_arr = [] # input stack의 reversed element가 들어갈 array 생성

for push\_elem in range(len(list)):

arr.push(list[push\_elem])

for pop\_elem in range(len(list)):

reversed\_arr.append(arr.pop())

return reversed\_arr

**A3-2. Suppose an initially empty stack \*\*S\*\* has executed a total of 25 push operations, 12 top operations, and 10 pop operations, 3 of which raised Empty errors that were caught and ignored. What is the current size of \*\*S\*\*?**

Empty error는 빈 리스트에서의 pop에서만 발생하므로, top operations의 경우 size에 관여하지 않는다. 따라서 25 – (10 – 3) = 18 이다. #3번의 error 횟수 제외

**A3-3. Suppose an initially empty queue \*\*Q\*\* has executed a total of 32 enqueue operations, 10 first operations, and 15 dequeue operations, 5 of which raised Empty errors that were caught and ignored. What is the current size of \*\*Q\*\*?**

A3-2와 같은 원리로, first operations는 생각하지 않는다. 따라서 32 – (15 – 5) = 22 이다. #5번의 error 횟수 제외

**A3-4. Had the queue of the previous problem A3-3 been an instance of ArrayQueue that used an initial array of capacity 30, and had its size never been greater than 30, what would be the final value of the `\_front` instance variable?**

이 문제의 ‘\_front’ index를 0으로 설정하자. Queue에서 ‘\_front’의 index는 dequeue의 횟수, resize에 따라 변화한다. 현재 고정된 capacity를 가짐에 따라 resize operation을 행하지 않으므로, dequeue의 횟수만 고려한다. enqueue하는 element = 라고 가정한다면, A3-3에서 dequeue의 횟수는 10회이므로, ‘\_front’의 index는 10, 그에 따른 variable은 이다.

**A3-5. Give a complete ArrayDeque implementation of the double-ended queue, implemething the following ADT:**

class ArrayDeque: #ArrayDeque 클래스 생성

def \_\_init\_\_(self, size = 10): #Default size는 10

self.size = size

self.front = 0

self.rear = 0

self.arr = [None] \* size

self.num\_elem = 0 # \_\_len\_\_ func을 위함

def \_\_len\_\_(self):

return self.num\_elem

def resize(self): #Doubling process

self.old\_arr = self.arr

self.new\_arr = [None]\*self.size\*2

for i in range(self.size):

k = self.front

self.new\_arr[i] = self.old\_arr[(k+i) % self.size]

self.arr = self.new\_arr

self.front = 0

self.rear = self.size - 1

self.size \*= 2

def is\_empty(self):

return self.front == self.rear

def is\_full(self): #Arr 안

return self.front == (self.rear + 1) % self.size

def add\_first(self, elem):

if self.is\_full():

self.resize()

self.front = (self.front - 1) % self.size

self.arr[self.front] = elem

self.num\_elem += 1

def add\_last(self, elem):

if self.is\_full():

self.resize()

self.rear = (self.rear + 1) % self.size

self.arr[self.rear] = elem

self.num\_elem += 1

def delete\_first(self):

if self.is\_empty():

raise EmptyError

else:

self.first\_elem = self.arr[self.front]

self.arr[self.front] = None

self.front = (self.front + 1) % self.size

self.num\_elem -= 1

return self.first\_elem

def delete\_last(self):

if self.is\_empty():

raise EmptyError

else:

self.last\_elem = self.arr[self.rear]

self.arr[self.rear] = None

self.rear = (self.rear - 1) % self.size

self.num\_elem -= 1

return self.last\_elem

def first(self):

if self.is\_empty():

raise EmptyError

else:

return self.arr[self.front]

def last(self):

if self.is\_empty():

raise EmptyError

else:

return self.arr[self.rear]

**A3-6. Suppose you have a deque \*\*D\*\* containing the numbers `(1,2,3,4,5,6,7,8)`, in this order. Suppose further that you have an initially empty queue \*\*Q\*\*. Give a code fragment that uses only \*\*D\*\* and \*\*Q\*\* (and no other variables) and results in \*\*D\*\* storing the elements in the order `(1,2,3,5,4,6,7,8)`.**

#initial statement: D=(1,2,3,4,5,6,7,8) Q=(\*,\*,\*,\*,\*,\*,\*,\*), Deque의 first는 array의 0번째, last는 array의 -1번째로 가정

Q.enqueue(D.delete\_first) #D=(\*,2,3,4,5,6,7,8) Q=(1,\*,\*,\*,\*,\*,\*,\*)

Q.enqueue(D.delete\_last) #D=(\*,2,3,4,5,6,7,\*) Q=(1,8,\*,\*,\*,\*,\*,\*)

Q.enqueue(D.delete\_first) #D=(\*,\*,3,4,5,6,7,\*) Q=(1,8,2,\*,\*,\*,\*,\*)

Q.enqueue(D.delete\_last) #D=(\*,\*,3,4,5,6,\*,\*) Q=(1,8,2,7,\*,\*,\*,\*)

Q.enqueue(D.delete\_first) #D=(\*,\*,\*,4,5,6,\*,\*) Q=(1,8,2,7,3,\*,\*,\*)

Q.enqueue(D.delete\_last) #D=(\*,\*,\*,4,5,\*,\*,\*) Q=(1,8,2,7,3,6,\*,\*)

Q.enqueue(D.delete\_first) #D=(\*,\*,\*,\*,5,\*,\*,\*) Q=(1,8,2,7,3,6,4,\*)

Q.enqueue(D.delete\_last) #D=(\*,\*,\*,\*,\*,\*,\*,\*) Q=(1,8,2,7,3,6,4,5)

D.add\_first(Q.dequeue) #D=(1,\*,\*,\*,\*,\*,\*,\*) Q=(\*,8,2,7,3,6,4,5)

D.add\_last(Q.dequeue) #D=(1,8,\*,\*,\*,\*,\*,\*) Q=(\*,\*,2,7,3,6,4,5)

D.add\_first(Q.dequeue) #D=(2,1,8,\*,\*,\*,\*,\*) Q=(\*,\*,\*,7,3,6,4,5)

D.add\_last(Q.dequeue) #D=(2,1,8,7,\*,\*,\*,\*) Q=(\*,\*,\*,\*,3,6,4,5)

D.add\_first(Q.dequeue) #D=(3,2,1,2,7,\*,\*,\*) Q=(\*,\*,\*,\*,\*,6,4,5)

D.add\_last(Q.dequeue) #D=(3,2,1,8,7,6,\*,\*) Q=(\*,\*,\*,\*,\*,\*,4,5)

D.add\_first(Q.dequeue) #D=(4,3,2,1,8,7,6,\*) Q=(\*,\*,\*,\*,\*,\*,\*,5)

D.add\_last(Q.dequeue) #D=(4,3,2,1,8,7,6,5) Q=(\*,\*,\*,\*,\*,\*,\*,\*)

Q.enqueue(D.delete\_first) #D=(\*,3,2,1,8,7,6,5) Q=(4,\*,\*,\*,\*,\*,\*,\*)

Q.enqueue(D.delete\_last) #D=(\*,3,2,1,8,7,6,\*) Q=(4,5,\*,\*,\*,\*,\*,\*)

Q.enqueue(D.delete\_first) #D=(\*,\*,2,1,8,7,6,\*) Q=(4,5,3,\*,\*,\*,\*,\*)

Q.enqueue(D.delete\_last) #D=(\*,\*,2,1,8,7,\*,\*) Q=(4,5,3,6,\*,\*,\*,\*)

Q.enqueue(D.delete\_first) #D=(\*,\*,\*,1,8,7,\*,\*) Q=(4,5,3,6,2,\*,\*,\*)

Q.enqueue(D.delete\_last) #D=(\*,\*,\*,1,8,\*,\*,\*) Q=(4,5,3,6,2,7,\*,\*)

Q.enqueue(D.delete\_first) #D=(\*,\*,\*,\*,8,\*,\*,\*) Q=(4,5,3,6,2,7,1,\*)

Q.enqueue(D.delete\_last) #D=(\*,\*,\*,\*,\*,\*,\*,\*) Q=(4,5,3,6,2,7,1,8)

D.add\_first(Q.dequeue) #D=(4,\*,\*,\*,\*,\*,\*,\*) Q=(\*,5,3,6,2,7,1,8)

D.add\_first(Q.dequeue) #D=(5,4,\*,\*,\*,\*,\*,\*) Q=(\*,\*,3,6,2,7,1,8)

D.add\_first(Q.dequeue) #D=(3,5,4,\*,\*,\*,\*,\*) Q=(\*,\*,\*,6,2,7,1,8)

D.add\_last(Q.dequeue) #D=(3,5,4,6,\*,\*,\*,\*) Q=(\*,\*,\*,\*,2,7,1,8)

D.add\_first(Q.dequeue) #D=(2,3,5,4,6,\*,\*,\*) Q=(\*,\*,\*,\*,\*,7,1,8)

D.add\_last(Q.dequeue) #D=(2,3,5,4,6,7,\*,\*) Q=(\*,\*,\*,\*,\*,\*,1,8)

D.add\_first(Q.dequeue) #D=(1,2,3,5,4,6,7,\*) Q=(\*,\*,\*,\*,\*,\*,\*,8)

D.add\_last(Q.dequeue) #D=(1,2,3,5,4,6,7,8) Q=(\*,\*,\*,\*,\*,\*,\*,\*), **FINISH!**

**A3-Bonus. Show how to use a stack S and a queue Q to generate all possible subsets of an n-element set T nonrecursively (Don't use a recursion algorithm).**

class Stack: # 간단한 Stack 클래스 생성

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

self.stack = []

self.pointer = 0

def \_\_len\_\_(self):

return self.pointer

def is\_empty(self):

return len(self.stack) == 0

def push(self, elem):

self.pointer += 1

return self.stack.append(elem)

def pop(self):

self.pointer -= 1

return self.stack.pop(-1)

class Queue: # 간단한 Queue 클래스 생성

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

self.queue = []

self.pointer = 0

def \_\_len\_\_(self):

return self.pointer

def is\_empty(self):

return len(self.queue) == 0

def enqueue(self, elem):

self.pointer += 1

return self.queue.append(elem)

def dequeue(self):

self.pointer -= 1

return self.queue.pop(0)

stack = Stack()

queue = Queue()

queue.enqueue(set())

for i in range(len(n)):

stack.push(n[i])

while stack.is\_empty() == False: #스택의 element가 없어질 때까지 생성한 집합과 뽑힌 element가 합집합을 생성함, 이를 queue에 저장

var = stack.pop()

for i in range(len(q)):

a = queue.dequeue()

queue.enqueue(a)

b = a|{var}

queue.enqueue(b)

while queue.is\_empty() == False: #큐의 elenent가 없어질 떄까지 생성한 부분집합을 print()함

new = queue.dequeue()

print(new)