



PROGRAMMING COMPETITION COMPETITORS PACKAGE

PROGRAMMING COMPETITION COMPETITORS PACKAGE

University of Toronto Engineering Kompetitions

Jan 16-17, 2016

##### **Contents**

[Schedule](#_Schedule)

[Overview](#_Overview)

[Problem Statement](#_Problem_Statement)

[Detailed Requirements](#_Detailed_Requirements)

[Presentation Guidelines](#_Presentation_Guidelines)

[Judging Criteria](#_Judging_Criteria)

##### **Schedule**

Programming Competition Schedule:

|  |  |  |  |
| --- | --- | --- | --- |
| **SATURDAY, JANUARY 16** | | | |
| **Time** | | **Events** | **Location** |
| **Start** | **End** |  |  |
| 8:00 | 9:00 | Set-Up |  |
| 9:00 | 9:30 | Registration (Programming) | BA Lobby |
| 9:30 | 10:00 | Briefing (Programming) | BA 1160 |
| 10:00 | 12:00 | Competition | Assigned Rooms |
| 12:00 | 01:00 | Lunch | BA Lobby |
| 01:00 | 06:00 | Competition | Assigned Rooms |

|  |  |  |  |
| --- | --- | --- | --- |
| **Sunday JANUARY 17** | | | |
| **Time** | | **Events** | **Location** |
| 9:00 | 9:30 | Check-in | BA Lobby |
| 10:00 | 12:00 | Presentations | BA 3008, 3012 |
| 12:00 | 1:00 | Lunch | BA Lobby |
| 1:00 | 3:00 | Presentations | BA 3008, 3012 |
| 1:00 | 2:00 | RBC Tech Talk | BA 2175 |
| 3:00 | 3:30 | Judges Deliberations |  |
| 3:30 | 5:00 | Closing Ceremonies & Awards | BA 1160 |

Team Allocation:

|  |  |  |
| --- | --- | --- |
| Team # | First Name | Last Name |
| 1 | Caroline | Chuang |
|  | Timothy | Chan |
|  | Yan | Fu |
|  | Sampson | Gao |
| 2 | Dixin | Wu |
|  | Yizhi | Xu |
|  | Zhihao | Sheng |
| 3 | Zain | Esmail |
|  | Jeremy | Stairs |
|  | Alfred | Kannampuzha |
|  | Beston | Leung |
| 4 | Wei-Kang | Tseng |
|  | Runjie | Shi |
|  | Haowei | Zhang |
|  | Tianchang | Shen |
| 5 | Johnson | Zhong |
|  | Zhuowei | Zhang |
| 6 | Site | Cao |
|  | Bowen | Gao |
|  | Haiqi | Xu |
|  | Zining | Zhu |
| 7 | Jordan | Lee |
|  | Jason | Piao |
|  | Javed | Nissar |
|  | Gary | Shin |
| 8 | Abhina | Sreeskantharajan |
|  | Tanushree | Podder |
|  | Nayeong | Park |
|  | Brameyaa | Ragupathy |
| 9 | Arnav | Goel |
|  | Anurag | Agarwal |
|  | Dhanyaa | Sudarsan |
|  | Harshun | Shukl |
| 10 | Moiz | Ganjifrakwala |
|  | Akshay | Kawlay |
|  | Kristofer | Sobon |
|  | Sheran | Cardoza |
| 11 | Rui Cheng | Zhang |
|  | Zhi Yang | Pan |
|  | Zidong | Weng |
|  | Newton | Xu |
| 12 | Navid | Korhani |
|  | Parimal | Edke |
|  | Armin | Nooshabadi |
| 13 | Lisa | Weng |
|  | Kelly | Zhang |
|  | Jack | Wong |
|  | Pakki | Lam |
| 14 | Nikita | Tsytsarkin |
|  | Alexei | Zenin |
|  | Nart | Barileva |
| 15 | Wenxin | Chen |
|  | Sally | Lee |
|  | Charlie | Yang |
|  | Netra | Unni |
| 16 | Xin | Geng |
|  | Yuzhang | Liao |
|  | Katherine | Huang |
|  | Chenyu | Wang |
| 17 | Imran | Ariffin |
|  | Muhammad | Aamir |
| 18 | Ken | Chen |
|  | Marian | Daniel |
|  | Tim | Fei |
|  | Shunzhe | Yu |

Day 1 Room Allocation:

|  |  |
| --- | --- |
| **Team #** | **Room #** |
| 1 | BA B024 |
| 2 | BA B026 |
| 3 | BA 1160 |
| 4 | BA 1200 |
| 5 | BA 1210 |
| 6 | BA 1220 |
| 7 | BA 1230 |
| 8 | BA 1240 |
| 9 | BA 2135 |
| 10 | BA 2185 |
| 11 | BA 2195 |
| 12 | BA 3008 |
| 13 | BA 3012 |
| 14 | BA 3116 |
| 15 | BA 1170 |
| 16 | BA 1180 |
| 17 | BA 1180 |
| 18 | BA 1170 |

##### **Overview**

In this competition, you will be working with City of Toronto ‘shapefiles’ which contain geospatial vector data representing the City of Toronto’s roadways (and buildings) to solve a classic computer science problem and expose your solution to the public as a service via a web-app.

Navigate to the Toronto Open Data catalogue and download the three datasets (in WGS84 format): “[Toronto Centrelines (TCL)](http://www1.toronto.ca/wps/portal/contentonly?vgnextoid=9acb5f9cd70bb210VgnVCM1000003dd60f89RCRD&vgnextchannel=1a66e03bb8d1e310VgnVCM10000071d60f89RCRD)”, “[Intersection File](http://www1.toronto.ca/wps/portal/contentonly?vgnextoid=e659522373c20410VgnVCM10000071d60f89RCRD&vgnextchannel=1a66e03bb8d1e310VgnVCM10000071d60f89RCRD)”, and “[Address Points (Municipal)](http://www1.toronto.ca/wps/portal/contentonly?vgnextoid=91415f9cd70bb210VgnVCM1000003dd60f89RCRD&vgnextchannel=1a66e03bb8d1e310VgnVCM10000071d60f89RCRD)”. Use this data to build a graph with intersections as nodes and streets (centrelines) as edges. For the purposes of this competition, you will only consider the following centerline types:

* 201200 Major Arterial Road
* 201201 Major Arterial Road Ramp
* 201300 Minor Arterial Road
* 201301 Minor Arterial Road Ramp
* 201400 Collector Road
* 201401 Collector Road Ramp
* 201500 Local Road
* 201600 Other Road
* 201601 Other Ramp
* 201700 Laneways
* 201800 Pending

You will ignore the rest, including trails. You may also ignore the fact that streets are one-way (e.g. you may traverse them backwards).

Part 1 is a classic algorithm problem with an optimal solution. You are provided two input files, one short and one long containing 10 and 100 test cases respectively, and must produce corresponding output files with the solutions to each of the test cases in both files.

Part 2 is more software-engineering orientated. You will extend the solution in part 1 into a practical implementation and create and polish a final product.

Finally, in part 3, you augment your web-app with a new feature targeting at a specific audience using another data set. It is purposefully left very open so that you may define your own problems, stakeholders and requirements, and make design decisions. We encourage you to be creative!

**An admissible solution only requires solutions for part 1 and part 2.**

## Problem Statement

## Part 1 Shortest Path

A common and very intuitive usage for a graph is to find a path between two nodes. Generally, there can be multiple paths to get from one node to another, but one of the paths of most interest is the shortest. In this part, you will learn about graphs and graph traversal by implementing a classic shortest path algorithm called ‘Dijkstra’s Shortest Path Algorithm.’ Write a function that takes two intersections and computes the length of the shortest path between them, as well as the path itself. Note that the output is only the shortest path length, but the path itself will be required in part 2.

### Input

The first line will give the number of test cases, T. T test cases follow. For each test case there will be two lines:

1. The start centerline intersection geo id
2. The end centerline intersection geo id

Note: two of such input files are going to be given (one has more test cases than the other to test efficiency of your algorithm), so two output files (output\_short.txt, output\_long.txt) will be expected for the submission. See Deliverables for details.

### Output

For each test case, output one line indicating the length of the shortest path in kilometers to get between the intersections

### Sample

|  |  |
| --- | --- |
| Input  2  13463981  13463893  13467033  14017894 | Output  12.9490530362  1.42154041893 |

Tips

* Do not try to write your own graph class. Use a library/module.
* Use another library/module to read the shapefiles
* Read the READMES of each dataset to understand what each field of a record means!
* The haversine formula can be used to calculate distances between two points on a sphere. Use R= 6371.0km

Restrictions

* You may not use a shortest path algorithm from a library or module

## Part 2 Building a Web-App

Writing an optimal algorithm and making a useful application are two different things. For instance, most people don’t tend to live on intersections and interpreting a list of numbers to determine a path is not very easy. Additionally, optimal solutions, although nice, can also be very computational expensive and take a long time to compute, but heuristic solutions that are not guaranteed to be optimal are often times ‘good enough’ and they can be computed much faster. For example, the A star algorithm (A\*) is a modification on Dijkstra’s algorithm that makes heuristic assumptions to find a short path quicker.

**In this section, you will write a web-application that allows a user to enter two human-readable addresses and outputs a list of instructions to get from the first address to the second and the length of the path.** You may optionally display the path via any other medium (e.g. points/lines on google maps). The final output path does not have to be an optimal path. You can use your implementation from part 1, but you may also modify it to create A\* (but keep the two functions/methods separate). **The final deliverable is a full web-application including back-end and front-end code. Also provide a README for installing dependencies and running your application.**

You may use any language, frameworks and tools. The format of the instructions and the user interface is up to you, but a sample direction format is provided below. Marks will be awarded to clean, usable interfaces, code quality and documentation.

|  |
| --- |
| From [a1] turn [LEFT/RIGHT] onto [STREET]  Head [NORTH/SOUTH/EAST/WEST] along [STREET] to [INTERSECTION]  Turn [LEFT/RIGHT] onto [STREET]  Head [NORTH/SOUTH/EAST/WEST] along [STREET] to [INTERSECTION]  …  [a2] is on the [LEFT/RIGHT] |

Figure Sample direction format

Tips

* Make extensive use of jQuery.
* Each address point actually sits on an edge between two nodes (intersections). Each address point has meta data for how far down the street it is in meters from the FNODE.
* Heuristically determine which pair of start and end nodes minimizes the total path length by applying 3D Vector math and using latitudes and longitudes (see Haversine formula)
* Think about the scalability of your application, and what will happen if multiple people use it

## Part 3 (Bonus)

Make use of the City of Toronto Open Data catalogue data or any other data set (e.g. [Bicycle Stations](http://www1.toronto.ca/wps/portal/contentonly?vgnextoid=ad3cb6b6ae92b310VgnVCM10000071d60f89RCRD&vgnextchannel=1a66e03bb8d1e310VgnVCM10000071d60f89RCRD) and [Bikeways](http://www1.toronto.ca/wps/portal/contentonly?vgnextoid=9ecd5f9cd70bb210VgnVCM1000003dd60f89RCRD&vgnextchannel=1a66e03bb8d1e310VgnVCM10000071d60f89RCRD) data) to add a feature for bikers to your web application. The design and functionality of the web app is totally up to you, so be creative!

# Detailed Requirements

## Deliverables

Your submission **MUST** include the following:

* output\_short.txt and output\_long.txt containing solutions to the test cases for parts 1
* zip copy of your source code
* all executable files
* A README detailing dependencies, installing and executing procedures
* Presentation Slides
* A few screenshots of your application

**Note:** DO NOT include the shapefile data in your submission

Deliverables should be sent to [programming@utek.skule.ca](mailto:programming@utek.skule.ca) before 6:00 PM. The title of the email should be “Programming 2016 Team X”. Put your team number in place of the ‘X’. Only 1 submission per team is required. A late penalty of 10% will be applied for each 5 minutes the deliverables are sent late. At 6:50 PM teams who have not submitted their deliverables will be disqualified.

The README must include information on how to run the executable file and how to compile the source code, including, if necessary, setting up your environment and installing third-party dependencies.

## Rules and Constraints

1. Cite all third-party libraries in the README of your code
2. Groups cannot inter-collaborate
3. A maximum of 4 people per team
4. Solutions to part 1 cannot use a shortest path algorithm from a library/module

# Presentation Guidelines

Participants are required to deliver a 7 minute presentation (+ 5 minute Q&A, so 15 minutes in total) on their program. The presentation should contain (but is not limited to):

* Major decisions and their justification
* Brief overview of algorithm and optimizations
* Critique of the design process
* Demonstration of a working version of the app
* Q&A with the judges

**NOTE:** You will be provided with a projector and a screen for you presentation. However if you plan on developing your program on a mobile platform, you are advised to install your program on a number of devices so that you can give your devices to each of the judges

# 

# Judging Criteria

## Scoring

The scoring criteria will be used to assess the ‘correctness’ of the application and to qualify the app for presentation (see rubric).

1. Participants must demonstrate correctness and efficiency of part 1 by providing output of the test cases in the files provided.
2. Participants must show a final app that takes as input a start location and an end location and outputs directions to get from start to end.

## Rubric

The apps shall be judged in 5 main areas:

1. **Correctness** of Code
2. **Quality** of Code
3. **Documentation** of Code
4. **Design** of Application
5. **Presentation** of Application

The **correctness** relates to whether the app is syntactically correct and is able to run and output the correct answer. The **Quality** relates to code cleanliness. **Documentation** is how well the code is described via comments. The **design** of the app is a wide-field relating to decisions made about the visuals, interaction, organization, etc. of the app. Finally, the **presentation** is the communication of the app’s utility, functionality and design (your design decisions).

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Criteria** |  |  |  |  | **Total** |
| **Correctness                                                                                                                                                                          /4** | | | | | |
| **Output (/2)** | Output is exactly correct for all test data | Output is mostly correct except for corner/special cases | Output is correct for simple cases | Output is not correct |  |
| **Efficiency (/2)** | Code is most optimal and able to run on large data sets | Code is able to run on normal sized data sets | Code can only run on small data sets | Code can only run on very small data sets |
| **Quality                                                                                                                                                                                  /4** | | | | | |
| **Readability (/1)** | Code is well indented and spaced and easy to follow and trace | Code is mostly well indented and spaced and easy to follow and trace | Code is indented and spaced | Code is not indented or spaced |  |
| **Modularity (/1)** | All processes are logically subdivided into reusable modular components (classes, functions) that are then linked together | Most code is logically divided into reusable, modular components. | Some code is logically divided into reusable, modular components. | Code is not modularized, multiple processing steps are conducted in a single method/function and code is not re-usable at all. |
| **Naming and Naming Conventions (/1)** | All methods and variables are named in a logical, and in an understandable way (pertaining to its role) and moreover, a strict spelling convention is always followed | Most methods and variables are named logically and a strict spelling convention is always followed | Some methods and variables are named logically and a spelling convention is mostly followed | Method and variable names are arbitrary and there is no spelling convention |
| **Error Handling (/1)** | All possible exceptions are caught and handled. All possible errors are checked and app fails gracefully with an error message for the user and log for the developer | All possible exceptions are caught and handled and app fails with a graceful message for the user | Most possible exceptions are caught and handled and app fails with a graceful message for the user | No error handling |
| **Documentation                                                                                                                                                                      /2** | | | | | |
| **Docstrings and in-line comments** | A strict and informative convention is consistently followed for methods and classes. All methods are commented with excellent explanations of functionality, parameters and return values. In-line comments effective and included where appropriate | All methods are commented with good explanations of functionality, parameters and return values. In-line comments are used well. | Most methods are commented with explanations of functionality, parameters and return values. | No comments or ineffective comments |  |
| **Design                                                                                                                                                                                     /8** | | | | | |
| **Usability (/2)** | User interface is clean and easy to read. App is simple and intuitive to use. | User interface is mostly clean and easy to read. App is mostly simple and intuitive to use | User interface is clunky and busy. App takes time to understand and use | User interface is messy with small, difficulty font. App is difficult to use - in entering finding functionality, entering input, and getting output |  |
| **Visuals and Aesthetics (/2)** | Excellent and appropriate use of images. App is very visual appealing and guides the eye. | Good use of images. App is visually appealing | Satisfactory use of images. App is usable | App is clunky. Difficult to find functionality. |
| **Creativity and Functionality (/4)** | Implemented new, creative designs and ideas using new data sets | Implemented new, creative ideas | Implemented some of the provided ideas | Only implemented required part 4 |  |
| **Presentation                                                                                                                                                                         /4** | | | | | |
| **Organization (/1)** | Presentation begins with overview of content and schedule, has distinct and well-timed introduction, content and conclusion and ends on time | Presentation begins with overview and schedule, has distinct introduction, content and conclusion and ends on time | Presentation has introduction, content and conclusion and ends on time | Presentation is disorganized with content brought up randomly out of order and context and presenters had to be cut-off |  |
| **Body Language (/1)** | Movements are fluid and help with articulation and visualization | Movements are mostly fluid and help with articulation and visualization | Slight nervousness and stiffness | Nervous movements, stiffness that distract from presentation |
| **Presentation Content (/1)** | Presentation introduces app, explains need for app and its functionality, the design process and design decisions, solutions to algorithmic problems and has conclusion | Presentation introduces app, explains need for app and its functionality, solutions to algorithmic problems and has conclusion | Presentation introduces app and explains its functionality | Explains functionality of app |
| **Grammar, Vocabulary and Sentence Structure (/1)** | No grammar, sentence structure. Appropriate and well-defined vocabulary is consistently used | Minimal grammar, sentence structure and word error. Appropriate and well-defined vocabulary is mostly used | Some grammar, sentence structure and word error. Appropriate and well-defined vocabulary is sometimes used | Many grammar and sentence structure errors and incorrect or undefined words are used to reference things. |
| **Total** | | | | |  |