# 1.제목:한국외대 자료구조 과제 4(4-1-1)  
# 2.날짜:20210501  
# 4-1)  
# 과제 3-1-1)을 연결구조(linked stack)로 구현한 스택을 이용하여 프로그램을 작성하시오.  
# ()[{a}]  
# 1  
# ()[{a}] {()]()  
# 0  
class Node: # 노드 클래스  
 def \_\_init\_\_(self, element):  
 self.data = element  
 self.link = None  
  
class LinkedStack:  
 def \_\_init\_\_(self):  
 self.top = None # 객체 초기화 함수  
  
 def peek(self):  
 if not self.isEmpty():  
 return self.top.data # stack이 비어있지 않을 경우 맨 앞 요소 리턴  
  
 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 not self.isEmpty():  
 e = self.top  
 self.top = e.link  
 return e.data  
  
 def print(self): # LinkedStack 출력  
 if self.top is None: # top이 비어있을 경우  
 print("Linked list is empty")  
 else: # top이 비어있지 않을 경우  
 ele = self.top  
 while ele is not None:  
 if ele.link == None:  
 print(ele.data)  
 else:  
 print(ele.data, '->', end='')  
 ele = ele.link  
 print("0") # 예시 1->2->3->4  
  
def solution(s):  
 d = {  
 ')' : '(',  
 '}' : '{',  
 ']' : '['  
 }  
 stack = LinkedStack()  
 for c in s: # input으로 문자열을 받아서 한 문자씩 루프  
 if c in '({[': # 한 문자가 열린괄호 일경우  
 stack.push(c) # 스택 리스트에 append  
 elif c in ')}]': # 한 문자가 닫힌괄호 일경우  
 if stack:  
 top = stack.pop() # top 변수에 스택에 저장되어있는 제일 위의 문자 pop  
 if d[c] != top: # d 변수의 set에서 괄호의 짝이 아닐 경우  
 return False # False 반환  
 else:  
 return False # 위의 모든 경우 아닐 경우 반환  
 return stack.isEmpty() # 스택을 비움  
  
p = input()  
if(solution(p) == False): # solution 함수에서 반환한 값이 False 일 경우  
 print("0") # 0출력  
else:  
 print("1") # 반환 값이 True 일 경우 1출력

# 1.제목:한국외대 자료구조 과제 4(4-1-2)  
# 2.날짜:20210501  
# 다리의 길이와 다리의 최대하중, 그리고 다리를 건너려는 트럭들의 무게가 순서대로 주어졌을 때,  
# 모든 트럭이 다리를 건너는 최단시간을 구하는 프로그램을 작성하시오.  
  
# \*\*\*\* 배열 큐가 아닌 LinkedQueue로 구했습니다. \*\*\*\*  
*'''  
4 2 10  
7 4 5 6  
'''*'''  
8  
'''  
class Node:  
 def \_\_init\_\_(self, element):  
 self.data = element  
 self.link = None  
class Queue:  
 def \_\_init\_\_(self):  
 self.front = self.rear = None # data 받기 전에는 front, rear는 None 상태  
  
 def isEmpty(self):  
 return self.front == None # front가 None일 경우 data 아무것도 없음  
  
 def enqueue(self, e):  
 newNode = Node(e)  
 if self.front == None:  
 self.front = self.rear = newNode  
 else:  
 self.rear.link = newNode  
 self.rear = newNode  
  
 def dequeue(self):  
 if self.isEmpty():  
 print("Queue empty")  
 return 0  
 e = self.front.data  
 self.front = self.front.link  
 if self.front == None:  
 self.rear = None  
 return e  
  
 def peek(self):  
 if self.isEmpty():  
 print("Queue empty")  
 else:  
 return self.front.data # self.front 만 리턴하면 type이 node가 됨  
 # 그러므로 self.front.data로 element를 뽑아냄  
answer = input().split(" ") # n == 다리를 건너는 트럭의 수 w == 다리의 길이 L == 다리의 최대하중  
 # 세 개의 정수 n(1≤n≤1000), w(1≤w≤100), L(10≤L≤1000)  
tmp = input().split(" ") # 입력의 두 번째 줄에는 n개의 정수 a1, a2, ..., an이 주어지는데,  
 # ai는 i번째 트럭의 무게를 나타낸다.  
  
truck = Queue() # truck변수와 bridge변수 Queue 클래스에 연결  
bridge = Queue()  
time = 0 # 최단시간  
bridge\_sum = 0 # 최대 하중  
  
for i in tmp:  
 truck.enqueue(int(i))  
for i in range(int(answer[1])):  
 bridge.enqueue(0) # 다리무게가 0 인 트럭을 세워놓고 enqueue하면 트럭의 무게를 할당해주는 방식으로 풀어냈습니다.  
  
while not bridge.isEmpty():  
 if truck.isEmpty(): # bridge를 할당했지만 truck이 비어 있을 경우  
 bridge.dequeue() # bridge를 dequeue함.  
 else: # bridge 할당하고 truck무게도 할당했을 경우  
 bridge\_sum -= bridge.dequeue() # 최대 하중 = 최대하중 - 브릿지 하중  
  
 if bridge\_sum + truck.peek() <= int(answer[2]): # 최대하중 + 맨앞 트럭 무게가 그다음 무게의 트럭보다 작을 경우  
 tmp = truck.dequeue() # 그다음 트럭 dequeue  
 bridge\_sum += tmp # 브릿지 최대하중에 건너는 트럭 무게 더함  
 bridge.enqueue(tmp) # 트럭이 건너면 브릿지 무게 enqueue  
 else:  
 bridge.enqueue(0) # 브릿지 최대하중 못버팀  
 time += 1 # 트럭 건널때마다 time++  
  
print(time)

# 1.제목:한국외대 자료구조 과제 4(4-1-3)  
# 2.날짜:20210504  
# 4-3) (간단한 연결리스트 만들기 연습)  
class Node:  
 def \_\_init\_\_(self, element):  
 self.data = element  
 self.link = None  
  
class StList:  
 def \_\_init\_\_(self):  
 self.head = None  
  
 def isempty(self): # 노드의 개수를 반환 해주는 함수에서 반환 값이 0인 경우  
 if self.sizelist() != 0:  
 return False # 비어있지 않음 -> False  
 else:  
 return True # 비어있음 ->True  
  
 def insert(self, st\_id): # 교수님께 피드백 받은 대로 노드 삽입하는  
 # 부분에서 새로운 data를 받을 때 마다 head 다음 첫 노드로 연결합니다.  
 newnode = Node(st\_id)  
 newnode.link = self.head  
 self.head = newnode  
 # newnode = Node(st\_id) # 새로받은 data를 노드 끝에 연결하는 것은 주석 처리했습니다.  
 # if self.head is None:  
 # self.head = newnode  
 # else:  
 # cur = self.head  
 # while cur.link is not None:  
 # cur = cur.link  
 # cur.link = newnode  
  
 def delete(self, st\_id):  
 # 지우는 값이 맨 앞 부분 일 때  
 if self.head.data == st\_id:  
 tmp = self.head  
 self.head = self.head.link  
 del tmp  
 else:  
 # 지우는 값이 중간값일때  
 node = self.head  
 while node.link:  
 if node.link.data == st\_id:  
 temp = node.link  
 node.link = node.link.link  
 del temp  
 return  
 else:  
 node = node.link  
  
 def sizelist(self) -> int: # list 크기 반환  
 count = 0  
 current = self.head  
 while current is not None:  
 count += 1  
 current = current.link  
 return count  
  
 def print(self):  
 arr1 = []  
 newfound = self.head  
 while newfound:  
 if newfound.link is not None:  
 arr1.append(newfound.data)  
 newfound = newfound.link  
 else:  
 arr1.append(newfound.data)  
 newfound = newfound.link  
 int\_li = sorted(arr1, key=str) # 맨 앞자리 수 기준 으로 정렬 key =str 이용  
 # int\_li = sorted(map(str, arr1))  
 return int\_li  
  
  
L = StList()  
while True:  
 command = input().split()  
 if command[0] == 'N': # st\_id(즉, command[1])를 수강자리스트에 insert: L.insert(command[1])  
 L.insert(command[1])  
 elif command[0] == 'C': # st\_id(즉, command[1])를 수강자리스트에서 삭제: L.delete(command[1])  
 L.delete(command[1])  
 elif command[0] == 'S': # 수강자리스트의 원소 수를 출력: L.size() 출력  
 print(L.sizelist()) # L.size하면 int형을 반환 할 수 없다고 에러나서 sizelist로 이름 변경했습니다.  
 elif command[0] == 'P': # 수강자리스트의 원소들을 오름차순으로 출력: L.print()  
 arr = L.print() # L.print()  
 for i in arr:  
 print(i, end=' ')  
 elif command[0] == 'Q': # 종료  
 break  
# 교수님께 질문 드렸던 delete 부분과 다른 학우가 질문했던 8번 케이스인 ex) 111-113 처리하는 부분을 고쳤습니다.  
# 입력 예  
'''  
N 111B  
N 111A  
N 111C  
  
  
N 111D  
S  
P  
Q  
'''  
# 출력결과  
'''  
4 # S  
1111 1113 1115 # P  
3 # S  
1111 1112 1113 1115 # P  
'''

# 1.제목:한국외대 자료구조 과제 5(5-1-1)  
# 2.날짜:20210521  
# 5-1-1) 교과목 수강자 관리를 위한 다음 명령어를 처리하는 프로그램을 작성하시오. 수강자는 학번정보(문자열)를 가진다.  
  
class Node:  
 def \_\_init\_\_(self, st\_id):  
 self.st\_id = st\_id # 학번 data  
 self.left = None # 왼쪽 노드 선택  
 self.right = None # 오른쪽 노드 선택  
  
class BSTree:  
 def \_\_init\_\_(self):  
 self.root = None # root노드  
 self.inorder\_list = [] # 중위 순회해서 inorder\_list에 대입  
 self.count = [] # 노드의 개수를 새기 위한 count 변수  
  
 def insert(self, st\_id): # 학번이 id인 학생이 수강신청을 함  
 if self.root is None: # 노드가 아무것도 없을 경우 root노드는 None인 상태  
 self.root = Node(st\_id) # root노드에 처음 학번 노드 대입  
 # root가 있으면 왼쪽배치 or 오른쪽배치  
 else:  
 self.\_\_insert\_node(self.root, st\_id) # root가 있을경우 어느 방향으로 배치 할 것인지 정해주는 \_\_insert\_node함수로 이동  
  
 # root가 있는 경우  
 def \_\_insert\_node(self, node, st\_id):  
 # root 값보다 작으면 왼쪽으로  
 if node.st\_id >= st\_id:  
 if node.left is not None:  
 self.\_\_insert\_node(node.left, st\_id)  
 else:  
 node.left = Node(st\_id)  
 # head 값보다 크면 오른쪽으로  
 elif node.st\_id < st\_id:  
 if node.right is not None:  
 self.\_\_insert\_node(node.right, st\_id)  
 else:  
 node.right = Node(st\_id)  
 else:  
 return node.st\_id  
  
 def delete(self, st\_id): # 학번이 id인 학생이 수강신청을 취소함  
 self.root, deleted = self.\_delete\_value(self.root, st\_id) # root노드가 None일 경우 deleted변수에 False담겨져 있음  
 return deleted # return False  
  
 def \_delete\_value(self, node, st\_id):  
 if node is None: # root노드가 None일 경우 못지움 -> return False  
 return node, False  
 deleted = False  
 if st\_id == node.st\_id: # 지우려는 st\_id와 노드의 st\_id가 같을 경우  
 deleted = True  
 if node.left and node.right:  
 # node.right의 가장 왼쪽에있는 노드를 교체합니다.  
 parent, child = node, node.right  
 while child.left is not None:  
 parent, child = child, child.left  
 child.left = node.left  
 if parent != node:  
 parent.left = child.right  
 child.right = node.right  
 node = child  
 elif node.left or node.right:  
 node = node.left or node.right  
 else:  
 node = None  
 elif st\_id < node.st\_id: # 지우려는 st\_id보다 노드의 st\_id 작을 경우 왼쪽으로 이동  
 node.left, deleted = self.\_delete\_value(node.left, st\_id)  
 else: # 지우려는 st\_id보다 노드의 st\_id 작을 경우 오른쪽으로 이동  
 node.right, deleted = self.\_delete\_value(node.right, st\_id)  
 return node, deleted  
  
 def print\_list(self):  
 if self.root is not None:  
 self.sizelist(self.root)  
  
 def sizelist(self, node): # list 크기 반환  
 if node.left is not None:  
 self.sizelist(node.left)  
  
 self.count.append(node.st\_id)  
  
 if node.right is not None:  
 self.sizelist(node.right)  
  
 # 오름차순으로 출력 (중위)  
 def in\_order\_traversal(self):  
 self.inorder\_list = []  
 def \_in\_order\_traversal(root):  
 if root is None:  
 pass  
 else:  
 \_in\_order\_traversal(root.left)  
 self.inorder\_list.append(root.st\_id)  
 \_in\_order\_traversal(root.right)  
 \_in\_order\_traversal(self.root)  
 int\_li = sorted(self.inorder\_list, key=str)  
 # int\_li = sorted(map(str, self.inorder\_list))  
 return int\_li  
  
classStudents = BSTree()  
while True:  
 command = input().split()  
 if command[0] == 'N': # st\_id(즉, command[1])를 수강자리스트에 insert  
 classStudents.insert(command[1])  
 elif command[0] == 'C': # st\_id(즉, command[1])를 수강자리스트에서 delete  
 classStudents.delete(command[1])  
 elif command[0] == 'S': # 수강자리스트의 원소 수를 출력  
 classStudents.count = []  
 classStudents.print\_list()  
 print(len(classStudents.count), end='\n')  
 elif command[0] == 'P': # 수강자리스트의 원소들을 오름차순으로 출력  
 arr = classStudents.in\_order\_traversal()  
 for i in arr:  
 print(i, end=' ')  
  
 elif command[0] == 'Q': # 종료  
 break  
'''  
N 111A  
N 111B  
N 111C  
P  
N 111-222  
P  
N 000-111  
P   
C 111-222  
P  
'''  
# 삽입되는 부분에서 문제있음

# 1.제목:한국외대 자료구조 과제 5(5-1-2)  
# 2.날짜:20210526  
# 2) 서점의 도서 재고관리 프로그램을 작성하시오.  
from operator import itemgetter  
  
  
class Node:  
 # st\_id 도서번호, title 도서이름, price 가격, num 수량  
 def \_\_init\_\_(self, st\_id, title, price, num):  
 self.st\_id = st\_id # 도서번호  
 self.title = title # 도서이름  
 self.price = price # 가격  
 self.num = num # 수량  
 self.book = [self.st\_id, self.title, self.price, self.num] # book이라는 리스트를 만들어  
 # 입력한 것들을 모두 포함하는 도서정보를 만듬듬  
 self.left = None # 왼쪽 트리  
 self.right = None # 오른쪽 트리  
  
  
class BSTree:  
 def \_\_init\_\_(self):  
 self.root = None # 루트 노드  
 self.sellbook\_list = [] # 판매한 모든 도서의 판매정보  
 self.order\_list = [] # 오름차순으로 출력하는 리스트  
 self.count = [] # 노드 개수 세는 변수  
  
 def insert(self, st\_id, title, price, num): # 입력: 도서번호, 도서이름, 가격, 입고수량  
 if self.root is None: # 루트 노드가 None일 경우(노드가 없을 경우)  
 self.root = Node(st\_id, title, price, num) # root노드에 첫번째 노드 insert  
 self.root.book = [st\_id, title, price, num] # 도서 목록  
 # root가 있으면 왼쪽배치 or 오른쪽배치  
 else:  
 if self.root.book[0] == st\_id:  
 return print("error: 1") # 유의사항: 신규도서가 재고도서 목록에 있을 경우 “error: 1” 출력  
 else:  
 self.\_\_insert\_node(self.root, st\_id, title, price, num) # 왼쪽, 오른쪽 자식노드 결정하는 함수로 이동  
 # root가 있는 경우  
  
 def \_\_insert\_node(self, node, st\_id, title, price, num): # 왼쪽, 오른쪽 자식노드 결정하는 함수  
 # root 값이 크면 왼쪽으로  
 if node.st\_id > st\_id: # 왼쪽 자식 노드  
 if node.left is not None:  
 self.\_\_insert\_node(node.left, st\_id, title, price, num)  
 else:  
 node.left = Node(st\_id, title, price, num)  
 # root 값이 작으면 오른쪽으로  
 elif node.st\_id < st\_id: # 오른쪽 자식 노드  
 if node.right is not None:  
 self.\_\_insert\_node(node.right, st\_id, title, price, num)  
 else:  
 node.right = Node(st\_id, title, price, num)  
 elif node.st\_id == st\_id: # 유의사항: 신규도서가 재고도서 목록에 있을 경우 “error: 1” 출력  
 return print("error: 1")  
 else:  
 return node.st\_id  
  
 def addbook(self, st\_id, num): # 재고도서 목록에 있는 도서 입고  
 if self.root.st\_id is None:  
 return print("error: 2") # 유의사항: 도서번호의 도서가 재고도서 목록에 없을 경우 “error: 2” 출력  
 else:  
 return self.\_\_add\_book(self.root, st\_id, num) # 왼쪽, 오른쪽 자식노드에서 st\_id 찾는 함수  
  
 def \_\_add\_book(self, node, st\_id, num):  
 if node.st\_id == st\_id: # 왼쪽 자식 or 오른쪽 자식 노드에서 입력한 학생 번호와 일치 할 경우  
 node.book[3] = str(int(node.book[3]) + int(num)) # 도서 입고후 더하는 부분  
 return True  
 else:  
 if node.st\_id > st\_id: # 왼쪽 자식 일 경우  
 if node.left is not None:  
 return self.\_\_add\_book(node.left, st\_id, num)  
 else:  
 return print("error: 2")  
 if node.st\_id < st\_id: # 오른쪽 자식 일 경우  
 if node.right is not None:  
 return self.\_\_add\_book(node.right, st\_id, num)  
 else:  
 return print("error: 2")  
  
 # 판매한 모든 도서의 판매정보(도서번호, 도서이름, 가격, 판매수량)를 도서번호 오름차순으로 출력  
 def sellbook(self, st\_id, num): # 재고도서 목록에 있는 도서를 판매함  
 if self.root.st\_id is None: # 재고도서 목록 root가 None일 경우(도서목록에 도서 없는 경우)  
 return print("error: 2") # 유의사항: 입력되는 도서번호가 재고도서 목록에 없을 경우, “error: 2”  
 else: # 재고도서 목록 root에 노드가 있는 경우  
 return self.\_\_sell\_book(self.root, st\_id, num) # 왼쪽자식, 오른쪽자식노드로 이동하는 함수  
  
 def \_\_sell\_book(self, node, st\_id, num): # 왼쪽자식, 오른쪽자식노드로 이동하는 함수  
 # root 값보다 작은 왼쪽자식으로  
 if node.st\_id > st\_id: # 왼쪽 자식 일 경우  
 if node.left is not None:  
 return self.\_\_sell\_book(node.left, st\_id, num)  
 else:  
 return print("error: 2")  
 # root 값보다 크면 오른쪽자식으로  
 elif node.st\_id < st\_id: # 오른쪽 자식 일 경우  
 if node.right is not None:  
 return self.\_\_sell\_book(node.right, st\_id, num)  
 else:  
 return print("error: 2")  
 elif node.st\_id == st\_id: # 판매하려는 학생번호와 노드의 학생번호가 일치 할 경우  
 if int(node.book[3]) < int(num): # 판매수량이 재고 수량보다 많을 경우  
 return print("error: 3") # 판매수량이 재고수량보다 많을 경우 “error: 3” 출력  
 else: # 판매수량이 재고수량보다 작을 경우(판매할 수 있는 경우)  
 node.book[3] = str(int(node.book[3]) - int(num)) # 도서목록에서 num(판매하려는 도서)만큼 뺌  
 if len(self.sellbook\_list) == 0: # 판매한 모든 도서의 판매정보가 없을 경우  
 self.sellbook\_list += [st\_id, node.title, node.price, num] # 판매 정보 입력  
 else: # 판매한 도서의 판매정보가 있을경우  
 for i in self.sellbook\_list: # 리스트의 한단어씩 검사함  
 if i == st\_id: # 검사해서 같은 도서를 또 파는 경우  
 n = int(self.sellbook\_list.index(i))  
 if n == 0:  
 p = 3  
 else:  
 p = n + 3 # 도서수량의 인덱스는 3 7 11 .... 점화식임  
 self.sellbook\_list[p] = str(int(self.sellbook\_list[p]) + int(num)) # 기존판매수량 +판매수량  
 return True  
 self.sellbook\_list += [st\_id, node.title, node.price, num]  
  
 def delete(self, st\_id): # 도서 폐기 (재고도서 목록에서 완전히 삭제함)  
 self.root, deleted = self.\_delete\_value(self.root, st\_id)  
 return deleted  
  
 def \_delete\_value(self, node, st\_id):  
 if node is None: # 도서목록에 삭제하려는 도서 없을 경우 == root에 노드 없을 경우  
 print("error: 2")  
 return node, False  
 deleted = False  
 if st\_id == node.st\_id: # 삭제하려는 노드와 도서번호가 같을경우  
 deleted = True  
 if node.left and node.right:  
 # node.right의 가장 왼쪽에있는 노드를 교체합니다.  
 parent, child = node, node.right  
 while child.left is not None: # 왼쪽 자식 노드가 없을경우  
 parent, child = child, child.left  
 child.left = node.left  
 if parent != node:  
 parent.left = child.right  
 child.right = node.right  
 node = child  
 elif node.left or node.right:  
 node = node.left or node.right  
 else:  
 node = None  
 elif st\_id < node.st\_id: # 삭제하려는 도서번호가 비교하는 노드보다 작을 경우  
 node.left, deleted = self.\_delete\_value(node.left, st\_id)  
 elif st\_id > node.st\_id: # 삭제하려는 도서번호가 비교하는 노드보다 클 경우  
 node.right, deleted = self.\_delete\_value(node.right, st\_id)  
 return node, deleted  
  
 def inquirybook(self, st\_id): # 도서의 재고 상태 조회  
 if self.root.st\_id is None:  
 return print("error: 2") # 유의사항: 입력되는 도서번호가 도서재고 목록에 없을 경우 “error: 2” 출력  
 else:  
 return self.\_\_inquiry\_book(self.root, st\_id)  
  
 def \_\_inquiry\_book(self, node, st\_id): # 왼쪽, 오른쪽 자식노드에서 st\_id 찾기  
 if node.st\_id == st\_id:  
 print(' '.join(node.book), end=' ') # 도서번호 일치할경우 해당 도서 출력  
 return True  
 else:  
 if node.st\_id > st\_id:  
 if node.left is not None:  
 return self.\_\_inquiry\_book(node.left, st\_id)  
 else:  
 return print("error: 2")  
 elif node.st\_id < st\_id:  
 if node.right is not None:  
 return self.\_\_inquiry\_book(node.right, st\_id)  
 else:  
 return print("error: 2")  
  
 def print\_list(self): # 도서재고 목록에 있는 모든 도서의 재고상태(도서번호, 도서이름, 가격, 재고수량)를 도서번호 오름차순으로 출력  
 if self.root is not None:  
 self.sizelist(self.root) # 왼쪽, 오른쪽 자식 노드에서 찾는 함수  
  
 def sizelist(self, node): # 왼쪽, 오른쪽 자식 노드에서 찾는 함수  
 if node.left is not None: # 노드를 리스트에 append 한후 개수를 샙니다.  
 self.sizelist(node.left)  
 self.count.append(node.st\_id)  
 if node.right is not None:  
 self.sizelist(node.right)  
  
 # 도서재고 목록에 있는 모든 도서의 재고상태(도서번호, 도서이름, 가격, 재고수량)를 도서번호 오름차순으로 출력  
 def in\_order\_traversal(self):  
 self.order\_list = []  
  
 def \_in\_order\_traversal(root):  
 if root is None:  
 pass  
 else:  
 \_in\_order\_traversal(root.left)  
 self.order\_list.append(root.book)  
 \_in\_order\_traversal(root.right)  
  
 \_in\_order\_traversal(self.root)  
 int\_li = sorted(self.order\_list, key=str)  
 return int\_li  
  
 def list\_chunk(self, lst, n): # 리스트를 일정한 수를 반복해서 자르는 함수 -> 중첩 리스트로 만들어줌  
 return [lst[i:i + n] for i in range(0, len(lst), n)]  
  
 def L\_listprint(self): # 판매한 모든 도서의 판매정보(도서번호, 도서이름, 가격, 판매수량)를 도서번호 오름차순으로 출력  
 int\_li = self.list\_chunk(self.sellbook\_list, 4) # 리스트를 4개씩 잘라서 중첩리스트 만듬  
 int\_li.sort(key=itemgetter(0)) # 인덱스 0를 기준으로 리스트를 정렬함  
 for item in int\_li:  
 print(' '.join(item), end='\n')  
  
  
classStudents = BSTree()  
while True:  
 command = input().split()  
 if command[0] == 'N': # 신규도서 입고  
 classStudents.insert(command[1], command[2], command[3], command[4])  
 elif command[0] == 'D': # 도서 폐기 (재고도서 목록에서 완전히 삭제함)  
 classStudents.delete(command[1])  
 elif command[0] == 'R': # 재고도서 목록에 있는 도서 입고  
 classStudents.addbook(command[1], command[2])  
 elif command[0] == 'S': # 재고도서 목록에 있는 도서를 판매함  
 classStudents.sellbook(command[1], command[2])  
 elif command[0] == 'I': # 도서의 재고 상태 조회  
 classStudents.inquirybook(command[1])  
 elif command[0] == 'L': # 수강자리스트의 원소 수를 출력  
 arr = classStudents.L\_listprint()  
 elif command[0] == 'P': # 수강자리스트의 원소들을 오름차순으로 출력  
 arr = classStudents.in\_order\_traversal()  
 for i in arr:  
 print(' '.join(i), end='\n')  
 elif command[0] == 'Q': # 종료  
 break

# 1.제목:한국외대 자료구조 과제 6(6-1-1)  
# 2.날짜:20210605  
# 사전 해싱  
class DoubleHashing:  
 def \_\_init\_\_(self, size):  
 self.M = size  
 self.hash\_table = [None for x in range(size+1)] # hashtable  
 self.data = [None for x in range(size+1)] # key관련 data저장  
 self.N = 0 # 항목수  
  
 def getkey(self, key): # 키를 [0]알파벳의 아스키코드값으로 받음  
 self.key = ord(key[0])  
 return self.key  
  
 def hash\_func(self, key):  
 return key % self.M  
  
 def getAddress(self, key): # 해시값을 구함 인덱스 번호  
 myKey = self.getkey(key)  
 hash\_address = self.hash\_func(myKey)  
 return hash\_address  
  
 def put(self, key, data): # 단어 삽입 함수  
 init\_num = self.getAddress(key) # 초기 위치  
 i = init\_num # 초기 인덱스  
 second\_i = 7 - (self.getkey(key) % 7) # hash 2  
 j = 0  
 while True:  
 if self.hash\_table[i] == None: # 삽입 위치 찾기  
 self.hash\_table[i] = key # key를 해쉬 테이블에 저장  
 self.data[i] = data # key값이 데이터를 동일한 인덱스에 저장  
 self.N += 1  
 return True  
 elif self.hash\_table[i] != None: # 해시값에 data가 있는경우  
 if self.hash\_table[i] == key: # 입력받은 키와 기존의 키가 같으면  
 return print("error1") # error1출력  
 if self.hash\_table[i] == key:  
 self.data[i] = data # data 갱신  
 return True  
 j += 1  
 i = (init\_num + j\*second\_i) % self.M # 충돌시 이중해쉬 하는 부분  
 if self.N > self.M: # 테이블이 꽉찰 경우  
 break  
 return print("error1")  
  
 def get(self, key): # 검색  
 init\_num = self.getAddress(key)  
 i = init\_num  
 second\_i = 7 - (self.getkey(key) % 7)  
 j = 0  
 while self.hash\_table[i] != None:  
 if self.hash\_table[i] == key: # 사전에서 단어를 검색하여  
 return self.data[i] # 뜻을 출력  
 j += 1  
 i = (init\_num + j\*second\_i) % self.M  
 return print("error2") # 단어가 사전에 없을 경우 error2 출력  
  
 def correct(self, key, data): # 당어 뜻 수정 함수  
 init\_num = self.getAddress(key)  
 i = init\_num  
 second\_i = 7 - (self.getkey(key) % 7)  
 j = 0  
 while self.hash\_table[i] != None:  
 if self.hash\_table[i] == key: # key가 일치할경우  
 self.data[i] = data # 입력받은 data로 수정  
 return  
 j += 1  
 i = (init\_num + j \* second\_i) % self.M  
 return print("error2") # 단어가 사전에 없을 경우 error2 출력  
  
 def print\_table(self): # 사전 출력 ( 쓰이진 않음)  
 for i in range(self.M):  
 if self.hash\_table[i] == None:  
 pass  
 else:  
 print(str(self.hash\_table[i]) + str(self.data[i]))  
  
  
  
h = DoubleHashing(13)  
while True:  
 command = input().split()  
 #print(command)  
 if command[0] == '단어삽입':  
 h.put(command[1], command[2])  
 elif command[0] == '단어뜻수정':  
 h.correct(command[1], command[2])  
 elif command[0] == '단어검색':  
 print(h.get(command[1]))  
 elif command[0] == 'p': # 사전 확인용  
 h.print\_table()  
 elif command[0] == '종료': # 종료  
 break  
  
'''  
단어삽입 time 시간  
단어삽입 bee 벌  
단어삽입 zoo 동물원  
단어삽입 school 학기  
단어삽입 trade 무역  
단어삽입 farm 농장  
단어삽입 zoo 동물원   
단어뜻수정 schol 학교  
단어뜻수정 school 학교  
단어검색 school  
종료  
'''