使用學生模型創建學生資料，使用acess()把資料放到節點裡面

使用search(id) 查詢學生ID，用在修改和刪除

用inorder(node) # 使用中序排列，把樹整個run過一遍

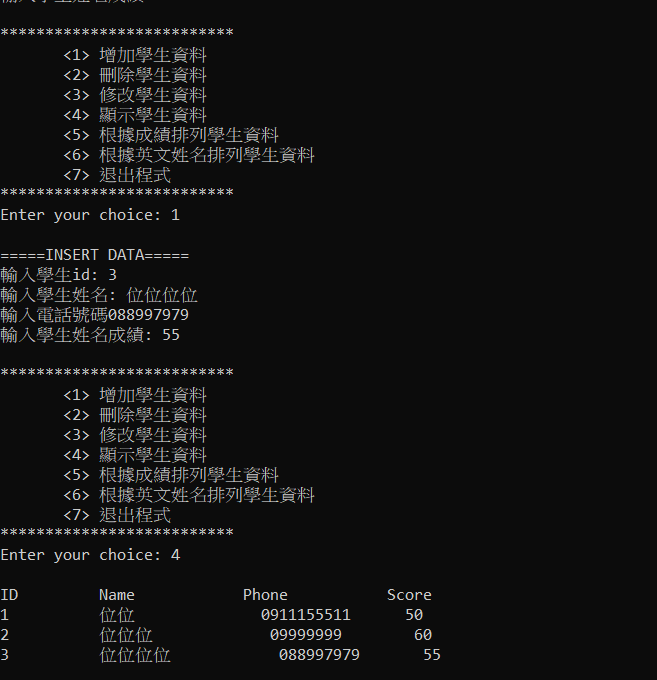
條件1:

可以輸入學生姓名、電話、成績

使用學生模型

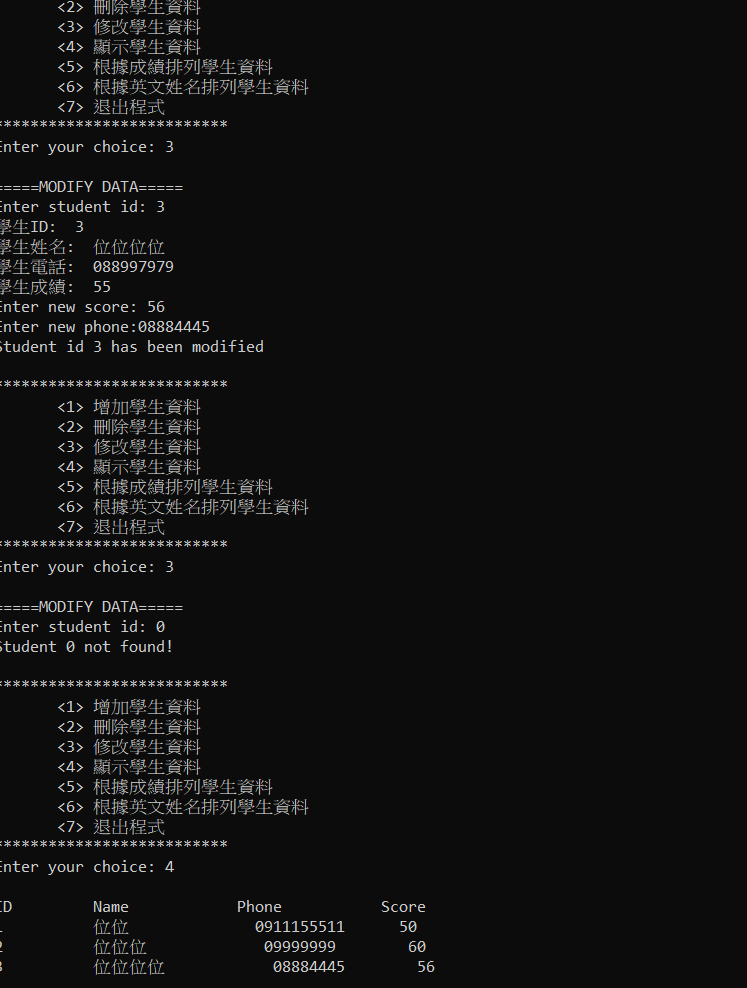
class Student:  
 def \_\_init\_\_(self):  
 self.id = 0 # 學生id  
 self.name = '' # 學生姓名  
 self.score = 0 # 學生成績  
 self.phone = ''  
  
 self.llink = None # 左子鏈結  
 self.rlink = None # 右子鏈結  
  
  
root = None

再對輸入和輸入做修改

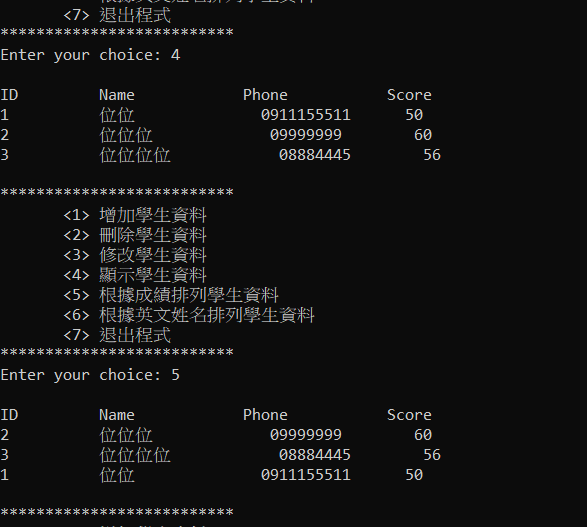


目前輸入，現在要修改電話和成績 (2

根據ID做修改



可以根據成績做排序 (3

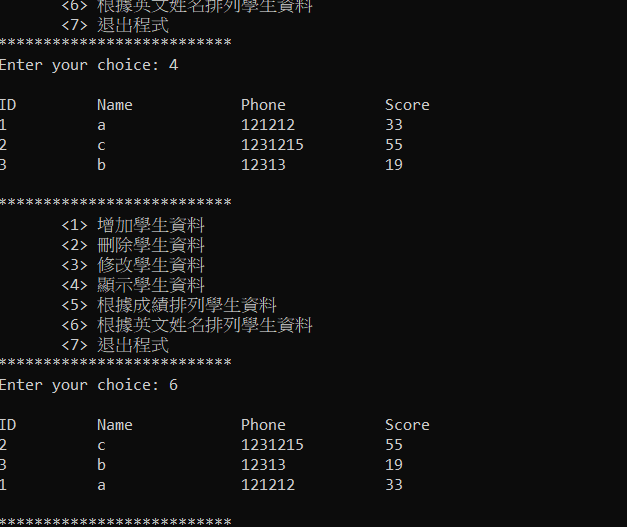


上面排序是使用氣泡法

def order\_by\_score(node):  
 dict01 = []  
 if node != None:  
 inorder(node.llink)  
 dict01.append({"id": node.id, "name": node.name, "phone": node.phone, "score": node.score})  
 inorder(node.rlink)  
  
 for i in range(len(dict01)-1):  
 temp = 0  
 for y in range(i+1, len(dict01)):  
 if dict01[i]["score"] > dict01[y]["score"]:  
 dict01[i], dict01[y] = dict01[y], dict01[i]  
  
 for i in range(len(dict01)):  
 print('%-10d %-15s %-15s %-3d' % (dict01[i]["id"], dict01[i]["name"], dict01[i]["phone"], dict01[i]["score"]))

最後是對姓名做排序 (4

跟上一提很像，我也是用氣泡法，英文ASCLL比較大在上面



程式碼如下

def order\_by\_name(node):  
 dict01 = []  
 if node != None:  
 inorder(node.llink)  
 dict01.append({"id": node.id, "name": node.name, "phone": node.phone, "score": node.score})  
 inorder(node.rlink)  
  
 for i in range(len(dict01)-1):  
 for y in range(i+1, len(dict01)):  
 if dict01[i]["name"] > dict01[y]["name"]:  
 dict01[i], dict01[y] = dict01[y], dict01[i]  
  
 for i in range(len(dict01)):  
 print('%-10d %-15s %-15s %-3d' % (dict01[i]["id"], dict01[i]["name"], dict01[i]["phone"], dict01[i]["score"]))

最後是我的完整程式碼

# 二元搜尋樹的加入、刪除與修改  
# File Name: binarySearchTree.py  
# version 4.0 (updated on Jan. 6, 2021)  
  
import sys  
  
  
class Student:  
 def \_\_init\_\_(self):  
 self.id = 0 # 學生id  
 self.name = '' # 學生姓名  
 self.score = 0 # 學生成績  
 self.phone = ''  
  
 self.llink = None # 左子鏈結  
 self.rlink = None # 右子鏈結  
  
  
root = None  
  
  
# 新增函數；新增一筆新的資料  
def insert\_f():  
 print('\n=====INSERT DATA=====')  
 id = eval(input('輸入學生id: '))  
 name = input('輸入學生姓名: ')  
 phone = input("輸入電話號碼")  
 score = eval(input('輸入學生姓名成績: '))  
  
 access(id, name, phone, score)  
  
  
# 刪除函數；將資料從二元搜尋樹中刪除  
def delete\_f():  
 if root == None:  
 print('No student record!')  
 return  
 print('\n=====DELETE DATA=====')  
 id = eval(input('Enter student id: '))  
  
 removing(id)  
  
  
# 修改函數；修改學生成績  
def modify\_f():  
 node = None  
 if root == None: # 判斷根節點是否為空  
 print('No student record!')  
 return  
 else:  
 print('\n=====MODIFY DATA=====')  
 id = eval(input('Enter student id: '))  
  
 node = search(id)  
 if node == None:  
 print('Student %d not found!' % (id))  
 else:  
 # 列出原資料狀況  
 print('學生ID: ', node.id)  
 print('學生姓名: ', node.name)  
 print('學生電話: ', node.phone)  
 print('學生成績: ', node.score)  
 node.score = eval(input('Enter new score: '))  
 node.phone = input("Enter new phone:")  
 print('Student id %d has been modified' % (id))  
  
  
# 輸出函數；依照人名由小至大輸出至螢幕  
def show\_f():  
 if root == None: # 判斷根節點是否為空  
 print('No student record!')  
 return  
 print('\n%-10s %-15s %-15s %-6s' % ('ID', 'Name', 'Phone', 'Score'))  
  
  
  
# 處理二元搜尋樹，將新增資料加入至二元搜尋樹中  
def access(id, name, phone, score):  
 global root  
 node = None  
 prev = None  
 if search(id) != None: # 資料已存在則顯示錯誤  
 print('Student id %d has existed!' % (id))  
 return  
 ptr = Student()  
 ptr.id = id  
 ptr.name = name  
 ptr.phone = phone  
 ptr.score = score  
 ptr.llink = None  
 ptr.rlink = None  
 if root == None: # 當根節點為空的狀況  
 root = ptr  
 else: # 當根節點不為空的狀況  
 node = root  
 while node != None: # 搜尋資料插入點  
 prev = node  
 if ptr.id < node.id:  
 node = node.llink  
 else:  
 node = node.rlink  
 if ptr.id < prev.id:  
 prev.llink = ptr  
 else:  
 prev.rlink = ptr  
  
  
# 將資料從二元搜尋樹中移除  
def removing(id):  
 global root  
 del\_node = search(id)  
 if del\_node == None: # 找不到資料則顯示錯誤  
 print('Student id %d not found!' % (id))  
 return  
  
 # 節點不為樹葉節點的狀況  
 if del\_node.llink != None or del\_node.rlink != None:  
 del\_node = replace(del\_node)  
 else:  
 if del\_node == root:  
 root = None  
 else:  
 connect(del\_node, 'n')  
 del\_node = None # 釋放記憶體  
 print('Student id %d has been deleted!' % (id))  
  
  
# 尋找刪除非樹葉節點的替代節點  
def replace(node):  
 re\_node = None  
 # 當右子樹找不到替代節點，會搜尋左子樹是否存在替代節點  
 re\_node = search\_re\_r(node.rlink)  
 if re\_node == None:  
 re\_node = search\_re\_l(node.llink)  
 if re\_node.rlink != None: # 當替代節點有右子樹存在的狀況  
 connect(re\_node, 'r')  
 elif re\_node.llink != None: # 當替代節點有左子樹存在的狀況  
 connect(re\_node, 'l')  
 else: # 當替代節點為樹葉節點的狀況  
 connect(re\_node, 'n')  
 node.id = re\_node.id  
 node.name = re\_node.name  
 node.score = re\_node.score  
 return re\_node  
  
  
# 調整二元搜尋樹的鏈結，link為r表示處理右鏈結、為l表示處理左鏈結、  
# 為n則將鏈結指向None  
def connect(node, link):  
 parent = search\_p(node) # 搜尋父節點  
 if node.id < parent.id: # 節點為父節點左子樹的狀況  
 if link == 'r': # link為r  
 parent.llink = node.rlink  
 elif link == 'l': # link為l  
 parent.llink = node.llink  
 else: # link為n  
 parent.llink = None  
 else: # 節點為父節點右子樹的狀況，  
 if link == 'r': # link為r  
 parent.rlink = node.rlink  
 elif link == 'l': # link 為 l  
 parent.rlink = node.llink  
 else: # link為n  
 parent.rlink = None  
  
  
# 以中序法輸出資料，採遞迴方式  
def inorder(node):  
 if (node != None):  
 inorder(node.llink)  
 print('%-10d %-15s %-15s %-3d' % (node.id, node.name, node.phone, node.score))  
 inorder(node.rlink)  
  
  
def order\_by\_score(node):  
 dict01 = []  
 if node != None:  
 inorder(node.llink)  
 dict01.append({"id": node.id, "name": node.name, "phone": node.phone, "score": node.score})  
 inorder(node.rlink)  
  
 for i in range(len(dict01)-1):  
 temp = 0  
 for y in range(i+1, len(dict01)):  
 if dict01[i]["score"] > dict01[y]["score"]:  
 dict01[i], dict01[y] = dict01[y], dict01[i]  
  
 for i in range(len(dict01)):  
 print('%-10d %-15s %-15s %-3d' % (dict01[i]["id"], dict01[i]["name"], dict01[i]["phone"], dict01[i]["score"]))  
  
def order\_by\_name(node):  
 dict01 = []  
 if node != None:  
 inorder(node.llink)  
 dict01.append({"id": node.id, "name": node.name, "phone": node.phone, "score": node.score})  
 inorder(node.rlink)  
  
 for i in range(len(dict01)-1):  
 for y in range(i+1, len(dict01)):  
 if dict01[i]["name"] > dict01[y]["name"]:  
 dict01[i], dict01[y] = dict01[y], dict01[i]  
  
 for i in range(len(dict01)):  
 print('%-10d %-15s %-15s %-3d' % (dict01[i]["id"], dict01[i]["name"], dict01[i]["phone"], dict01[i]["score"]))  
  
# 搜尋target所在節點  
def search(target):  
 global root  
 node = root  
 while node != None:  
 if target == node.id:  
 return node  
 elif target < node.id: # target小於目前節點，往左搜尋  
 node = node.llink  
 else: # target大於目前節點，往右搜尋  
 node = node.rlink  
  
 return node  
  
  
# 搜尋右子樹替代節點  
def search\_re\_r(node):  
 re\_node = node  
 while re\_node != None and re\_node.llink != None:  
 re\_node = re\_node.llink  
 return re\_node  
  
  
# 搜尋左子樹替代節點  
def search\_re\_l(node):  
 re\_node = node  
 while re\_node != None and re\_node.rlink != None:  
 re\_node = re\_node.rlink  
 return re\_node  
  
  
# 搜尋node的父節點  
def search\_p(node):  
 global root  
 parent = root  
 while parent != None:  
 if node.id < parent.id:  
 if node.id == parent.llink.id:  
 return parent  
 else:  
 parent = parent.llink  
 elif node.id == parent.rlink.id:  
 return parent  
 else:  
 parent = parent.rlink  
 return None  
  
  
def main():  
 while True:  
 print()  
 print('\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*')  
 print(' <1> 增加學生資料 ')  
 print(' <2> 刪除學生資料 ')  
 print(' <3> 修改學生資料 ')  
 print(' <4> 顯示學生資料 ')  
 print(' <5> 根據成績排列學生資料 ')  
 print(' <6> 根據英文姓名排列學生資料 ')  
 print(' <7> 退出程式 ')  
 print('\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*')  
  
 try:  
 option = input('Enter your choice: ')  
 except ValueError:  
 print('Not a correct number.')  
 print('Try again\n')  
  
 if option == '1':  
 insert\_f()  
 elif option == '2':  
 delete\_f()  
 elif option == '3':  
 modify\_f()  
 elif option == '4':  
 show\_f()  
 inorder(root) # 以中序法輸出資料  
 elif option == '5':  
 show\_f()  
 order\_by\_score(root)  
 elif option == '6':  
 show\_f()  
 order\_by\_score(root)  
 elif option == '7':  
 sys.exit(0)  
 else:  
 print('Wrong option!')  
  
  
main()

: