Assignment 2 Solution

James Zhang zhany111

February 25, 2019

In this assignment we are going to solve a preblem similar to A1, that we want to allocate the first year enigineering students into their respective second year programs. We wil use doxygen, make, LaTeX and Python that are same as A1. Also we are going to use pytest and flake8 to test and check code.

1 Testing of the Original Program

I find that the sort(f), average(f) and allocate() functions from SALst() are not working. I had trouble with translating the access routine semantics the lambda function. I understand it now by reading my partner's code

2 Results of Testing Partner's Code

My partner has excellent code.

3 Critique of Given Design Specification

I like that in this assignment every module(ADP) are connected together. It feels like we are doing an actual computer science project instead of a simple assignment. I think that one thing that the assignment can be improved on is that the class DCapALst and SALst are very similar in designing. They both have add(), remove() and elm() methods. Maybe we can define a parent class like a list of students and departments, then define the children classes DCapALst and SALst using the parent class. It saves time defining the similar methods.

4 Discussion

By doing the exercise, I learned that by representing different variables as new ADP instead of list or dictionary, the whole model will be more organized and easier for us to test the code. By using pytest, it is more efficient for us to find the error. Pytest can be very helpful when test on the complex code including different methods and functions such as the code for this assignment.

5 Answers

1. For A1, the advantage is that it gives us the freedom to construct the project by our own thoughts. We can decide how each function works to output the result we want. However, in A1 everybody has completely different code, it gives the difficulty to read and understand the other's code. For A2, the advantage is we do not need to spend time on designing the modules, since we just need to translate the semantics from the instruction to python code. However, we did not practice on designing our own program in this assignment.

2. I can change the assumption to an exception by: if gpa ¡ 0.0 or gpa ¿ 12.0: raise keyError. We do not need to modify the specification to replace a record type with a new ADT because an abstract object can raise error.

3. As I said in question 7), the possible way to modify the project is define a new list type to store department and student as the parent class. Then define the DCapALst and SALst using the parent class.

4. A2 is more general than A1 because it takes objects as abstract objects or ADPs other than regular list or dictionary in A1. Also, in A1, each student must have three choices of department. However, in A2, each student can have any number of choices of department that is greater than 0.

5. Using SeqADP is better than using a regular list because regular list is mutable, which means the elements in it can be changed. However, for SeqADP, there is only add() method but no any removing or deleting method, which makes SeqADP an immutable type, that we can only add new element into it, but we cannot change any element after we create it. SeqADP makes the information storing much safer. We can never ruin the information stored in SeqADP by any mistake.

6. Enums have the advantage that it is easier to type. We do not need to type the name of the department every time we input the value, because using Enums, the

name of the department can be represented by numbers. Student's macid cannot be represented as Enums because every student has a unique macid. It is impossible to enum them all.

F Code for StdntAllocTypes.py

```
## @file StdntAllocTypes.py
#  @author James Zhang
#  @brief StdntAllocTypes
#  @date 08/02/2019

from SeqADT import *
import typing


## @brief A abstract data type that represents gender
class GenT:
    def male(self):
        return self
    def female(self):
        return self


## @brief A abstract data type that represents differert departments
class DeptT:
    ## @brief DeptT constructor
    #  @details create the list containing the different departments
    def __init__(self):
        self.l = [DeptT.civil, DeptT.chemical, DeptT.electrical, DeptT.mechanical, DeptT.software,
            DeptT.materials, DeptT.engphys]

        self.civil = 0
        self.chemical = 1
        self.electrical = 2
        self.mechanical = 3
        self.software = 4
        self.materials = 5
        self.engphys = 6


## @brief A abstract data type that represents student's info
class SInfoT:
    ## @brief SInfoT constructor
    #  @details Take t as the student, generate the student info
    #  @param t NamedTuple of student's info
    def __init__(self, t):
        max_gpa = 12.0
        min_gpa = 0.0
        self.t = t
        if self.t.gpa > max_gpa or self.t.gpa < min_gpa:
            raise IOError


## @brief A NamedTuple that stores student's info
class student(typing.NamedTuple):
    fname: str
    lname: str
    gender: GenT
    gpa: float
    choices: SeqADT
    freechoice: bool
```

G Code for SeqADT.py

```
## @file SeqADT.py
#  @author James Zhang
#  @brief SeqADT
#  @date 08/02/2019


## @brief A  generic module that represents a sequence
class SeqADT:

    s = []
    i = 0

    ## @brief SeqADT constructor
    #  @details take l as the initial set, set i to 0
    #  @param l sequence of any type
    def __init__(self, l):
        self.s = l
        self.i = 0

    ## @brief start the sequence
    #  @details set i to 0
    def start(self):
        self.i = 0

    ## @brief pop the seq
    #  @details increment i and output s[i]
    def next(self):
        if not self.end():
            self.i += 1
            return self.s[self.i]
        else:
            raise StopIteration

    ## @brief check if reach the end
    #  @details comparing the len(s) and i, check if there is anything left
    def end(self):
        return self.i >= len(self.s)
```

H Code for DCapALst.py

```
## @file SeqADT.py
#  @author James Zhang
#  @brief SeqADT
#  @date 08/02/2019

from StdntAllocTypes import *


## @brief A abstract data type for departments and their capacity
class DCapALst:

    s = {}

    ## @brief DCapALst constructor
    #  @details create DCapAlst by set s to empty dictionary
    def init():
        DCapALst.s = {}

    ## @brief add n to d
    #  @details add n to dictionary d
    #  @param d the key
    #  @param n the element
    def add(self, d, n):
        if d not in self.s.keys():
            self.s[d] = n
        else:
            raise KeyError

    ## @brief delete d
    #  @details delete everything about key d
    #  @param d the key
    def remove(self, d):
        del self.s[d]

    ## @brief output the elements of d
    #  @param d the key
    def elm(self, d):
        return d, self.s[d]

    ## @brief output the length of the element of d
    #  @param d the key
    def capacity(self, d):
        if d in self.s.keys():
            return len(self.s[d])
        else:
            raise KeyError
```

I Code for AALst.py

```
## @file SeqADT.py
#  @author James Zhang
#  @brief SeqADT
#  @date 08/02/2019

from StdntAllocTypes import *

## @brief A NamedTuple representing the different lists of
class AllocAssocListT(typing.NamedTuple):
    DeptT.civil: list
    DeptT.chemical: list
    DeptT.electrical: list
    DeptT.mechanical: list
    DeptT.software: list
    DeptT.materials: list
    DeptT.engphys: list

## @brief class that allocate student to different department
class AALst:
    ## @brief SeqADT constructor
    #  @details take l as the initial set, set i to 0
    #  @param l sequence of any type
    def init(self):
        self.s = AllocAssocListT([],[],[],[],[],[],[])

    ## @brief add student to department
    #  @details add the student to the list according to the input
    #  @param dep the department
    #  @param m the student's id
    def add_stdnt(self, dep, m):
        self.s[dep] += [m]

    ## @brief output the element from d
    #  @param d the key
    def lst_alloc(self, d):
        return self.s[d]

    ## @brief output the length of element of d
    #  @param d the key
    def num_alloc(self, d):
        return len(self.s[d])
```

J Code for SALst.py

```
## @file SeqADT.py
#  @author James Zhang
#  @brief SeqADT
#  @date 08/02/2019

from StdntAllocTypes import *
from AALst import *
from DCapALst import *

## @brief A NamedTuple representing the different lists of
class studentT(typing.NamedTuple):
    macid: str
    info: SInfoT

## @brief get the gpa of student
#  @detail
#  @param m macid of student
#  @param s set of students
def get_gpa(m, s):
    return True

## @The class that associate with students
class SALst():

    ## @brief SAlst constructor
    #  @details set s to empty dictionary
    def init():
        SALst.s = {}

    ## @brief add i to s[m]
    #  @param m the key
    #  @param i the element
    def add(self,m,i):
        if m in self.s.keys():
            raise KeyError
        else:
            self.s[m] = i

    ## @brief remove s[m]
    #  @param m the key
    def remove(self, m):
        if m in self.s.keys():
            del self.s[m]
        else:
            raise KeyError

    ## @brief check if m is an key in s
    #  @param m the key
    def elm(self, m):
        return m in self.s.keys()

    ## @brief output the element from m
    #  @param m the key
    def info(self, m):
        if m in self.s.keys():
            return self.s[m]
        else:
            raise KeyError

    ## @brief output the sorted students list by their gpa
    #  @param f function representing the student we want
    def sort(self, f):
        return True

    ## @brief output the average of student's gpa
    #  @param m the key
    def average(f):
        return 's'

    ## @brief output the element from m
    def allocate():
        return 's'
```

K Code for Read.py

```
## @file Read.py
#  @author James Zhang
#  @brief Read
#  @date 08/02/2019

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


def load_stdnt_data(s):
    with open(s, 'r') as infile:
        contents = infile.readlines()

    contents = [x.strip().split(',') for x in contents]
    SALst.init()
    for i in range(0, len(contents)):

        ok = contents[i]



def load_dcap_data(s):
    with open(s, 'r') as infile:
        contents = infile.readlines()

    contents = [x.strip().split(',') for x in contents]

    a = []
    b = []
    for i in range(0, len(contents)):
        ok = contents[i]
        a += [ok[0]]
        b += [int(ok[1])]

    print(a,b)
    DCapALst.init()
    for j in range(0, len(a)):
        DCapALst.add(DCapALst, a[j], b[j])



load_dcap_data('DeptCap.txt')
```

L Code for Partner's SeqADT.py

```
## @file SeqADT.py
#  @author janzej2
#  @brief Implementation of an abstract data type for a sequence.
#  @date 02/11/19


## @brief This class represents an abstract data type of type sequence.
#  @details This class represents a sequence initialised with a list s and a count
#  variable i.
class SeqADT:

    ## @brief Constructor for SeqADT
    #  @details Constructor accepts one parameter consisting of a list of a type T
    #   and initialises its parameter i to 0.
    #  @param s List containing a specific data type.
    def __init__(self, s):
        self.__s = s
        self.__i = 0

    ## @brief Initialiser for i variable, resetting it to its original 0 value.
    def start(self):
        self.__i = 0

    ## @brief Function to iterate through the sequence and return the most recent value.
    #  @return Returns the i-th element before it is incremented.
    def next(self):
        if self.end():
            raise StopIteration
        self.__i += 1
        return self.__s[self.__i - 1]

    ## @brief Function to determine if the end of the sequence has been reached.
    #  @return Returns True if the function has reached the end, or False otherwise.
    def end(self):
        if self.__i >= len(self.__s):
            return True
        return False
```

M Code for Partner's DCapALst.py

```
## @file DCapALst.py
#  @author janzej2
#  @brief A module which defines the abstract data type for various departments
#  @date 02/10/19

# from StdntAllocTypes import *


## @brief This class represents an abstract data type of department capacities.
#  @details Initialising this object creates a new tuple used to store departments
#  and their relevant capacities.
class DCapALst:
    s = []
    ## @brief Initialises an empty tuple.
    @staticmethod
    def init():
        DCapALst.s = []

    ## @brief Adds a department and its capacity to the DCapALst.
    #  @param d Represents a department of type DeptT.
    #  @param n Represents the capacity of that department.
    @staticmethod
    def add(d, n):
        for dept in DCapALst.s:
            if dept[0] == d:
                raise KeyError
        DCapALst.s.append([d, n])

    ## @brief Removes a given department from the DCapALst.
    #  @param d Represents the DeptT to remove from the list.
    @staticmethod
    def remove(d):
        removed = False
        for dept in DCapALst.s:
            if dept[0] == d:
                DCapALst.s.remove(dept)
                removed = True
        if not removed:
            raise KeyError

    ## @brief Checks if a given department is already in the list.
    #  @param d Represents the DeptT to check for in the list.
    #  @return Returns a boolean value representing whether the
    #   department is in the list.
    @staticmethod
    def elm(d):
        for dept in DCapALst.s:
            if dept[0] == d:
                return True
        return False

    ## @brief Checks the capacity of a given department.
    #  @param d Represents the DeptT to check.
    #  @return Returns the capacity of the given department if
    #   it is in the list.
    @staticmethod
    def capacity(d):
        for dept in DCapALst.s:
            if dept[0] == d:
                return dept[1]
        raise KeyError
```

N Code for Partner's SALst.py

```
## @file SALst.py
#  @author janzej2
#  @brief A module which defines the behvaiour of the Student Association List.
#  @date 02/11/19
#  shared ideas with Ahmed Al Koasmh and David Thompson

from StdntAllocTypes import *
from AALst import *
from DCapALst import *


## @brief A class representing the data type SALst.
#  @details This module contains function for initialising a new list,
#  adding a student to the list (given their MacID and student information),
#  removing a student from the list, determining if a student is in the list
#  by MacID, returning a student's student information by MacID, sorting a
#  group of students by a given f, taking the average of a set of students
#  based on a condition f, and a function to allocate students to their programs.
class SALst:

    s = []

    ## @brief Initialises the empty student allocation list.
    def init():
        SALst.s = []

    ## @brief Adds a student to the student allocation list.
    #  @param m A string representing the student's MacID.
    #  @param i A SInfoT object representing a student's info.
    @staticmethod
    def add(m, i):
        for element in SALst.s:
            if element[0] == m:
                raise KeyError
        SALst.s.append([m, i])

    ## @brief Removes a student from the student allocation list.
    #  @param m A string representing a student's MacID.
    @staticmethod
    def remove(m):
        removed = False
        for element in SALst.s:
            if element[0] == m:
                SALst.s.remove(element)
                removed = True
        if not removed:
            raise KeyError

    ## @brief Determines if a student is an element of the list.
    #  @param m A string representing a student's MacID.
    #  @return True if the student is in the list, and false otherwise.
    @staticmethod
    def elm(m):
        for element in SALst.s:
            if element[0] == m:
                return True
        return False

    ## @brief Checks a student's MacID and returns their student information.
    #  @param m A string representing a student's MacID.
    #  @return The student's information as type SInfoT.
    @staticmethod
    def info(m):
        for element in SALst.s:
            if element[0] == m:
                return element[1]
        raise KeyError

    ## @brief Sorts the SALst based on parameters given through f.
    #  @param f Represents a function to determine the method of sorting
    #  @return Returns a list of MacIDs ordered based on the input condition.
    @staticmethod
    def sort(f):
        sorted_list = sorted(SALst.s, key=lambda i: i[1].gpa, reverse=True)
        return [item[0] for item in sorted_list if f(item[1])]
```

```
## @brief Returns the average of a group of students based on the input criteria.
#  @param f Represents a function to filter based on.
#  @return Returns the average of the students that match the given criteria.
@staticmethod
def average(f):
    avg_list = [item[1].gpa for item in SALst.s if f(item[1])]
    if len(avg_list) == 0:
        raise ValueError
    sum = 0
    for item in avg_list:
        sum += item
    return sum / len(avg_list)

## @brief Allocates students based on their student information.
#  @details Creates a list of freechoice students first, allocating
#  them to their first choice program. Then, the function sorts the student
#  list by GPAs >= 4.0, and allocates them to their first, second or third
#  choice programs based on remaining space.
@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 l: not l.freechoice and l.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
```