

## **Setting up your environment**

For these first steps if any of these commands ask you y/n question always type y.

### **First make sure everything is updated:**

```
sudo apt-get update
```

```
sudo apt-get upgrade
```

### **Now we are going to make sure you have python3:**

```
sudo apt-get install python3
```

### **Now let's get pip3**

```
sudo apt-get install python3-pip
```

### **Next step is to get matplotlib and its dependencies like numpy**

```
sudo apt-get install python3-matplotlib
```














**Next we need to get a fortran compiler I suggest you use gfortran this is what I used. We need this to create a binary of MODFLOW to run FloPy and its tutorials.**

```
sudo apt-get install gfortran
```

**Next we need to get PyMake. This will do all the heavy lifting of compiling our MODFLOW binary.**

```
pip3 install https://github.com/modflowpy/pymake/zipball/master
```

Now we need to go to this site: <https://github.com/modflowpy/pymake/tree/master/examples> and download a file.

..		
 <a href="#">buildall.bat</a>	Added a buildall.py script that downloads and builds many of the USGS...	3 months ago
 <a href="#">buildall.py</a>	examples(modpath7): Update MODPATH 7 path case	3 months ago
 <a href="#">make_mf2000.py</a>	Update url addresses to https. Added modpath7 example and autotest fo...	a year ago
 <a href="#">make_mf2005.py</a>	Update pymake to use forward slashes in makefile, even when creating ...	a year ago
 <a href="#">make_mflgr.py</a>	Update url addresses to https. Added modpath7 example and autotest fo...	a year ago
 <a href="#">make_mfnwt.py</a>	Update current version of mfnwt	8 months ago
 <a href="#">make_mfusg.py</a>	Updated MODFLOW-USG autotest and example to use latest version (1.4)...	a year ago
 <a href="#">make_modflow6.py</a>	feat(pymake): Add support for mpiifort	4 months ago
 <a href="#">make_modpath6.py</a>	Update to modpath6 scripts to add code to prevent division by zero wh...	a year ago
 <a href="#">make_modpath7.py</a>	examples(modpath7): modify MODPATH 7 source	3 months ago
 <a href="#">make_mt3d.py</a>	Try using requests instead of urlretrieve to eliminate travis timeouts	a year ago
 <a href="#">make_mt3dusgs.py</a>	Update url addresses to https. Added modpath7 example and autotest fo...	a year ago
 <a href="#">make_swtv4.py</a>	Update url addresses to https. Added modpath7 example and autotest fo...	a year ago

You should be able to download any of these example files and get it to work but I stuck with the highlighted file: `make_mf2005.py`

```
44 lines (32 sloc) | 1.18 KB
Raw Blame History
1 from __future__ import print_function
2 import os
3 import shutil
4 import pymake
5 from pymake.download import download_and_unzip
6
```

After you clicked the file click Raw and then right click save-as and put it some where on your computer in its own folder.

Now navigate to that folder you downloaded `make_mf2005.py` then type:

```
python3 make_mf2005.py
```

This is going to take a while so sit back and do something else while it runs.

Once it is done you should have something like this in your folder:

 <b>temp</b>	12/21/2018 12:44 ...	File folder	
 <b>make_mf2005.py</b>	12/21/2018 12:39 ...	PY File	2 KB

Inside the temp folder is your binary for MODFLOW called mf2005. Remember where this is because we will reference it later.

 mf2005	12/21/2018 12:43 ...	File	3,016 KB
 mf2005dbf	12/21/2018 12:44 ...	File	3,004 KB

Now we need to get FloPy installed. To do that we will run this command:

```
pip3 install flopy
```

If you get an error like this:

```
kuaku@MSI:~$ pip3 install flopy
Collecting flopy
Collecting enum34 (from flopy)
  Using cached https://files.pythonhosted.org/packages/af/42/cb9355df32c69b553e72a2e28daee25d1611d2c0d9c272aa1d34204205b2/enum34-1.1.6-py3-none-any.whl
Requirement already satisfied: numpy>=1.9 in /usr/lib/python3/dist-packages (from flopy) (1.11.0)
Installing collected packages: enum34, flopy
Could not install packages due to an EnvironmentError: [Errno 13] Permission denied: '/usr/local/lib/python3.5/dist-packages/enum'
Consider using the '--user' option or check the permissions.
```

Try using sudo in front of it. If you then get this error:

```
kuaku@MSI:~$ sudo pip3 install flopy
[sudo] password for kuaku:
Traceback (most recent call last):
  File "/usr/bin/pip3", line 9, in <module>
    from pip import main
ImportError: cannot import name 'main'
```

Then try this command:

```
sudo python3 -m pip install flopy
```

Once you have FloPy installed we can start running things. The first thing I'd like to show you is how to run the FloPy tutorial. Locate the file called tutorial02.py and open it up in your favorite text editor like nano or vim. Once you have it up we need to change a line of code to get this to work.

```
# Flopy objects
modelname = 'tutorial2'
mf = flopy.modflow.Modflow(modelname, exe_name='/home/hierophant-green/Downloads/temp/mf2005')
dis = flopy.modflow.ModflowDis(mf, nlay, nrow, ncol, delr=delr, delc=delc,
                                top=ztop, botm=botm[1:],
                                nper=nper, perlen=perlen, nstp=nstp,
                                steady=steady)
bas = flopy.modflow.ModflowBas(mf, ibound=ibound, strt=strt)
lpf = flopy.modflow.ModflowLpf(mf, hk=hk, vka=vka, sy=sy, ss=ss, laytyp=laytyp,
                                ipakcb=53)
pcg = flopy.modflow.ModflowPcg(mf)
```

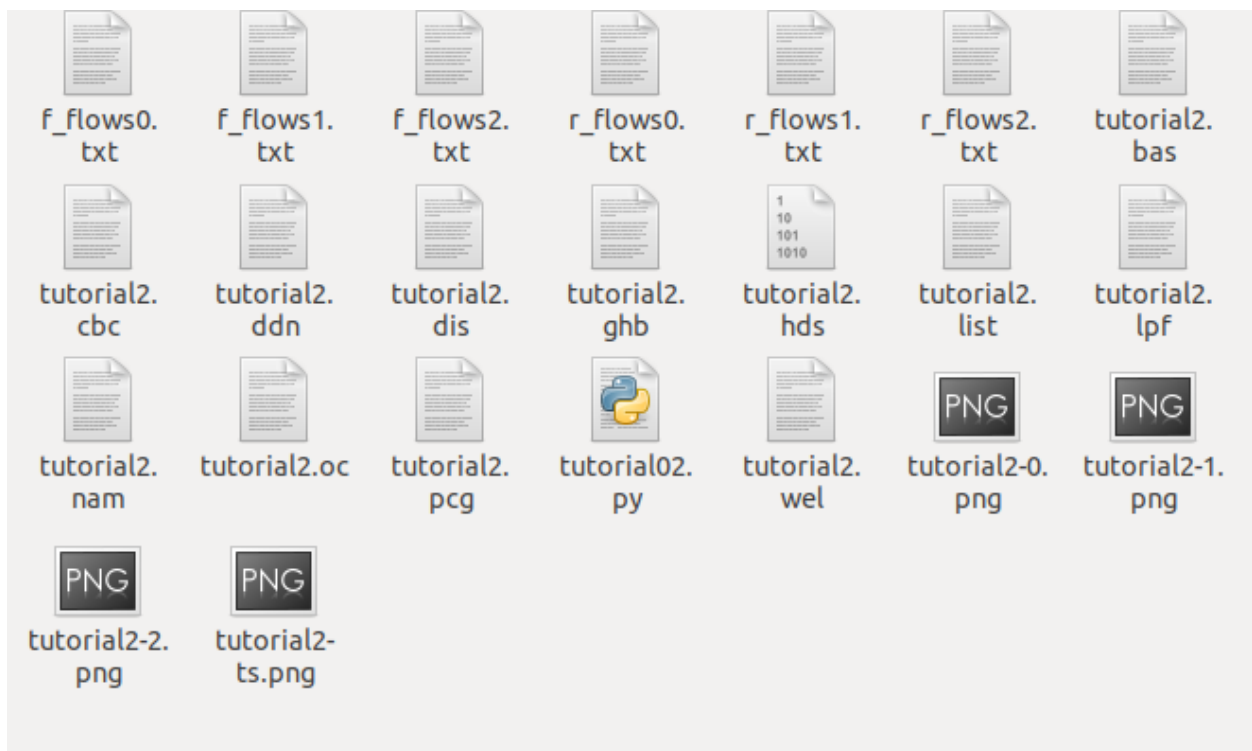
When you find this block of code you'll want to change what is in the single quotes after exe\_name=

Change it to the path of your earlier made binary file. You can see the location of my file in the above screenshot. Once you edited the file make sure you save it and then we should be able to run it.

You can run the tutorial by typing this command:

```
python3 tutorial02.py
```

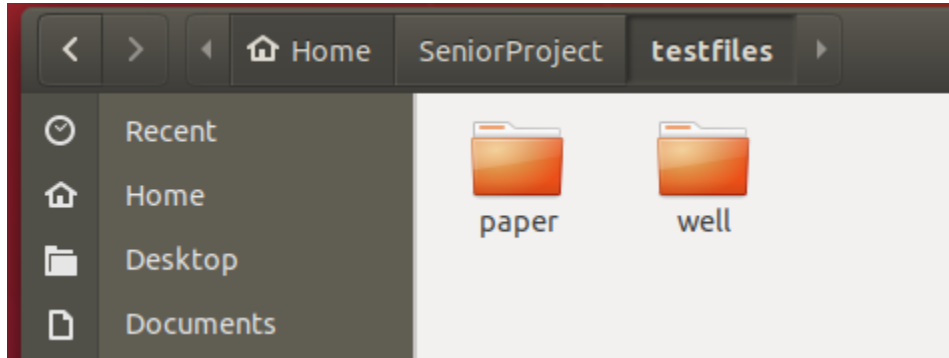
When it runs it will generate a ton of files so I would recommend putting this in its own folder so you don't clutter up your work area.



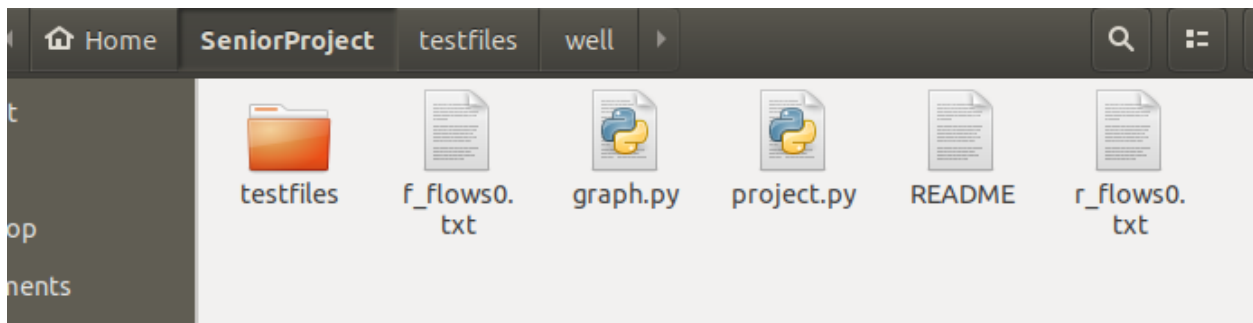
The png files are the images of the waterflows as generated by FloPy using matplotlib. The important thing though is that this script when ran created your f\_flows and r\_flows texts you'll need for later. This tutorial makes 3 for each the right and the left. You only need r\_flows2.txt and f\_flows2.txt. You will need to edit them later if you want to use these specific files though to work with FlowSourcePy.

## How to run the project

To run FlowSourcePy you'll first need your `f_flows` and your `r_flows` files. Inside the `testfiles` folder you'll see two other folders that contain your `f_flows` and `r_flows` text files. I'll show you how to run both of them. Let's first run the files in the `well` folder.



Go ahead and open the `well` folder and then copy the two text files and paste them in the same folder as your two python scripts called `graph.py` and `project.py` like so:



Next open up your terminal and we will get to running this code. To run it use this command:

```
python3 project.py
```

```
hierophant-green@hierophantgreen:~/SeniorProject$ python3 project.py
Number of time periods:
█
```

You will now be greeted by the prompt above. For the number of periods input 1

```
hierophant-green@hierophantgreen:~/SeniorProject$ python3 project.py
Number of time periods:
1
Number of nodes:
100
What is the width of the graph:
10
```

The next prompt will ask you for the number of nodes. For these files you'll say 100 nodes because this is a 10x10 grid.

For the width of the graph prompt you'll enter 10 because it is 10 nodes wide.

```
Which command would you like to run?
Enter the number of your desired command:
1 - Display Neighbors
2 - Fraction Through
3 - Display Fraction Through Animation
5 - Change time period
Type 'quit' to quit

```

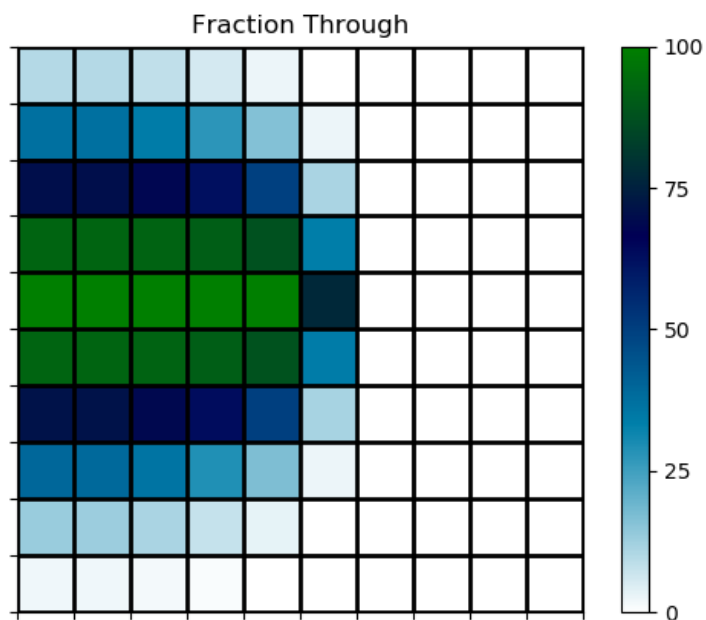
You'll now be greeted with a similar image to the one above. Option 3 will not be in the version you'll be looking at so ignore it. The only thing we want to take note of is option 1 which will print out a list of neighbors and option 2 which will create our beautiful graph.

Let's start with option 2. So type 2 and press enter.

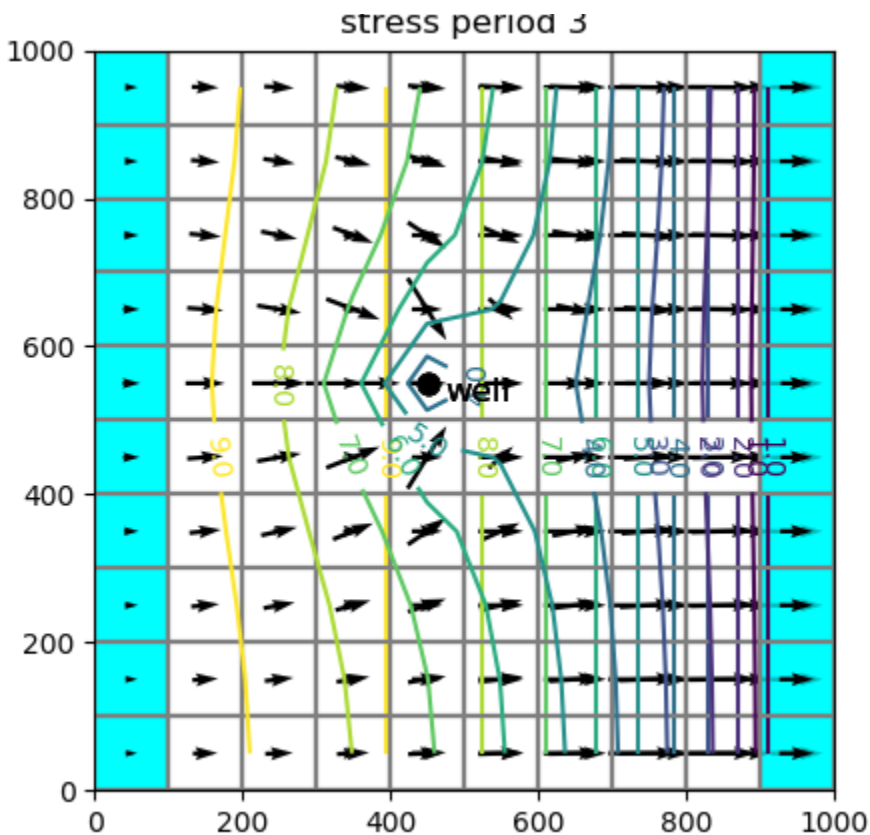
```
Which command would you like to run?
Enter the number of your desired command:
1 - Display Neighbors
2 - Fraction Through
3 - Display Fraction Through Animation
5 - Change time period
Type 'quit' to quit
2
Which node?

```

You are now presented with the prompt above. Type in 44 and press enter. It will generate your beautiful graph right before your eyes. It should look like this:



You might be wondering why we select node 44. That is because our grid starts at 0 and if you recall from our earlier when we ran the tutorial02.py file it generated us a bunch of files including some png files. If you take a look at the file named tutorial2-1.png you'll see a striking resemblance to the graph we just made.



That's because the `r_flows0.txt` and `f_flows0.txt` use the same data as the data that generated this image above. Now take a look at the big black dot in the middle of the graph. That is where a well is located and a bunch of water is flowing to that exact spot. That cell is number 44. This is why we chose it to run our test because it will show us a great example of how its neighbors are affecting it.

Alright let's try a more textbook example. Copy the files from the other folder in testfiles called paper and paste them next to our two python scripts just like we did the other files but overwrite the old ones.

Run the script again with:

```
python3 project.py
```

```

hierophant-green@hierophantgreen:~/SeniorProject$ python3 project.py
Number of time periods:
1
Number of nodes:
20
What is the width of the graph:
5

```

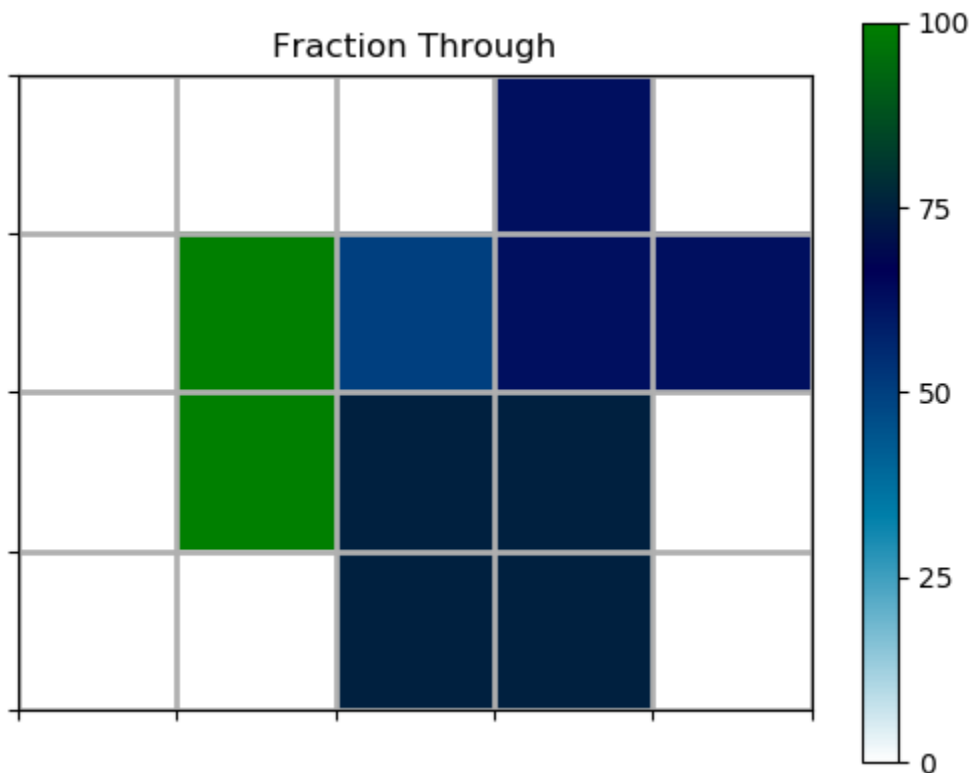
The number of periods you enter will be 1 again, but the number of nodes we will enter will be 20. Our data is no longer a 10x10 it is now a 5x4. So for the next question on the width you'll now enter 5.

```

Which command would you like to run?
Enter the number of your desired command:
1 - Display Neighbors
2 - Fraction Through
3 - Display Fraction Through Animation
5 - Change time period
Type 'quit' to quit
2
Which node?
6

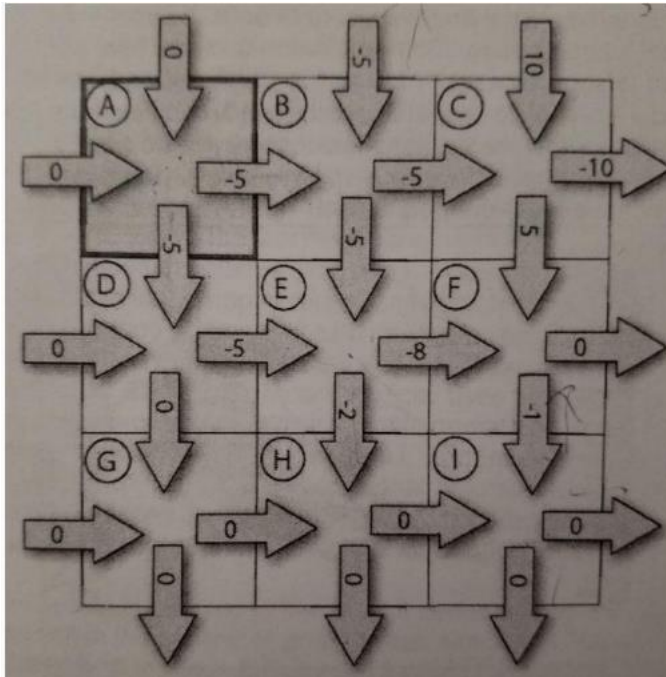
```

When you see the menu pop up you'll once again press 2 to run the Fraction Through. When it asks you which node to display you'll enter 6.





You should then be greeted with this graph. Originally on paper this was supposed to be a 3x3 grid but the data given to us had a lot more data than for a 3x3.



You need the extra data from the top row as well as the data from the last column. This makes it 5x4. This gives us this result.

(A) 100%	(B) 50%	(C) 62.5%
(D) 100%	(E) 75%	(F) 75%
(G) 0%	(H) 75%	(I) 75%

Handwritten notes: A diagonal arrow points from (E) to (I) with the label "75%". A handwritten "B" is written below (H).

Alright that's about all there is to how to run the file. The next thing I'll talk about is some tips on looking at the code and also a high level overview of what is going on under the hood.

To print out the adjacency list you'll want to uncomment the following lines of code in the project.py file:

```
98     #for k in range (0, (int(num_nodes))):
99         #print(k, ': ', g[i].graph[k])
100         #print(k, ' cost: ', g[i].vert_dict[k])
101
```

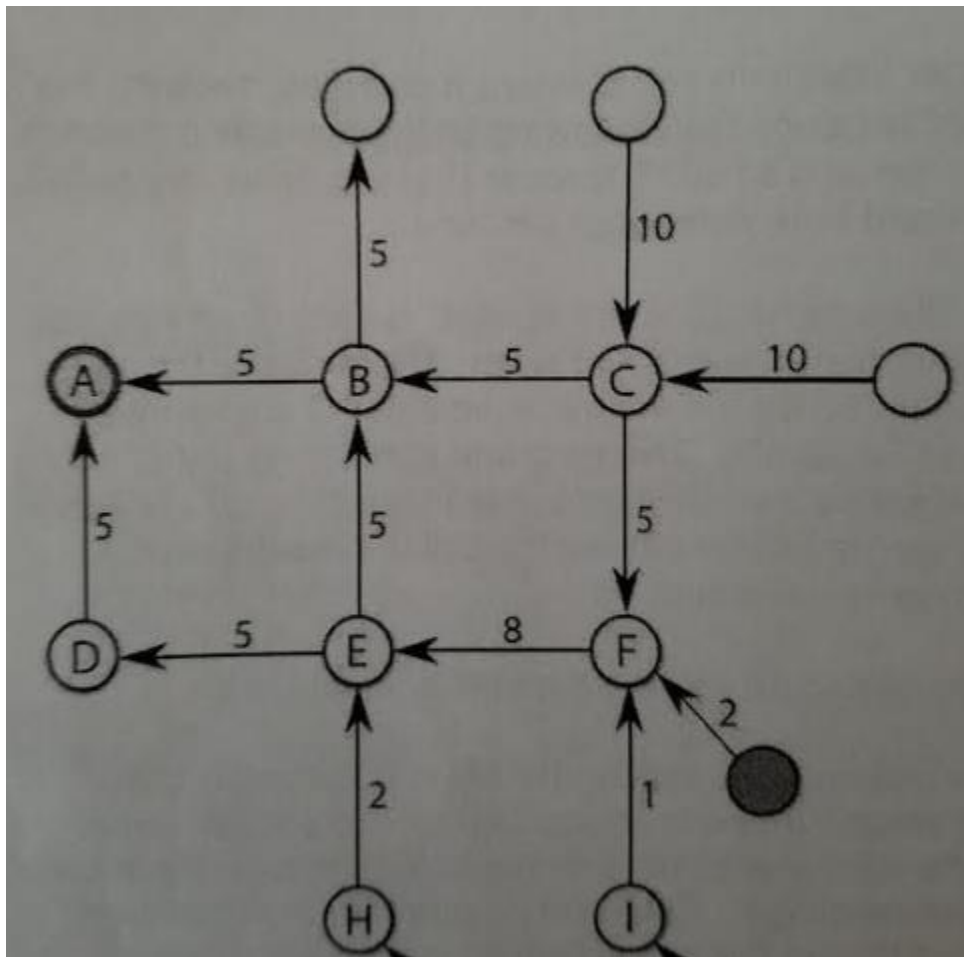
Lines 98 and line 100. This will print out the node and then it will print out its neighbors and their respective weights. If you try to run this with the r\_flows and f\_flows text files from the paper folder then you will most likely encounter an error for some reason so the screenshot you see below is from running the text files from the well folder.

```
hierophant-green@hierophantgreen:~/seniorbackup$ python3 project.py
Number of time periods:
1
Number of nodes:
100
What is the width of the graph:
10
0 cost: {1: 39.210255, 10: 0.0218381733}
1 cost: {0: -39.210255, 2: 42.00273, 11: 2.21358061}
2 cost: {1: -42.00273, 3: 46.704857, 12: 4.82445002}
3 cost: {2: -46.704857, 4: 52.04686, 13: 7.75454807}
4 cost: {3: -52.04686, 5: 57.702507, 14: 9.60365391}
5 cost: {4: -57.702507, 6: 65.56117, 15: 7.81935406}
6 cost: {5: -65.56117, 7: 74.73498, 16: 5.03564215}
7 cost: {6: -74.73498, 8: 82.90432, 17: 2.74556303}
8 cost: {7: -82.90432, 9: 87.785255, 18: 1.16785026}
9 cost: {8: -87.785255, 19: 0.011520816}
```

Now if you instead comment back out line 100 and then uncomment line 99 you'll then see the directed graph.

```
98     for k in range(0, (int(num_nodes))):
99         print(k, ': ', g[i].graph[k])
100        #print(k, ' cost: ', g[i].vert_dict[k])
```

The graph just shows the neighbors that are associated with the current node you selected. It is a representation of the dependency list we need to run the topological sort on. Here is the high level look at it from the paper.



Alright let's take a look at a high level of what is going on under the hood.

`g = []`

G is a list that holds objects from the graph class that is seen in the graph.py file. The reason for this is because of the prompt you see at the beginning of running the code called period. Each period is represented by data from two files. These two files are an r\_flows and an f\_flows. You can have as many of these files as you want but they always need to start at 0 that's why are examples are named r\_flows0.txt and f\_flows0.txt.

The next bit you'll need to know is that the text files are space separated.

```
0.00 0.00 -5.00 10.00 0.00
0.00 -5.00 -5.00 5.00 0.00
0.00 0.00 -2.00 -1.00 0.00
0.00 0.00 0.00 0.00 0.00
```

They look like this and they are read in from left to right top to bottom. The first row you see will be the same as the first row you see in the graphs from left to right.

```
class Graph:
    def __init__(self, vertices, columns):
        self.graph = defaultdict(list)
        self.V = vertices ## No. of vertices
        self.C = columns
        self.vert_dict = {} ## Dictionary of dictionaries for nodes and weights
        self.top_order = [] ## Stores result of topologicalSort
```

`self.graph`

As I said above this is holding the directed graph that is the dependency list to calculate our flows.

`self.V` and `self.C`

These are pretty self explanatory and they just hold the number of nodes and how wide the grid is.

`self.vert_dict`

This is pretty much our adjacency list. It is a dictionary of dictionaries that holds each node that is a neighbor and their respective weights.

`self.top_order`

This is holding the results of topological sort.