PROGRAMMING COMPETITION
COMPETITORS' PACKAGE

University of Toronto Engineering Kompetition
January 21-22, 2016

Director: Henry Xu

Schedule

Day 1: Saturday, January 21, 2017

10:00am - 10:30am	Registration / Sign-in
10:30am - 11:30am	Keynote
11:30am - 12:30pm	Lunch
12:30pm - 01:00pm	Problem Statement Briefing
01:00pm - 07:00pm	Working Time (submission due at 07:00pm)
07:00pm - 08:00pm	Testing

Day 2: Sunday, January 22, 2017

09:30am - 10:00am	Sign-in
10:00am - 10:30am	Presentation Briefing
11:00am - 01:00pm	Lunch
10:30am - 03:00pm	Testing + Presentations
03:30pm - 04:30pm	Closing Ceremony

Introduction

I. Rules

- You may not collaborate with anyone other than members of your team.
- Cite all of your libraries

II. Deliverables

a. Code

All code and documentation you have written for the competition, including a README with installation instructions and a list of libraries. Code can be written in any language that has a compiler available for all students. Any libraries that are available to the public or all U of T students may be used except for Parts 4 and 6.

b. Presentation*

A presentation summarizing the key aspects of the problem and proposal.

	Time	Additional Information
Team Presentation	10 minutes	Reminders will be given at 3 minute, 1 minute, and 30 second marks. There is a grace period of 30 seconds, after which the presenters **will** be cut off.
Judges Q&A	5 minutes	All members of the team must be ready to answer questions.

Submission Method

Send the code in a zipped folder and the presentation as a .pptx or .odp file to programming@utek.skule.ca before **7:00 PM** on Saturday January 21st. Please name the subject "**UTEK Programming 2017 - Team X**", where X is substituted with your team number. Only the last submission before the deadline will be considered. Late submissions will **not** be accepted.
Testing on Day 1 will take place from 7:00 - 8:00PM in the auditorium.
*** Only 10 teams will advance to the presentation stage on Day 2.** You will be informed if you have advanced by midnight on Day 1, so each team will need to submit a presentation.

At 7:00 PM, you will receive some number of test cases for each part in files named partXcaseY.in, where X, Y $\in \mathbb{N}$. The values of Y will not necessarily be contiguous because some of these will be unique to your team. Submit the outputs in files named partXcaseY.out, where X and Y match the input file. These must be received by 8:00 PM. For these, disable your cache so that everyone has comparable results.

Background

Advances in batteries and growing environmental concern have led sales of electric vehicles to skyrocket in the last few years. Car manufacturers have taken this trend seriously, resulting in cars such as the Chevrolet Volt, Nissan Leaf and Tesla Model S. However one limitation is the relative paucity of electric charging stations compared to traditional fueling stations. In this challenge, you will help drivers of electric cars to navigate between Tesla's 337 supercharging stations around the US.

Problem Statement

Part 1: Finding a Charging Station
Knowing the location of the closest charging station is important in deciding whether to purchase an electric car. For this and the rest of the problem, you are to use the list of charging stations provided here. You will be given the GPS coordinates of a point and will need to output the names and coordinates of the three closest charging stations by their spherical distance.

The list of charging stations is provided as a JSON object, a common way of representing data in a machine and human-readable format. A complete schema is available here, but all scripting languages should have a built-in way to parse it.

Input: Latitude and longitude of a point in decimal degrees, separated by a single space.

e.g. `38.8977 -77.0365`

Output: Latitude and longitude of the three closest charging stations with the latitude, longitude, spherical distance in kilometres to the original point and station name separated by a single space, with each station separated by a newline.

e.g.
```
39.0250017 -77.148335 Westfield Montgomery Mall - Tesla 17.137
39.0942239 -76.85709 Towne Center at Laurel - Tesla 26.787
38.6434359 -77.2952552 Potomac Mills - Tesla 36.089
```

Part 2: Using the Google Maps API
Use the Google Maps API to get the driving distance and time between two points.

Input: Latitude and longitude of a pair of points in the same format as part 1.

e.g.
```
37.773972 -122.431297
37.4931367 -121.9453883
```

Output: Driving distance and time between the points in kilometers and hours respectively. Note that the time depends slightly on traffic. Use the default settings for the query.

e.g. `67.123 0.85`

Part 3: Caching
Because of Google's limit of 2500 API calls per day, you will need to build a caching system to store the distances between the charging stations that will be queried in Part 2. There are technically no deliverables for this part, but if you do not, you will need to create many new Google accounts and we cannot guarantee that you will not be blocked. Also, using the API is slower than a local server, slowing down your testing.

Only use the cache for testing. When we give you the actual test cases, do not use the cache.

Part 4: Pathfinding
Write an algorithm that calculates the path that takes the minimum time from the origin to the destination, outputting the latitude, longitude and names of the charging stations at which your car stops to charge, as well as the duration of the trip.

The Tesla Model X has a maximum range of 480 kilometers, so you cannot drive for more than that without refuelling. It takes 20 minutes to recharge from an empty to fully charged battery. Assume that charging is linear and that you start with a fully charged battery. Also assume that your car travels at 80 kilometers per hour.

You must implement the pathfinding yourself, i.e. you cannot use external libraries, including implementations that you have written yourself before the competition.

Input: Latitude and longitude of the origin and destination in the same format as above.

e.g.
```
37.773972 -122.431297
40.7128 -74.0059
```

Output: The space separated latitude, longitude, city and id of all charging stations that your car stopped to charge. For the start and end points, the names are "Start" and "End" and ids are 0 and 1 respectively. At the end, also output the duration in hours and distance travelled in kilometers in that order.

e.g.
```
37.773972 -122.431297 Start 0
39.3272037 -120.2064298 Truckee 63115
40.9571761 -117.7488086 Winnemucca 67209
40.738399 -114.058998 West Wendover 65734
39.6016773 -110.8331845 Price 71841
39.552525 -107.340878 Glenwood Springs 53219
39.77512 -104.794648 Denver 65086
40.9172852 -100.1697798 Gothenburg 80211
41.220464 -95.836025 Council Bluffs 67206
41.5773185 -90.5138484 Davenport 79682
41.7188212 -86.1878432 Mishawaka 53862
41.312212 -81.5183629 Macedonia 60107
41.0164106 -76.4918828 Bloomsburg 79204
40.7128 -74.0059 End 1
49.123 4932.842
```

Part 5: User application (Bonus)
Construct a user-friendly application to display the waypoints along the route. This can be on any platform (mobile, web, desktop), and will be demonstrated during the presentation. You must submit screenshots of your application by the deadline for it to be considered. Points will be awarded for UI and UX. You may want to use a library such as leafletjs (Javascript).

Part 6: Travelling Salesman Problem… sort of (Bonus)
Plan a route that visits a charging station in every state represented in the data (Alaska, Arkansas, Hawaii and North Dakota are not). You may visit charging stations more than once. You may not use a library implementation of the TSP. Entries will be scored against each other. We are not interested in the runtime, you can take as much time as you want. The distance and charging limitations from part 4 still apply.

Report your output in the same format as part 4.

Scoring Criteria

Parts 1-4 will be scored out of 100 in total. Parts 5 and 6 are worth an additional 50 points. The presentation is worth 50 points.

Metric	**Score**	**Notes**
Basic Feature: Parts 1-4	*(/100)*	*This score will determine who moves to presentation round*
Part 1: Correctness	15 (5 per test case)	
Part 2: Correctness	15 (5 per test case)	
Part 4: Correctness	50 (10 per test case)	
Code Quality	20	Code quality is scored based on readability, modularity and error-handling. Documentation of the code will also be considered.
Bonus Features: Parts 5-6	*(/50)*	
Part 5: Aesthetics (UI)	10	Scored by judges
Part 5: Ease of use (UX)	10	Scored by judges
Part 6: Correctness	10	Visit all 46 states represented

	20	Teams will be ranked by the time driven in their solution, and scored linearly according to rank
Presentation	*(/50)*	
Justification	10	Justify why your solution to the problem is effective
Presentation Quality	20	Organization, flow, professionalism
Clarity of explanation	20	Solution is clearly explained