Lab 3: Circuit Solver

You are not logged in.

If you are a current student, please Log In for full access to the web site.
Note that this link will take you to an external site (`https://shimmer.csail.mit.edu`) to authenticate, and then you will be redirected back to this page.

Table of Contents

1) Preparation

This lab assumes you have Python 3.7 or later installed on your machine.

The following file contains code and other resources as a starting point for this lab: `lab3.zip`

Most of your changes should be made to `lab.py`, which you will submit at the end of this lab. Importantly, you should not add any imports to the file.

This lab is worth a total of 4 points. Your score for the lab is based on:

- answering the questions on this page (0.5 points)
- passing the test cases from `test.py` under the time limit (2.5 points), and
- a brief "checkoff" conversation with a staff member to share your code, including discussion of the bullets below and a review of your code's clarity/style (1 point).

For this lab, you will only receive credit for your tests if they run to completion within the indicated time limit on the server.

Please also review the collaboration policy before continuing.

The questions on this page (including your code submission) are due at 4pm on Friday, Sept 27.

2) Introduction

In this lab, you will implement a simple circuit solver that works by generating and solving a system of linear equations. Specifically, given a DC circuit consisting only of electric cells (batteries) and resistances, your program should output the current flowing through each wire. There is also an optional section at the end where you may implement a checker which, given an assignment of currents to the wires, should determine how far off the assignment is to the correct solution.

The lab is divided into 3 parts, each of which can be done independently. The first part is implementing the function to solve a system of independent linear equations (we recommend using the substitution method), the second is implementing the circuit solver using the linear equation solver, and the third is optionally implementing the checker.

Since the second part depends on the first, we have provided a dummy implementation of the linear equation solver, which you may use for testing and debugging. Note that you may not import the module with the dummy implementation in your final submission. **You must therefore have a working implementation of the linear equation solver to obtain credit for the circuit solver.**

2.1) `lab.py` and `test.py`

These files are yours to edit in order to complete this lab. You should implement the main functionality of the lab in `lab.py`, and you may optionally implement additional test cases to help you debug in `test.py`.

In `lab.py`, you will find skeletons for the functions we expect you to write.

3) Solving a System of Linear Equations

First, you will need to implement a linear equation solver.

3.1) Background

A linear equation with the k variables x_i for $1 \leq i \leq k$ can generally be written in the form below.

$$c_0 + \sum_{i=1}^{k} c_i x_i = 0$$

or

$$c_0 \cdot 1 + c_1 x_1 + c_2 x_2 + ... + c_k x_k = 0.$$

Here, c_i for $1 \leq i \leq k$ are the coefficients of the variables and c_0 is the coefficient of the constant term.

A system of equations is just a set of n equations. An equation in a system is said to be independent if no combination of the other equations can be used to derive or disprove it. A solution to a system of equations is an assignment of values to each of the variables such that all of the equations in the system are satisfied.

It can be shown that if $n = k$ and all of the equations are independent of each other, the system of equations has a unique solution. For our purposes, we assume that this criterion is met by the input. **Note that we would need to keep this in mind while using this part for the circuit solver.**

3.2) Representing Equations

Each equation is represented as a `dictionary` mapping the variables to their coefficients. If there is a constant term, the coefficient of the constant term is present under the key `1`. The coefficients are either integers or floating point numbers.

For instance, the equation $2x + 3.5y + 4z + 5 = 0$ could be represented as `{'x': 2, 'y': 3.5, 'z': 4, 1: 5}` whereas $x + y = 0$ might be represented as `{'x': 1, 'y': 1}` or `{'x': 1.0, 'y': 1, 1: 0}`.

A system of equations can simply be represented as a list of the equation dictionaries.

Answer the following concept question to ensure that you have understood the input format.

How would the following system of equations be represented? Enter a Python `list` each of whose elements is a `dictionary`. Note that the variable names are case sensitive.

$$\begin{aligned} x + y &= 0 \\ x - y + 1 &= 0 \end{aligned}$$

This question is due on Friday September 27, 2019 at 04:00:00 PM.

3.3) The Substitution Method

Although there are many techniques for solving linear equations, we recommend that you use the substitution method. In this section we will walk through the steps involved in the substitution method using the following example system of independent linear equations.

$$\begin{aligned} x + 2y + 4z - 1 &= 0 \\ 2x + y &= 0 \\ 2x + 3y + 4z + 5 &= 0 \end{aligned}$$

Step 1: Choose an equation to substitute.

Our first step is to choose an equation from our system of independent linear equations to substitute. This equation will be known as the *substitution equation*. We could choose any equation to be the *substitution equation*, but to prevent speed issues we will choose the equation with the fewest variables. In this case, we will choose $2x + y = 0$ as our *substitution equation* because it only has two terms.

Step 2: Choose a variable to substitute.

Once we have selected a *substitution equation*, we will select a variable from that equation to be our *substituted variable*. While we could choose any variable in the *substitution equation* to be the *substituted variable*, to prevent precision issues we will choose the variable with the largest absolute value coefficient. In this case, we will choose x as our *substituted variable* because its coefficient, 2, is larger than that of y (1).

Step 3: Solve the *substitution equation* for the *substituted variable*.

Next we rearrange the *substitution equation* to solve for the *substituted variable* in terms of the other variables.

$$x = -0.5y$$

Step 4: Sub the *substituted variable* into the remaining equations.

Now that we have an expression for the *substituted variable*, we can plug that value into the remaining equations. Doing so allows us to eliminate the *substituted variable* from every remaining equation. For our example, we can use the result from step

3 and plug it into the other equations as follows.

$$(-0.5y) + 2y + 4z - 1 = 0$$
$$2(-0.5y) + 3y + 4z + 5 = 0$$

to get the equations

$$1.5y + 4z - 1 = 0$$
$$2y + 4z + 5 = 0$$

We recommend you implement the `substituteEquation` helper function in `lab.py`. This function should perform the substitution in step 4 on a single equation. The function takes in the following parameters.

- `equation`: The equation to be simplified, given in the format described [earlier] (COURSE/labs/lab3#_representing_equations).
- `substitutedVariable`: The name of the variable to be substituted out of the equation.
- `substitutionEquation`: A `dictionary` representing the substitution equation.

This function should return a `dictionary` representing the resulting equation after the substitution takes place. Although implementing this function is not strictly required (there are no test cases for this part), we recommend coding the overall `solveLinear` function in logical parts.

Step 5: Recursively solve the resulting reduced system.

These equations we derived in step 4 constitute their own system of independent linear equations. This new system has one fewer equation and one fewer variable than our original system. Therefore, we may apply this procedure recursively to obtain the solution to this reduced system.

Since we are performing recursion, we would need to have a base case. For our purposes, we can take that to simply be the case when we have one equation left. alternatively, you might find it easier to consider the case when there are no equations left. In either of these cases, the system can be solved trivially.

In our example, on recursing, we end up with:

$$y = -12$$
$$z = 4.75$$

Step 6: Plug back to find the value of the substituted variable.

Finally, we need to calculate the value of the *substituted variable* using the *substitution equation* and the partial solution obtained in step 5. To do so, you can plug in the values obtained from step 5 into the *substitution equation* to solve for the *substituted variable*. For our example, we get:

$$2x - 12 = 0 \implies x = -(-12)/2 = 6$$

Therefore, the final assignments are given by $x = 6$, $y = -12$, and $z = 4.75$.

Answer the following concept questions to ensure that you understand the substitution method.

If the substitution equation is `{'x': 1, 'y': 2, 1: 11}` and the substituted variable is `'y'`, what would the equation `{'x': -1, 'z': 10, 1: 30}` look like after performing the substitution? Enter your answer as a Python `dictionary` representing an equation equivalent to the chosen substitution equation.

This question is due on Friday September 27, 2019 at 04:00:00 PM.

If the substitution equation is `{'w': 2, 'x': 1, 'y': 2, 1: 11}` and the substituted variable is `'y'`, what would the equation `{'w': 3, 'x': -1, 'y': 3, 'z': 10}` look like after performing the substitution? Enter your answer as a Python `dictionary` representing an equation equivalent to the chosen substitution equation.

This question is due on Friday September 27, 2019 at 04:00:00 PM.

If the substitution equation is given by `{'x': 2, 'y': 1.5, 1: 3}` and the **solution** to the smaller system of equations is given by `{'y': -1, 'z': 2}` (in other words, $y = -1$ and $z = 2$), what is the value of substituted variable (`'x'`) in the satisfying assignment? Your answer should be an integer or a floating point number.

This question is due on Friday September 27, 2019 at 04:00:00 PM.

3.4) Task

Implement the `solveLinear` function in `lab.py`. The function takes in the arguments:

- `variables`: A `set` of all the variables in the system of equations. Each variable is either a **`string` or a `tuple`**.
- `equations`: A `list` of equations in the format described [earlier](COURSE/labs/lab3#_representing_equations). The length of this `list` is given to equal that of `variables` and the equations are given to be independent.

The function should return a dictionary mapping each variable to its value in the satisfying assignment (the solution). **Note that this function may modify its input as it pleases.** In fact, copying the entire system of equations at every recursive step will likely lead to memory errors.

To pass some of the larger test cases, you might need to consider the factors detailed in the following sections.

3.5) Precision

There are inherent precision issues resulting from dealing with floating point values (try typing 10 / 3 - 10 * (1 / 3) into the Python shell or look at this page for a brief description). To account for this, we will accept a value as long as it is within `1e-5` of the correct value. However, if you find yourself running into precision issues in your `solveLinear` function here are some tactics that can help you reduce the error from floating point arithmetic:

1. **Avoid unnecessary arithmetic operations especially divisions.** The only quantity you will need to divide by is the coefficient of the substituted variable in the substitution equation. If you find yourself dividing by other values, those divisions are likely unnecessary and should be avoided.
2. **Choose the variable with the greatest absolute value coefficient in the substitution equation to be the substituted variable.**[1]
3. **Delay divisions for as long as possible.** For instance, you could use `(a + b) / c` instead of `a / c + b / c` or `(a * b) / c`) instead of `a * (b / c)`.

3.6) Speed

Depending on how efficient your implementation is, you might need to perform a few additional optimizations to pass all of the test cases under the time limit. Particularly, you might want to consider the following.

1. **Use proper data structures and avoid slow operations.** For instances, inserting or deleting in the middle of a list can be inefficient and should be avoided.
2. When you first receive a system of equations, you can make each of the equations slightly smaller by **removing any variable which has a coefficient of zero**. This gives a decent boost as the systems of equations encountered while

solving circuits have a fair number of zero-coefficient variables. Note that this should be done only when you first obtain the system of equations and need not be repeated every time you recurse.

3. **If a coefficient becomes exactly zero during a substitution, the variable can be removed from the equation.** This is similar to the previous optimization, but you only need to check the variables whose values are modified during a substitution. This will particularly help with the test case in which the substitutions form a chain (`test_11`).
4. To determine the best equation to substitute, note that the time needed to do a substitution increases with the size of the equation. Therefore, it makes sense to **choose the equation whose corresponding dictionary has the fewest elements** as the substitution equation.

To ensure that you understand the prescribed way to choose the substitution equation and the substituted variable (based on the precision section), please answer the following concept questions based on the system of equations given below.

```
[
  {'x': 1, 'y': -1, 'z': 2, 1: 2},
  {'x': 1, 'y': 1, 'z': 3, 1: 1},
  {'x': 11, 'z': -13, 1: -20}
]
```

According to the given prescription, which of the equations in the above system should be chosen as the substitution equation? Enter your answer as Python `dictionary` representing an equation equivalent to the chosen substitution equation.

This question is due on Friday September 27, 2019 at 04:00:00 PM.

According to the given prescription, which variable should be chosen as the substituted variable in the above system? Enter your answer as a Python `string`.

This question is due on Friday September 27, 2019 at 04:00:00 PM.

4) Circuit Solver

Now, we are ready to use our linear equations solver to analyze circuits. However, you must first understand some basic concepts of circuit analysis. If you are familiar with the topic, you may skim through the background section.

4.1) Background

We model a circuit as a set of J junctions with W wires connecting certain pairs of junctions. Charges can flow through any of the wires in the circuit. The flow of charges is known as current $\left(\text{usually denoted as } I\right)$.

In addition to wires and junctions, our circuits also contain electric cells (batteries) and resistors. Each of these components is placed on a wire.

Electric cells have a positive and a negative terminal. They create a constant potential difference $\left(\text{colloquially known as voltage and denoted by } V\right)$ across their terminals and drive some current through the circuit.

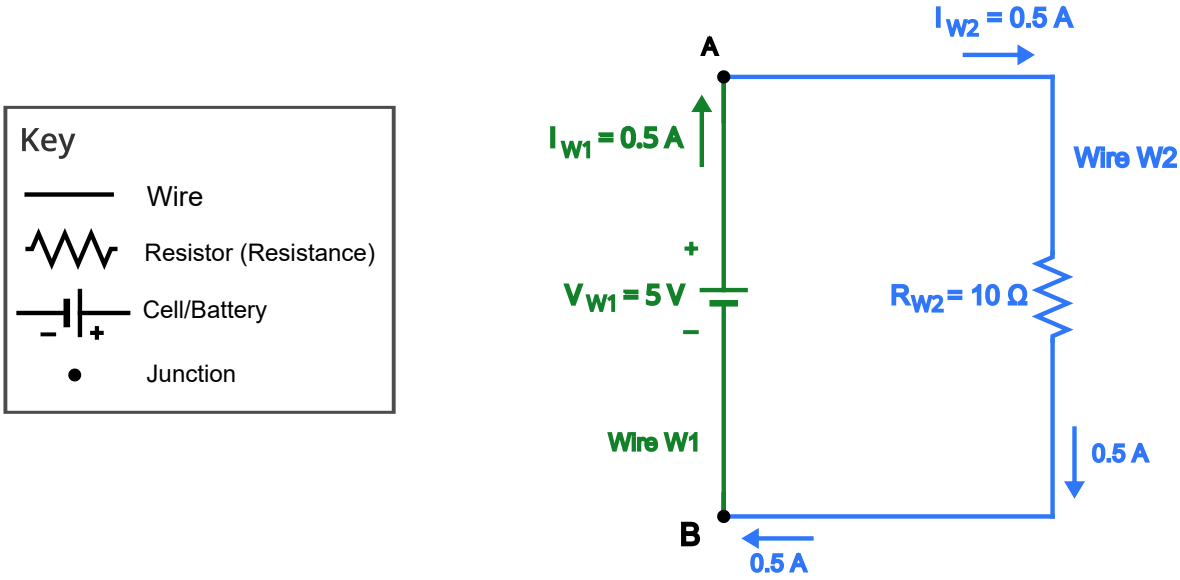
On the other hand, resistors $\left(\text{with resistance denoted by } R\right)$ try to slow down the flow of current (in the absence of resistances, an infinite amount of current would flow on connecting the terminals of a battery).

Every wire in the circuit has some (potentially zero) amount of current flowing through it. It is important to note that current has an orientation. Specifically, if current I is flowing through a wire from junction A to junction B, we say that a current of

precisely $-I$ is flowing from junction B to junction A through the same wire. When a single battery is connected to a resistor as shown below, the current flows from the positive terminal to the negative terminal.

It is often convenient to also define the electrical potential at each junction. If a wire has a battery but no resistance on it, we can find the potential difference (voltage) of the battery by first finding the junctions closest to its terminals and then subtracting the negative junction's potential from the positive junction's potential.

In our circuits, we work in SI units with the resistances measured in Ohms (Ω), current in Amperes (A), and voltages/potentials in Volts (V). For simplicity, units are omitted in all further discussion. We assume that all components are ideal and neglect effects due to inductance, capacitance, degradation of the batteries, etc.



In the following two sections we define two relationships—Kirchhoff's Current Law and Ohm's Law—that can be used to relate the currents, potentials, resistances, and battery voltages in our circuits. This will be sufficient to determine the current along every wire.

4.1.1) Kirchhoff's Current Law

Kirchhoff's Current Law (KCL) states that **current is conserved**. In other words, at every junction, the amount of current entering the junction is equal to the amount of current leaving it. For example, in the circuit above, the current flowing along wire $W1$ and into junction A must equal the amount of current leaving junction A through wire $W2$. This can be written as:

$$I_{W1} = I_{W2}$$

where I_W is the current from along W. This relation can be easily generalized to junctions adjacent to more than two wires.

Intuitively, this is because current is just a flow of charge. Since charge does not pile up or disappear at any junction (at least for the ideal components we deal with), any charge entering a junction must leave it. This is analogous to the flow of water through a system of pipes.

Current conservation can be applied on each junction giving a total of J relations. However, it turns out that all of these relations are not independent. In fact, exactly one of the obtained equations can be considered redundant (it does not matter which one). Therefore, **any $J-1$ of the equations obtained by using Kirchhoff's Current Law will comprise an independent set of equations.**[2]

4.1.2) Ohm's Law

We can use Ohm's Law to relate the electric potential across a wire to the current and resistance of that wire. Particularly, for a wire W from junction A to junction B, we can write

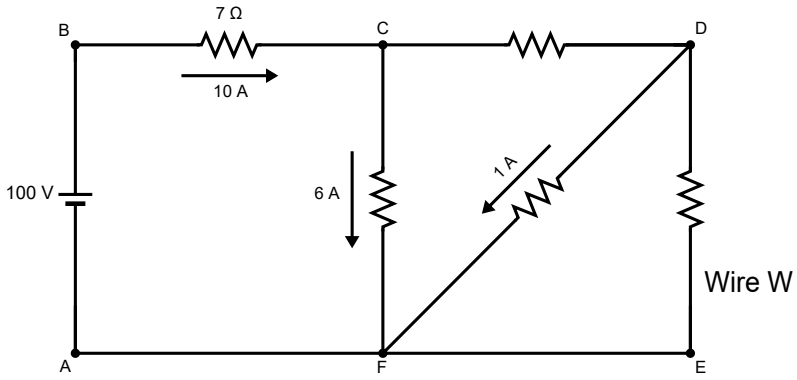
$$V_B - V_A = V_W - I_W R_W$$

where V_B and V_A are the electric potentials at junctions B and A respectively, V_W is the voltage of the battery connected along W (taken to be positive when the positive terminal of the battery is next to B and negative otherwise), R_W is the resistance connected along W, and I_W is the current from along W.

We can get one such relation for each wire, giving a total of W relations. It can be shown that all of these relations are in fact independent of each other and also independent of those obtained from Kirchhoff's Current Law. Therefore, combining these gives us a total of $J + W - 1$ relations. The currents along every wire and the potentials at every junction constitute the unknowns in these equations. Therefore, they can be treated like a system of equations with $J + W$ variables to solve for. Notice that we now seem to be short of one equation.

Note that the Ohm's Law relations only depend on the electric potential difference across two junctions, whereas Kirchhoff's Current Law does not depend on the potentials at all. Therefore, adding the same value to all the potentials in the circuit should not affect these relationships. **One may use this property to fix the potential at a particular junction or impose some other suitable constraint on them.** This could involve getting rid of a variable (such as a potential at a junction) or adding another suitable equation to impose some constraint on the potentials. In either case, the number of variables now matches the number of equations, ensuring that there is a unique solution.

Answer the following concept questions to ensure that you understand these relations.



What is the current (in Amperes) flowing from E to D along wire W in the above circuit? Your answer should be an integer or a floating point number with the units omitted. Hint: You might want to consider using current conservation.

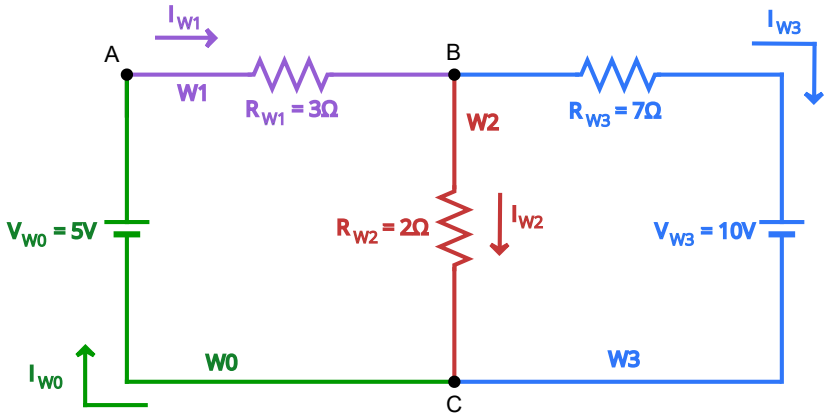
This question is due on Friday September 27, 2019 at 04:00:00 PM.

What is the potential (in Volts) at C given that the potential at A is -2? Your answer should be an integer or a floating point number with the units omitted. Hint: You can try Ohm's Law.

This question is due on Friday September 27, 2019 at 04:00:00 PM.

4.2) Solved Example

In this section we will walk through solving the circuit below by hand.



Step 1: Determine the Unknowns

Here we have 7 unknowns. The potentials at each junction: V_A, V_B, and V_C, and the currents across each wire: I_{W0}, I_{W1}, I_{W2}, and I_{W3}.

Step 2: Determine the KCL Equations

We get a KCL equation for each junction.

$$\begin{aligned} I_{W0} &= I_{W1} && \text{(Junction A)} \\ I_{W1} &= I_{W2} + I_{W3} && \text{(Junction B)} \\ I_{W2} + I_{W3} &= I_{W0} && \text{(Junction C)} \end{aligned}$$

Step 3: Determine the Ohm's Law Equations

We get an Ohm's Law equation for each wire.

$$\begin{aligned} V_A - V_C &= V_{W0} - I_{W0}R_{W0} = 5 - I_{W0}(0) = 5 && \text{(Wire W0)} \\ V_B - V_A &= V_{W1} - I_{W1}R_{W1} = 0 - I_{W1}(3) = -3I_{W1} && \text{(Wire W1)} \\ V_C - V_B &= V_{W2} - I_{W2}R_{W2} = 0 - I_{W2}(2) = -2I_{W2} && \text{(Wire W2)} \\ V_C - V_B &= V_{W3} - I_{W3}R_{W3} = -10 - I_{W3}(7) = -10 - 7I_{W3} && \text{(Wire W3)} \end{aligned}$$

Step 4: Ensure the Equations are Independent

As expected, the KCL equations we derived are not independent. This is easy to see because from $I_{W0} = I_{W1}$ and $I_{W1} = I_{W2} + I_{W3}$ we can derive the third equation $I_{W2} + I_{W3} = I_{W0}$ by simply replacing I_{W1} with I_{W0} since they are equal. Any two of them will be a set of independent equations so we only keep the first two.

If we remove a KCL equation, then we only have 6 equations left. However, we need 7 equations to solve for 7 unknown variables. Using the fact that we may impose an additional constraint on the potentials, we add $V_A = 0$ to our list of equations to get 7 independent equations as desired. This process of setting a particular potential to zero is known as grounding.

Step 5: Solve Equations

We have 7 equations and want to solve for the currents flowing through each wire. The equations are

$$\begin{aligned} I_{W0} &= I_{W1} \\ I_{W1} &= I_{W2} + I_{W3} \\ V_A - V_C &= 5 \\ V_B - V_A &= -3I_{W1} \\ V_C - V_B &= -2I_{W2} \\ V_C - V_B &= -10 - 7I_{W3} \\ V_A &= 0, \end{aligned}$$

which can be solved to obtain

$$I_{W0} = \frac{25}{41}\ \text{A} \approx 0.60976\ \text{A}$$
$$I_{W1} = \frac{25}{41}\ \text{A} \approx 0.60976\ \text{A}$$
$$I_{W2} = \frac{65}{41}\ \text{A} \approx 1.58537\ \text{A}$$
$$I_{W3} = -\frac{40}{41}\ \text{A} \approx -0.97561\ \text{A}.$$

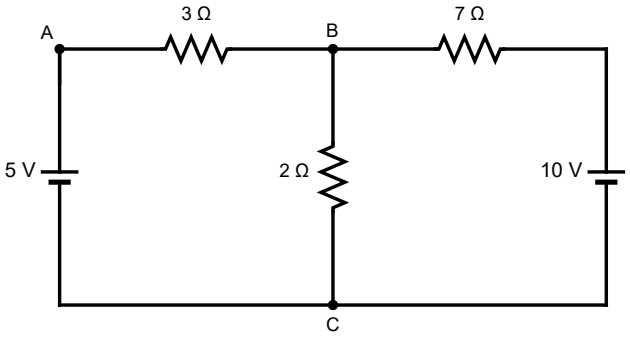
4.3) Task

Implement the `solveCircuit` function in `lab.py`. The function takes in the arguments:

- `junctions`: A `set` of all the junctions in the circuit. Each junction is labeled by a `string` or a `tuple`.
- `wires`: A `dictionary` mapping a unique wire ID (each of which is a `string` or a `tuple`) to a tuple of two distinct elements representing the starting and ending junctions of the wire. No wire will have the same ID as a junction. Moreover, the circuit is given to be connected (you can travel from each junction to every other junction by following some sequence of wires). Note that although current can flow in either direction along a wire, each wire will only appear once in the dictionary. Also note that there might be multiple wires between the same pair of junctions.
- `resistances`: A `dictionary` mapping the unique wire ID of each wire to the resistance along the wire. Every wire ID appears as a key. In case there is no resistance across a wire, the value is set to 0. You are given that there are no loops with zero resistance on every wire. This ensures that there is a unique solution.
- `voltages`: A `dictionary` mapping the unique wire ID of each wire to the voltage difference caused by any battery along the wire. A positive voltage indicates that the positive terminal is next to the end junction whereas a negative voltage indicates that the positive terminal is next to the starting junction of the wire. Every wire ID appears as a key. In case there is no battery across a wire, the value is set to 0.

The function should return a `dictionary` mapping each wire ID to the current along that wire (positive when it is flowing from the starting junction to the ending junction as defined in the `wires` dictionary). We tolerate the same error threshold as for the linear equation solver (maximum additive difference of `1e-5`, or 0.00001). You are advised to use an approach analogous to the one in the solved example to obtain a system of linear equations in terms of the currents and potentials and then use your equation solver to solve them.

Answer the following concept question to ensure that you have understood the input format.



What would be the `wires`, `resistances`, and `voltages` dictionaries for the above circuit? Note that the three junctions are `'A'`, `'B'`, and `'C'` (the names are case sensitive). Note that removing the junction `'A'` will essentially not change the circuit. However, you must include it in your answer. You may assigns IDs to the wires as you please (provided they satisfy the constraints mentioned above). Enter your answer as a Python list of the form `[wires, resistances, voltages]` where each element is a `dictionary`.

This question is due on Friday September 27, 2019 at 04:00:00 PM.

4.4) Using the UI

To help you debug your `solveCircuit` implementation, we have provided an interactive visualization system that displays the circuit diagrams for the test cases (excluding some circuits for `test_10`) and compares the result of your implementation to the expected result. To use the visualization, run `server.py` and use your web browser to navigate to localhost:8000. There you will have the option to select a circuit (via the "Select Circuit" button) and explore it by hovering over each element. You can also run your implementation of `solveCircuit` on the circuit (by pressing the "Run Code" button) and the circuit wires will be highlighted green or red depending on whether your code correctly calculated the current on that wire. If you hover on a wire, it will display both the expected current flowing along that wire as well as what your code produced. Please note that if you make changes to your code, you will need to refresh the page.

4.5) Using The Sample Linear Equations Solver

If you wish to complete this task before writing solveLinear, we have included a sample implementation of `solveLinear` in `solve_linear_sample.py`. You can import this code by uncommenting the appropriate line at the top of the `lab.py` file. Note that our implementation may have different behavior if the system of equations given to it is invalid (particularly if the equations are not independent or if the number of equations does not match the number of variables). Also, note that you may not import this function while submitting since it depends on `numpy`, which is not available on the server. In case you do not have `numpy` on your computer, you may install it by running the following command.

```
pip3 install numpy
```

If this does not work, try replacing `pip3` with `pip`. If you still encounter issues, look at the instructions here.

5) Code Submission

Once you have debugged your code locally and are satisfied, upload your `lab.py` below:

Select File No file selected

This question is due on Friday September 27, 2019 at 04:00:00 PM.

6) Checkoff

Once you are finished with the code, please come to a lab session or office hour and add yourself to the queue asking for a checkoff. **You must be ready to discuss your code in detail before asking for a checkoff.** Since the clarity of your code will be evaluated as part of the checkoff, you may wish to take some time to comment your code, use good variable names, avoid repetitive code (create helper methods), etc.

Be prepared to discuss:

- Your implementation of `solveLinear`.
- How you converted a circuit into a system of equations and your implementation of `solveCircuit`.

6.1) Grade

You have not yet received this checkoff. When you have completed this checkoff, you will see a grade here.

7) Verifying a Solution (Optional)

Note that this part is completely optional and will not affect your grade in any way. While this part is optional, we suggest you attempt this part to get more experience with recursion and Python's build in data types. To test this part, simply uncomment the last two test suites as indicated in `test.py`.

In this section, we will use the laws governing circuits to check if a given solution to a circuit (such as the result of `solveCircuit`) is correct. In particular, we can check if a solution is valid by checking that is satisfies Ohm's Law and Kirchhoff's Current Law.

7.1) Checking Kirchhoff's Current Law

As discussed in `solveCircuit`, Kirchhoff's Current Law states that current is conserved at every junction. In order to show that a circuit solution satisfies this condition, you will implement a function that finds the junction in the circuit with the largest absolute deviation from 0. In other words, the function should return the junction which has the greatest amount of current appearing from or disappearing into it. If we were to use this function to check a solution, we could check that the maximum deviation junction returned to us has a deviation of 0.

Implement the `findMaximumDeviationJunction` function in `lab.py`. The function takes in the following arguments.

- `junctions`: A `set` of all the junctions in the circuit. It satisfies the same conditions as in `solveCircuit`.
- `wires`: A `dictionary` mapping a unique wire ID (each of which is a `string` or a `tuple`) to a tuple of two distinct elements representing the starting and ending junctions of the wire. It satisfies the same conditions as in `solveCircuit`.
- `resistances`: A `dictionary` mapping the unique wire ID of each wire to the resistance along the wire. It satisfies the same conditions as in `solveCircuit`.
- `voltages`: A `dictionary` mapping the unique wire ID of each wire to the voltage along the wire. It satisfies the same conditions as in `solveCircuit`.
- `currents`: A `dictionary` mapping the unique wire ID of each wire to the indicated (potentially incorrect) current along the wire. It has the same format as the return type of `solveCircuit`.

`findMaximumDeviationJunction` should return the junction with the maximum absolute deviation from Kirchhoff's Current Law. If there are multiple such junctions, it may return any one of them.

7.2) Checking Ohm's Law via Kirchhoff's Voltage Law

Now that we are able to check that a circuit solution satisfies Kirchhoff's Current Law, we need to make sure that every wire in the solution satisfies Ohm's Law: $V_B - V_A = V_W - I_W R_W$. Although we know the battery voltage along each wire (V_W), the resistance on each wire (R_W), and the current along each wire (I_W), we do not know the potentials at the junctions $(V_B$ and $V_A)$. Therefore, we cannot directly check if a circuit satisfies Ohm's Law.

However, we note that the sum of the potential differences along any loop in the circuit should be zero as the contributions would cancel out. For instance, for the loop $A \to B \to C \to A$, we would get $(V_B - V_A) + (V_C - V_B) + (V_A - V_C) = 0$. This is known as Kirchhoff's Voltage Law (KVL). Therefore, we can check that our circuit satisfies Ohm's Law by checking that it satisfies Kirchhoff's Voltage Law for every loop. We can calculate the potential difference of a wire using Ohm's Law with the current, voltage, and resistance of the wire given in the solution as the inputs. Note that we only consider loops without repeating junctions (known as simple loops).

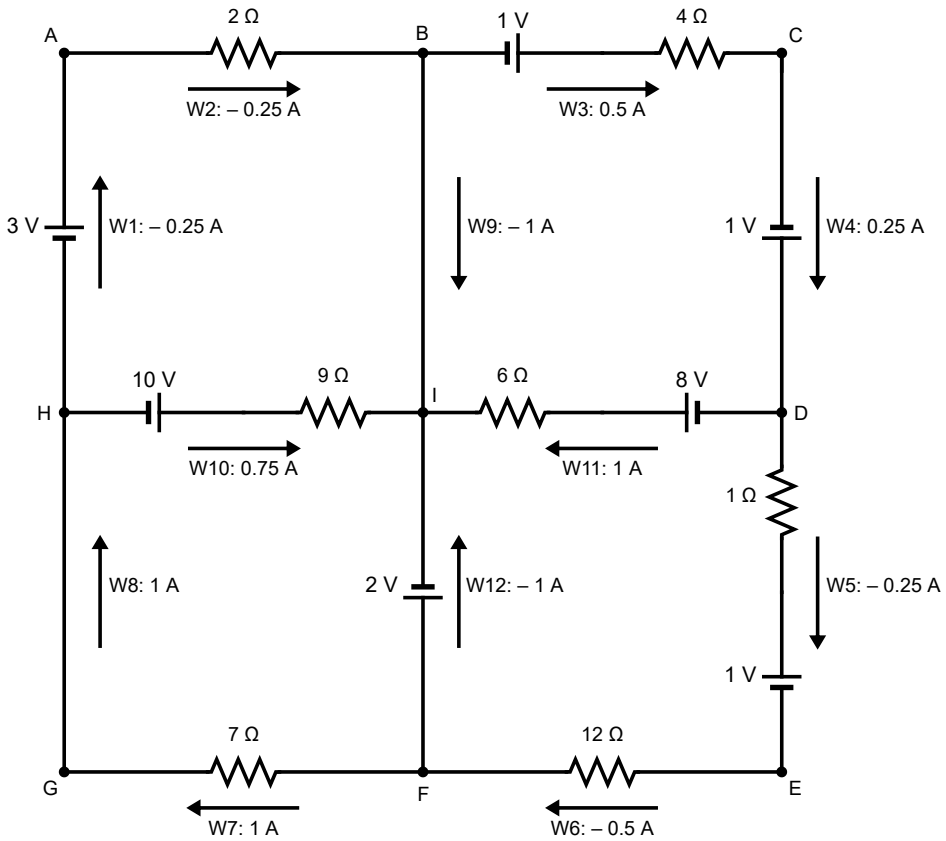
Implement `findMaximumDeviationLoop` function in `lab.py`. The function takes in the following arguments.

- `junctions`: A `set` of all the junctions in the circuit. It satisfies the same conditions as in `solveCircuit`.

- `wires`: A `dictionary` mapping a unique wire ID (each of which is a `string` or a `tuple`) to a tuple of two distinct elements representing the starting and ending junctions of the wire. It satisfies the same conditions as in `solveCircuit`.
- `resistances`: A `dictionary` mapping the unique wire ID of each wire to the resistance along the wire. It satisfies the same conditions as in `solveCircuit`.
- `voltages`: A `dictionary` mapping the unique wire ID of each wire to the voltage along the wire. It satisfies the same conditions as in `solveCircuit`.
- `currents`: A `dictionary` mapping the unique wire ID of each wire to the indicated (potentially incorrect) current along the wire. It has the same format as the return type of `solveCircuit`.

`findMaximumDeviationLoop` should return the loop with the maximum additive deviation from Kirchhoff's Voltage Law. If there are multiple such loops, it may return any one of them. The return type should be a list of the wire IDs along the loop in order (the starting junction and the direction within the loop could be arbitrary). We suggest recursively enumerating all cycles. You will need to be careful to avoid enumerating things that are not parts of cycles. **Note that this part is much more challenging than the previous ones.**

In case you want to make sure that you understand the above parts, you can try answering the following concept questions. **Note that they are optional and won't contribute to your lab grade.**



What are the maximum deviation junctions in the above circuit? For instance, the deviation at junction F is

$$\left|(-0.5) - (1 + (-1))\right| = 0.5.$$

Enter your answer as a Python set such as `{'A', "B"}`. Note that the junction names are case sensitive.

This question is due on Friday September 27, 2019 at 04:00:00 PM.

What is the deviation of the loop $A \to B \to C \to D \to I \to H \to A$ in the above circuit? For instance, the deviation of the loop $A \to B \to I \to H \to A$ is given by

$$\begin{aligned}\left|(0-(-0.25)\times 2)+(0-(-1)\times 0)-(10-0.75\times 9)+(3-(-0.25)\times 0)\right| &= |0.5+0-3.25+3| \\ &= 0.25.\end{aligned}$$

Enter your answer as a positive integer or floating point number.

This question is due on Friday September 27, 2019 at 04:00:00 PM.

What is the maximum deviation loop in the above circuit? Enter your answer in the same format as the output of `findMaximumDeviationLoop`. For instance, the loop $A \to B \to I \to H \to A$ can be represented as `['W1', 'W2', 'W9', 'W10']`, `['W2', 'W1', 'W10', 'W9']`, or six other ways corresponding to cyclic permutations of these. Hint: The maximal deviation is $3.5\ V$. You might find it helpful to first calculate all the potential differences and then just look at the circuit carefully.

This question is due on Friday September 27, 2019 at 04:00:00 PM.

Footnotes

[1] One reason this is really effective is that sometimes the coefficients should exactly cancel as a result of some substitutions (see the first precision test case). However, due to the lack of arbitrary precision, these values remain in the equations with really small coefficients (of the order of `1e-16`). Substituting one of such variables can have disastrous consequences for the precision of the final results. Picking the variable with the largest norm coefficient mitigates this possibility.

[2] Intuitively, this redundancy is because current conservation can never be violated at exactly one junction. If it is, some current must be flowing into or out of that junction. In that case, that current would have to flow out of or into some other junction(s) respectively. However, if current is conserved everywhere else, this could not happen.