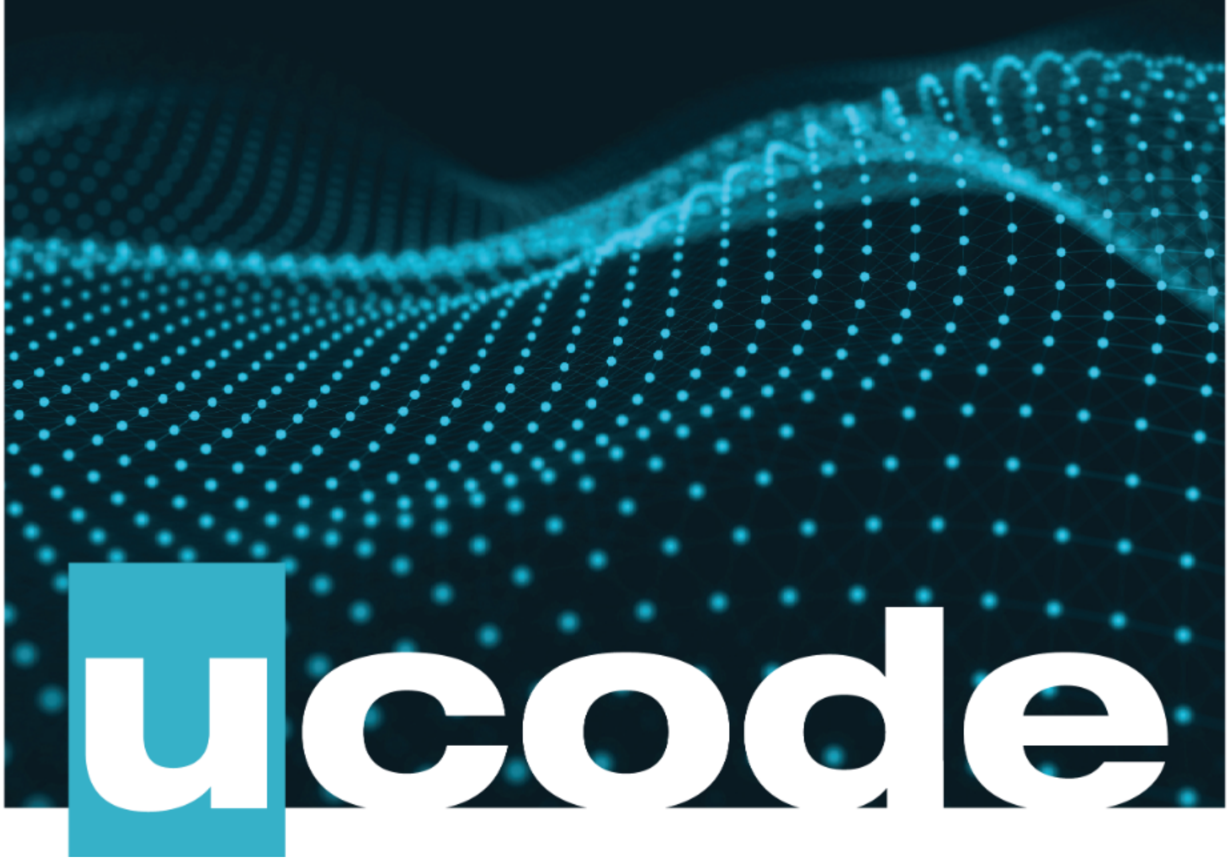
PathFinder

Track C

December 24, 2020



Contents

ucode

Engage

DESCRIPTION

Hi, challenger.

Finding the shortest path is a relevant topic at the present. This topic is widely used in everyday life: searching for the shortest path from one point to another, in GPS when analyzing traffic congestion and choosing the optimal route, in switching information packets on networks, and so on. There are many ways to apply this topic in life.

Very often a person has to visit several places, and for this, it is necessary to think over the optimal path. The pace of human life is often intense: work, occasions, business meetings, hobbies, etc. This is an important issue that needs automation. Nowadays, keeping up with everything, correctly allocating time and the optimal path is an exact science. During this challenge, you will improve your algorithmic skills and create an automation solution that helps to find the shortest between the points. Also, try out your library in this challenge and see what it is capable of.

Remember that this is an individual challenge, so you must develop all the code yourself. Of course, do not forget about consulting and brainstorming with your peers, use the power of P2P to the fullest. Your success is in your hands.

Good luck!

BIG IDEA

The development of algorithmic mindset.

ESSENTIAL QUESTION

What algorithms can help to use time and resources effectively?

CHALLENGE

Develop an algorithm that finds the shortest paths between points.

Investigate

GUIDING QUESTIONS

We invite you to find answers to the following questions. By researching and answering them, you will gain the knowledge necessary to complete the challenge. To find answers, ask the students around you and search the internet. We encourage you to ask as many questions as possible. Note down your findings and discuss them with your peers.

- What is a path?
- Is it possible to interpret the places and paths? How?
- What does the term shortest path mean?
- What is a graph?
- What is the graph theory?
- What is the use of graph theory in real life?
- How to find the shortest path between two places?
- What algorithms of shortest path search exist?
- What is combinatorics?
- Is it possible to use mathematical approaches of combinatorics in solving such problems?
- What does the term FIFO mean? How is it used?

GUIDING ACTIVITIES

Complete the following activities. Don't forget that you have a limited time to overcome the challenge. Use it wisely. Distribute tasks correctly.

- Read the story carefully. Find out what you have to do.
- Make a plan for this challenge.
- Learn the basic algorithms for finding the shortest paths in graphs.
- Read and understand the basic rules of combinatorics.
- Collaborate with other students to get a deeper understanding of the challenge. Learn from others and share your knowledge.
- Brainstorm with other students and find more than one solution.
- Clone your git repository that is issued on the challenge page.
- Start developing your program. Try to realize your thoughts in code.
- Read about header files and structures.
- Open Auditor and follow the rules.
- Test your code.

ANALYSIS

Analyze your findings. What conclusions have you made after completing guiding questions and activities? In addition to your thoughts and conclusions, here are some more analysis results.

- Be attentive to all statements of the story.
- The challenge must be performed in `C`.
- The challenge must have the following structure:
  - `src` directory contains files with extension `.c`. Or the `src` directory can consist of subdirectories with `.c` files
  - `obj` directory contains files with extension `.o` (you must not push this directory in your repository, only `Makefile` creates it during compilation)
  - `inc` directory contains header files `.h`. Or the `inc` directory can consist of subdirectories with `.h` files
  - `libmx` directory contains source files of your library including its `Makefile`. You must use it
  - `Makefile` that compiles the library `libmx` firstly and then compiles and builds `pathfinder`
- Complete the challenge according to the rules specified in the `Auditor`.
- Submit your files using the layout described in the story. Only useful files allowed, garbage shall not pass!
- Compile C-files with clang compiler and use these flags: `clang -std=c11 -Wall -Wextra -Werror -Wpedantic`.
- Pay attention to what is allowed. Use of forbidden stuff is considered a cheat and your challenge will be failed.
- Your program must manage memory allocations correctly. A memory that is no longer needed must be freed, otherwise, the challenge is considered incomplete.
- Oracle will check the memory leaks with the `leaks tool` on the macOS.
- Oracle will check the memory leaks with the `valgrind tool` on the Ubuntu.
- The solution will be checked and graded by students like you. `Peer-to-Peer learning`.
- Also, the challenge will pass automatic evaluation which is called `Oracle`.
- If you have any questions or don't understand something, ask other students or just Google it.

Act

ALLOWED FUNCTIONS

`malloc, malloc_size, malloc_usable_size, free, open, read, write, close, exit`

BINARY

`pathfinder`

DESCRIPTION

Create a program that finds all the shortest paths between all the islands, using information about the bridges that connect them. The program:

- reads data from a file. Its name is specified as a command-line argument
- finds all the shortest paths between each combination of two islands
- prints the paths using the FIFO (first) rule for the islands to the standard output. More examples can be found in the CONSOLE OUTPUT

Input file description:

- the first line in the file is the number of islands
- the remaining lines describe the distance between the two islands, one per line. Each line consists of two islands, the length of the bridge between them and a newline in a format:
  `island1-island2,length_of_bridge` :
  - the names of the islands contain only alphabetic characters and cannot be empty or identical
  - the length of the bridge contains only numbers, cannot be empty and has a positive value
  - the sum of the lengths of all bridges in the file does not exceed `INT_MAX`

The output consists of information blocks about a specific shortest path. Each block contains:

- `========================================` - the upper boundary that consists of 40 `=` characters
- `Path: <island1> -> <island2>` that shows the initial point and the final destination
- `Route: <island1> -> <all_islands_between> -> <island2>` that shows the full route between the two islands
- `Distance: <length1> + <length2> = <sum>` that shows the distance between every island in the route, as well as their sum that indicates the distance to the final destination
- `========================================` - the bottom boundary that consists of 40 `=` characters

Error handling. The program prints errors to the standard error stream `stderr` in the following order of priority:

- `usage: ./pathfinder [filename]` if there is an invalid number of command-line arguments
- `error: file [filename] does not exist` if the file does not exist
- `error: file [filename] is empty` if the file is empty
- `error: line 1 is not valid` if the first line contains something other than digits or has a non-positive value
- `error: line [line_number] is not valid` if one of the remaining lines does not match the format above
- `error: invalid number of islands` if the number received in the first line does not match with the number of islands
- `error: duplicate bridges` if there is more than one bridge between the islands
- `error: sum of bridges lengths is too big` if the sum of the lengths of all bridges in the file exceeds `INT_MAX`

CONSOLE OUTPUT

```
>./pathfinder | cat -e
usage: ./pathfinder [filename]
>cat -e islands
cat: islands: No such file or directory
>./pathfinder islands | cat -e
error: file islands does not exist
>cat -e empty
>./pathfinder empty | cat -e
error: file empty is empty
>cat -e invalid1
4f$
Greenland-Bananal,8$
Fraser-Greenland,10$
Bananal-Fraser,3$
Java-Fraser,5$
>./pathfinder invalid1 | cat -e
error: line 1 is not valid
>cat -e invalid2
4$
Greenland-Bananal,8$
Fraser--Greenland,10$
Bananal-Fraser,3$
Java-Fraser,5$
>./pathfinder invalid2 | cat -e
error: line 3 is not valid
>cat -e invalid3
3$
Greenland-Bananal,8$
Fraser-Greenland,10$
Bananal-Fraser,3$
Java-Fraser,5$
>./pathfinder invalid3 | cat -e
error: invalid number of islands
>cat -e invalid4
```

```
3$
Blade-Wizard,1$
Drive-Blade,1$
Wizard-Wizard,1$
>./pathfinder invalid4 | cat -e
error: line 4 is not valid
>cat -e invalid5
3$
Blade-Wizard,1$
Drive-Blade,1$
Drive-Blade,2$
Wizard-Drive,1$
>./pathfinder invalid5 | cat -e
error: duplicate bridges
>cat -e invalid6
3$
Blade-Wizard,1$
Drive-Blade,2147483647$
Wizard-Drive,1$
>./pathfinder invalid6 | cat -e
error: sum of bridges lengths is too big
>cat -e example1
4$
Greenland-Bananal,8$
Fraser-Greenland,10$
Bananal-Fraser,3$
Java-Fraser,5$
>./pathfinder example1 | cat -e
========================================$
Path: Greenland -> Bananal$
Route: Greenland -> Bananal$
Distance: 8$
========================================$
========================================$
Path: Greenland -> Fraser$
Route: Greenland -> Fraser$
Distance: 10$
========================================$
========================================$
Path: Greenland -> Java$
Route: Greenland -> Fraser -> Java$
Distance: 10 + 5 = 15$
========================================$
========================================$
Path: Bananal -> Fraser$
Route: Bananal -> Fraser$
Distance: 3$
========================================$
========================================$
Path: Bananal -> Java$
Route: Bananal -> Fraser -> Java$
Distance: 3 + 5 = 8$
========================================$
========================================$
Path: Fraser -> Java$
```

```
Route: Fraser -> Java$
Distance: 5$
========================================$

>cat -e example2
5$
A-B,11$
A-C,10$
B-D,5$
C-D,6$
C-E,15$
D-E,4$
>./pathfinder example2 | cat -e
========================================$
Path: A -> B$
Route: A -> B$
Distance: 11$
========================================$
========================================$
Path: A -> C$
Route: A -> C$
Distance: 10$
========================================$
========================================$
Path: A -> D$
Route: A -> B -> D$
Distance: 11 + 5 = 16$
========================================$
========================================$
Path: A -> D$
Route: A -> C -> D$
Distance: 10 + 6 = 16$
========================================$
========================================$
Path: A -> E$
Route: A -> B -> D -> E$
Distance: 11 + 5 + 4 = 20$
========================================$
========================================$
Path: A -> E$
Route: A -> C -> D -> E$
Distance: 10 + 6 + 4 = 20$
========================================$
========================================$
Path: B -> C$
Route: B -> D -> C$
Distance: 5 + 6 = 11$
========================================$
========================================$
Path: B -> D$
Route: B -> D$
Distance: 5$
========================================$
========================================$
Path: B -> E$
```

```
Route: B -> D -> E$
Distance: 5 + 4 = 9$
========================================$
========================================$
Path: C -> D$
Route: C -> D$
Distance: 6$
========================================$
========================================$
Path: C -> E$
Route: C -> D -> E$
Distance: 6 + 4 = 10$
========================================$
========================================$
Path: D -> E$
Route: D -> E$
Distance: 4$
========================================$
>
```

Document

DOCUMENTATION

One of the attributes of Challenge Based Learning is documentation of the learning experience from challenge to solution. Throughout the challenge, you document your work using text and images, and reflect on the process. These artifacts are useful for ongoing reflection, informative assessment, evidence of learning, portfolios, and telling the story of challenge. The end of each phase (Engage, Investigate, Act) of the challenge offers an opportunity to document the process.

Much of the deepest learning takes place by considering the process, thinking about one's own learning, analyzing ongoing relationships with the content and between concepts, interacting with other people, and developing a solution. During learning, documentation of all processes will help you analyze your work, approaches, thoughts, implementation options, code, etc. In the future, this will help you understand your mistakes, improve your work, and read the code.

At the learning stage, it is important to understand and do this, as this is one of the skills that you will need in your future job. Naturally, the documentation should not be voluminous, it should be drawn up in an accessible, logical, and connected form.

So, what must be done?

- a nice-looking and helpful README file. In order for people to want to use your product, their first introduction must be through the README on the project's git page. Your README file must contain:
  - Short description. This means, that there must be some info about what your project actually is. For example, what your program does.
  - Screenshots of your solution. This point is about screenshots of your project "in use".
  - Requirements and dependencies. List of any stuff that must be installed on any machine to build your project.
  - How to run your solution. Describe the steps from cloning your repository to the first launch of your program.
- a full-fledged documentation in any forms convenient for you. By writing this, you will get some benefits:
  - you have an opportunity to think through implementation without the overhead of changing code every time you change your mind about how something should be organized. You will have very good documentation available for you to know what you need to implement
  - if you work with a development team and they have access to this information before you complete the project, they can confidently start working on another part of projects that will interact with your code
  - everyone can find how your project works
- your documentation must contain:
  - Description of progress after every competed CBL stage.
  - Description of the algotithm of your whole program.

Keep in mind that the implementation of this stage will be checked by peers at the assessment!

Also, there are several links that can help you:

- Make a README
- How to write a readme.md file?
- A Beginners Guide to writing a README
- Google Tools - a good way to journal your phases and processes:
  - Google Docs
  - Google Sheets
- Dropbox Paper - a tool for internally developing and creating documentation
- Git Wiki - a section for hosting documentation on Git-repository
- Haroopad - a markdown enabled document processor for creating web-friendly documents
- Canva - a good way to visualize your data
- QuickTime - an easy way to capture your screen, record video or audio
- code commenting - source code clarification method. The syntax of comments is determined by the programming language
- and others to your taste

Share

PUBLISHING

Last but not least, the final stage of your work is to publish it. This allows you to share your challenges, solutions, and reflections with local and global audiences. During this stage, you will discover ways of getting external evaluation and feedback on your work. As a result, you will get the most out of the challenge, and get a better understanding of both your achievements and missteps.

To share your work, you can create:

- a text post, as a summary of your reflection
- charts, infographics or other ways to visualize your information
- a video, either of your work, or a reflection video
- an audio podcast. Record a story about your experience
- a photo report with a small post

Helpful tools:

- Canva - a good way to visualize your data
- QuickTime - an easy way to capture your screen, record video or audio

Examples of ways to share your experience:

- Facebook - create and share a post that will inspire your friends
- YouTube - upload an exciting video
- GitHub - share and describe your solution
- Telegraph - create a post that you can easily share on Telegram
- Instagram - share photos and stories from ucode. Don't forget to tag us :)

Share what you've learned and accomplished with your local community and the world. Use #ucode and #CBLWorld on social media.