Computer Science and Creative Technologies

Coursework or Assessment Specification

Module Details

Module Code	UFCFW4-30-2
Module Title	Design and Analysis of Data Structures & Algorithms
Module Leader	Dr Gordon Downie & Dr Elias Pimenidis
Module Tutors	Dr Kun Wei & Mr. James Lear
Year	2020-21
Component/Element Number	B / 1
Total number of assessments for this module	2
Weighting	40%
Element Description	Coursework A

Dates

Date issued to students	30 November 2020
Date to be returned to students	
Submission Date	11 February 2021
Submission Place	Blackboard - Online
Submission Time	14:00
Submission Notes	Please submit all files in a zipped folder **clearly labelled "DADSA2021A – Your Name"**

Feedback

Feedback provision will be	Formative / Verbal, during practical sessions. Written on return of the marked work.

Contents

Section 1: Overview of Assessment

This assignment assesses the following module learning outcomes:
Analyse requirements and select appropriate solutions
Design programmes that use appropriate data structures.
Implement data structures and the algorithms that maintain them, allowing for secure processing of the data.

The assignment is worth ***20%*** of the overall mark for the module.
Broadly speaking, the assignment requires you to select and implement appropriate data structures, design and implement algorithms and create the relevant software applications that will allow a user to store, update and manipulate the data relating to the operations of an animal sanctuary.
The assignment is described in more detail in section 2.
This is an individual assignment.
Working on this assignment will help you to develop the ability to relate organisational requirements into design choices for the storage and management of data. It will also help you practice your programming skills on Python and explore means by which the efficiency of your programming can improve.
If you have questions about this assignment, please ask your practical session tutor for clarifications.
Python libraries should not be used in creating code, neither should built-in data structure and algorithms templates should be utilised. The only readily available code you can reuse and customise is that for importing and reading a CSV file.
You can build your own data structures using the Python default structures as building blocks – see example of building a linked list using the default Python array (week 3 TB1 practical worksheet).

Section 2: Task Specification

Shopping for the neighbours
It is the Covid-19 era in the spring of 2020, in a neighbourhood in Bristol Danny and Carla, a young couple, that have found time in their hands during the lockdown, have decided to help their elderly neighbours with their shopping while they are shielding from the virus. There 15 different households that they have decided to help and their elderly neighbours have responded with enthusiasm.
By week 2 of their new endeavours the young couple have realised that picking up lists (often vague) and delivering groceries for others is not an easy chore. This has become increasingly more challenging as after the first few days the "clients" have become more daring. Despite being extremely grateful to Danny and Carla, the neighbours have started adding preferences and constraints with various preferences as to quality and range of products. This has made organising the shopping much more complex that our young volunteers have imagined. Carla, the more practical minded one, has thought that all the little notes placed in front of their doors by their neighbours contain data. If the data can be organized and stored somehow, then they can think of how they can use it to make their shopping forages much more efficient.

Some of the key things that they want to control are

- The number of trips per shop per week
- The number of different times they have to buy the same product per week
- The number of deliveries they need to make per household per week
- The cost of each shopping for each household per week
- The petrol cost for themselves per week

Note: Danny and Carla are not charging their neighbours anything for the deliveries
At the end of the second week Carla has accumulated the data attached in the following files. Now she wants your help to keep the following at a minimum

- Number of trips per shop
- Number of different times they have to buy each item.

NOTE that during the first two weeks there have only been 7 households that took advantage of their help, but the number is set to increase to the whole 15.
Carla who manages the whole voluntary enterprise has given us her data in the attached CSV files. She needs your support in organising and managing the shopping trips to minimise them.

- One shop per store,
- Visit one store per day in a week
- Deliver to each household once a week
- Do not keep shopping in her house more than one day

Task 1 – Design (pseudocode & diagrams) and implement software in Python to deliver the following results.

- Produce a shopping schedule – shop to be visited on each day & items to be bought from each shop on the day.
- Produce a delivery schedule – households per day. An example of such output on the screen has been demonstrated and explained during the coursework spec presentation (week starting 30 November 2020). Please view the relevant video for your group. A copy of the example has been uploaded on BB along with this specification document.
- While producing the schedules observe the following constraints in the order of preference given below:
  - Visit one store per day (note that you can make more than one shopping trips to the same store during the 7-day week window).
  - Do not keep shopping in the house more than one day
  - Deliver to each household once a week
  - Your algorithm should attempt to meet all the constraints. If these cannot be met then the algorithm should explore the alternative of substituting items for one of the same kind. This should allow the shopping for each household to be completed from two stores only and therefore allow you to meet all the constraints above. You can define your rules according to which substitutions (among the given data) will be identified and made within your algorithm.

Use the data in the two CSV files labelled week 1 to test and demonstrate your program.

New customers and a new store

By week three Carla has been receiving a lot more orders as the remaining eight households have joined the initial seven and now the number of Carla's and Danny's "customers" stand at 15.
With the increase of households comes an increase in the types of products in demand. Finally as pensioners are feeling the stretch of the cost of mass shopping and they are starting to exhaust their previous supplies, they are looking for cheaper alternatives.
A little further than the other three stores, there is a fourth store called the "cheap store".
At least a third of the 15 households have shown preference to some of their products…
This will add at least one more trip to the stores per week for Danny and a further puzzle to solve for Carla.
Desperate with the new development Carla has created some new tables and has asked us to help with her planning of the shopping and deliveries for the next two weeks.
Note that Carla and Danny have decided to do all their shopping and deliveries Monday to Thursday and keep Friday – Sunday to themselves.

Task 2 – Design (pseudocode & diagrams) and implement software in Python to deliver the following results.

- Produce a shopping schedule – shop to be visited on each day & items to be bought from each shop on the day.
- Produce a delivery schedule – households per day. An example of such output on the screen has been demonstrated and explained during the coursework spec presentation (week starting 30 November 2020). Please view the relevant video for your group. A copy of the example has been uploaded on BB along with this specification document.
- While producing the schedules observe the following constraints in the order of preference given below:
    - Visit one store per day
    - Do not keep shopping in the house more than one day
    - Deliver to each household once a week
    - Shopping and delivery will take place Monday to Thursday only
    - Your algorithm should attempt to meet all the constraints. If these cannot be met then the algorithm should explore the alternative of substituting items for one of the same kind. This should allow the shopping for each household to be completed from two stores only and therefore allow you to meet all the constraints above. You can define your rules according to which substitutions (among the given data) will be identified and made within your algorithm.
- Use the data in the two CSV files labeled week 4 to test and demonstrate your program.

Carla and Danny get some help
As things have become quite desperate with Carla and Danny struggling and sometimes missing their targets in fulfilling all the orders, they asked around for help.
Carla's mum has lent them her husband's old minivan, Danny's boss has lent them a spare freezer, and one of Danny's mates has lent them a spare fridge.
Danny has managed to fit both the spare fridge, and freezer in his shed at the back of the house.
Now they can keep goods for more than one day, they can do two shop visits per day and split the deliveries between the two of them. This has put a smile back in Carla's previously worried face.

Task 3 Design (pseudocode & diagrams) and implement software in Python to deliver the following results.
Explain the differences in your design and choice of data structures and algorithms for each of the three phases of the development. Justify what improvements in efficiency and effectiveness each choice has contributed. (400 words maximum).

- Produce a shopping schedule – shop to be visited on each day & items to be bought from each shop on the day.
- Produce a delivery schedule – households per day. An example of such output on the screen has been demonstrated and explained during the coursework spec presentation (week starting 30 November 2020). Please view the relevant video for your group. A copy of the example has been uploaded on BB along with this specification document.
- While producing the schedules observe the following constraints in the order of preference given below:
    - Carla and Danny to visit one store per day maximum, each – More than one visits to each store during the week are possible now. Also sopping items can be kept in the house for longer than one day. Thus compiling a shopping list from more than two stores is possible.
    - Deliver to each household once a week
    - Shopping and delivery will take place Monday to Thursday only
    - Your algorithm should attempt to meet all the constraints. If these cannot be met (unlikely for this task) then the algorithm should explore the alternative of substituting items for one of the same kind. This should allow the shopping for each household to be completed from two stores only and therefore allow you to meet all the constraints above. You can define your rules according to which substitutions (among the given data) will be identified and made within your algorithm.
- Use the data in the two CSV files labelled week 7 to test and demonstrate your program.

Section 3: Deliverables

One folder in zip format (only) must be uploaded via the relevant link on the module's space on Blackboard. The link will be available two weeks before the due date and will be communicated to students via an email announcement.
You can reuse code and design that you have produced for one of the tasks here to fulfil the requirements of another task.
The folder must contain:

- The software modules in Python clearly labelled.
- A word / text file that provides clear instructions as to how to run the program – This is particularly important where more than one Python files have been submitted.
- All the data files packaged in a way that the software will run and access them without any problems.
- A short video demonstrating your software running and fulfilling each of the tasks in the correct order.
- A word / PDF file with your justification of the design choices.
- A word /PDF file with your pseudocode

- Design diagrams – these should be accessible through software available at UWE.

All program files saved in Python format (3.6 - 3.8) – any other version will not be accepted and the work will not be marked resulting in a 0 (zero) mark for this coursework.
NOTE 1: You are advised to develop and present your software as three separate solutions, even if this means that you'll end up reusing parts of the solutions from previous tasks. If you choose to deliver one piece of software with a menu of choices of running tasks and this fails to run for whatever reason, the marker will NOT try to edit your code to see which parts actually work.

Section 4: Marking Criteria

The following table (please see next page) gives details of the marking criteria for this coursework.
Marks will be awarded for clear rationale justifying design choices.
Clarity in the pseudo code submitted allowing to map the full logic of the solution implemented is expected.
Code must be well structured, appropriately commented, neat and efficient. Clear use of functions and reduced repetitions of blocks of code are expected.
The use of GUI or other user interface will not attract any specific marks, but simplicity and efficiency of its design will be considered when awarding for an overall efficient system developed.

NOTE – No hard coded data will be allowed. Hard coded data in the submitted work will result in the work marked at 0 (zero).

Section 5: Feedback mechanisms

Formative / Verbal will be provided during practical sessions.
Written feedback will be provided on blackboard along with the marked for the submitted work on 17 March 2021.

Marking Criteria Table

	0-29	30-39	40-49	50-59	60-69	70-84	85-100	Mark & Advice for Improvement
Task 1 **40%**	Pseudo code lacks clarity / is incomplete. The program does not deliver a complete solution.	The program works but the constraints are not met in the order they have been specified. Pseudo code and design diagrams show that about half the required features are met.	Pseudo code addresses 51 to 60% of required features. The program runs observing up to three constraints.	Pseudo code addresses 61 to 70% of required features. The program runs observing up to three constraints, with at least the first two in the correct order for the task.	Pseudo code addresses 71 to 80% of required features. The program runs observing up to three constraints, in the correct order for the task.	Pseudo code addresses over 80% of required features. The program runs observing all constraints, in the correct order for the task.	The design delivers all of the required features and goes beyond the requirements in such a way as to propose a solution that is fully efficient and will result in elegant program code.	
Task 2 **30%**	Pseudo code lacks clarity / is incomplete. The program does not deliver a complete solution.	The program works but the constraints are not met in the order they have been specified. Pseudo code and design diagrams show that about half the required features are met.	Pseudo code addresses 51 to 60% of required features. The program runs observing up to three constraints.	Pseudo code addresses 61 to 70% of required features. The program runs observing up to three constraints, with at least the first two in the correct order for the task.	Pseudo code addresses 71 to 80% of required features. The program runs observing up to three constraints, in the correct order for the task.	Pseudo code addresses over 80% of required features. The program runs observing all constraints, in the correct order for the task.	The design delivers all of the required features and goes beyond the requirements in such a way as to propose a solution that is fully efficient and will result in elegant program code.	
Task 3 **30%**	Pseudo code lacks clarity / is incomplete. The program does not deliver a complete solution. The justification of the design choices is minimal at best.	The program works but the constraints are not met in the order they have been specified. Pseudo code and design diagrams show that about half the required features are met. The justification shows some clarity of thought, but	Pseudo code addresses 51 to 60% of required features. The program runs observing up to three constraints. There is an effort to justify the design choices, but there is no	Pseudo code addresses 61 to 70% of required features. The program runs observing up to three constraints, with at least the first two in the correct order for the task. The justification of design	Pseudo code addresses 71 to 80% of required features. The program runs observing All constraints, in the correct order for the task. The justification is good showing how the design	At least 90% of the code works efficiently and delivers the required functionality. All constraints are met and the code is neat, well adorned with comments. The comments match the logic	All tasks have been fully met. The code is elegant, well documented and efficient. The software exceeds the requirements offering a more complete and intelligent solution.	

		with no reference to efficiency.	clarity as their contribution to efficiency.	choices traces the changes across the three tasks but does not explain why the chosen data structures and algorithms offer efficiency.	choices map the changing requirements from task to task.	of effort to achieve efficiency.		