# FIT1045 Algorithmic Problem Solving – Assignment (18%).

Due: 23:55:00, Friday 12th May, 2017.

## Submission Procedure

1. Put you name and student ID as a comment on each file.
2. Make sure your programs strictly follow the instructions given at the end of each task. Use the automated checkers provided to ensure that your programs produce correct results.
3. Save your files named `task1.py` and `task2.py` into *a single folder* called `assignment` and create a zip file of this folder called yourFirstName_yourLastName.zip.
4. Submit your zip file containing your solution to Moodle. Make sure that your assignment is not in "Draft" mode. You need to click "Submit" to successfully submit the assignment.
5. Your assignment will not be accepted unless it is a readable zip file.

**Important Notes:**

1. Please ensure that you have read and understood the university's policies on plagiarism and collusion available at `http://www.monash.edu.au/students/policies/academic-integrity.html`. You will be required to agree to these policies when you submit your assignment. The assignments will be checked for plagiarism using an advanced plagiarism detector and the students will be interviewed by tutors to demonstrate the understanding of their code. Last year, many students were detected by the plagiarism detector and almost all got zero mark for the assignment and, as a result, failed the unit. "Helping" others is NOT okay. Please do not share your solutions with others. If someone asks you for help, ask them to visit us during consultation hours for help.
2. You must not use the built-in functions to sort (e.g., `sort()` or `sorted()`), orto find the maximum/minimum element in the list (e.g., `max()` or `min()`): please write your own Python code for these if required. You are not allowed to import any Python library or module except the math module.
3. Your program will be checked against a number of test cases. Do not forget to include comments in your code explaining your algorithm. If your implementations have bugs, you may still get some marks based on how close your algorithm is to the correct algorithm. Also, make sure that your program handles the boundary cases properly.
4. For each task, you need to write a program that properly decomposes the problem. You will learn more details of functions and decomposition in Week 6.

**Marks:** This assignment has a total of 100 marks and contributes to 18% of your final mark. Late submission will have 10% off the total assignment marks per day (including weekends) deducted from your assignment mark, i.e., 10 marks per day. So, if you are 1 day late, you will lose 10 marks. Assignments submitted 7 days after the due date will normally not be accepted.

## Marking Guide:

**Task 1: 50 marks**

(a) Code readability (Non-trivial comments where necessary and meaningful variable names) – 5 marks

(b) Code decomposition – 5 marks

(c) Correctly reading input from `matches.txt` – 5 marks

(d) Correctly computing stats for each team – 15 marks

(e) Correctly sorting the table – 20 marks

**Task 2: 50 marks**

(a) Code readability (Non-trivial comments where necessary and meaningful variable names) – 5 marks

(b) Code decomposition – 5 marks

(c) Correctly reading input from `POIs.txt` – 5 marks

(d) Correctly computing the minimum distance from a house to a given type of POI (e.g., schools) – 10 marks

(e) Correctly computing walkability score of a house – 10 marks

(f) Correctly computing the most walkable house – 10 marks

(g) Correctly writing the required information to the text file `walkable.txt` – 5 marks

# Task 1: FIFA World Cup Statistics 50 Marks

In this task, you are given scores of all matches played in 2010 FIFA World Cup[1] (see the file `matches.txt` under task1 folder in the zipped file provided on moodle). Some of the lines in `matches.txt` are shown below.

```
South Africa:Mexico:1:1
Uruguay:France:0:0
Korea Republic:Greece:2:0
Argentina:Nigeria:1:0
```

The values in each line are separated by colon. The values in first two columns correspond to the names of home team and away team, respectively. Third and fourth columns represent the scores of home team and away team, respectively. For example, the third line above shows that Korea Republic scored 2 goals and Greece scored 0 goal. You need to write a Python program that first computes the following statistics for each team.

```
Played: Number of matches played
Won: Number of matches won
Drawn: Number of matches drawn
Lost: Number of matches lost
GS: Total number of goals scored
GA: Total number of goals the opponent scored against the team
GD: The difference between GS and GA (i.e., GS-GA)
Points: Total number of points calculated as 3*Won + Drawn.
```

After computing the above statistics for each team, you need to rank the teams according to the following criteria (see the sample output shown on the next page for explanation of the ranking criteria).

- The team are first ranked by the total number of points (e.g., Germany is ranked higher than Argentina).
- If two teams have equal number of points then the team that played fewer matches is ranked higher (e.g., Australia and Mexico have 4 points each but Australia is ranked higher because Australia has played 3 matches as compared to 4 matches played by Mexico).
- If the tie is still not broken, the team with higher goal difference (GD) is ranked higher (e.g., Australia and South Africa have the same points and have played the same number of matches but South Africa is ranked higher because its GD is -2 as compared to the GD (-3) of Australia).
- If the two teams have equal number of points, have played the same number of matches and have the same goal difference, the team that has scored more goals (i.e., higher GS) is ranked higher (e.g., Nigeria is ranked higher than Algeria because Nigeria scored more goals than Algeria).
- In the unlikely event that the tie still cannot be broken, the teams are ranked alphabetically. We do not have such a case in this data set but assume that Nigeria and Algeria both had same number of points, matches played, GD and GS, then we would rank Algeria higher because "Algeria" < "Nigeria" in alphabetical order.

```
A sample output for the file matches.txt.
```

[1] Scraped from `http://www.fifa.com/worldcup/archive/southafrica2010/matches/`

| Team | Played | Won | Drawn | Lost | GS | GA | GD | Points |
|---|---|---|---|---|---|---|---|---|
| Netherlands | 7 | 6 | 0 | 1 | 12 | 6 | 6 | 18 |
| Spain | 7 | 6 | 0 | 1 | 8 | 2 | 6 | 18 |
| Germany | 7 | 5 | 0 | 2 | 16 | 5 | 11 | 15 |
| Argentina | 5 | 4 | 0 | 1 | 10 | 6 | 4 | 12 |
| Uruguay | 7 | 3 | 2 | 2 | 11 | 8 | 3 | 11 |
| Brazil | 5 | 3 | 1 | 1 | 9 | 4 | 5 | 10 |
| Ghana | 5 | 2 | 2 | 1 | 5 | 4 | 1 | 8 |
| Japan | 4 | 2 | 1 | 1 | 4 | 2 | 2 | 7 |
| Chile | 4 | 2 | 0 | 2 | 3 | 5 | -2 | 6 |
| Paraguay | 5 | 1 | 3 | 1 | 3 | 2 | 1 | 6 |
| Portugal | 4 | 1 | 2 | 1 | 7 | 1 | 6 | 5 |
| USA | 4 | 1 | 2 | 1 | 5 | 5 | 0 | 5 |
| England | 4 | 1 | 2 | 1 | 3 | 5 | -2 | 5 |
| Ivory Coast | 3 | 1 | 1 | 1 | 4 | 3 | 1 | 4 |
| Slovenia | 3 | 1 | 1 | 1 | 3 | 3 | 0 | 4 |
| Switzerland | 3 | 1 | 1 | 1 | 1 | 1 | 0 | 4 |
| South Africa | 3 | 1 | 1 | 1 | 3 | 5 | -2 | 4 |
| Australia | 3 | 1 | 1 | 1 | 3 | 6 | -3 | 4 |
| Mexico | 4 | 1 | 1 | 2 | 4 | 5 | -1 | 4 |
| Korea Republic | 4 | 1 | 1 | 2 | 6 | 8 | -2 | 4 |
| Slovakia | 4 | 1 | 1 | 2 | 5 | 7 | -2 | 4 |
| New Zealand | 3 | 0 | 3 | 0 | 2 | 2 | 0 | 3 |
| Serbia | 3 | 1 | 0 | 2 | 2 | 3 | -1 | 3 |
| Denmark | 3 | 1 | 0 | 2 | 3 | 6 | -3 | 3 |
| Greece | 3 | 1 | 0 | 2 | 2 | 5 | -3 | 3 |
| Italy | 3 | 0 | 2 | 1 | 4 | 5 | -1 | 2 |
| Nigeria | 3 | 0 | 1 | 2 | 3 | 5 | -2 | 1 |
| Algeria | 3 | 0 | 1 | 2 | 0 | 2 | -2 | 1 |
| France | 3 | 0 | 1 | 2 | 1 | 4 | -3 | 1 |
| Honduras | 3 | 0 | 1 | 2 | 0 | 3 | -3 | 1 |
| Cameroon | 3 | 0 | 0 | 3 | 2 | 5 | -3 | 0 |
| Korea DPR | 3 | 0 | 0 | 3 | 1 | 12 | -11 | 0 |

## Important Instructions

- Download the files provided on Moodle. In the task1 folder, a file `task1.py` is provided. Write your code in this file. We have provided two functions in this file. **Do not modify these functions - These are critical for auto-marking as explained later.**
- You will need to create a table (i.e., list of lists) that contains the statistics for each team and then sort the table as shown in the above input.
- Once you have sorted the table according to the above criteria, call the function `printTable()` provided in `task1.py` and pass your table as a parameter. This should display you the output as well as it will create an output file called `FIFA_stats.txt`.
- You can check whether your program is generating correct results for the provided file `matches.txt`. The program `checker_task1.py` will auto-mark your program. Therefore, it is important that you strictly follow the instructions. Open `checker_task1.py` in IDLE and run the program. The checker will run your code in `task1.py` and will compare your results stored in `FIFA_stats.txt` with the results in `expected_FIFA_stats.txt`. If there is an error, it will show the first row that does not match with the expected output.
- The checker has been tested on Windows environment on the lab machines. If you receive some strange errors, test your code on the lab machine before submitting or contact us with the specific details of the error message.

## Task 2: Selecting the Most Walkable House 50 Marks

Your friend Alice is moving to USA. She is considering to rent a house in North West USA (Washington and Oregon area). Her ideal house would be close to a school, a fast food restaurant, a post office and a hospital. She has come up with a scoring function to compute the score of each available house, called walkability score[2]. Given a house $h$, let $mindist(h, school)$ denote the distance of the closest school to the house $h$. Similarly, $mindist(h, fastfood)$, $mindist(h, postoffice)$, and $mindist(h, hospital)$ denote the distances to the house $h$'s closest fast food restaurant, post office, and hospital, respectively. Walkability score of the house $h$ is denoted as $h.walkability$ and is computed using the equation below.

$$h.walkability = mindist(h, school) + mindist(h, fastfood) + mindist(h, postoffice) + mindist(h, hospital)$$

She wants to find the most walkable house, i.e., the house with the smallest walkability score. She has downloaded a file from OpenStreetMap (www.openstreetmap.org) that has the locations of all relevant points of interest (POIs), i.e., schools, post offices, fast food restaurants, hospitals and the houses available for renting in Washington and Oregon (see `data.txt` under task2 folder). Below are some records from this file.

```
1:47.068818:-122.89045:house
2:45.513901:-123.065825:school
3:43.35308:-124.201326:school
4:45.639439:-122.658908:hospital
```

Each line represents a unique point of interest (POI). The values in each line are separated by colon. First column in each line is the unique id of the POI. The second column is the latitude and the third column is the longitude[3] of the POI. The fourth column specifies the type of POI (e.g., house, school etc.). The distance between two points is computed using haversine formula[4] that returns the shortest distance (i.e., as-the-crow-flies) between two points on earth. The function to compute haversine distance is provided in the file `task2.py`. The function `haversine(point1,point2)` takes two lists as parameters where each list contains latitude and longitude of the point. The function returns the distance between the two points in kilometers. Below is an example on how to call this function to get the distance between the house with ID 1 and the school with ID 2 in the above sample input.

```
p1 = [47.068818,-122.89045]
p2 = [45.513901,-123.065825]
dist = haversine(p1,p2)
print("The distance is:",dist, "km")
```

This will print "`The distance is: 173.42303260967046 km`".

Alice needs your help in writing a program that can find the most walkable house in a given area. Specifically, the program should take a rectangular area (called window) as an input from the user and return the most walkable house among the houses that lie in the window. The window is represented using the locations of its lower left corner and upper right corner. Consider the example of Figure 1 where the window (the shaded area) is represented by its lower left corner `[47.7, -122.5]` and upper right corner `[47.9, -122.4]`. Note that, in `[47.7, -122.5]`, 47.7 is the latitude and -122.5 is the longitude of the point.

The user will enter the input in a string where values are separated by colon. The first two columns correspond to the latitude and longitude of the lower left corner and the third and fourth column represent the latitude and longitude of the upper right corner. Below is a sample input entered by the user for the window in Figure 1.

```
47.7:-122.5:47.9:-122.5
```

Your program must find the most walkable house in the given window. In the example of Figure 1, there are only two houses $H_1$ and $H_3$ that lie within the window. The closest school to $H_1$ is $S_1$ and assume that the haversine distance between $H_1$ and $S_1$ (denoted as $dist(H_1, S_1)$) is 1 km, i.e., $mindist(H_1, school) = dist(H_1, S_1) =$

[2] Walkability score is inspired by Walk Score (`https://www.walkscore.com/`) and the paper "*Finding the sites with the best accessibilities to amenities*" published in DASFAA 2011.

[3] These are real world POIs except that houses are actually parks. Try entering 45.513901, -123.065825 in Google Maps. This data set was collected by Tenindra Abeywickrama for his paper "*k-Nearest Neighbors on Road Networks: A Journey in Experimentation and In-Memory Implementation*", published in VLDB 2016.

[4] `https://en.wikipedia.org/wiki/Haversine_formula`

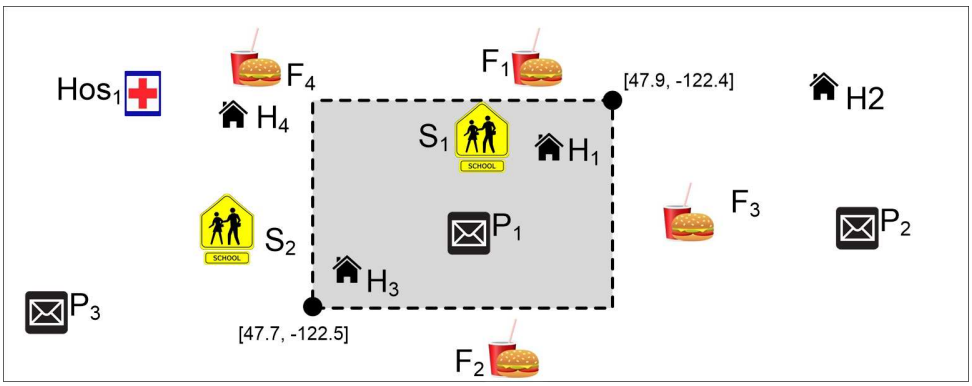


Figure 1: Selecting the most walkable house in a given area

$1km$. Similarly, assume that $mindist(H_1, fastfood) = dist(H_1, F_1) = 1km$, $mindist(H_1, hospital) = dist(H_1, Hos_1) = 7km$ and $mindist(H_1, postoffice) = dist(H_1, P_1) = 2km$. The walkability score of $H_1$ is $H_1.walkability = 1 + 1 + 7 + 2 = 11km$. Similarly, assume that for house $H_3$ in the window, the distances to the closest POIs are as follows. $mindist(H_3, school) = dist(H_3, S_2) = 2km$, $mindist(H_3, fastfood) = dist(H_3, F_2) = 3km$, $mindist(H_3, hospital) = dist(H_3, Hos_1) = 6km$ and $mindist(H_3, postoffice) = dist(H_3, P_1) = 2km$. The walkability score of $H_3$ is $H_3.walkability = 2+3+6+2 = 13km$. Thus, the most walkable house in the window is the house $H_1$.

Your program must output the location of the most walkable house, its walkability score and its distance to the closest POI of each type. The output must be printed to a file called `walkable.txt`. Below are two sample outputs for two different windows using the data provided in `POI.txt`.

A sample output.

```
Enter the window: 47.701460:-122.573313:47.916884:-122.397654
Location of the most walkable house: 47.800229, -122.502084
Walkability Score: 12.64256128910096
The closest school is 0.30596846398819993 km
The closest fast food is 0.06558006467957564 km
The closest post office is 0.19904259701867597 km
The closest hospital is 12.071970163414509 km
```

Another sample output.

```
Enter the window: 42.3:-124.2:43.3:-123.3
Location of the most walkable house: 42.449755, -123.328187
Walkability Score: 5.315847211532855
The closest school is 0.6055333221300785 km
The closest fast food is 1.0371239069242966 km
The closest post office is 0.395888284834936 km
The closest hospital is 3.2773016976435434 km
```

If the window does not contain any house, your program must print "`No house found in the window`".

## Important Instructions

- Download the files provided on Moodle. In the task2 folder, a file `task2.py` is provided. Write your code in this file. We have provided two functions in this file. **Do not modify these functions - These are critical for auto-marking as explained later.**
- Your program must take the window as an input in the format described above and must return the most walkable house along with other stats as shown above.
- The results must be written to a file `walkable.txt`. We have provided a function `writeResults` to assist in following the correct format. See the details below.

- You can check whether your program is generating correct results for the provided file `POI.txt` file and the window `42.3:-124.2:43.3:-123.3`. The program `checker_task2.py` will auto-mark your program. Open `checker_task2.py` in IDLE and run the program. The checker will run your code in `task2.py` and will compare your results stored in `walkable.txt` with the results in `expected_walkable.txt`. If there is an error, it will show the first line that does not match.
- The checker has been tested on Windows environment on the lab machines. If you receive some strange errors, test your code on the lab machine before submitting or contact us with the details of the error message.

Since the program will be auto-marked, it is important that your output strictly follows the sample output shown above including the whitespace, case of characters and spellings. We have provided a function `writeResults` that takes care of correctly printing the output and writing it to the required file. To call this function, you will need to create a list say called `results` containing 7 floats. Below is a description of each element in this list.

```
results[0] = latitude of the most walkable house
results[1] = longitude of the most walkable house
results[2] = the walkability score of the most walkable house
results[3] = distance of the closest school
results[4] = distance of the closest fast food restaurant
results[5] = distance of the closest post office
results[6] = distance of the closest hospital
```

For example, for the results shown above for window `42.3:-124.2:43.3:-123.3`, your list `results` will contain the following values.

```
results[0] = 42.449755
results[1] = -123.328187
results[2] = 5.315847211532855
results[3] = 0.6055333221300785
results[4] = 1.0371239069242966
results[5] = 0.395888284834936
results[6] = 3.2773016976435434
```

After you have successfully created the list `results`, call `writeResults(results)` and it will print the output and write the results in `walkable.txt` in the required format.

## Nuggets: NOT ASSESSABLE

Feel free to ignore everything below this. This is not assessed!!! This information is provided for the students who are interested in exploring beyond the scope of this unit.

For task 1, the scores of all matches played in 2010 FIFA World Cup were scraped from the official website of FIFA World cup using a simple Python program. See the file `crawlFIFA.py` in task1 folder. This program uses BeautifulSoup library which makes it very easy to scrape data from websites. For more details, see `https://www.crummy.com/software/BeautifulSoup/`. Using `crawlFIFA.py`, you can also download the matches for other world cups on FIFA website, e.g., set `url = www.fifa.com/worldcup/archive/brazil2014/matches/index.html` in `crawlFIFA.py` and run the program to get the scores for matches played in FIFA 2014 World Cup. I have not tested the program for previous world cups but the program should work fine unless the websites for the other world cups follow a different format. You may need to install BeautifulSoup before you can run the code.

For task 2, we are using real world data set obtained from OpenStreetMap. There are many libraries that you can use to do many interesting things. For exmaple, you can use `geopy` (`https://pypi.python.org/pypi/geopy`) to obtain the address of any POI. Below is a sample code to print the address of a POI.

```
from geopy.geocoders import Nominatim
geolocator = Nominatim()

# string contains the latitude and longitude of the POI
string = "45.513901, -123.065825"
location = geolocator.reverse(string)
print(location.address)
```

The above program gives the following output.

```
Emmaus Christian School, 460, South Heather Street, Cornelius, Washington County,
Oregon, 97113, United States of America
```

Similarly, you may draw different POIs on Google Maps quite easily using `gmplot` library[5]. Below is a sample code that draws the most walkable house in red circle and its nearest POI of each type in a different color (school in blue, fast food in black, post office in purple and hospital in green).

```
import gmplot
def drawPOI(color, POI):
    gmap.scatter([POI[0]], [POI[1]], color, size=30, marker=False)

gmap = gmplot.GoogleMapPlotter(POIs[0][0], POIs[0][1], 13)
color = ["red","blue","black","purple","green"]
for i in range(len(color)):
    drawPOI(color[i],POIs[i])
gmap.draw("mymap.html")
```

This code generates a html file called `mymap.html` (see Figure 2 below). This code assumes that POIs is a list of lists containing the latitude and longitude of the most walkable house and its nearest POI of each type. For Figure 2, POIs is [[48.753228, -122.481751], [48.757628, -122.4747], [48.748028, -122.479301], [48.751455, -122.478373], [48.753928, -122.476601]]. You may need to install `gmplot` library before you can use it.

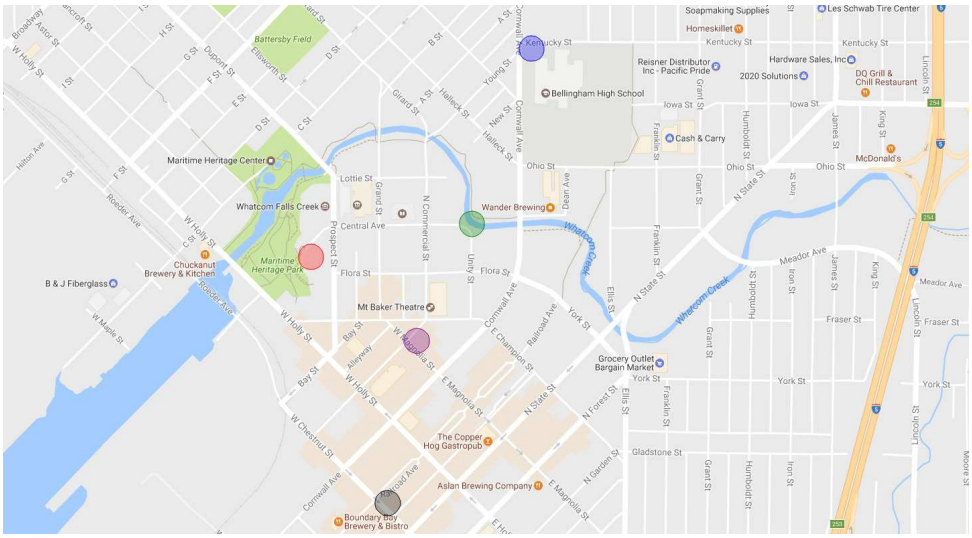


Figure 2: Drawing POIs on Google Maps using gmplot library

Note that I am not an expert in Python and there may be better ways/libraries to do similar or other interesting stuff. However, the above shows that there is a rich resource of very useful Python libraries that can do many interesting things rather easily.

DO NOT SUBMIT ANYTHING SHOWN IN THE NUGGETS SECTION WITH YOUR ASSIGNMENT. THIS MAY AFFECT THE AUTO-MARKING. HOWEVER, IF YOU HAVE DONE SOMETHING INTERESTING, AND WOULD LIKE TO SHARE WITH US, SEND IT TO US VIA EMAIL.

[5] https://pypi.python.org/pypi/gmplot/1.0.5