School of Information Systems

COR-IS1702 Computational Thinking

Project Assignment[1] (v1.1)

Release: 1 Nov 2020 (Sunday, Week 12)

Due: 15 Nov 2020 (Sunday, 11pm, Week 14)

Teaming:

(1) This project is to be done in teams of two. Unless you have requested to attempt this project alone, you have already been allocated a teammate.

(2) From 3rd Nov 2020 onwards, go to http://red.smu.edu.sg and request for a new password. Remember to use your full email address when doing so (i.e. include your school in your email address: xxx@<SCHOOL>.smu.edu.sg)

(3) After logging into red, you will be able to see your team ID at the upper right corner of the browser screen. Your team looks like this "G<section>_T<number>" (e.g. G1_T30).

(4) Click on your team ID to see the members of your team. If you are working alone, you will only see your name in "Members". Contact your team member to start working together.

(5) You will not be doing a peer review at the end of this project, but we have a whistle-blowing policy in place. Do contact your respective instructor ASAP if your teammate is uncontactable, does not respond, or is not contributing to the group effort at all.

General Instructions:

(1) Timeline:

- Release of project requirements: 1st Nov 2020
- red will be ready for submission from 3rd Nov 2020.
- Deadline: **15 Nov 2020**, 23:00hrs (for all deliverables):
    - Late submissions before 16 Nov 2020 23:00 hrs will be accorded a 25% penalty.
    - Submissions beyond 16 Nov 2020 23:00hrs will not be marked.

(2) Both questions are to be done in a team of 2 members. Each team submits only one solution.

(3) Refer to "Announcements" at red.smu.edu.sg for bug discoveries/fixes, dates and new data sets that will be provided to test your solutions.

(4) **WARNING**: Plagiarism is strictly not tolerated and plagiarism cases will be referred to the

[1] Acknowledgement: Questions are set by Senior TA Han Chung-Kyun.

university's disciplinary committee. You MUST acknowledge all third-party contributions (including assistance acquired) in your write-up and list all sources in a reference list there.

(5) The following Python modules have been installed on red, and you may use them by importing them in your python file: **numpy**, **pandas**, **scipy**, **sklearn**[2]. You are not required to use them and no additional modules will be installed on red. You are allowed to use other 3rd party code in your solutions (just upload the code in a separate .py file or include the relevant functions in **q1/2.py**), and you MUST acknowledge and reference all such usage in your write-up.

(6) Only python code can constitute part of your solution. Do not include binary files, code from other languages or pre-compiled python code. Your code should not communicate with any other servers/programs. You are also not allowed to use multi-threaded code to improve performance.

(7) Any member of a team can submit for the team. You may submit your solution to red any number of times, but the latest submission by any member before the deadline will be treated as the final submission for the team. (Please coordinate so that you will not "overwrite" your teammate's submission.)

Deliverables:

Please submit the following by the stipulated deadlines:

(i) a working **p1.py** and **p2.py** to red.smu.edu.sg

(ii) identical copies of **p1.py** and **p2.py** to eLearn Assignments (this portal: http://smu.sg/ct2020)

(iii) a single write-up (PPT or PPTX) to eLearn Assignments. Name your write-up <Gx_Tyy>.ppt. Your write-up should include:

- Your team ID and names of all members - not counted in slide limit
- Acknowledgement and reference list (if any) - not counted in slide limit
- Content: Explain how your algorithm works for both solutions and, if possible, derive the time complexity of your algorithm (with clear steps on how it is derived). The **combined content** for both solutions **may NOT exceed FOUR slides.** Any content beyond the slide limit may not be considered for assessment. You may include content in the "Notes" section of your slides, but they may not be considered during assessment.

Grading:

This project is worth 20% of your final course grade. Q2 is worth more marks than Q1, and quality is higher weighted than performance. This is the breakdown:

Question	Marks (%)	
Q1 & Q2 write-up	2	
Q1 quality	~~4~~ 5	~~6%~~ 7%
Q1 performance	2	
Q2 quality	8	~~12%~~ 11%
Q2 performance	~~4~~ 3	

[2] To use **sklearn**, you need to insert this statement at the top of your code: **from pandas import ***

Write-up: you will be graded on the clarity of your explanation and steps to derive the time complexity, and not on the quality of your algorithm. Use "point form" and diagrams if relevant.

Quality and performance: scoring is competitive, and determined by your relative rank on red. Performance and quality of your algorithm will be scored separately by referring to your solution's respective rank on the various scoreboards. For example, a Q1 solution that ranks 1st for quality and somewhere in the middle for performance on the scoreboards may get 4/4 marks for quality and ~1.25/2 marks for performance. The best solutions will rank highly for both quality and performance. Your relative rank on the scoreboard at red is only indicative and not final. After the deadline, new data sets will be used at red in order to determine your final score.

The time limit for each question, as well as errata announcements, will be shown as an "Announcement" at the home page of red. Also, do modify the **W** and **n** values in order to test your algorithm for different scenarios (by modifying the file names of the various CSV files - see **readme.txt** in the data folder).

Files given

You are given the following files:

- **q1** and **q2.py**: You are expected to edit the code in **select_packageSet()** and **select_packageSets()** respectively. These are the only 2 files you should submit to red. (You may submit them individually or together - just ensure that your team's name is on the scoreboard for both questions before the deadline.)
- **q1_main** and **q2_main.py**. Use them to test your q1/q2.py on your machine. Do not submit q1/q2_main to red.
- **utility.py.** Contains functions used by q1/q2_main.
- multiple **packages<x>.csv** files:
    - These CSV files are used by q1/q2_main as input. The **s_package** and **m_package** files are for Q1 and Q2 respectively.
    - Each CSV file represents a list of packages that your algorithm may choose from. Each line in a CSV file represents one package and contains the package ID, reward and weight.
    - You may assume that package IDs are unique and always starts with P000 and runs in order.
    - Your solution should work with different CSV files.
    - **readme.txt** in the data folder provides more information.

The context

You need some amount of money for your mother's present, so you decide to do package deliveries through an app for a while. The app automatically estimates each delivery's reward (how much pay you will get for that delivery) based on the properties of packages and the

traveling costs. Another information the app provides is each item's weight. You want to earn money as much as possible, but the vehicle you will use for deliveries can load stuff up to a certain weight only. If the loaded weight exceeds the limit, an accident is highly likely to happen.

For simplicity, though the traveling costs would depend on the travel sequence for deliveries, you can assume that that portion is not significant. Also, you can assume that there is no perishable item and the delivery time doesn't matter.

Q1

Given a list of packages for delivery (with the package ID, reward and weight of each item), develop an algorithm that returns the set of packages that you will accept to deliver. Your objective is to maximize the total reward without exceeding your vehicle's weight limit. Your algorithm is not expected to return the optimal solution though.

Requirement

You are required to fill up the body of this function in **q1.py**:

select_packageSet (W, packages)

where:

- **W** is a weight limit of the vehicle used for deliveries (**W**>0).
- **packages** is a 2D list that contains the unique package ID, reward, and weight of each package. You can assume that there will be at least 1 package in **packages** and that reward and weight will always be integers (not floats).

  e.g., the following represents three packages that you may choose to accept for delivery:

  [["P000", 35, 45], ["P001", 10, 20],["P002", 40, 51]]

  Package P000 is worth $35, and its weight is 45kg. Package P001 is worth $10, and its weight is 20kg. Package P002 is worth $40 and weights 51kg.

select_packageSet should return a list of package IDs that have chosen to accept for delivery. For example, your function may return this:

["P000", "P002", "P010"]

The total reward of of your selection is $117 (= $35 + $40 + $42) and the sum of weights is 146kg (= 45kg + 51kg + 50kg). It is important that the sum of weights is $\leq$ **W**.

Scoring

Your algorithm will be scored in 3 aspects:

1) Correctness: the function should return a valid set of packages, which means the sum of weights must be $\leq$ **W**. Performance and quality will only be considered if your solution is correct. It is VERY important that your solution is "correct".
2) Performance: the time taken for your algorithm to complete (the faster, the better). Your function MUST return within the time limit on red server. This time limit will be mentioned in "Announcement" at red.
3) Quality: the quality score will be the sum of rewards of packages you select (the higher,

the better). Quality is given a significantly higher priority than performance for grading. (See "Grading" section at the beginning of this document.)

Q2

Your friends heard that you have developed the algorithm for selecting delivery tasks from the app and want you to share the knowledge and experience. After a discussion, your group decides to cooperate and fulfil a set of delivery tasks together as a team. Everybody wants to get high rewards but does not want to confront a scenario whereby few members would get low rewards. Also, there is the weight limit that each vehicle faces. For simplicity, you can assume that all members of your team will use have the same weight limits.

For Q2, you will develop a new algorithm for selecting multiple sets of packages. While getting high rewards from each set of packages, the algorithm should deal with the fairness issue. Specifically, the algorithm needs to maximize the minimum reward that each member gets. This will ensure that everyone gets a good reward.

Requirement

You are required to fill up the body of this function in **q2.py**:

select_packageSets (n, W, packages)

where:

n is the number of members in your team (**n**>1).

select_packageSets should return a 2D list of package IDs that represent the set of packages that each of your **n** members will deliver. For example, if **n**=4, your function may return this:

[["P001", "P003"], ["P011", "P007"], ["P004", "P005", "P006"], ["P012"]]

This is to be interpreted as:

- Member 1 in your group is assigned this set of packages: P001, P003
- Member 2 in your group is assigned this set of packages: P011, P007
- Member 3 in your group is assigned this set of packages: P004, P005, P006
- Member 4 in your group is assigned this set of packages: P012

Scoring

Like Q1, your algorithm will be scored in 3 aspects:

1) Correctness: the function MUST return a valid list of sets of packages, and each set's weight sum must be $\leq$ **W**. Also, there should be no duplicated package IDs among package sets (i.e. each package may only be included in at most one set). There should be exactly **n** sets in your solution. Empty sets are allowed.

   e.g. if **n**=4, and the available package IDs are P001, P002…. P012, the following solutions are incorrect:

- **[["P001", "P003"], ["P011", "P007"], ["P004", "P006"]]** ← 3 sets (need **n** sets exactly)
- **[["P001", "P003"], ["P011", "P005"], ["P005", "P006"], []]** ← P005 appears in 2 sets
- **[[], ["P011", "P007"], [], ["P012", "P001", "P001"]]** ← P001 appears twice in the same set

- **[["P001", "P003"], ["P011", "P007"], ["P004", "P013"], ["P006"]]** ← P013 is invalid (no such package)

2) Performance (time taken) - as for Q1.
3) Quality: the quality score will be the minimum sum of rewards among **n** members (the higher, the better).

   e.g. if **n**=4, and the sum of rewards for the members are: $100, 124, 98, 150. The minimum sum of rewards for this solution will be $98, and that will be your quality score.

Have fun! :D

~End

Appendix: About the Test Cases on Red

Notes:

- The time limit for your code to run on red has been set to 1 minute each for Q1 and Q2.
- You will see 4 test cases for each question on red. Click on the "Scoreboard" menu and you will see 8 different scoreboards; each scoreboard corresponds to one of the 8 test cases.
- Here are the details for each test case. You should be able to reproduce the same quality score on your own laptop using the correct parameters:

Q1:

Scoreboard name	**Question project_q1**	**Question project_q1 24pkg**	**Question project_q1 40pkg**	**Question project_q1 131pkg**
CSV file used	s_packages_26.csv	s_packages_6404180.csv	s_packages_40.csv	m_packages_5_165.csv
W	26	1000,000	40	165
no. of packages	5	24	40	131

Q2:

Scoreboard name	**Question project_q2**	**Question project_q2 24pkg**	**Question project_q2 131pkg**	**Question project_q2 201pkg**
CSV file used	m_packages_2_98.csv	s_packages_6404180.csv	m_packages_5_165.csv	m_packages_4_100.csv
W	98	1000,000	165	100
n	2	12	5	12
no. of packages	10	24	131	201

- Remember that after the deadline, more test cases will be added to get a more accurate ranking of your solutions. What you see on red before the deadline is only indicative.
- The scoreboards have been anonymized; you will only be able to see your own team's name on the scoreboards.