UTEK

SUSTAINABLE CITIES
& COMMUNITIES

PROGRAMMING KOMPETITION PACKAGE

University of Toronto Engineering Kompetitions

November 11-12, 2023

Table of Contents

Please click the title of the section to head to it!

Programming

Please join the UTEK Competition Slack here, and the UTEK Programming Channel here!

PROBLEM STATEMENT

In the heart of Emerald City, where towering skyscrapers touched the clouds and bustling streets pulsed with energy, a new challenge had arisen. The visionary Mayor, known for her commitment to sustainability, had set her sights on transforming the city's energy landscape. Fossil fuels no longer held a place in her vision; instead, she sought to harness the power of technology to pave the way towards a greener, more sustainable future.

Enter our team of expert programmers, a group handpicked for their prowess in the intricate world of coding. Charged with the monumental task of revolutionizing Emerald City's power infrastructure, they eagerly embarked on a journey to optimize the city's electricity distribution system. The mayor's vision was clear – eliminate the reliance on fossil fuels, and usher in a new era of clean, efficient energy. The team knew that the challenge was daunting, but they were determined to turn Emerald City into a beacon of sustainability.

The programmers dove deep into the data, analyzing consumption patterns, peak hours, and potential bottlenecks. The goal was not just to shift from fossil fuels to renewable sources but to do so in the most efficient and cost-effective manner. They envisioned a network where energy flowed seamlessly, adapting to the city's needs with the precision of a well-tuned symphony.

PROBLEM DESCRIPTION

The problem has been broken down into parts. Once you complete all parts, a solution to the main problem will be formed. The Mayor of Emerald City and her directors will then judge your solution.

Part 1: Graph Construction

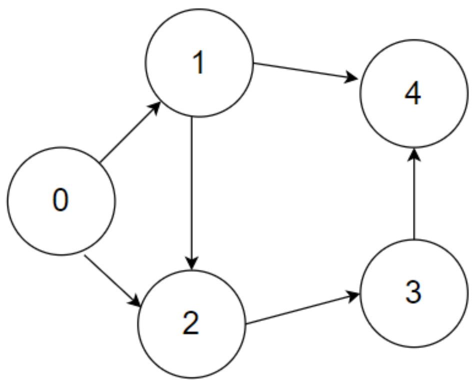
In this subproblem, you are given an input of intersections in pairs, separated by the arrow "->" and comma "," characters. An example input is shown below:

> ***Input:*** *a->b, b->c, c->d, d->b*

This input can be visualized as a graph where one 'node' points to another 'node', where all of these pairs are connected in a network. Your job is to parse through this input, and create and output a resulting *adjacency matrix/list*. The images on the following page show how these structures work, tracking the connections between nodes in a *graph data structure*.

You may visit https://mathworld.wolfram.com/AdjacencyMatrix.html for more information about how adjacency matrices and lists work.

Adjacency Matrix



	0	1	2	3	4
0	0	1	1	0	0
1	0	0	1	0	1
2	0	0	0	1	0
3	0	0	0	0	1
4	0	0	0	0	0

Your final step is to output this in an organized fashion, up to your own discretion (the more organized, the better score you will receive). Be aware that intersection names may not only be from "a" to "d" and can vary case to case. Part marks will be provided if only some constraints are met.

Example Input:

a->b, b->c, c->d, d->b

// Assume we are using the adjacency matrix method

Example Output:

```
      a  b  c  d
___________________
a |  0  1  0  0
b |  0  0  1  0
c |  0  0  0  1
d |  0  1  0  0
```

Part 2: Optimal Path

Now, the mayor mentions that traveling from intersection to intersection requires a certain amount of fuel. He gives you an input of intersections, now with a weight associated with each pair of intersections representing the cost of fuel to travel between them. Assume that when an input such as "a->b ($2)" is given, this means it costs 2 dollars to travel FROM intersection a TO intersection b. An example input is shown below:

Input: *a->b ($4), b->c ($5), c->d ($3), d->b ($7), a->c ($4), d->a ($1)*

Your job is to optimize the fuel consumption/cost, while also visiting the least amount of intersections. In other words, visit all intersections (i.e. visit a, b, c, and d) given a starting and ending intersection name (i.e. start at "a" and end at "d") and ensure the fuel costs to travel from each intersection is minimized as well. Assume that you have to continue your traversal using the last intersection you stopped at.

You may find it helpful to use your adjacency matrix/list from before, as well as any new data structures to develop an algorithm/program for this problem.

Example Input 1:

Starting intersection: "a"

Ending intersection: "b"

a->b ($4), b->c ($5), c->d ($3), d->b ($4), a->c ($4), d->a ($1)

Example Output 1:

Best path: a->c, c->d, d->b

Cost: $11

// a->b is not a valid path since we need to visit all intersections

// a->c, d->b is not valid as we stopped at c in our first path, and need to continue from c for the

// next path

// a->b, b->c, c->d, d->b is not optimal as it costs $16

Example Input 2:

Starting intersection: "a"

Ending intersection: "d"

a->b ($4), b->d ($5), c->d ($3), d->a ($4), a->c ($4), d->a ($1), c->b ($1)

Example Output 2:

Best path: a->c, c->b, b->d

Cost: $10

// a->b, b->d, d->a, a->c, c->d is not the most optimal path as it costs $16 to visit all intersections

Part 3: Time Constraint

The mayor calls you at 2AM because she found out there is a new issue. There is also a cooldown required when visiting each intersection. So, rather than simply calculating the cost to travel from one to another, there is also a cooldown period after doing so.

You need to make a modified algorithm from the previous parts, this time finding the minimum cost to get power from a start intersection to an ending intersection within a certain amount of time. If it is not possible to get power to the end location within the time given, return an empty array for the path and then -1 for any numeric values.

Example inputs and outputs are shown below:

Example Input 1:

Starting intersection: "a"

Ending intersection: "d"

maxTime = 5min

a->b ($4, 1 min cooldown) , b->c ($5, 1 min cooldown) , c->d ($3, 3 min cooldown) , b->d ($7, 3 min cooldown), a->c ($4, 3 min cooldown), d->a ($1, 4 min cooldown)

Example Output 1:

bestPath: a -> b -> c -> d

totalCost: $12

totalTime: 5min

Explanation: Although the cheapest path is a -> c -> d, costing $7, it violates the time constraint of 5 minutes

Example Input 2:

Starting intersection: "a"

Ending intersection: "d"

maxTime: 2min

a->b ($4, 3 min cooldown), b->d ($5, 10 min cooldown), c->d ($3, 9 min cooldown), d->a ($4, 7 min cooldown), a->c ($4, 1 min cooldown), c->b ($1, 1 min cooldown)

Example Output 2:

Best path: [ ]

Cost: -1

Time: -1

DESCRIPTION OF DELIVERABLES

Your submission must include a zip file with all your program files and your PowerPoint presentation. The zip file should be named TEAM_NUMBER.zip. The README file should contain instructions on how to run each part of your code.

Please submit the following:

1. All code and documentation you have written for the competition, including a README that contains running instructions. The code is due at 6:00 pm on Saturday. Submit your code as a .zip file through this form. Please note only one teammate should make the submission.
2. A *presentation* summarizing the key aspects of the problem, your approach, and your solution. The slides are due at 11:59 pm, and should be submitted through this form. Please note that due to space constraints, **only the top 5 teams will be presenting on Sunday.** You will be notified by us through email on **Saturday night** if you have been chosen along with the specific time/location.

Failure to submit files in the correct format or without the defined requirements will result in deduction of points.

In case the participants face a problem during submission they should immediately contact the Programming Co-Directors through this email: programming@utek.skule.ca.

	Time allocated	**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.

TESTING (INPUT/OUTPUT)

We will run your code through a total of 8 test cases, 3 will be public and 5 will be hidden test cases. You will be awarded points based on your correctness, efficiency, and quality of code (see the rubric on Page 10 and 11 for more details on grading).

Parts 1 to 3 have three test cases, which will be in files (i.e. 1a.in, 1b.in, 1c.in). One test case (a) will be provided at the beginning of the competition, the remaining two will be provided one hour before the end of the competition. There will be 5 hidden test cases that we will run on your code not provided to you.

RUBRICS

Teams are responsible for submitting their solutions through a zip file with all the program files and attaching their PowerPoint presentation. There will be 2 presentation judges. There will be penalties for plagiarism (ChatGPT, copying from other groups, etc.), missing teammates during presentation, and late submissions.

Presentation Judging Rubric

Category	**Points**	**Extra Details**
Explanation	50	Your solution should be explained in a detailed, clear way. The solution should be well-justified through examples. Competitors should be prepared to answer

		questions about their solution.
Presentation	30	The presentation should be easy to follow and flow naturally.
Visuals	20	The presentation includes meaningful visual aids.
Total	100	

Solution Grading Rubric

	4 Excellent	3 Great	2 Fair	1 Poor	0 Insufficient
Correctness (16 points)					
Syntax & bugs	Code compiles without errors or warning and no runtime error occurs	Code compiles without errors, but some warnings.	Code compiles without errors, but some warnings. Non-fatal runtime errors occur	Code compiles with errors and runtime errors occur	Code does not compile
Output	Output is correct for all test cases	Output is mostly correct, but not for some hidden edge cases	Output is correct for some cases, not for any edge/special cases	Output is correct for few test cases	Output is incorrect for all cases
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	Code cannot run on very small data sets.
Error Handling	All possible exceptions are caught and handled in code.	Most possible exceptions are caught and handled.	Some exceptions are caught and handled.	No exceptions are caught, but there is effort for error handling.	No error handlings in methods.
Quality of Code (12 points)					
Readability	Code is always	Code is mostly	Code is	Code is rarely	Code is not

	well-indented and spaced. Easy to follow and trace.	well indented and spaced. Easy to follow and trace	sometimes well-indented and spaced.	indented or spaced.	indented or spaced (unreadable).
Modularity	All processes are logically subdivided into reusable modular components (classes and functions) that are linked together	Most code is logically divided into reusable, modular components.	Some code is logically divided into reusable, modular components.	Little code is divided into modular components. Most steps are conducted in a single method/function.	Code is not reusable, no effort to modularize,
Naming	All methods and variables are named in a logical/intuitive and understandable way related to its context. A strict naming convention (snake_case, CamelCase, etc.) is always followed.	Most methods and variables are named logically and strict naming convention is always followed.	Some methods and variables are named logically and naming conventions are sometimes followed.	Methods and variables are rarely logically named and naming conventions are rarely followed.	Methods and variables are arbitrary and there is no naming convention