한국외국어대학교
HANKUK UNIVERSITY OF FOREIGN STUDIES

프로그래밍 과제 4

HUFS

■ 담 당 교 수 : 김희철 교수님
■ 제 출 일 : 4/13
■ 학 과 : 컴퓨터전자시스템공학부
■ 학 번 : 201902520
■ 성 명 : 이상윤

contents

문제 기술

1) 후위 (postfix) 표기 수식을 읽어 그 결과값을 출력하는 프로그램을 작성하시오. **모든 피연산자는 정수 상수이고, 연산자는 이항연산자인 '+', '-', '*', '//', '%'이다.** **여기서,** //는 몫 연산자이다. 식의 마지막은 ;으로 끝난다. **피연산자와 연산자 사이에는 공백이 있다.**

제약조건: Linked 스택으로 구현한 스택을 사용한다.

입력: 후위 표기 수식이 한 줄에 주어진다.

출력: 후위 표기 수식의 결과값을 출력한다. **단, 후위 표기 수식에 오류가 있을 경우 error를 출력한다.**

2) 공항에서 사람들의 입국을 심사하는 심사대가 k개있다. 사람들이 입국심사를 받기 위하여, 심사대 바로 앞에서 한 줄로 서서 기다린다. k개의 심사대 중 하나가 비어있으면(심사받는 사람이 없으면), 먼저 도착한 사람이 비어 있는 이 심사대에서 심사를 받는다. 입국심사를 받기 위한 각 사람의 심사대 앞에 도착하는 시간과 심사받는데 걸리는 시간이 주어진다. 각 사람이 심사대 앞에 도착한 시간에 대한 정보는, 바로 이전 사람이 심사대 앞에 도착한 시간과의 차이(분)로 주어진다. 각 사람이 심사를 받기 위하여 기다리는 시간 (심사시간은 제외)의 평균을 **소수점 이하 2자리까지 (소수점 이하 3째 자리에서 반올림)** 출력하는 프로그램을 작성하시오.

예를 들어, k = 1이라 하자. 그리고 5명의 사람들의 도착시간이 0, 2, 1, 2, 1이고, 심사받는데 걸리는 시간이 각각 4분, 7분, 6분, 5분, 3분이라 하자. 5명의 도착시간이 0, 2, 1, 2, 1라는 것은, 2번째 사람은 첫 번째 사람이 심사대 앞에 도착한 시간 2분 후에 심사대 앞에 도착하고, 3번째 사람은 2번째 사람이 도착한 시간 1분 후에 심사대 앞에 도착하고, 4번째 사람은 3번째 사람이 도착한 시간 2분 후에 심사대 앞에 도착하고, 5번째 사람은 4번째 사람이 도착한 시간 1분 후에 심사대 앞에 도착한다.

첫 번째 사람은 기다리지 않고 심사를 받는다. 두 번째 사람은 첫 번째 사람이 심사를 받은 직후, 심사를 받으므로 2분을 기다린다. 세 번째, 4번째, 5번째 사람은 각각 8분, 12분, 16분 기다린다. 따라서 5명이 심사대 앞에서 기다리는 평균시간은 (2+8+12+16)/5 =38/5 = 7.6 분이다.

위의 예에서 k = 2라 하자. 심사대가 2개이므로 첫 번째 사람과 두 번째 사람은 각각 심사대 1과 심사대 2에서 기다리지 않고 심사를 받는다. 3번째 사람은 1분을 기다리고 심사대 1에서 심사를 받는다. 4번째 사람은 4분을 기다리고 심사대 2에서 심사를 받는다. 5번째 사람은 4분을 기다리고 심사대 1에서 심사를 받는다. 따라서 5명이 심사대 앞에서 기다리는 평균시간은 (0 + 0 + 1 + 4 + 4) / 5 = 1.8분이다.

제약조건: 심사대 앞에 도착하는 사람들을 관리하는 큐를 이용하여야 함 (큐는 파이썬 모듈을 사용하지 말고, 원형큐(파이썬 리스트)로 구현하고 큐가 full일 경우 리스트 길이를 2배로 확장하도록 한다.)

2-1) k = 1일 때, 해를 구하는 프로그램을 작성하시오.

2-2) k = 2일 때, 해를 구하는 프로그램을 작성하시오.

입력
첫 번째 줄에 심사대 앞에 도착하는 사람들의 수 n이 주어진다. 다음의 각 줄에 먼저 온 사람부터 각 사람의 심사대 앞 도착시간(분 단위 양의 정수)과 심사받는데 걸리는 시간(분 단위의 양의 정수)이 주어진다.

자료구조 및 알고리즘 설명

1. Linkedstack를 이용합니다. 연결된 구조의 특징은 용량이 고정되지 않고 필요한 만큼만 할당해 사용하고 크기 제한도 없어서 효율적인 장점이 있습니다. 또 중간에 자료를 삽입하거나 식제하는 것이 용이합니다. 연결리스트는 노드, 헤드 포인터로 구성되어 있습니다. 노드는 데이터 필드와 하나 이상의 링크 필드로 구성되어 있고 헤드 포인터는 연결리스트의 첫 번째 노드를 가리킵니다. 알고리즘은 먼저 노드와 Linkedstack을 구현합니다. 값을 push할때는 값을 넣고 top값을 바꿔주고 pop할때는 top값을 반환합니다. 그리고 top을 top이 링크하고 있던 노드로 다시 바꿔줍니다 (빼내고 난 후 stack에 가장 최근에 들어 온 값). 문제 풀이 알고리즘으로 후위표기 계산법은 입력받은 수식에서 차례로 검사합니다. 값이 피연산자(숫자)면 스택에 push합니다. 그러다가 연산자가 나오면 stack에 있던 피연산자를 pop 2번 합니다. 먼저 나온 값이 2번째 피연산자가 됩니다(제일 나중에 들어온 값이 먼저 반환되기 때문에 순서 고려해야 합니다. a(?)b순으로 계산하려고 하기 때문). 계산을 마치고 ';'가 나올 경우에 상황에 따른 결과를 출력합니다.

2. 큐를 이용합니다. 큐는 선입선출(FIFO)의 자료구조입니다. 삽입은 후단에서 삭제는 전단에서 이루어집니다. 선형큐와 원형큐 2가지로 이루어져 있고 선형큐는 리스트의 맨 앞에서 항목을 삭제하면 이후의 모든 항목을 한 칸씩 앞으로 이동해야 하므로 비효율적입니다. 원형 큐는 이런 단점을 보완한 큐입니다. 원형 큐는 전단과 후단을 위한 'front 와 Rear' 2개의 변수를 사용합니다. 항목을 삽입할 때에는 rear값을 늘리고 삭제할 때는 front값을 늘리고 크기로 나눠 나머지 값으로 관리합니다. 저는 front를 첫 번째 요소 바로 이전의 인덱스, Rear을 큐의 마지막 요소의 인덱스로 사용하여 큐를 구현하였습니다. 문제플이 알고리즘으로 심사대의 사람을 관리하는 큐를 2개 생성합니다. 하나는 앞사람이 도착하고 얼마나 뒤에 다음사람이 오는지를 관리하는 큐, 다른 하나는 심사하는데 걸리는 시간을 관리하는 큐입니다. 저는 사람이 들어오는 경과된 시간(앞사람이 도착하고 자신이 도착하는 시간에 앞사람이 도착한 시간을 더함)을 구하기 위해 큐를 하나 더 만들었습니다. 대기시간은 앞사람이 끝나는 시간에서 내가 도착한 시간을 빼어서 구하였습니다. 여기서 시간은 첫 번째 사람이 도착하고 지난

시간입니다. 심사받을 때 심사대와 비어있는 경우와 비어있지 않는 경우를 다르게 적용하여 계산하였습니다. 비어있는 경우에는 대기시간이 음수값이 나오고 심사가 끝나는 시간이 제대로 나오지 않으므로 빈 시간만큼 심사시간에 빼주고 대기시간은 0으로 빠꿔주어 원하는 결과를 얻을 수 있었습니다. 마지막으로는 파이썬 반올림이 우리가 보통 알고있는 반올림과 다르다는 것을 알고 직접 구현하여 결과값을 구했습니다.

3. 도착시간과 심사시간을 각각 1번심사대와 2번심사대로 나누어 4개의 큐를 생성합니다. 2-1문제와 마찬가지로 입장시간의 합 큐를 각각 1개씩 더 만들었습니다. 그리고 다음사람이 1번과 2번 둘중 앞사람이 빨리 끝나는 심사대로 넣게 하는 알고리즘을 구현했습니다. 이것도 마찬가지로 들어갔을 때 심사대가 비어있는 경우와 그렇지 않은경우를 분리하여 계산해야 앞사람이 끝나는 시간을 정확하게 계산할 수 있었습니다. 2번문제와 대기시간을 구하는 방식은 같습니다. 1번심사대와 2번심사대를 따로 계산하여 각각 나온 대기시간을 합쳤습니다.

느낀 점

이번 과제는 linkedstack 사용하여 후위표기 수식을 계산하고 queue를 사용하여 입국심사대를 관리하는 프로그램을 구현하였습니다.

1. Linkedstack을 구현하는 방법과 특징을 알게 되었습니다. 1번문제는 지난번 과제와 유사한 부분이 많아 비교적 어렵지 않게 끝낼 수 있었습니다.

2-1. 대기 줄은 큐와 유사한 특성이 있어 큐를 사용하는 것이 가장 적절하다고 생각합니다. 2-1과 2-2를 풀면서 좀 많이 애를 먹었습니다. 해결할 방법을 생각해내도 생각보다 잘 안돼서 어떻게 구현해야 할지 많은 고민을 했었습니다. 정확성을 높이기 위해 많은 테스트케이스를 임의로 만들어서 연습장에 그려보고 비교해 보면서 프로그램을 완성시킬 수 있게 되었습니다. 예시들을 비교하다가 대기시간이 음수가 나오는 경우가 있었는데 뭐가 문제일까 고민하다가 심사대에 들어갈 때 심사대가 비어있는 경우에 그런 결과가 나온다는 것을 파악했고 그 값만큼 심사시간에 빼면 심사가 끝나는 시간과 대기시간을 계산할 때 원하는 값이 나온다는 것을 알게 되었습니다. 그런 다음 심사대가 비어있었기 때문에 당연히 대기시간이 없으므로 0으로 바꿔주었습니다. 답을 제출할때는 2-1과 2-2문제의 한 테스트에서 값이 틀린 것을 보고 질문한 결과 파이썬 반올림의 방식이 다르다는 것을 알게되었습니다. 그래서 반올림을 하는 방법을 알아보다가 구현을 해보는게 제일 낫겠다 생각하여 직접 구현하였습니다.

2-2. 이 문제는 2-1과 푸는 방식은 전체적으로 비슷했습니다. 차이점은 심사대가 2개입니다. 대기시간을 최소화하기 위하여 각 심사대의 이전 사람이 심사가 끝나는 시간을 관리하는 큐를 만들어서 빨리 끝나는 심사대로 다음 사람을 보내는 방법을 사용하는게 가장 빠르고 효율적이라고 생각했습니다.

프로그램 코드

1.

```
import sys
class Node :   #Node 구현
   def __init__(self ,element ):
     self .data =element
     self .link =None

class LinkedStack :  #Linkedstack 구현
   def __init__(self ):
     self .top =None

   def isEmpty (self ):
     return self .top ==None

   def push (self ,e ):
     newNode =Node (e )
     newNode .link =self .top
     self .top =newNode

   def pop (self ):
     if (self .isEmpty ()):
       print ("Stack is empty")
       return
     e =self .top .data
     self .top =self .top .link
     return e
def Postfix_cal (question ):  #후위표기 수식 계산 알고리즘
   stack =LinkedStack ()
   operator =('+','-','*','/','//','%') #연산자 구분
   for i in range (len (question )):
     str =question [i ]
     if str in operator : #값이 연산자일 경우
       if i ==0 :  #첫번째로 연산자가 나올경우 error출력
         print ('error')
         return
       if stack .isEmpty (): #연산자가 있지만 피연산자가 없는 경우 error출력
         print ("error")
         return
       else :  #그 외의 경우 val2 pop
         val2 =stack .pop ()

       if stack .isEmpty (): #또 다른(2번째) 피연산자가 없는 경우 error출력
         print ("error")
         return
       else :  #아니면 val1 pop
         val1 =stack .pop ()
       if str =='+': stack .push (val1 +val2 )    #연산자와 알맞는 연산
       elif str =='-': stack .push (val1 -val2 )
```

```
        elif str =='*': stack .push (val1 *val2 )
        elif str =='/': stack .push (val1 /val2 )
        elif str =='//': stack .push (val1 //val2 )
        elif str =='%': stack .push (val1 %val2 )
      elif str ==';':
        result =stack .pop ()  #';'을 만난경우 결과 pop
        if stack .isEmpty (): # 방금 pop한 경우가 stack의 마지막 값일 경우 진
짜 답
          print ("%.0f " %result )
          return
        else : #다른 값이 더 있을 경우 error 출력
          print ("error")
          return
      else :  #연산자가 아닌 피연산자는 stack에 push
        stack .push (float (str ))

solution =sys .stdin .readline ().split ()  #후위 표기 수식 입력받음
Postfix_cal (solution )
```

2-1.

```
import math #나중에 반올림 계산을 위함
class Queue :   #원형 큐 구현
   Max_Qsize =200
   def __init__(self ):
     self .items =[None ]*Queue .Max_Qsize
     self .front =-1
     self .rear =-1
     self .size =0

   def isEmpty (self ):
     return self .size ==0

   def enqueue (self ,e ):
     if self .size == len (self .items ):
       print ("Queue is full")
       self .resize (2 *len (self .items ))
     else :
       self .rear =(self .rear +1 )%(len (self .items ))
       self .items [self .rear ]=e
       self .size =self .size +1

   def dequeue (self ):
     if self .isEmpty ():
       print ("Queue is empty")
     else :
       self .front =(self .front +1 )%(len (self .items ))
       e =self .items [self .front ]
       self .size =self .size -1
       return e

   def resize (self , cap ):
     olditems = self .items
     self .items = [None ]*cap
     walk = self .front
     for k in range (self .size ):
       self .items [k ] = olditems [walk ]
       walk = (walk + 1 )% len (olditems )
     self .front = -1
     self .rear = self .size - 1


people =int (input ()) #사람의 수 입력받음 200이하
cometime_queue =Queue ()  #이전사람 도착후 다음사람 도착하는데 걸리는 시간 저장하는 큐
dotime_queue =Queue ()   #한사람마다 심사하는데 걸리는 시간을 저장하는 큐
for i in range (people ): #도착하는데 걸리는 시간과 심사하는데 걸리는 시간을 입력받고 정수로 변환
   cometime_1man , dotime_1man = input ().split ()
   cometime_1man =int (cometime_1man )
   dotime_1man =int (dotime_1man )
```

```
   #도착하는데 걸리는 시간과 심사하는데 걸리는 시간을 분리해서 큐에 enqueue
   cometime_queue .enqueue (cometime_1man )
   dotime_queue .enqueue (dotime_1man )

cometime =Queue ()   #도착하는데 걸리는 시간보다는 첫사람이 오고 난 후 흐르는 시간으로 계산을 편리하게 하려고 만든 큐
cal_cometime =0
cal_endtime =0
for i in range (cometime_queue .size ):
   cal_cometime += cometime_queue .dequeue ()

   cometime .enqueue (cal_cometime )

cometime .dequeue ()  #1번사람이 오는데 걸리는 시간은 대기시간에서 무의미하므로 미리 빼냄
wait_1man =0 #한사람당 대기시간
sum_waittime =0  #대기시간 합
endtime =0  #이전사람이 끝나는 시간
for i in range (cometime .size ):  #반복문으로 모든사람이 심사대로 가는 알고리즘
   endtime +=dotime_queue .dequeue ()
   wait_1man =endtime - cometime .dequeue ()
   if wait_1man <0 : #도착했을 때 심사대가 비어있는 경우의 계산
     endtime -=wait_1man
     wait_1man =0
   sum_waittime +=wait_1man
result =sum_waittime /people
def my_round (a ):   #파이썬의 반올림은 우리가 보통 알고있는 반올림과 다르기 때문에 직접 구현
   a =math .trunc (a *1000 )
   a =a /1000
   b =a
   b =math .trunc (b *100 )
   b =b /100

   if a -b <0.005 :
     return b
   else :
     return b +0.01
result =my_round (result )
print ("%.2f " %result )
```

2-2

```
import math   #나중에 반올림 계산을 위함
class Queue :   #원형 큐 구현
   Max_Qsize =200
   def __init__(self ):
     self .items =[None ]*Queue .Max_Qsize
     self .front =-1
     self .rear =-1
     self .size =0

   def isEmpty (self ):
     return self .size ==0

   def enqueue (self ,e ):
     if self .size == len (self .items ):
       self .resize (2 *len (self .items ))

     else :
       self .rear =(self .rear +1 )%(len (self .items ))
       self .items [self .rear ]=e
       self .size =self .size +1

   def dequeue (self ):
     if self .isEmpty ():
       print ("Queue is empty")
     else :
       self .front =(self .front +1 )%(len (self .items ))
       e =self .items [self .front ]
       self .size =self .size -1
       return e

   def resize (self , cap ):
     olditems =self .items
     self .items =[None ]*cap
     walk =self .front
     for i in range (self .size ):
       self .items [i ]=olditems [walk ]
       walk =(walk +1 )% len (olditems )
     self .front =-1
     self .rear =self .size -1

people =int (input ()) #사람의 수 입력받음 200이하
cometime_queue =Queue ()  #이전사람 도착후 다음사람 도착하는데 걸리는 시간 저
장하는 큐
dotime_queue =Queue ()   #한사람마다 심사하는데 걸리는 시간을 저장하는 큐
for i in range (people ): #도착하는데 걸리는 시간과 심사하는데 걸리는 시간을 입
력받고 정수로 변환
   cometime_1man , dotime_1man = input ().split ()
   cometime_1man =int (cometime_1man )
   dotime_1man =int (dotime_1man )
   #도착하는데 걸리는 시간과 심사하는데 걸리는 시간을 분리해서 큐에 enqueue
```

```
   cometime_queue .enqueue (cometime_1man )
   dotime_queue .enqueue (dotime_1man )
come_nowtime =Queue () #도착하는데 걸리는 시간보다는 첫사람이 오고 난 후 흐르는 시간으로 계산을 편리하게 하려고 만든 큐
cal_cometime =0
for i in range (cometime_queue .size ):
   cal_cometime += cometime_queue .dequeue ()
   come_nowtime .enqueue (cal_cometime )



d1_cometime_queue =Queue () #1번심사대 인원정보를 관리하는 큐 각각 온 시간과 심사시간
d1_dotime_queue =Queue ()
d2_cometime_queue =Queue () #2번심사대 인원정보를 관리하는 큐 각각 온 시간과 심사시간
d2_dotime_queue =Queue ()
end_d1 =0   #대기시간이 적은 심사대로 가기위해 앞사람이 심사를 끝내는 시간을 저장하는 변수
end_d2 =0
e1 =come_nowtime .dequeue ()  #첫사람은 1번심사대로 가는 알고리즘
d1_cometime_queue .enqueue (e1 )
f1 =dotime_queue .dequeue ()
d1_dotime_queue .enqueue (f1 )
f2 =0
end_d1 =f1
while (come_nowtime .isEmpty ()==False ):  #반복문으로 대기인원이 모두 대기시간이 적은 심사대로 가는 알고리즘
   if end_d1 <= end_d2 :   #1번심사대의 마지막 사람이 2번심사대 마지막 사람보다 빨리 끝나거나 같이 끝날경우 다음사람을 1번 심사대로 보냄
     e1 =come_nowtime .dequeue ()    #1번심사대 관련 큐에 정보 enqueue
     d1_cometime_queue .enqueue (e1 )
     if (end_d1 -e1 <0 ): #대기없이 입장할 경우
       f1 =dotime_queue .dequeue ()
       d1_dotime_queue .enqueue (f1 )
       end_d1 =e1 +f1
     else :  #대기하고 입장할 경우
       f1 =dotime_queue .dequeue ()
       d1_dotime_queue .enqueue (f1 )
       end_d1 += f1

   else :  #2번심사대가 먼저 빠지는 경우
     e2 =come_nowtime .dequeue ()
     d2_cometime_queue .enqueue (e2 )
     if (end_d2 -e2 <0 ):  #대기없이 입장할 경우
       f2 =dotime_queue .dequeue ()
       d2_dotime_queue .enqueue (f2 )
       end_d2 =e2 +f2
     else :  #대기하고 입장할 경우
       f2 =dotime_queue .dequeue ()
       d2_dotime_queue .enqueue (f2 )
       end_d2 += f2
```

```
sum_waittime =0
endtime1 =d1_cometime_queue .dequeue ()
endtime2 =d2_cometime_queue .dequeue ()
for i in range (d1_cometime_queue .size ):   #1번심사대에서의 개인마다 대기시
간 계산
   endtime1 +=d1_dotime_queue .dequeue ()
   d1_waittime =endtime1 - d1_cometime_queue .dequeue ()
   if d1_waittime <0 :  #도착했을 때 심사대가 비어있는 경우의 계산
     endtime1 -=d1_waittime
     d1_waittime =0
   #print(d1_waittime)
   sum_waittime += d1_waittime

for i in range (d2_cometime_queue .size ):   #2번심사대에서의 개인마다 대기시
간 계산
   endtime2 +=d2_dotime_queue .dequeue ()
   d2_waittime =endtime2 - d2_cometime_queue .dequeue ()
   if d2_waittime <0 :  #도착했을 때 심사대가 비어있는 경우의 계산
     endtime2 -=d2_waittime
     d2_waittime =0
   #print(d2_waittime)
   sum_waittime += d2_waittime

result =sum_waittime /people


def my_round (a ):     #파이썬의 반올림은 우리가 보통 알고있는 반올림과 다르기
때문에 직접 구현
   a =math .trunc (a *1000 )
   a =a /1000
   b =a
   b =math .trunc (b *100 )
   b =b /100

   if a -b <0.005 :
     return b
   else :
     return b +0.01
result =my_round (result )
print ("%.2f " %result )
```

자료구조 4

컴퓨터전자시스템공학부

201902520

이상윤