2021 UofT
Engineering Kompetitions

Making it easier to dream bigger

PROGRAMMING

COMPETITOR'S PACKAGE

University of Toronto Engineering Kompetitions
January 16, 2021 - January 17, 2021

Directors: JHANAVI GERA and YI FEI TANG
Contact Us: programming@utek.skule.ca

Thank you to our sponsors:

Table of Contents

1.0 Introduction

Welcome to the 20th University of Toronto Engineering Kompetitions and thank you for choosing to participate! UTEK 2021 aims to provide students with an opportunity to apply class knowledge and acquired skills to real-world problems. It is a chance to challenge oneself, meet like-minded peers, and network with company professionals. We hope you enjoy UTEK 2021!

In this competition, you will solve a programming challenge, compete with other teams, and end with a presentation on your solution. Teams are evaluated based on practicality, ease-of-use, and capability of their coded solution. We are looking forward to your participation. Let the games begin!

1.1 UTEK Theme

This year, the UTEK theme is *making it easier to dream bigger,* as we challenge competitors to think about accessibility issues in a variety of industries and how we can address them either through accessible or universal design.

Accessible design is a design process in which the needs of people with disabilities are specifically considered. Accessibility sometimes refers to the characteristic that products, services, and facilities can be independently used by people with a variety of disabilities.[1] Universal design is a broader concept defined by The Center for Universal Design at North Carolina State University as "the design of products and environments to be usable by all people, to the greatest extent possible, without the need for adaptation or specialized design." It is meant to benefit all people at little or no cost.[2] This year, each competition will contain some component of accessibility either through the problem statement itself or the sections of the rubric. We are so excited to see how you incorporate accessibility considerations during the competition!

---

[1] https://www.washington.edu/accesscomputing/what-difference-between-accessible-usable-and-universal-design#:~:text=Accessible%20design%20is%20a%20design,with%20a%20variety%20of%20disabilities.

[2] the design of products and environments to be usable by all people, to the greatest extent possible, without the need for adaptation or specialized design

2.0 Competition Schedule

The competition will be delivered online this year. The problem statement will be released on the day of the competition. The competition will be conducted using Quercus. The coding section will be made available from 11 to 6pm EST. The team will select a team member to start the Quercus quiz and submit all the files. Once that team member starts the quiz, the team **will have 7 hours to complete the coding challenge and submit their code**. The presentation should take 1-2 hours to complete. Presentation submission will be done using a quercus quiz which will be available from 6pm to 8pm. The schedule will be as follows:

Day 1: Saturday, Jan 16, 2020

Time	Event
9:00 - 10:00	Opening Ceremonies with Keynote Speaker
10:00 - 10:45	Company Networking
10:45 - 11:00	Competition Problem Statement Briefings
11:00 - 6:00 PM	Work Time - Coding
6:00 - 8:00 PM	Work Time - Presentations

Day 2: Sunday, Jan 17, 2020

Time	Event
9:30 - 10:00 AM	Judge Presentation Briefing
10:00 - 12:00 PM	Presentations
12:00 - 1:00 PM	Lunch
1:00 PM - 3:00 PM	Presentations
3:00 - 4:00 PM	Awards & Closing Ceremony 1st place: $80/team member 2nd place: $60/team member 3rd place: $30/team member

4:00 - 5:00 PM	Judge Feedback & Networking OEC Information for Winning Teams

3.0 Rules

- Students from **all years** can participate in the competition.
- You can **only** collaborate with the members of your team.
- You can use **any programming language and its standard libraries** to solve the given problem (eg. if there is a Python package needed to be installed that does Breadth First Search for you, it cannot be used. If there is a standard list sorting function, it can be used)
- There can be a **maximum of 4 University of Toronto students** in a team.
- In order to participate in the 2021 Ontario Engineering Competition, at least half of the team must comprise engineering students. In order to participate in the Canadian Engineering Competition, the entire team must comprise engineering students.
- You are allowed to use online resources and the resources mentioned in the zip file provided to you. All the online resources used by competitors must be cited in a references slide at the end of the presentation (in whichever academic citation format they select). Competitors are not permitted to submit work completed by anyone other than the members of their team. If they decide to recycle their own or someone else's code it must be **clearly cited** in the presentation.

4.0 Problem Background

According to recent studies, many Canadians feel accessible transportation is a major issue. [1] A prominent problem is the lack of access to subway stations. [2] Until all subway stations are accessible, we can create a service that plans trips that are accommodating to individuals with specific needs.

5.0 Design Problem/Goals

As a TTC employee, you are tasked with finding out how accessible people's commutes are, as well as creating custom paths that emphasize accessible stations.

In order to achieve the highest number of possible points, the most optimal algorithm must be implemented (eg. Quicksort is more efficient than Bogosort and will receive more points). Points will be removed for redundant code. You may find the recorded workshop video about pathfinding a useful starting point.

Part 1: Determine the accessibility of the nodes in a path - 20 Points
a) You decide the best place to start is by handling basic inputs and outputs. You want to figure out which nodes are accessible and which nodes are not. Given a set of subway stations, output the subway stations that are accessible. The format of the input file will be:

1a.json

```
{
  "Nodes":
    [{
    "Name": "Yonge and Dundas",
    "Accessible": false,
    "Neighbours": ["College"]
    },
    {
    "Name": "College",
    "Accessible": true,
    "Neighbours": ["Yonge and Dundas", "Queens"]
    },
    {
    "Name": "Queens",
    "Accessible": true,
    "Neighbours": ["College"]
    }]
}
```

In this case, the output file should read like so:

1a.out
College, Queens

b) Now that you can find out which individual stations are accessible, your job is to write a function that determines the top k most accessible paths given a set of paths. This means the path that has the highest ratio of accessible stations to non-accessible stations. The input will be formatted like so:

```
{
  "Paths": [
    {
      "PathName":"Path 1",
      "Nodes":
      [{
        "Name": "Yonge and Dundas",
        "Accessible": false,
        "Neighbours": ["College"]
      },
      {
        "Name": "College",
        "Accessible": true,
        "Neighbours": ["Yonge and Dundas", "Queens"]
      },
      {
        "Name": "Queens",
        "Accessible": true,
        "Neighbours": ["College"]
      }]
    },
    {
      "PathName":"Path 2",
      "Nodes":
      [{
        "Name": "Yonge and Dundas",
        "Accessible": false,
        "Neighbours": ["Node5"]
      },
      {
        "Name": "College",
```

```
            "Accessible": false,
            "Neighbours": ["Node4", "Node6"]
        },
        {
            "Name": "Queens",
            "Accessible": false,
            "Neighbours": ["Node5"]
        }]
    },
    {
        "PathName":"Path 3",
        "Nodes":
        [{
            "Name": "Yonge and Dundas",
            "Accessible": true,
            "Neighbours": ["Node5"]
        },
        {
            "Name": "College",
            "Accessible": false,
            "Neighbours": ["Node4", "Node6"]
        }]
    }
  ]
}
```

For your output file, run your function with the value of k being set to 3. For this example, the output file should read like so (where Path 1 is first place and Path 2 is third place):

1b.out

Path 1, Path 3, Path 2

Part 2: Find the shortest path given a set of points - 40 Points

In this section, you are given a series of nodes in a 2.json. For simplicity, the coordinates will represent an x and y value on a theoretical map. Each node will be accompanied with a list of neighbours. The “Distance” field represents the distance the node is from the neighbour. You want to find the shortest path from one location to another, minimizing the total distance. A list of these nodes will be given in 2.in. You may not use a library implementation of pathfinding.

Format of 2a.json:

```
{
 "Nodes":
    [{
    "Name": "Yonge and Dundas",
    "Accessible": false,
    "Coordinates": [100,200],
    "Neighbours": [{"Name":"College", "Distance":15},{"Name":"Queens", "Distance":30}]
    },
    {
    "Name": "College",
    "Coordinates": [200,250],
    "Accessible": true,
    "Neighbours": [{"Name":"Yonge and Dundas","Distance":15}, {"Name":"Queens","Distance":8}]
    },
    {
    "Name": "Queens",
    "Coordinates": [389,240],
    "Accessible": true,
    "Neighbours": [{"Name":"College","Distance":8},{"Name":"Yonge and  Dundas", "Distance":30}]
    }]
}
```

The list of paths will be a start location and end location separated by a comma. It will look like this:

2.in

Yonge and Dundas,College
Queens,Yonge and Dundas

In this case, you would need to find the path between Yonge and Dundas, and College, as well as the path between Queens and College. Your 2.out should look like the following where each row follows the format:
Source, Path, To, Destination, Total Distance

2.out

Yonge and Dundas, College, 15
Queens, College, Yonge and Dundas, 23

Part 3: A path that is both short and accessible - 30 Points
The format of the input and output files for this section is the same as in part 2 (the test cases will be provided in 3.in). However, in this section you must also take into account the accessibility of the path. The goal is to find the best path that is both short and accessible. Every node in your path that is not accessible will result in an addition of 5 distance points. If more than 50% of the nodes are inaccessible, either the total points will be doubled or the previous rule will apply (whichever is a greater addition). The total distance value of your 3.out file must account for these deductions. You will need to make the necessary tradeoffs in your algorithm. You may not use a library implementation of pathfinding.

*Note: Although the Subway station names are real the way they are connected are randomly generated and do not reflect Toronto maps.

Part 4: Travelling Salesman Problem - 30 Points (BONUS Points)
Given a list of subway stations, find the shortest route that is able to reach every single station exactly once: starting and ending with the first station. You may visit subway stations more than once. You may not use a library implementation of the TSP. Your output should follow the format Start, List Of Stations, End, Path Cost. Accessibility does not need to be considered, and the format of 4.json is identical to 2.json and 3.json. However, again the paths connecting them will be randomly generated.

4.in
College,Bathurst,Spadina
Dundas,Bloor,St. George

4.out
College, Queen's Park, Bathurst, Yonge, Spadina, College, 15
Dundas, Bloor, St. George, Bloor, Dundas, 10

6.0 Competition Deliverables

You are responsible for submitting your code, input and output files in a zip file on the Code section of the Quercus quiz before your deadline. The zip file should be named TEAM_NUMBER.zip. The README file should contain instructions on how to run each part of your code. For example if the team programmed in Python, the zip file should contain:

1a.py, 1b.py, 2.py, 3.py, 1a.out, 1b.out, 2.out, 3.out, 1a.json, 1b.json 2.json, 3.json, 2.in, 3.in, 4.json, 4.in, 4.out, README.

Failure to submit all the files in a correct format will result in deduction of points.

You are furthermore required to create a 10 minute presentation. Your presentation slides along with your code, should be submitted on the Presentation section of the Quercus quiz. If you submit multiple times, then your most recent submission will be considered. In case the participants face a problem during submission they should immediately contact the Programming Co-Directors through the Quercus discussion board.

	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.

7.0 Rubric

Teams will be responsible for uploading their solution to each section of the problem, as well as the outputs to their programs in a file. We will use scripts that will grade each team by parsing through their outputs. Representatives from OEC will judge the presentations and look through the code. There will be a penalty for plagiarized content, absent teammates and late submissions.

Competition Director's Rubric

Category	Points	Details
Question 1	20	10 points for successfully passing each section.
Question 2	40	30 points for finding the shortest path. 10 points for optimal implementation.
Question 3	30	Teams will be ranked by how short the route is (30 points for first place, 25 for second place, 20 for third place, the rest receive 10 points).
Question 4 (BONUS)	30	Teams will be ranked by how short the route is (30 points for first place, 25 for second place, 20 for third place, the rest receive 10 points).

Code Quality	10	Code quality is scored based on readability, modularity and error-handling. Documentation of the code will also be considered.
Total	100	130 total possible points

Judge's Rubric

Criteria	4	3	2	1	Total
Correctness					**/12**
Syntax and bugs	Code compiles without errors or warning and no runtime error occurs	Code compiles without errors or warnings and non-fatal runtime errors occur	Code compiles with errors and runtime errors occur	Code does not compile	
Output	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	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					**/16**
Readability	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	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	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	All possible exceptions are caught and handled or all possible errors	Most possible exceptions are caught and handled	Some exceptions are caught and handled	No error handling	

Presentation					**/16**
Organization	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	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	Presentation clearly explains the projects, gives appropriate examples and each third party code is cited properly. All the questions asked on the content are answered properly.	Presentation explains the projects, gives examples.Some citations are not done properly. All the questions asked on the content are answered properly.	Presentation explains the projects, gives appropriate examples and the citations are not done properly. Some of the questions are answered properly.	Presentation fails to explain the project properly,no examples are provided and the citations are not done properly. None of the questions are answered properly.	
Grammar, Vocabulary and Sentence Structure	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 /44					

8.0 References

[1]
https://www.rickhansen.com/news-stories/blog/canadians-see-massive-gaps-accessibility-people-disabilities

[2]
https://www.blogto.com/city/2018/11/ttc-subway-parody-map-highlights-accessibility-issues/