

**자료 구조 과제 5**

**과목명 자료구조**

**담당교수 김희철교수님**

**제출일 20210531**

**전공 컴퓨터전자시스템**

**학번 201904458**

**이름 이준용**

* **문제 1-1 번**

1) 교과목 수강자 관리를 위한 다음 명령어를 처리하는 프로그램을 작성하시오. 수강자는 학번정보(문자열)를 가진다.

**명령어 종류**

N id // 학번이 id인 학생이 수강신청을 함

C id // 학번이 id인 학생이 수강신청을 취소함

S // 수강 학생수를 출력

P // 수강 학생들을 학번순서(오름차순)대로 출력

Q // 끝내기

**명령어 입력에 있어서 오류는 없다고 가정한다.** 예를 들어, id가 1111인 학생이 수강신청을 하였을 경우, N 1111의 명령어는 주어지지 않는다. id가 1112인 학생이 수강신청을 하지 않았을 경우, C 1112인 명령어는 주어지지 않는다.

**요구조건: 이진탐색트리(연결구조 표현)를 이용한다.**

class Node:

def \_\_init\_\_(st\_id, link = None):

self.st\_id = st\_id

self.link = link

class BSTree

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

self.root = None

...

**main 프로그램 예**

classStudents = BST() # 교과목 수강자 (이진탐색트리)

while True:

command = input()

if command == 'N':

st\_id = input().split()

# st\_id을 classStudents에 insert: classStudents.insert(st\_id);

...

elif command == 'Q':

break

입력 출력 예:

|  |  |
| --- | --- |
| 입력 예 | 출력 예 |
| N 1113  N 1112  N 1111  N 1115  S  C 1112  P  S  N 1112  P  Q | 4 # S  1111 1113 1115 # P  3 # S  1111 1112 1113 1115 # P |

* **알고리즘 및 자료구조**

Class Node - st\_id 학번 data 저장

* Left 왼쪽 자식 노드 이동
* Right 오른쪽 자식 노드 이동

Class BSTree – root 루트 노드 저장

* Inorder\_list 중위순회해서 배열에 원소 하나씩 저장
* Count 중위순회한 list 원소들의 개수를 셈

insert 학번이 id인 학생이 수강신청을 함

\_\_insert\_node 노드를 왼쪽 배치할지 오른쪽 배치할지 정하는 함수

delete 학번을 지우는 함수

\_delete\_value 학번을 지우고 부모 노드와 연결하는 함수

print\_list 노드 개수를 새서 반환하는 함수

sizelist 노드 개수를 새는 함수

in\_order\_traversal 중위순회해서 오름차순으로 출력

* **느낀 점**

교수님께 질문했었던 delete부분들과 self.root에 None으로 바꿔서 처음 노드 생성시 root에는 None에서 시작하는 것으로 바꿨습니다.

* **프로그램 코드**

# 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

* **문제 1-2 번**

2) 서점의 도서 재고관리 프로그램을 작성하시오.

(a) **재고 도서들은 이진탐색트리(연결구조로 표현)로 구현**

도서는 도서번호(문자열), 도서이름(공백이 없는 문자열), 가격(정수), 수량(정수) 정보를 가진다.

(b) 명령어 종류는 다음과 같다.

• N # 신규도서 입고

입력: 도서번호, 도서이름, 가격, 입고수량

입력 예: N 1111 DataStructures 20000 10

유의사항: 신규도서가 재고도서 목록에 있을 경우 “error: 1” 출력

• R # 재고도서 목록에 있는 도서 입고

입력: 도서번호, 입고수량

입력 예: R 1111 3

유의사항: 도서번호의 도서가 재고도서 목록에 없을 경우 “error: 2” 출력

• S # 재고도서 목록에 있는 도서를 판매함

입력: 도서번호 판매수량

입력 예: S 1111 2

유의사항: 입력되는 도서번호가 재고도서 목록에 없을 경우, “error: 2” 출력하고, 판매수량이 재고수량보다 많을 경우 “error: 3” 출력

• D # 도서 폐기 (재고도서 목록에서 완전히 삭제함)

입력: 도서번호

입력 예: 1112

유의사항: 입력되는 도서번호가 재고도서 목록에 없을 경우 “error: 2” 출력

• I # 도서의 재고 상태 조회

입력 : 도서번호

입력 예: 1111

출력 : 입력된 도서번호의 재고 상태(도서번호 이름 가격 재고수량)를 출력

**※ 재고 수량이 0인 경우도 출력함**

출력 예

1111 DataStrctures 20000 11

유의사항: 입력되는 도서번호가 도서재고 목록에 없을 경우 **“error: 2” 출력**

• P # 도서재고 목록에 있는 모든 도서의 재고상태(도서번호, 도서이름, 가격, 재고수량)를 도서번호 오름차순으로 출력

**※ 재고 수량이 0인 도서도 출력함**

출력 예

1111 DataStrctures 20000 11

1112 Algorithms 18000 4

...

• L # 판매한 모든 도서의 판매정보(도서번호, 도서이름, 가격, 판매수량)를 도서번호 오름차순으로 출력

• 종료(Q) #

|  |  |
| --- | --- |
| 입력 예 | 출력 예 |
| N 1111 DataStructures 20000 10  N 1114 Algorithms 22000 20  S 1111 1  R 1111 15  P  N 1113 C++Programming 21000 5  S 1114 3  I 1114  D 1111  P  S 1114 2  L  S 1114 15  S 1112 4  N 1112 DiscreteMathematics 20000 5  P  Q | 1111 DataStructures 20000 24 // P  1114 Algorithms 22000 20  1114 Algorithms 22000 17 // I 1114  1113 C++Programming 21000 5 // P  1114 Algorithms 22000 17  1111 DataStructures 20000 1 // L  1114 Algorithms 22000 5  error: 2 // S 1112 4  1112 DiscreteMathematics 20000 5 // P  1113 C++Programming 21000 5  1114 Algorithms 22000 0 |

프로그램을 종료함

**main 프로그램 예**

while True:

command = input()

if command == 'N':

....

elif command == 'R':

elif command == 'Q':

break;

* **알고리즘 및 자료구조**

Class Node - st\_id 도서번호 저장

* title 도서이름
* price 가격
* num 수량
* book [st\_id, title, price, num] 재고 도서 목록
* Left 왼쪽 자식 노드 이동
* Right 오른쪽 자식 노드 이동

Class BSTree – root 루트 노드

* order\_list 중위순회해서 배열에 원소 하나씩 저장
* count 중위순회한 list 원소들의 개수를 셈

insert 도서를 입고하는 함수

\_\_insert\_node 노드를 왼쪽 배치할지 오른쪽 배치할지 정하는 함수

addbook 재고도서 목록에 있는 도서 입고

\_\_add\_book 재고도서 번호와 입고하려는 도서번호와 일치할 경우

delete 도서를 지우는 함수. 노드의 st\_id(도서번호)와 비교해서 지움

\_delete\_value 학번을 지우고 부모 노드와 연결하는 함수

Inquirybook 도서의 재고 상태 조회

\_\_inquiry\_book 도서의 재고 상태 조회시 자식노드에서 찾을 경우 찾는 함수 도서번호 일치할 경우 출력

print\_list 노드 개수를 새서 반환하는 함수

sizelist 노드 개수를 새는 함수

in\_order\_traversal 도서재고 목록에 있는 모든 도서의 재고상태 오름차순으로 출력

list\_chunk 리스트를 일정한 수를 반복해서 자르는 함수 -> 중첩 리스트로 만들어줌

L\_listprint 판매한 모든 도의 판매정보 오름차순으로 출력

인덱스 0(도서 번호)을 기준으로 리스트 정렬

* **느낀 점**

Delete부분(도서번호 삭제하는 부분)에서 질문했던 부분 테스트 케이스 4번과 11번 오류부분이 어떤 이유 때문에 런타임 오류가 나는지 고치지 못했습니다.

* **프로그램 코드**
* # 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.root1 = None  
   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): # 재고도서 목록에 있는 도서 입고  
   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