# Project 2 (v1.0)

**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) 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: 18 Mar 2020 (Wed, Week 11)
- red will be ready for submission from 19 March 2020.
- Deadline: **3 Apr 2020**, 23:00hrs (Fri, Week 13). Late submissions will be severely penalized.

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

(3) Teams with the best algorithms for each section may be invited to present their solutions.

(4) 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.

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

(6) The following modules have been installed on red, and you may use them by importing them in your python file: **numpy**, **pandas**, **scipy**, **sklearn**[1]. 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 **p2q<X>.py**), and you MUST acknowledge and reference all such usage in your write-up.

(7) Only python code can constitute part of your solution. Do not include binary files, or code from other languages, 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.

(8) Any member of a team can submit for the team. 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.)

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

**Deliverables:**

Please submit the following by the stipulated deadlines:

(i) a working **p2q1/2.py** to red.smu.edu.sg

(ii) identical copies of **p2q1/2.py** to eLearn (Assignments)

(iii) a single write-up (PDF) to eLearn (Assignments). Name your write-up <Gx_Tyy>.pdf. Your write-up should include:

- Your team ID and names of all members
- Acknowledgement and reference list (if any)
- 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 a single A4-sized page.** Any content beyond the page limit may not be considered for assessment.

**Grading:**

This project is worth 20% of your final course grade. Each question is worth 10%. This is the breakdown:

(i) 2% for write-up

You will be graded on the clarity of your explanation and steps to derive the time complexity. Use "point form" and diagrams if relevant.

(ii) 12% for quality

(iii) 6% for performance

Scoring is competitive, and determined by your relative rank in terms of quality and performance on red. Your relative rank on the scoreboard at red is only indicative and not final. After the deadline, new data sets will be loaded to 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.

Performance (time taken) and quality of your algorithm will be scored separately by referring to your solution's respective rank on the various scoreboards. Hence a solution that ranks first for quality and somewhere in the middle for performance on the scoreboards may get 12/12 marks for quality and roughly 4/6 marks for performance. The best solutions will rank highly for both quality and performance. Do modify the values of **flags_csv**, **p** and **v** in your **main.py** in order to test your algorithm for different scenarios.

---

**The context**


A game is to be played in a huge open flat field where several flags are planted. Each flag is labelled with a positive number that indicates the number of points the flag is worth. Players touching a flag the first time will receive these points (no points for a subsequent touch). All players start from the same starting point (SP) (**x**=0.0, **y**=0.0). When the whistle blows, players are free to move from flag to flag to collect as many points as possible by touching them.

For simplicity, you can assume that all players run at the same speed (which implies that the total distance travelled determines the time taken by each player). You can also assume that players move from between flags directly in straight lines. This means that the distance between two flags can be easily calculated as the Euclidian distance between these two points. In order to facilitate planning, players are given the coordinates of every single flag, as well as the points of each flag before the game starts.

You are given multiple **flags.csv** files:

- **flags<x>.csv**: The list of flags is shown in this file with the following attributes: flag ID, points, x-coordinate, y-coordinate. Each line in this CSV file represents one flag. For example, the following three lines represent three flags in the field:

  F001, 3, -39.0888931396, 14.9550391799
  F002, 1, -10.3413339776, -14.6545825259
  F003, 4, 25.7389470222, 32.5374354386

Each CSV file represents one particular game field. Your solution should work with different CSV files.

---

**Q1**

You are a player in this game. The objective is to collect at least **p** points. (Since players run at the same speed, this means that you want to minimize the distance taken in your route.) It does not matter how many points you manage to accumulate; as long as you get at least **p** points. Plan the route that you will take in your attempt to win the game. There are two variations of this game:

(i) In the first variation, players stop at the last flag in their route to end the game; there is no need to move back to the SP.

(ii) in the second variation, all players must get back to the SP to end the game.

In both variations, the objective is still the same: minimize the distance the player has to travel to collect at least **p** points.

**Requirement**

You are required to fill up the body of this function in **p2q1.py**:

**get_route(p, v, flags)**

where:

- **p** is the target number of points that a player must collect before ending the game. Players who fail to accumulate at least p points lose the game immediately. You can assume that **p** is >0 and that there are enough flag points in the field to meet **p**'s requirement.
- **v** is the game variation. If **v** is 1, players do not need to move back to the SP. If **v** is 2, players

must get back to the SP.

- **flags** is a 2D list that contains the unique ID, points, and x and y coordinates of each flag. The following shows two flags:

  **[[F001, 3, -39.088893, 14.9550392], [F002, 1, -10.341334, -14.654583]]**

  Flag F001 is worth 3 points, and is positioned at (x=-39.088893, y=14.9550392). Flag F002 is worth 1 point, and is positioned at (x=-10.341334, y=-14.654583).

This function should return a list of flag IDs that represent the route that you will take. For example, your function may return this:

**[F002, F005, F003, F009]**

This is to be interpreted as:

1) Player starts from (0, 0) and goes to F002 and touches F002.
2) Player then goes from F002 to F005 to touch F005.
3) Player then goes from F005 to F003 to touch F003.
4) Player then goes from F003 to F009 to touch F009.
5) Player then goes from F009 back to (0, 0) in order to end the game.

The total distance travelled by the player would be the sum of Euclidian distances between each of these locations that constitute the route. Let d(A to B) represent the Euclidian distance between A and B, then:

If v = 1, total distance =

d(SP to F002) + d(F002 to F005) + d(F005 to F003) + d(F003 to F009)

If v = 2, total distance =

d(SP to F002) + d(F002 to F005) + d(F005 to F003) + d(F003 to F009) + d(F009 to SP)

**Scoring**

Your algorithm will be scored in 3 aspects:

1) Correctness: the function should return a valid route (i.e. a list containing unique and valid flag IDs), and the total number of points collected must be >=**p**. 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). Check the time limit that will be allowed under "Announcements" at red. Your function MUST return within the time limit.
3) Quality: the quality score will be the distance travelled by the player using your proposed route (lower is better). Quality is given a significantly higher priority than performance for grading.

**Q2**

You manage a team of **n** players in this game (where n is a number from 1 to 8). The rules and objective of the game is the same as for Q1 except that:

(i) Players in your team do not get points for touching the same flag more than once. If player 1 has already touched F0009, no other player in your team should touch the same flag.

(ii) The total number of points collected by the whole team need to be at least **p** to end the game.

Plan the routes for each player in your team so as to minimize the total distance travelled by all players in order to collect at least **p** points as a team.

**Requirement**

You are required to fill up the body of this function in **p2q2.py**:

**get_routes(p, v, flags, n)**

where:

**n** is the number of players in your team. You can assume that **n** is a number between 1 and 8 (inclusive).

This function should return a 2D list of flag IDs that represent the routes that each of your **n** players will take. For example, if **n**=4, your function may return this:

**[[F012, F003], [F001, F005, F004], [F009], [F002, F006, F007]]**

This is to be interpreted as:

1) Player 1 in your team is assigned this route: SP, F012, F003
2) Player 2 in your team is assigned this route: SP, F001, F005, F004
3) Player 3 in your team is assigned this route: SP, F009
4) Player 4 in your team is assigned this route: SP, F002, F006, F007

Like Q1, if v = 2, the total distance travelled by each player includes the distance between the last flag in his respective route and SP.

**Scoring**

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

1) Correctness: the function should return a valid list of routes and the total number of points collected by all players in your team must be >=**p**. Routes should not have duplicated flag IDs (i.e. each flag may only be visited at most once by any player in your team.) There should be exactly **n** routes in your solution. It is permissible for a player not to move (i.e. his route is [] as long as the team meets the **p** requirement.
2) Performance
3) Quality: the quality score will be the total distance travelled by all **n** players (the lower the better).

---

**END**