2023학년도 2학기 자료구조 중간고사

2022. 10. 23

점수	

학번: **성명:**

※ (1 - 10) 아래 문제를 읽고 물음에 답하시오. [50점 각 5점]

1. 다음은 입력 개수 n에 대한 알고리즘 A ~ D의 수행시간 복잡도를 나타낸 것이다. 알고리즘 A ~ D를 수행시간 효율이 좋은 것부터 순서대로 나열한 것은?

알고리즘	수행시간 복잡도
A	$O(n\log n)$
B	$O(2^n)$
C	$O(n^2)$
D	$O(n!)$

① A, C, B, D ② A, C, D, B
③ C, A, B, D ④ C, A, D, B

2. 다음 하노이 탑 문제를 해결하는 파이썬(Python) 코드의 시간 복잡도는? (단, n은 원판의 개수이고 first, second, third는 기둥 번호이다)

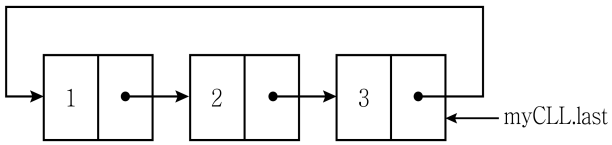
```
def hanoiT(n, first, second, third):
  if n == 1:
    print("원판1 : %s 에서 %s"%(first, third))
  else:
    hanoiT(n - 1, first, third, second)
    print("원판%d : %s 에서 %s"%(n, first, third))
    hanoiT(n - 1, second, first, third)
```

① $O(n)$
② $O(n^2)$
③ $O(2^n)$
④ $O(\log_2 n)$

3. 양방향 연결 리스트(doubly linked list)에 대한 설명으로 옳지 않은 것은?

① 이전 노드와 이후 노드에 대한 링크(link)를 가지고 있어 양방향 탐색이 가능하다.
② 한방향 연결 리스트(singly linked list)에 비해 기억 장소가 추가로 소요된다.
③ 데크(deque)를 구현하는데 활용이 가능하다.
④ 이항힙(binomial heap)이나 피보나치힙(Fibonacci heap)과 같은 우선순위큐를 구현하는데는 활용할 수 없다.

4. 그림과 같은 원형 연결 리스트 myCLL을 만들기 위해 다음 파이썬 코드를 작성하였다. (가) ~ (다)에 들어갈 코드를 A ~ D에서 바르게 연결한 것은?

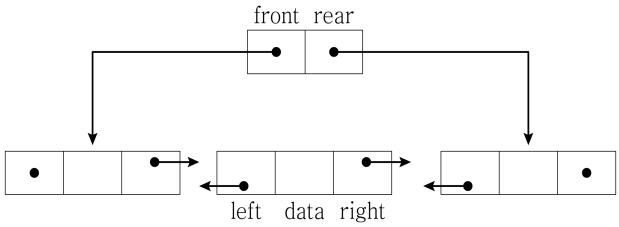



```
class Node:
    def __init__(self, data, next):
        self.data = data
        self.next = next

class CircularLinkedList:
    def __init__(self):
        self.last = None

    def insert(self, data):
        n = Node(data, None)
        if self.last is None:
            n.next = n
            self.last = n
        else:
            (가)
            (나)
            (다)

myCLL = CircularLinkedList()
myCLL.insert(1)
myCLL.insert(2)
myCLL.insert(3)
```

```
A. n.next = self.last
B. n.next = self.last.next
C. self.last = n
D. self.last.next = n
```

	(가)	(나)	(다)
①	A	C	D
②	A	D	C
③	B	C	D
④	B	D	C

5. 다음은 양방향 연결 리스트를 이용하여 덱(deque)을 구현한 파이썬 코드이다. (가) ~ (라)에 들어갈 내용을 A ~ F에서 바르게 연결한 것은?




```
class DQNode:
    def __init__(self, data, left, right):
        self.data = data
        self.left = left
        self.right = right

class DoublyLinkedDQ:
    def __init__(self):
        self.front = None
        self.rear = None

    def insertFront(self, data):
        node = DQNode(data, None, self.front)
        if self.front == None:
            self.front = node
            self.rear = self.front
        else:
            (가)
            (나)

    def deleteRear(self):
        if self.front != None:
            data = self.rear.data
            (다)
            if self.rear == None:
                (라)
            else:
                self.rear.right = None
            return data
```

A. self.front = node
B. self.front = self.rear
C. self.front.left = node
D. self.front.right = node
E. self.rear = self.rear.left
F. self.rear = self.rear.right

	(가)	(나)	(다)	(라)
①	A	C	F	A
②	C	A	E	B
③	C	A	F	D
④	D	C	E	C

6. 크기가 4인 빈 스택에 다음 연산을 차례로 수행한 후의 스택 상태를 바르게 표현한 것은?

push(3) → push(7) → pop() → push(5) → push(8) → peek() → push(10) → pop()

① 5 / 3

② 8 / 5 / 3

③ 10 / 8 / 5

④ 10 / 8 / 5 / 3

7. 숫자 분석법(digit analysis)은 모든 키 값이 미리 알려져 있을 때 유용한 해시 함수이다. 키 집합이 다음과 같을 때, 숫자 분석법을 이용하여 십진수 두 자리의 주소를 생성하고자 한다. 두 자리 주소에 들어갈 자릿수로 가장 적절한 것은?

0001	1006	2011	3016	4021	5026	6031	7036	8041	9046
0002	1007	2012	3017	4022	5027	6032	7037	8042	9047
0003	1008	2013	3018	4023	5028	6033	7038	8043	9048
0004	1009	2014	3019	4024	5029	6034	7039	8044	9049
0005	1010	2015	3020	4025	5030	6035	7040	8045	9050

① 100의 자릿수, 1의 자릿수
② 100의 자릿수, 10의 자릿수
③ 1000의 자릿수, 1의 자릿수
④ 1000의 자릿수, 10의 자릿수

8. 다음 비어 있는 크기가 11인 해시 테이블(hash table)에 입력키가 4, 6, 15, 26, 17 순서로 저장되었다. 입력키 26이 저장된 해시 테이블의 주소는? (단, 해시 함수는 h(k) = k mod 11이고, 충돌 해결은 선형 탐사법(linear probing)을 사용한다)

해시 테이블 주소	0	1	2	3	4	5	6	7	8	9	10
해시 테이블											

① 1
② 4
③ 6
④ 7

9. 다음 조건에 따라 공백 해시 테이블에 키(key) 값 12, 15, 25, 34, 37, 75, 62, 33, 47, 55, 21, 42, 53을 순서대로 삽입할 때, 충돌 횟수가 가장 많은 키 값은? (단, 해시 함수 또는 선형 탐사법을 적용하여 찾아간 버킷에 빈 슬롯이 없을 때마다 충돌이 발생한 것으로 본다)

> ○ 해시 함수: h(x) = x mod 13
> ○ 충돌 해결책: 선형 탐사법(linear probing)
> ○ 해시 테이블의 크기: 13개 버킷(0부터 12까지 인덱스를 가짐)
> ○ 해시 테이블 버킷당 슬롯 수: 1개

① 21
② 42
③ 53
④ 62

10. 다음 파이썬 프로그램의 실행 결과는?

```
import queue

values=queue.Queue()
for i in range(20):
   if i % 3 == 0:
      values.put(i)
   elif i % 4 == 0:
      values.get()
   elif i % 5 == 0:
      values.get()
print(values.get(), values.get())
```

① 15
② 18
③ 15 18
④ 12 15 18

11. 컴파일러는 스택을 이용하여 프로그래머가 작성한 중위표기(Infix Notation) 수식을 후위표기(Postfix Notation) 수식으로 변환하여 계산을 수행한다. 다음 물음에 답하시오. (단, 스택은 오른쪽으로 커지고, +, -, *, /는 산술연산자이고 eos는 문자열의 끝이다.)

(1) 다음 표는 중위표기 수식 a/(b-c*d)*e+f 를 후위표기 수식으로 변환하는 과정을 보여주고 있다. 토큰(연산자 또는 피연산자) 일부를 처리한 결과를 참고하여 나머지 토큰들을 처리할 때 스택과 출력결과를 나타내시오.(15점)

다음 토큰	스택					출력(후위표기 수식)
	[0]	[1]	[2]	[3]	[4]	
없음	공백 (empty)					없음
a						a
/	/					a
(	/	(				a
b	/	(				ab
-	/	(	-			ab
c	/	(	-			abc
*	/	(	-	*		abc
d	/	(	-	*		abcd
)	/					abcd*-
*	*					abcd*-/
e	*					abcd*-/e
+	+					abcd*-/e*
f	+					abcd*-/e*f
eos						abcd*-/e*f+

(2) 후위표기 수식 abc+d**e/의 계산 과정에서 토큰 별로 처리할 때 스택을 나타내시오.(10점)

다음 토큰	스택				
	[0]	[1]	[2]	[3]	[4]
없음	공백 (empty)				
a	a				
b	a	b			
c	a	b	c		
+	a	b+c			
d	a	b+c	d		
*	a	(b+c)*d			
*	a*((b+c)*d)				
e	a*((b+c)*d)	e			
/	(a*((b+c)*d))/e				
eos	공백 (스택에 있는 계산 결과 출력)				

(3) 컴파일러가 중위표기 수식을 후위표기 수식으로 변환하여 계산되는 이유를 설명하시오. (5점)

- 괄호를 사용하지 않아도 계산 순서를 알 수 있음. or 식 자체에 우선순위가 이미 포함되어 있어, 우선순위를 생각할 필요 없음. (3점)
- 중위표기 수식은 괄호와 연산자의 우선순위 때문에 수식을 끝까지 읽은 다음에야 계산 가능하나 후위표기 수식은 수식을 읽으면서 바로 계산 가능함. (2점)