Assignment 2 Solution

Shunbo Cui
cuis13

February 25, 2019

Introductory blurb.

1 Testing of the Original Program

The several test cases are implemented on the add, remove and sorting functions in SALst section and they passed. However when I test the allocating function there are some problems. There is always only 1 student can be allocated in the department. The reason for the problem is that there are problem in the AALst module. Originally I used one new tuple for each of the student in the AAlst, containing their macid and choice. So there will be many tuples for the same department in the list. However the expected result is that for every department there is just 1 tuple containing the name of department and macid of students in it. That's why when testing the allocating function, as soon as it find the tuple containing the name of the department, it will return the single macid in the tuple. I have fixed this problem and put the code in a new file called AALst_new.py. This took me long time to fix and the test case is not completed.

2 Results of Testing Partner's Code

The test cases are not completed. The Partner's file passes existing cases.

3 Critique of Given Design Specification

There is no natural language in the specification so it's hard to try to figure out what each function does when building them.

4 Answers

1. The advantage of natual language is easier for the programmer to understand the flow of the program. Its faster to implement the correct methods with natual language instructions. However the solution generated is not general. Every designer can have different approach to the solution. The formal specification is more general. Writer is highly constrained by the logic, the proofsystem, and the fact that every detail must be verified.

2. In the add_stdnt function, there will be an exception of ValueError for the students with gpa out of the range. The information type can still be Named tuple.

3. If two modules are similar, the similar functions can be put together for the programmer to use the same template of code. Then the difference can be stressed for modifying the template.

4. In A2 the problem is divided into several modules, including Department Capacity Association List, Allocation Association List and Student Association List. They all use the types defined in StdntAllocTypes. If we want to work on the student information we just need to call the key of Named tuple, instead of searching for the location it is stored. And when modifying the information lists there are built-in method like remove and add. We don't need to know the actual inner structure of the dataset. However in A1, the information is simply stored in dictionary. We need the detail of the data structure when implementing other functions.

5. There are specific operations in ADT. In this case there are next and end methods. It determines how the choices will be operated in the allocate function in advance. If it is a single list, we need to add operations on the list in the allocate function which will make it complicated.

6. The members in enum are stored as interger which makes the program more efficient. And the value of members cannot be modified in outside operations so the data is stable. Macid is the label of the student. It is seperated so other functions can find the student more easily.

E Code for StdntAllocTypes.py

```
## @file StdntAllocTypes.py
#  @author Shunbo Cui
#  @brief StdntAllocTypes
#  @date 11/2/2019
from enum import Enum
from collections import namedtuple


## @An abstract data type that represents gender type
class GenT(Enum):
    male = 0
    female = 1


## @An abstract data type that represents departments
class DeptT(Enum):
    civil = 0
    chemical = 1
    electrical = 2
    mechanical = 3
    software = 4
    materials = 5
    engphys = 6


SInfoT = namedtuple("SInfoT", ["fname", "lname", "gender", "gpa", "choices", "freechoice"])
```

F Code for SeqADT.py

```
## @file SeqADT.py
#  @author Shunbo Cui
#  @brief SeqADT
#  @date 11/2/2019


## @An abstract data type that represents a sequence
class SeqADT:
    def __init__(self, x):
        self.s = x
        self.i = 0

## @brief SeqADT constructor
    def start(self):
        self.i = 0

## @brief moves to next element in the sequence
#  @return the pointer of the location of the element
    def next(self):
        if self.i >= len(self.s):
            raise StopIteration(”Sequence length exceeded”)
        temp = self.s[self.i]
        self.i = self.i + 1
        return temp

    def end(self):
        return self.i >= len(self.s)


## @brief defining the exception
#  @return the value of the string
class StopIteration(Exception):
    def __init__(self, value):
        self.value = value

    def __str__(self):
        return str(self.value)
```

G Code for DCapALst.py

```
## @file DCapALst.py
#  @author Shunbo Cui
#  @brief DCapALst
#  @date 11/2/2019


## @An abstract data type that represents Department capacity type
class DCapALst:

    s = []

    ## @brief constructor of the data type
    @staticmethod
    def init():
        DCapALst.s = []

    ## @brief appending elements to the departments list
    @staticmethod
    def add(d, n):
        tup = (d, n)
        for tup1 in DCapALst.s:
            if d in tup1:
                raise KeyError(”tuple already in set”)
        DCapALst.s.append(tup)

    ## @brief deleting elements in the departments list
    @staticmethod
    def remove(d):
        for tup1 in DCapALst.s:
            if d not in tup1:
                continue
            raise KeyError(”tuple not in set”)
        for tup1 in DCapALst.s:
            if d in tup1:
                DCapALst.s.remove(tup1)

    ## @brief check if element in the list
    #  @return the boolean value representing if exists
    @staticmethod
    def elm(d):
        for tup1 in DCapALst.s:
            if d in tup1:
                return True
        return False

    ## @brief check the capacity of the department
    #  @return the interger of the capacity
    @staticmethod
    def capacity(d):
        for tup1 in DCapALst.s:
            if d == tup1[0]:
                return tup1[1]
        raise KeyError(”tuple not in set”)


## @brief defining the exception
#  @return the value of the string
class KeyError(Exception):
    def __init__(self, value):
        self.value = value

    def __str__(self):
        return str(self.value)
```

H Code for AALst.py

```
## @file AALst.py
#  @author Shunbo Cui
#  @brief AALst
#  @date 11/2/2019


## @An abstract data type that represents allocation association list
class AALst:

    s = []

    ## @brief constructor of the data type
    @staticmethod
    def init():
        AALst.s = []

    ## @brief appending elements to the association list
    @staticmethod
    def add_stdnt(dep, m):
        tup = (dep, m)
        AALst.s.append(tup)

    ## @brief get the list in the tuple
    #  @return the list related to input d
    @staticmethod
    def lst_alloc(d):
        for tup1 in AALst.s:
            if d in tup1:
                return(tup1[1])

    ## @brief get the length of list in the tuple
    #  @return the length of list related to input d
    @staticmethod
    def num_alloc(d):
        for tup1 in AALst.s:
            if d in tup1:
                return(len(tup1[1]))
        return 0
```

I Code for SALst.py

```
## @file SALst.py
#  @author Shunbo Cui
#  @brief SALst
#  @date 11/2/2019
from StdntAllocTypes import *
from AALst import *
from DCapALst import *


## @An abstract data type that do operations to the data
class SALst:

    s = []

    ## @brief constructor of the data type
    @staticmethod
    def init():
        SALst.s = []

    ## @brief appending elements to the student list
    @staticmethod
    def add(m, i):
        tup = (m, i)
        for tup1 in SALst.s:
            if m in tup1:
                raise KeyError(”tuple already in set”)
        SALst.s.append(tup)

    ## @brief removing elements in the student list
    @staticmethod
    def remove(m):
        for tup1 in SALst.s:
            if m not in tup1:
                continue
            raise KeyError(”tuple not in set”)
        for tup1 in SALst.s:
            if m in tup1:
                SALst.s.remove(tup1)

    ## @brief check if element in the list
    #  @return the boolean representing if the element exists
    @staticmethod
    def elm(m):
        for tup1 in SALst.s:
            if m in tup1[1]:
                return True
        return False

    ## @brief getting the information in the tuple
    #  @return the information of the student responding to the macid
    @staticmethod
    def info(m):
        for tup1 in SALst.s:
            if m in tup1:
                return tup1[1]
        raise ValueError(”tuple not in set”)

    ## @brief sorting the student list
    #  @return the sorted list of students
    @staticmethod
    def sort(f):
        L1 = []
        Lm = []
        for tup1 in SALst.s:
            if f(tup1[1]):
                L1.append(tup1)
        L2 = sorted(L1, key=lambda x: (x[1].gpa), reverse=True)
        for tup2 in L2:
            Lm.append(tup2[0])
        return Lm

    ## @brief calculating the average gpa in the list
    #  @return the float of the average gpa value
    @staticmethod
    def average(f):
        L3 = []
        sum = 0
        count = 0
        for tup1 in SALst.s:
            if f:
                print(f)
                L3.append(tup1)
        if L3 == []:
            raise ValueError(”List is empty”)
        for tup2 in L3:
            sum += tup2[1].gpa
            count += 1
        average = sum / count
        return average

    ## @brief allocate the students to departments
    @staticmethod
    def allocate():
        AALst.init()
        F = SALst.sort(lambda t: t.freechoice and t.gpa >= 4.0)
        for m in F:
            ch = SALst.info(m).choices
            AALst.add_stdnt(ch.next(), m)
        S = SALst.sort(lambda t: (not (t.freechoice)) and t.gpa >= 4.0)
        for m in S:
            ch = SALst.info(m).choices
            alloc = False
            while (not alloc) and (not ch.end()):
                d = ch.next()
                if AALst.num_alloc(d) < DCapALst.capacity(d):
                    AALst.add_stdnt(d, m)
                    alloc = True
            if not alloc:
                raise RuntimeError(”Run time error”)


## @brief defining the exception
#  @return the value of the string
class ValueError(Exception):
    def __init__(self, value):
        self.value = value

    def __str__(self):
        return str(self.value)


## @brief defining the exception
#  @return the value of the string
class RuntimeError(Exception):
    def __init__(self, value):
        self.value = value

    def __str__(self):
        return str(self.value)
```

J Code for Read.py

```
from StdntAllocTypes import *
from SALst import *
from DCapALst import *
from SeqADT import *


def load_stdnt_data(s):
    f = open(s, "r", -1, "utf-8-sig")
    SALst.init()
    for line in f.readlines():
        v = line.strip().split(',')
        if v[3] == "male":
            gen = GenT.male
        else:
            gen = GenT.female
        sinfo1 = SInfoT(v[1], v[2], gen, float(v[4]), SeqADT(v[5]), v[6] == str(True))
        SALst.add(v[0], sinfo1)
    f.close()


def load_dcap_data(s):
    f = open(s, "r", -1, "utf-8-sig")
    DCapALst.init()
    for line in f.readlines():
        v = line.strip().split(',')
        if v[0] == "civil":
            dept = DeptT.civil
        elif v[0] == "chemical":
            dept = DeptT.chemical
        elif v[0] == "electrical":
            dept = DeptT.electrical
        elif v[0] == "mechanical":
            dept = DeptT.mechanical
        elif v[0] == "software":
            dept = DeptT.software
        elif v[0] == "materials":
            dept = DeptT.materials
        elif v[0] == "engphys":
            dept = DeptT.engphys
        SALst.add(dept, v[1])
    f.close()
```

K Code for test_All.py

```
from StdntAllocTypes import *
from SeqADT import *
from DCapALst import *
from AALst import *
from SALst import *

from pytest import *

class TestSALst:
    def SALsttest1(self):
        SALst.init()
        sinfo1 = SInfoT(”first”, ”last”, GenT.male, 12.0, SeqADT([DeptT.civil, DeptT.chemical]), True)
        SALst.add(”stdnt1”, sinfo1)
        assert SALst.elm(”stdnt1”) == True

    def SALsttest2(self):
        SALst.init()
        sinfo1 = SInfoT(”first”, ”last”, GenT.male, 12.0, SeqADT([DeptT.civil, DeptT.chemical]), True)
        SALst.add(”stdnt1”, sinfo1)
        SALst.remove(”stdnt1”)
        assert SALst.elm(”stdnt1”) == False

    def SALsttest3(self):
        SALst.init()
        sinfo1 = SInfoT(”first”, ”last”, GenT.male, 12.0, SeqADT([DeptT.civil, DeptT.chemical]), True)
        SALst.add(”stdnt1”, sinfo1)
        assert SALst.info(”stdnt1”) == SInfoT(fname=’first’, lname=’last’, gender=<GenT.male: 0>,
            gpa=12.0, choices=<SeqADT.SeqADT object at 0x000002AC29F1CC88>, freechoice=True)

    def SALsttest4(self):
        SALst.init()
        sinfo1 = SInfoT(”first”, ”last”, GenT.male, 12.0, SeqADT([DeptT.civil, DeptT.chemical]), True)
        sinfo2 = SInfoT(”first2”, ”last2”, GenT.female, 8.0, SeqADT([DeptT.civil, DeptT.chemical]),
            True)
        assert SALst.sort(lambda t: t.freechoice and t.gpa >= 4.0) == [’stdnt1’, ’stdnt2’]
```

L Code for AALst_new.py

```
## @file AALst.py
#  @author Shunbo Cui
#  @brief AALst
#  @date 11/2/2019
from StdntAllocTypes import *


## @An abstract data type that represents allocation association list
class AALst:

    s = {}

    ## @brief constructor of the data type
    @staticmethod
    def init():
        AALst.s = {
        DeptT.civil: [],
        DeptT.chemical: [],
        DeptT.electrical: [],
        DeptT.mechanical: [],
        DeptT.software: [],
        DeptT.materials: [],
        DeptT.engphys: []}

    ## @brief appending elements to the association list
    @staticmethod
    def add_stdnt(dep, m):
        tup = (dep, m)
        AALst.s[dep].append(tup)

    ## @brief get the list in the tuple
    #  @return the list related to input d
    @staticmethod
    def lst_alloc(d):
        return AALst.s[d]

    ## @brief get the length of list in the tuple
    #  @return the length of list related to input d
    @staticmethod
    def num_alloc(d):
        return len(AALst.s[d])
```

M Code for Partner's SeqADT.py

```
##  @file SeqADT.py
#   @author Mengxi (William) Lei, leim5
#   @date Created 2019/02/05
#   @date Last modified 2019/02/09
#   @brief ADT for a sequence (list)


##  @brief Takes a sequence and manipulate it
class SeqADT:

    ##  @brief Construct the object, set count to 0 and sequence to input
    #   @param seq sequence of the type of the class
    #   @return object of SeqADT
    def __init__(self, seq):
        self.count = 0
        self.sequence = seq

    ##  @brief Reset the count to 0
    def start(self):
        self.count = 0

    ##  @brief Return the next item in the sequence
    #   @return next item in the sequence
    #   @throws StopIteration Reached the end of the array, no more item in the sequence
    def next(self):
        if (self.end()):
            raise StopIteration
        else:
            self.count += 1
            return self.sequence[self.count - 1]

    ##  @brief Check if at the end of the sequence
    #   @return boolean state whether the end of the sequence have been reached
    def end(self):
        if (self.count >= len(self.sequence)):
            return True
        else:
            return False
```

N Code for Partner's DCapALst.py

```
##  @file DCapALst.py
#   @author Mengxi (William) Lei, leim5
#   @date Created 2019/02/05
#   @date Last modified 2019/02/09
#   @brief Class for the department capacity


##  @brief Holds a dictionary of the department capacity and manipulate it
class DCapALst:

    seq = None

    ##  @brief Construct the object, set the dictionary to a empty one
    @staticmethod
    def init():
        DCapALst.seq = {}

    ##  @brief Add the department capacity information to dictionary
    #   @param department department being added
    #   @param size capacity of the department
    #   @throws KeyError Department already in dictionary
    @staticmethod
    def add(department, size):
        if DCapALst.elm(department):
            raise KeyError
        else:
            DCapALst.seq[department] = size

    ##  @brief Delete the department capacity information from dictionary
    #   @param department department being deleted
    #   @throws KeyError Department not in dictionary
    @staticmethod
    def remove(department):
        if DCapALst.elm(department):
            del DCapALst.seq[department]
        else:
            raise KeyError

    ##  @brief Check if the element is inside the current dictionary
    #   @param department department being checked
    #   @return boolean state if the department is in the dictionary or not
    @staticmethod
    def elm(department):
        if department in DCapALst.seq:
            return True
        else:
            return False

    ##  @brief Check the capacity of the given department
    #   @param department department being checked
    #   @return the capacity of the department
    #   @throws KeyError the department is not in the dictionary
    @staticmethod
    def capacity(department):
        if DCapALst.elm(department):
            return DCapALst.seq[department]
        else:
            raise KeyError
```

O Code for Partner's SALst.py

```
## @file SALst.py
#  @author Mengxi (William) Lei, leim5
#  @date Created 2019/02/07
#  @date Last modified 2019/02/09
#  @brief Class for the allocation of student

from operator import itemgetter
from StdntAllocTypes import *
from AALst import *
from DCapALst import *


## @brief Class for the allocation of student
class SALst:

    seq = None

    ## @brief Construct the object, initialize a empty dictionary
    @staticmethod
    def init():
        SALst.seq = {}

    ## @brief Add the student to the dictionary
    #  @param macid student's macid
    #  @param info the student's information
    #  @throws KeyError student with the macid already in dictionary
    @staticmethod
    def add(macid, info):
        if SALst.elm(macid):
            raise KeyError
        else:
            SALst.seq[macid] = info

    ## @brief Remove the student from the dictionary
    #  @param macid macid of student being removed
    #  @throws KeyError macid not in dictionary
    @staticmethod
    def remove(macid):
        if SALst.elm(macid):
            del SALst.seq[macid]
        else:
            raise KeyError

    ## @brief Check if the element is inside the current dictionary
    #  @param macid macid being checked
    #  @return boolean state if the macid is in dictionary or not
    @staticmethod
    def elm(macid):
        if macid in SALst.seq:
            return True
        else:
            return False

    ## @brief Retrieve the information of the student
    #  @param macid macid of student whose information is being retrieved
    #  @return the information of the student
    #  @throws ValueError macid not in dictionary
    @staticmethod
    def info(macid):
        if SALst.elm(macid):
            return SALst.seq[macid]
        else:
            raise ValueError

    ## @brief Sort the students who satisfies the given condition is decreasing gpa
    #  @param condition condition of which students are being sorted
    #  @return list of student's macid sorted by gpa (decreasing order)
    @staticmethod
    def sort(condition):
        id_list = SALst.seq.keys()
        info_list = []
        list = []
        for element in id_list:
            if condition(SALst.seq[element]):
                info_list.append({”id”: element, ”grade”: SALst.seq[element].gpa})
        info_list.sort(key=itemgetter(”grade”), reverse=True)
        for item in info_list:
            list.append(item["id"])
        return list

    ## @brief Return the average of the students that satisfies the given condition
    #  @param condition condition of which students are being used for calculation
    #  @return the average of the students
    #  @throws ValueError no student meet the condition
    @staticmethod
    def average(condition):
        id_list = SALst.seq.keys()
        grade = 0
        count = 0
        for element in id_list:
            if condition(SALst.seq[element]):
                grade += SALst.seq[element].gpa
                count += 1
        if count == 0:
            raise ValueError
        else:
            return (grade / count)

    ## @brief Allocate students depend on freechoice, gpa and their choices
    #  @throws RuntimeError Student not allocated
    @staticmethod
    def allocate():
        AALst.init()
        sorted_student = SALst.sort(lambda t: t.freechoice and t.gpa >= 4.0)
        for element in sorted_student:
            choice_list = SALst.seq[element].choices
            AALst.add_stdnt(choice_list.next(), element)
        sorted_student = SALst.sort(lambda t: (not t.freechoice) and t.gpa >= 4.0)
        for element in sorted_student:
            choice_list = SALst.seq[element].choices
            allocated = False
            while (not allocated) and (not choice_list.end()):
                choice = choice_list.next()
                if AALst.num_alloc(choice) < DCapALst.capacity(choice):
                    AALst.add_stdnt(choice, element)
                    allocated = True
            if not allocated:
                raise RuntimeError
```