

Title: Methods and Tools for Software Engineering
Course ID: ECE 650 Section 01
WWW: <https://ece.uwaterloo.ca/~agurfink/ece650/>
LEARN: <https://learn.uwaterloo.ca>
Campuswire: <https://campuswire.com>
Instructor: Prof. Arie Gurfinkel, arie.gurfinkel@uwaterloo.ca, EIT 4021
TA: Aosen Xiong @TAAosen a4xiong@uwaterloo.ca

Office hours by appointment. Begin all email subjects with [ECE650]. Use **Campuswire** instead of email whenever possible!

Assignment 1 - Due February 2nd, 2024

The skeleton for this assignment is available at the main branch of <https://git.uwaterloo.ca/ece650-1241/skeleton> in directory `a1`. Follow the instructions in Assignment 0 to correctly fetch and merge the files from the skeleton!

This is the first in a series of assignments that is part of a single large project. The project is to help the local police department with their installation of security cameras at traffic intersections. You will solve a particular kind of optimization problem, called the Vertex Cover problem, in this context. The idea is for the police to be able to minimize the number of cameras they need to install, and still be as effective as possible with their monitoring.

For this assignment, you need to:

1. Take as input a series of commands that describe streets.
2. Use that input to construct a particular kind of an undirected graph.
3. Write your code in Python.
4. Ensure that your code works on `eceubuntu.uwaterloo.ca`. (You are allowed to use only those Python libraries that are already installed on those machines. You are not allowed to install any new libraries.) (Use `eceterm.uwaterloo.ca` to connect when you are outside of the campus network; then follow the instructions to connect to `eceubuntu`.)

Sample Input

The input comprises lines each of which specifies a command. There are 4 kinds of commands. (1) add a street, (2) modify a street, (3) remove a street, and, (4) generate a graph. Here is an example of how your program should work. Visualizing this example using the Cartesian coordinate system may help you understand what's going on.

```
add "Weber Street" (2,-1) (2,2) (5,5) (5,6) (3,8)
add "King Street S" (4,2) (4,8)
add "Davenport Road" (1,4) (5,8)
gg
V = {
  1: (2,2)
  2: (4,2)
  3: (4,4)
  4: (5,5)
  5: (1,4)
  6: (4,7)
```

```

7: (5,6)
8: (5,8)
9: (3,8)
10: (4,8)
}
E = {
  <1,3>,
  <2,3>,
  <3,4>,
  <3,6>,
  <7,6>,
  <6,5>,
  <9,6>,
  <6,8>,
  <6,10>
}
mod "Weber Street" (2,1) (2,2)
gg
V = {
  2: (4,2)
  5: (1,4)
  6: (4,7)
  8: (5,8)
  10: (4,8)
}
E = {
  <2,6>,
  <6,5>,
  <6,8>,
  <6,10>
}
rm "King Street S"
gg
V = {
}
E = {
}

```

Commands

- `add` is used to add a street. It is specified as: “`add "Street Name" (x1, y1) (x2, y2) ... (xn, yn)`”. Each (x_i, y_i) is a GPS coordinate. We interpret the coordinates as a poly-line segment. That is, we draw a line segment from (x_i, y_i) to (x_{i+1}, y_{i+1}) . You are allowed to assume that each x_i and y_i in the input is an integer. (Note, however, that the coordinates at which streets intersect might be not integers.)
- `mod` is used to modify the specification of a street. Its format is the same as for `add`. It is a new specification for a street you’ve specified before.
- `rm` is used to remove a street. It is specified as “`rm "Street Name"`”.
- `gg` causes the program to output the corresponding graph.

Input and Output

Your program should take input from standard input. Your program should output to the standard output. Error should be output to standard error. You can use exceptions in your code to catch errors.

Errors

The above example is that of a “perfect” user — someone that did not make any mistakes with specifying the input. You should account for errors in the input. If a line in the input is erroneous, you should immediately output an error message. The format of the message is to be the string “Error:” followed by a brief descriptive message about the error. For example:

```
Error: `mod' or `rm' specified for a street that does not exist.
```

Your program should recover from the error as well. That is, your program should reject the erroneous line, but continue to accept input. Your program should not crash because of an error. Any erroneous input we try will be of a relatively benign nature that mimics honest mistakes a user makes. We will not try malicious input, such as unduly long lines or weird control characters.

The Output Graph

The output graph consists of a set of vertices V and a set of edges E .

The set V is defined so that there is exactly one vertex corresponding to: (a) each intersection, and, (b) to each end-point of a line segment of a street that intersects with another street.

In our initial example, vertex 3 satisfies condition (a), and vertex 1 satisfies condition (b).

The identity (or the name) of a vertex can be any string of letters or integers (but no special or other characters). For example, v1xyz is acceptable as the identity of a vertex, but not v1 !!#xyz. (The space is unacceptable, as are '!' and '#'. In our initial example, we have used numbers as identifiers.

The set E is defined so that there is an edge between two vertices iff: (a) at least one of them is an intersection, (b) both lie on the same street, and, (c) one is reachable from the other without traversing another vertex.

In our initial example, the edge $\langle 1, 3 \rangle$ connects the end-point of a line segment to an intersection, and the edge $\langle 3, 6 \rangle$ connects two intersections.

Ambiguity

You may have noticed that the description above is rather informal. That is, the format of the input, the format of the output, and the expected behaviour are described in prose, without using formal mathematical notation. This is intentional. In real life, the requirements are informal and it is up to the Software Engineer to work with the stakeholders to formalize and clarify the requirements. Informal requirements are inherently *ambiguous* – there are multiple reasonable interpretations. However, this does not mean that any interpretation of the requirements is correct. Part of your assignment is to discover the behaviour intended by the stakeholders.

You will do the discovery by asking clarification questions on Campuswire.

Here is first such clarification. *The amount of white space that is used to separate components of the input or the output does not matter. That is, multiple spaces should be treated as a single space.*

Marking

Your output has to perfectly match what is expected. You should also follow the submissions instructions carefully. The reason is that our marking is automated.

- Does not compile/make/crashes: automatic 0

- Your program runs, awaits input and does not crash on input: + 20
- Passes Test Case 1: + 20
- Passes Test Case 2: + 20
- Passes Test Case 3: + 15
- Correctly detects errors: + 25
- Programming style: + 0 for this assignment, but will be > 0 for future assignments.

Get Skeleton Code from Upstream

To get the code, remember to pull from `upstream/main`.

Python Resources

If you're looking for some practice using and understanding Python3 you may wish to use <http://pythontutor.com/>. There is one example on the main page that you can step through line-by-line. More examples can be found on the live online editor page (<http://pythontutor.com/visualize.html>). The generated visualization of the code may be useful to you. The python tutor webpage also has code visualization for C++. You may find this useful for the next assignment.

If you have Python code on your computer that you wish to step through in a similar way, either line-by-line or some other way, you can run the code using the Python debugger. The Python debugger (pdb) can be invoked on the command line by providing the flag and argument as follows:

```
| python3 -m pdb <YOUR-PYTHON-FILE.py>
```

Documentation on how to use the debugger can be found on <https://docs.python.org/3/library/pdb.html>.

Submission Instructions

Your code can be in multiple Python source files. The main python file should be `a1/ece650-a1.py`. The assignment should be submitted as a `main` branch on YOUR GitLab in directory `a1`. Don't forget to enter your student identification in `a1/user.yml` and follow any additional instructions in `a1/README.md`.