



Computer Science Department

Data Structures

RUTGERS School of Arts and Sciences

Infinity War – 100 course points

The purpose of this assignment is to practice your understanding of graphs and the adjacency matrix graph storage.

This assignment WILL TAKE A SUBSTANTIAL AMOUNT OF TIME TO COMPLETE.

Start your assignment early! You need time to understand the **assignment** and to answer the many questions that will arise as you read the description and the code provided.

Refer to our Programming [Assignments FAQ](#) for instructions on how to install VSCode, how to use the command line and how to submit your assignments.

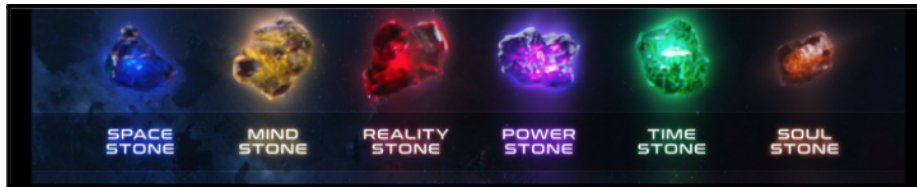
Overview

We do not expect that you have seen the movie in order to complete this assignment. This description will give you enough background information for you to understand how to complete each task.

In the MCU (Marvel Cinematic Universe), right after the Big Bang occurred there were 6 singularities contained within Infinity Stones that were spread across the infant universe.

These stones include:

Stone	Ability	Color	Object	First appearance
Space	Travel between places instantaneously	Blue	Tesseract	<i>Captain America: The First Avenger</i>
Mind	Control minds	Yellow	Loki's scepter Vision's head	<i>The Avengers</i>
Reality	Alter reality	Red	Aether	<i>Thor: The Dark World</i>
Power	Manipulate energy; increased strength	Purple	Orb	<i>Guardians of the Galaxy</i>
Time	Control and manipulate time	Green	Eye of Agamotto	<i>Doctor Strange</i>
Soul	Control souls	Orange		<i>Avengers: Infinity War</i>

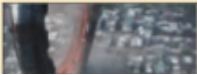


Backstory of Thanos

Before the events of *Avengers: Infinity War*, a titan named Thanos witnessed the death of his home planet due to overpopulation. Reeling from his experience, Thanos made it his life mission to exterminate half of all life in the universe to delay overpopulation on other worlds like his. Thanos determined that using the 6 Infinity Stones which he could hold in his Infinity Gauntlet, a simple snap of his fingers would painlessly eliminate 50% of all life, and he can finally rest in peace victorious.



Background Information and Characters

<u>Locations:</u>	<u>Main Avengers:</u>	<u>Guardians of the Galaxy</u> (hereby referred to as "Guardians")	<u>Other Characters:</u>
NYC: New York City on Earth. 	Thor: King of Asgard, known as the God of Thunder. Can control lightning and electricity, but lost his hammer when his evil sister Hela destroyed it.	Starlord (Peter Quill): The half-human, half-Celestial leader of the Guardians who	Black Panther (T'Challa): The king of the African nation of Wakanda who gained his enhanced strength by ingesting the Heart-Shaped Herb.

The Infinity War

If you are interested in the film's story the following videos will contextualize it for you.

TRAILER: [Marvel Studios' Avengers: Infinity War Official Trailer](#)

The film begins with Thanos destroying Thor's homeworld (Asgard) after acquiring the Power and Space Stones.

1. [Avengers: Infinity War \(2018\) – "Attack On The Statesman" I Movie Clip](#)
2. [Thanos Kills Loki Loki Death Scene Avengers Infinity War 2018 Lifeline HD](#)

A defeated Hulk is sent to NYC to warn Dr. Strange and Iron Man about the impending arrival of Thanos and his army (the Black Order) to retrieve the Time Stone.

His ship arrives soon and a battle ensues as Iron Man, Spiderman, and Dr. Strange battles Ebony Maw and Cull Obsidian (two of Thanos' "children").

- [AVENGERS INFINITY WAR – Battle in New York – IRON MAN vs BLACK ORDER Movie Clip HD](#)

After Cull Obsidian dies, Dr. Strange is captured onto Maw's ship, and Iron Man and Spiderman latch onto it as it departs Earth. Iron Man blows a hole in the ship and Maw is shot into space.

- [Avengers Infinity War Iron Man And Spider Man Saves Doctor Strange](#)

Meanwhile, the Guardians of the Galaxy (hereby referred to as "Guardians") rescue Thor and the group splits, where Thor, Rocket, and Groot (a tree-human hybrid) go to Nidavellir to forge Thor's Stormbreaker, a weapon capable of destroying Thanos.

- [The Guardians of the Galaxy respond to a distress call and rescue Thor. Avengers: Infinity War 2018](#)

Implementation Notes

The structure of this assignment is quite different from the previous assignments.

- **DO NOT** use static variables on your code.

- In each given Java class, you will read from a given set of input files (passed in as command line arguments), and write to a given output file (passed in as a command line argument).
- **DO NOT** change the names of any of the given Java files, or the project structure itself (do not change directory names or create new directories).
- **DO NOT** remove the package statement from any of the given input files.
- **DO NOT** use `System.exit()` in your code.
- Unlike any previous assignment, **YOU MAY** (and should) create your own classes in the `src/avengers` folder. **YOU MAY** import `"java.util.*"`, but **DO NOT** import anything else. Make sure any new classes have a package statement: `package avengers;`
- The classes that you create **MAY NOT** have spaces in their names.
- In order to grade a problem, we run the corresponding Java class and verify the output file. This means you have full freedom in your project structure, as long as our provided classes output the correct answer to the correct output file. Take this opportunity to practice your project design skills, and write clean code that avoids redundancy.
- **DO NOT** remove the package statement from the provided classes.

Using StdIn and StdOut

- Use `StdIn.setFile(fileName)` to set the current input file you want to read from.
- You can now use methods like `StdIn.readInt()`, `StdIn.readString()` and `StdIn.readLine()` to operate on the input file as if it was standard input.
- The methods `StdIn.readInt()` and `StdIn.readString()` actually leave the newline character unread, so if you use `StdIn.readLine()` after one of these methods, it will read this character rather than the next line. If you want to read the next line with `StdIn.readLine()`, you will need to call `StdIn.readLine()` once to read the newline character and then again to read the next line. `StdIn.readInt()` and `StdIn.readString()` ignore spaces and newlines by default.
- Use `StdOut.setFile(fileName)` to set the current output file you want to write to. It creates the file if it doesn't already exist.
- You can now use methods like `StdOut.print()` and `StdOut.println()` to operate on the output file as if it was standard output.
- Autolab ignores empty lines and extra spaces the your output files may have.

Overview of Files

1. *ForgeStormBreaker.java*

- Print to the output file the sum of the 2D array from the input file.

2. *LocateTitan.java*

1. Create the adjacency matrix using the input file. Each vertex represents a generator.
2. Divide each value in the adjacency matrix (so `matrix[i][j]`) with the functionality values of vertex *i* and vertex *j* (meaning `functionalities[i]` and `functionalities[j]`)
3. Use Dijkstra's algorithm to find the minimum cost from **vertex 0** and the **last vertex**.
4. Print the minimum cost to the output file.

3. *MindStoneNeighborNeurons.java*

1. Create a graph representation from the input file. Each vertex represents a nerve and each edge represents a synapse. All the vertices have a varying out-degree EXCEPT the **Mind Stone vertex which has out-degree 0**.
2. Find the Mind Stone vertex, and identify the vertices that neighbor (have an edge) the Mind Stone vertex.
3. Print Mind Stone vertex neighbors to the output file.

4. *UseTimeStone.java*

1. Create the adjacency matrix using the input file. Each vertex is an event, each path with connecting vertices is a timeline. Each vertex has an EU value.
2. Determine the TOTAL number of timelines.
3. Determine the number of timelines where *sum_of_EU_of_all_vertices_in_the_timeline* \geq *threshold EU*.
4. Print the number from step (3) to the output file.

5. *PredictThanosSnap.java*

1. Create the adjacency matrix using the input file. Each vertex represents a person, and each edge represents a connection.
2. Use the `StdRandom.uniform()` function and the given pseudocode to determine which vertices to delete.
3. Delete the vertices (this can be done directly on the adjacency matrix).
4. Determine if the resulting graph is connected. A graph is connected with there is a path from each vertex to every other vertex.
5. Print **true** to the output file if the graph is connected, **false** otherwise.

Tasks

1. ForgeStormBreaker

To forge the Stormbreaker, Thor has to endure the power of a Neutron Star near Nidavellir. The star's plasma rushes through the opening that Thor is inside, as depicted in the image to the right.

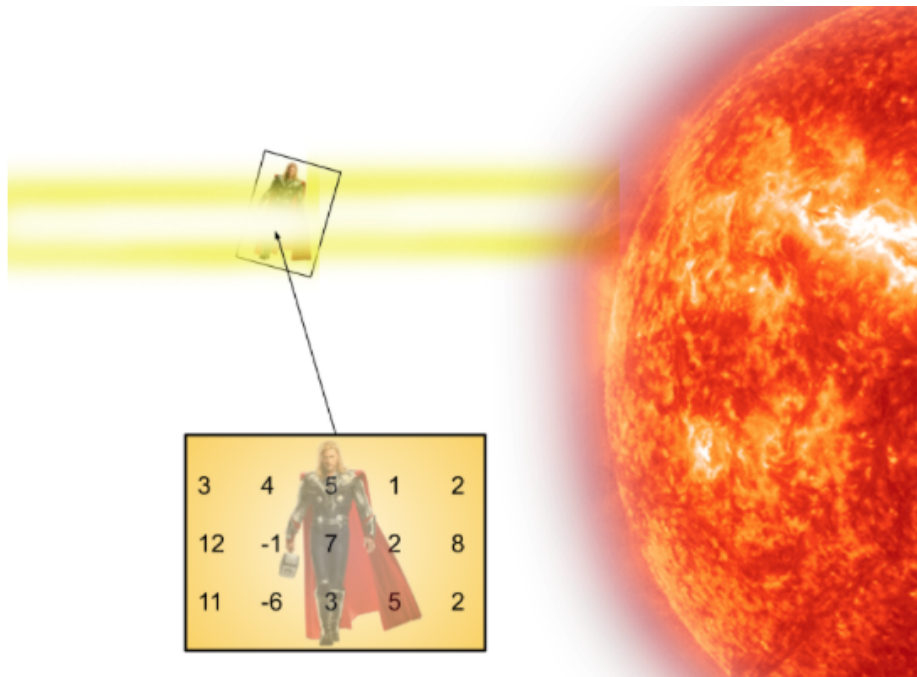


In physics, *flux* is defined as the amount of fluid (or field strength) flowing through an area.

In the image to the right, the box with the matrix of numbers shows the *flux intensity* of the sun's plasma through the opening that Thor is inside. Taking the sum of all the numbers reveals the total flux of the plasma through the opening.

Thus, **the flux would be $(3 + 4 + 5 + 1 + 2) + (12 - 1 + 7 + 2 + 8) + (11 - 6 + 3 + 5 + 2) = 58$.**

Task: given a 2D array where the values are the flux intensity data of the Neutron Star, calculate the total flux that Thor has to endure. Write this number into the output file.



An example input file looks like this:

3	←	Number of rows
3	←	Number of columns
1 2 3		
4 5 6	←	2D Array for Flux Intensity Data
7 8 9		

The corresponding output file looks like this:

45

Some preconditions:

- The numbers in the 2D array are guaranteed to be integers.
- The dimensions of the 2D array may not be the same.

On Earth, Vision (who has the Mind Stone contained in his head) is protected by the Scarlet Witch (Wanda Maximoff) against more of Thanos' forces as they try to acquire the Mind Stone. They are joined by Captain America (Steve Rogers), Falcon, and Black Widow (Natasha Romanoff).

- [Scarlet Witch & Vision vs The Black Order | The Avengers: Infinity War](#)

Meanwhile, the Guardians who did not go to Nidavellir (basically Peter Quill, Gamora, Drax, and Mantis) head to Knowhere to prevent Thanos from acquiring the Reality Stone. However, this fails as Thanos has already gained the Reality Stone and uses it to trick the Guardians into an illusion, and captures Gamora (Thanos' daughter) since she knows the location of the Soul Stone.

1. [GOTG Arrives at Knowhere – Thanos Tortures Collector Scene – Avengers: Infinity War Movie Clip HD](#)
2. [Avengers: Infinity War | Knowhere Scene | Thanos: “I like you.”](#)

Thanos tortures his daughter Nebula until Gamora reveals that the Soul Stone is in Vormir. Afterwards, Thanos and Gamora depart there while Nebula begs the Guardians to meet her on Titan, Thanos' homeworld.

- [Thanos Tortures Nebula Scene – Avengers Infinity War \(2018\) Movie Clip HD](#)

2. Locate Titan

After hijacking the Q-ship from Earth, Dr. Strange, Iron Man, and Spider Man decide to go to Titan. In the MCU, interstellar travel happens via wormholes (Einstein-Rosen Bridges).

- [AVENGERS INFINITY WAR “Avengers Vs Guardians of the Galaxy”](#)

Suppose that the route from Earth to Titan can be represented by a Weighted Undirected Graph where the vertices are *wormhole generators* and the edges are the *wormholes* themselves. A wormhole is created by two wormhole generators, so the wormhole would be a “path” between these generators.

Some important points:

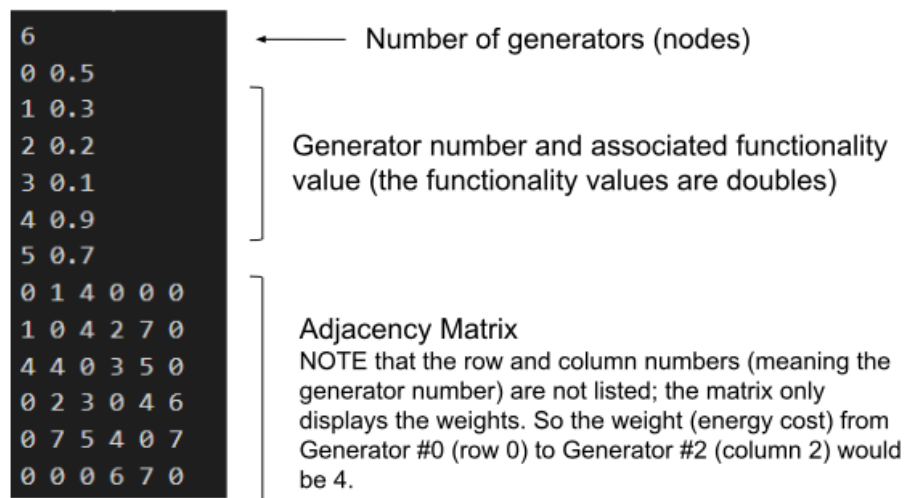
- There is *no* direct wormhole that can extend from Earth to Titan, so going from Earth to Titan requires intermediate “stops” at different wormhole generators.
- For instance, a hypothetical path could be from a generator on Earth to a generator on Planet X, and from the generator on Planet X to a generator on Titan.
- Each edge (Wormhole) would have an associated weight, which is the energy cost of creating the wormhole that the particular edge represents.
- Each vertex (Wormhole Generator) would also have a weight, and this number (a *double* data type) represents how functional that generator is. Due to Thanos' conquest so far across the universe, not all of the wormhole generators would be 100% functional.

- The Wormhole Generators would be identified by cardinal numbers (vertex 1 would be generator 1, vertex 2 would be generator 2, etc).

SUPPOSE there are n vertices IN THE GRAPH. ASSUME THAT GENERATOR #0 (vertex 0, the source) IS EARTH AND GENERATOR #(n-1) (vertex n-1, the destination) IS TITAN.

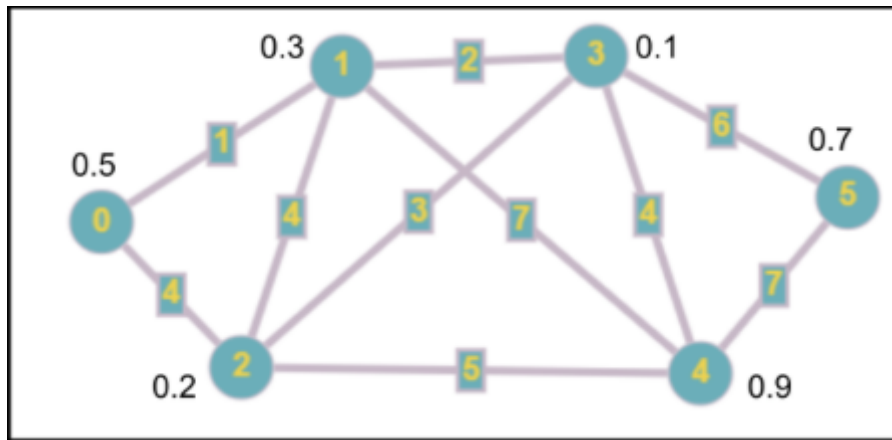
Your input is made of 2 portions:

- The generator number and the associated functionality.
- An adjacency matrix displaying the energy cost of traveling from one generator to another.



The corresponding graph looks like this.

- The functionality of each generator is the number in black that is next to each vertex. For instance, the functionality of Generator #1 is 0.3
- The energy cost between Generators i and j (in this image, that would be the number inside the *square* on each edge) would be the value at the i 'th row and j 'th column in the adjacency matrix.
- For instance, the edge between Generator #0 and Generator #2 has the number 4 boxed. Thus, the energy cost between Generator #0 and Generator #2 would be 4.



Right now, the adjacency matrix only shows the energy cost of each edge. However, we also need to factor in how each generator is *not* fully functional. Thus, if the *total cost* acknowledges the energy cost of each wormhole *and* the functionalities of the generators creating the wormhole, then the *less* functional the generators are, the *higher* the total cost of creating and using the wormhole would be.

Thus, if we want the adjacency matrix to hold the *total cost* of each wormhole, we can **DIVIDE** the energy cost by the functionality of **BOTH** generators that the wormhole connects.

Hence, once you read the input in and create your adjacency matrix, change **EVERY** edge weight (energy cost) by **DIVIDING** it by the functionality of **BOTH** vertices (generators) that the edge points to. Then, typecast this number to an integer (this is done to avoid precision errors).

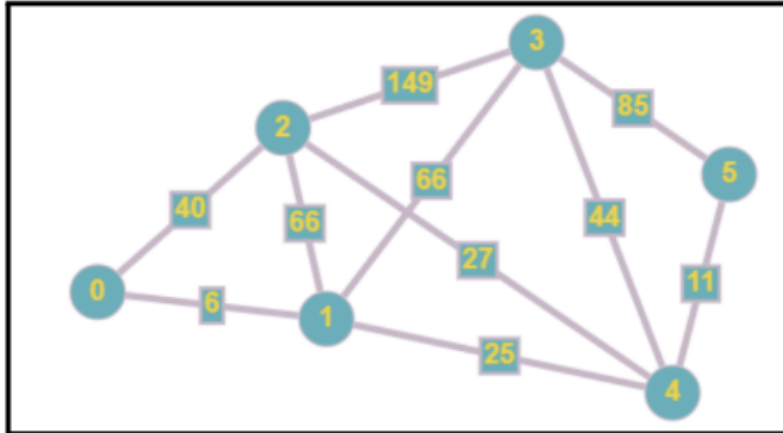
For instance, the edge from Generator #1 (row 1) to Generator #4 (column 4) has an energy cost of 7.

- Divide this by the functionality values of Generator #1 (which is 0.3) and Generator #4 (which is 0.9). As such, you would change the 7 to a $7/(0.3*0.9) = 25.925$.
- Next, typecast this to an int, so $(\text{int})(25.925) = 25$.
- Thus, the value at row 3 and column 4 of the matrix would *now* be 25.
- This means the *total cost* between Generator #1 and Generator #4 is 25.
- Do this for every edge!

Using the example above, the new matrix looks like this:

```
0 6 40 0 0 0
6 0 66 66 25 0
40 66 0 149 27 0
0 66 149 0 44 85
0 25 27 44 0 11
0 0 0 85 11 0
```

And the corresponding graph looks like this:



Then, using your *new* graph, calculate the minimum cost from Earth (Generator #0) and Titan (the final Generator; this would be Generator #5 since there are 6 Generators so the last Generator index is $6-1 = 5$) in the example above.

Use Dijkstra's Algorithm to find the path of minimum cost between Earth and Titan. Write this number into your output file!

Here is a good reference video for Dijkstra's Algorithm:

- [Dijkstra algorithm | Single source shortest path algorithm](#)

Pseudocode for Dijkstra's Algorithm:

```

// Create an array that keeps track of the MINIMUM cost
// to reach every vertex FROM vertex 0.
minCost = new array of ints;

// Create an array that keeps track of which nodes are in
// the path already.
DijkstraSet = new array of booleans;

// Set each minCost value to infinity (Integer.MAX_VALUE)
// except vertex 0 since we are STARTING at node 0.
for ( every element in minCost ) {
    if ( vertex is 0 ) {
        set minCost to 0
    } else {
        set minCost to Integer.MAX_VALUE
    }
}

for ( # of vertices - 1 iterations ) {

    /* determine the vertex with the MINIMUM cost from vertex 0.
       Store this vertex in currentSource. */
    currentSource = getMinCostNode();

    // Add currentSource to DijkstraSet (we are visiting it now)
    DijkstraSet[currentSource] = true;

    // Update the distance from 0 to each currentSource's neighbors IF it CAN BE lowered.
    for ( each currentSource's neighbor w ) {
        if ( w has not been visited AND
            minCost[currentSource] is not Integer.MAX_VALUE AND
            minCost[w] > (minCost[currentSource] + cost from currentSource to w) {
            minCost[w] = minCost[currentSource] + cost from currentSource to w
        }
    }
}

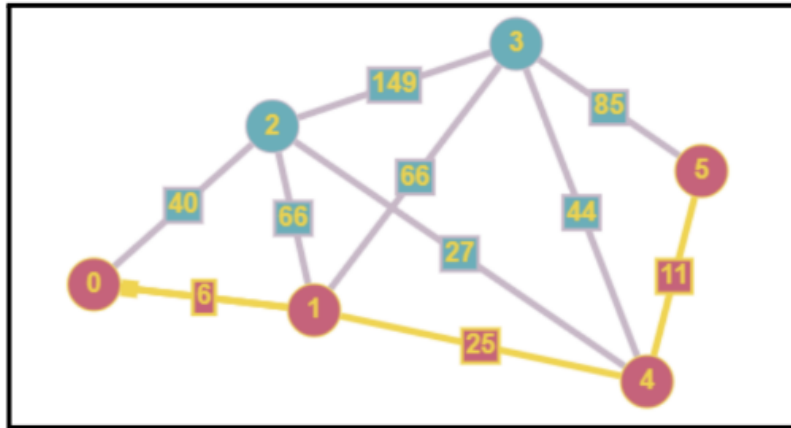
```

Task: using the Adjacency Matrix of n vertices and starting from Earth (**vertex 0**), modify the edge weights using the functionality values of the vertices that each edge connects, and then determine the minimum cost to reach Titan (vertex $n-1$) from Earth (vertex 0). Write this number into the output file.

Some preconditions:

- The generator functionality values are guaranteed to be *between 0 and 1*.
- The energy costs are guaranteed to be positive integers.
- The graph is guaranteed to be undirected.
- The graph is guaranteed to be connected.

For the example above, the path of lowest cost from Earth to Titan is highlighted in red in this image.



Since $(6+25+11) = 42$, you would print 42 into an output file.

42

Iron Man, Dr. Strange, and Spiderman arrive at Titan but are ambushed by the Guardians who think they are part of the Black Order. After some tension, they plan to work together to kill Thanos.

Meanwhile, to protect Vision, the Avengers on Earth head to Wakanda, a country in East Africa which has the technology to remove the Mind Stone in Vision's head *without* killing him in the process.

- [Arriving on Wakanda Scene – Avengers: Infinity War \(2018\) Movie Clip HD \[1080p 50FPS\]](#)

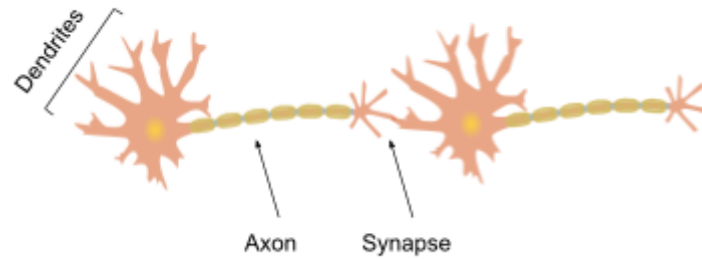
3. MindStoneNeighborNeurons

A Wakandan named Shuri attempts to figure out how to remove the Mind Stone from Vision's head.

- [I'm Sure You Did Your Best – Shuri Helps Vision Scene – Avengers Infinity War Movie Clip HD](#)

Extracting the Mind Stone from Vision without killing him requires the detection of a specific set of neurons that **CONNECT** to the Mind Stone. Given a Directed Graph that contains vertices (neurons) and edges (synapses), the MindStoneNeighborNeurons method returns a list of neurons that connect to the Mind Stone. In the brain, cranial nerves usually have dendrites and an axon. A synapse occurs *between* a dendrite and an axon. The dendrites *receive* information from other nerves, and the axon *transmits* information to other nerves. As such, information always flows *from* dendrites of a nerve, and

then *into* the axon of that nerve. That is why the graph is



For this task, you can assume that the neurons have multiple dendrites and *only* one axon. You can also assume that a nerve's axon *always* connects to **ONLY 1** dendrite of another nerve (or it connects to the Mind Stone).

Also, the Mind Stone does not have any axons, *but* it does receive information from other nerves. As such, the Mind Stone always has an out-degree of 0. Use this to your advantage!

Your input file will be a Set of Edges representation of Vision's neural network.

- The first number n is the number of vertices (neurons). In the below example, $n = 17$.
- The next n lines are the names of the neurons. These are Strings. In the below example, the Strings are a, b, c, d, etc.
- The next line is the number m of edges (synapses). In the below example, $m = 16$.
- The next m lines are the synapses. Each line contains two Strings, so the synapse is between the neurons represented by the first and second String. In the below example, these synapses would be a d, b d, c d, etc.

Preconditions:

- The names of the nerves may not be single letters or characters (instead, they may be strings).
- The input is guaranteed to be a set of edges representation of a neural network (each nerve has 1 axon, every axon connects to only 1 dendrite of another nerve, and the Mind Stone has no outgoing axons). As such, the graph is guaranteed to be connected.

The input file to the right represents the following graph (a neural network). NOTE that in this example, vertex x represents the Mind Stone.

17

a

b

c

d

e

f

h

h

i

j

k

l

m

n

o

p

x

16

a d

b d

c d

d x

f e

g e

h e

e x

j i

k i

i x

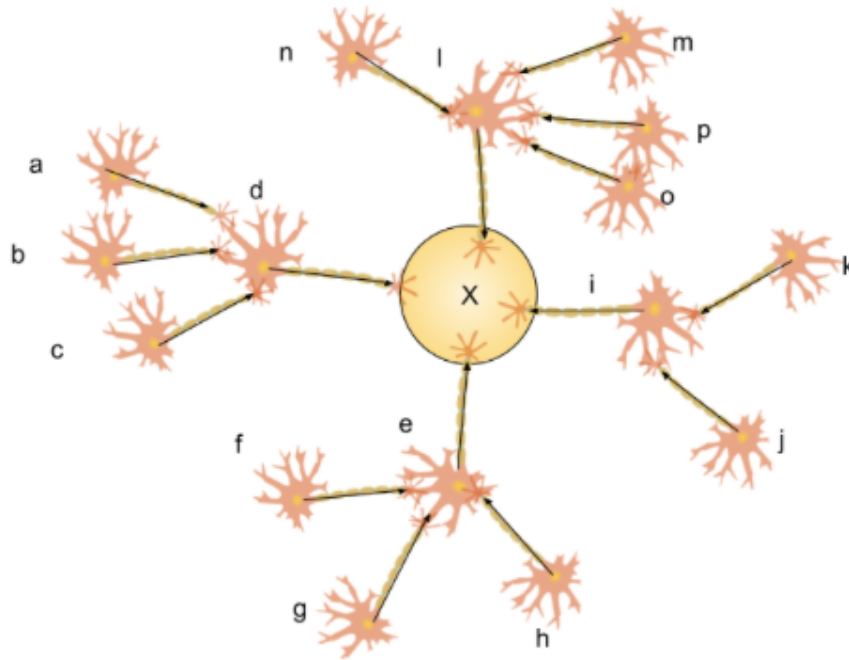
o l

p l

m l

n l

l x



The neurons connecting to the Mind Stone are d, e, i, and l. Print these into the output file, and EACH neuron in a *new line*! It is also fine if there is a new line after the *last* printed neuron also. As such, the corresponding output file looks like this:

```
d
e
i
l
```

Preconditions:

- The names of the nerves may not be single letters or characters (instead, they may be strings).
- The input is guaranteed to be a set of edges representation of a neural network (each nerve has 1 axon, every axon connects to only 1 dendrite of another nerve, and the Mind Stone has no outgoing axons). As such, the graph is guaranteed to be connected.

Task: given a Set of Edges representing Vision's Neural Network, identify all of the vertices that connect to the Mind Stone. List the names of these neurons in the output file.

On Titan, the Guardians plan out with Iron Man and Spiderman regarding an action scheme to stop Thanos. Meanwhile, using his Time Stone, Dr. Strange tries to predict the possible future outcomes, discovering that out of around 14 million, they win only one.

4. Challenge – UseTimeStone (25 points extra credit)

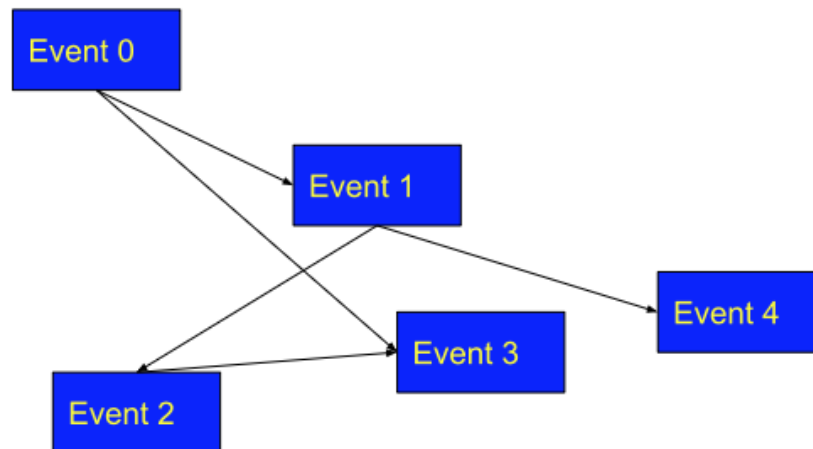
Dr. Strange uses his Time Stone to view possible alternate realities.

- [Avengers: Infinity War \(2018\) – “14,000,605” I Movie Clip HD](#)

When Dr. Strange calculates all the possible futures, the network of events that can potentially occur can be represented by a Directed Graph. The value that Dr. Strange obtained would basically be the number of possible paths through this graph – in other words, the number of possible timelines. Think of each timeline as a “path” through the graph where each vertex represents an event that could potentially occur.

For instance, starting from Event 0 in the graph below, there are 6 possible timelines that could occur.

These timelines are 0, 0-1, 0-1-4, 0-1-2, 0-1-2-3, and 0-3.



First determine the number of possible timelines in the graph, starting from the first event (event 0, in the above example). Print this value into the output file.

Each “event” also has an expected utility (EU) which is a number that shows how “desirable” the event is. The EU of a timeline would be the SUM of the EUs of all the events in that timeline (meaning all the vertices in the path).

In *Avengers: Infinity War*, Dr. Strange only sees 1 timeline where the Avengers win against Thanos. We want to determine the number of timelines starting from event 0 where the EU of the timeline is *at least* the EU of the Avengers winning! The EU of the Avengers winning is given as the threshold EU.

As such, given an input threshold EU value, next determine the number of timelines starting from event 0 where the total EU is GREATER than OR equal to the threshold. Print this value into the output file in a NEW line!

The input file looks like this:

```
5
8
0 2
1 -1
2 3
3 4
4 -6
5 2
6 4
7 -3
0 1 0 0 1 0 0 0
0 0 1 1 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 1 0 0
0 0 0 1 0 0 0 0
0 0 0 0 0 0 1 1
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
```

← Threshold EU
← Number of events (nodes)

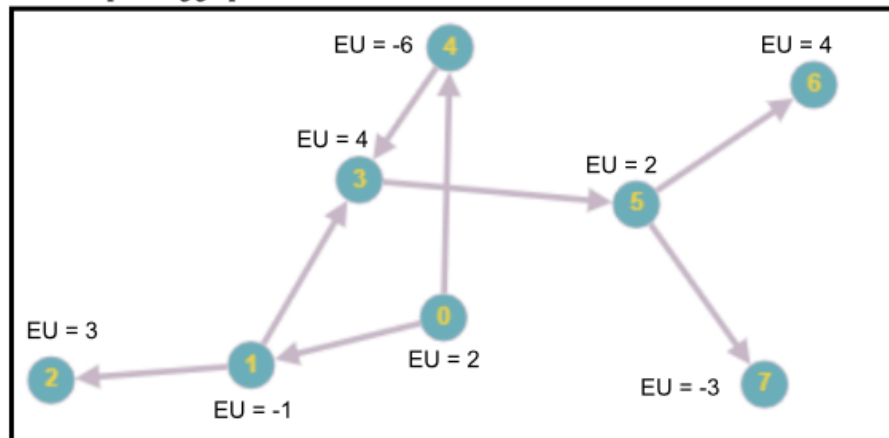
Event number and associated EU value

Adjacency Matrix

NOTE that the row and column numbers (meaning the event number) are not listed; the matrix only displays if there is an edge between 2 nodes.

For instance, there IS an edge between node 1 and node 2 since the value at row 1 and column 2 is 1. Likewise, there is NO edge between node 3 and node 1 since the value at row 3 and column 1 is 0.

The corresponding graph looks like this:



Again, the timelines start from event 0. As such, the possible timelines and corresponding EU values are listed below.

The possible timelines are:

- 0 (EU = 2)
- 0-1 (EU = 2-1 = 1)
- 0-1-2 (EU = 2-1+3 = 4)
- 0-1-3 (EU = 5)
- 0-1-3-5 (EU = 7)
- 0-1-3-5-6 (EU = 11)
- 0-1-3-5-7 (EU = 4)

- 0-4 (EU = -4)
- 0-4-3 (EU = 0)
- 0-4-3-5 (EU = 2)
- 0-4-3-5-6 (EU = 6)
- 0-4-3-5-7 (EU = -1)

Here, there are 12 possible timelines, and there are 4 timelines with an EU higher than or equal to the threshold of 5 (in the input file). Thus, you would print “12” and “4” in separate lines in the output file.

Thus, the output file for the above example looks like this:

```
12
4
```

Some preconditions:

- The EU values are guaranteed to be integers.
- The graph is guaranteed to be connected.

Task: given a starting event and an Adjacency Matrix representing a graph of all possible events once Thanos arrives on Titan, determine the total possible number of timelines that could occur AND the number of timelines with a total EU at least the threshold value. Write this number into the output file.

On Vormir, Thanos sacrifices Gamora’s soul for the Soul Stone by throwing her off a cliff.

1. [Thanos Sacrifices Gamora Scene | Avengers: Infinity War \(2018\) Marvel Movie Clip](#)
2. [Gamora’s Death Scene – Avengers Infinity War \(2018\) Movie Clip HD \[1080p 50FPS\]](#)

Thanos arrives on Titan and after an intense showdown, Dr. Strange gives up the Time Stone to Thanos who was about to kill Iron Man using the 4 Infinity Stones he already had in his gauntlet.

- [Avengers infinity war | battle of titans HD 4k](#)

The Wakandans hold a fierce battle against the Black Order at Wakanda.

- [Avengers: Infinity War \(2018\) – “Battle Of Wakanda” | Movie Clip HD](#)

Eventually, Thor, Rocket, and Groot arrive in Wakanda to assist in defeating the Black Order.

- [Thor Arrives at the Battle of Wakanda in Avengers: Infinity War \(2018\)](#)

Finally, Thanos shows up in Wakanda to take the Mind Stone. Using his Infinity Stones, Thanos easily blocks and incapacitates many of the characters fighting him there including Captain America, Black Panther, and Bruce Banner.

Using all of her power, Wanda destroys the Mind Stone (reluctantly killing Vision in the process) before Thanos reaches her. However, using the Time Stone, Thanos reversed time until Vision was still alive, and then grabs the Mind Stone in Vision's head and kills him.

5. PredictThanosSnap

A final attack by Thor has him dig his Stormbreaker through Thanos' chest. However, Thanos snaps his fingers and teleports away, causing half of all life to disintegrate shortly afterward.

- [Avengers: Infinity War \(2018\) – “Snap Of Disintegration” I Movie Clip HD](#)

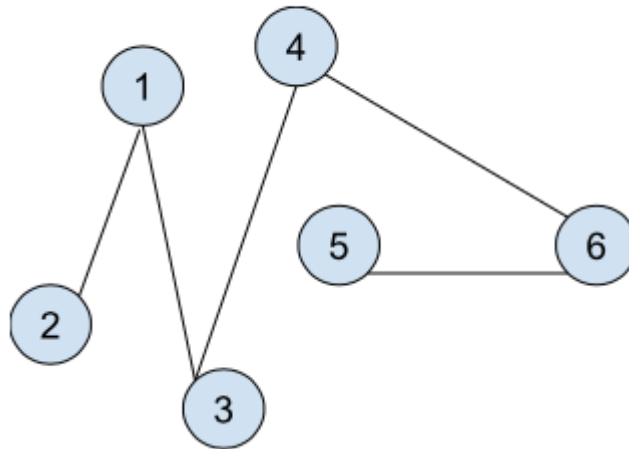
Thanos' snap would have had an immeasurable impact on planets such as Earth. Eliminating half of the population without notice would lead to unprecedented accidents, crashes, power outages, and loss of governmental control.

In the PredictThanosSnap method, we work with a social network (an Adjacency Matrix) where each vertex represents a person. We want to find out if half of the people (vertices) are removed, are the remaining people still able to contact each other? In other words, if half of the vertices are removed, is the graph still connected?

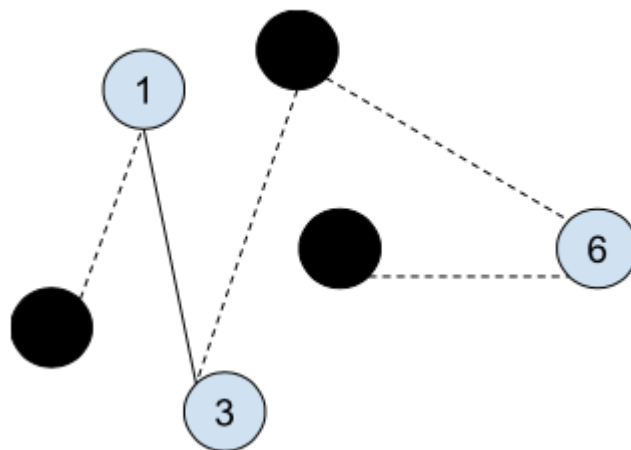
Because Thanos' snap was impartial and unbiased, we use a `random()` function to determine which vertices to eliminate. In essence, each person has a 50% probability of disappearing, and you will use a seed so that the randomization yields the same results in Autolab as well as your program.

Then, return a boolean (true or false) which indicates if the graph is still connected after the removal.

Example (before snap):



Example (after snap):



The social network in the example (after the snap) is *not* connected since person 6 has *no* way of contacting person 1 and person 3 (since person 4 disappears).

The input file looks like this:

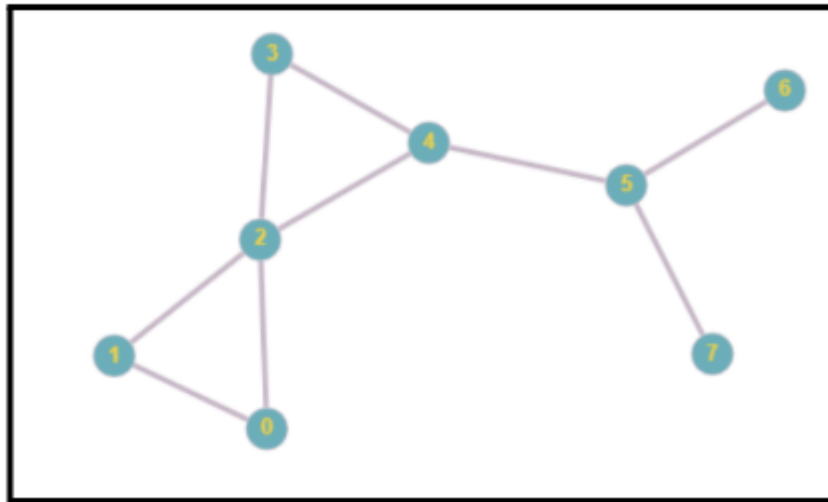
```
1933723380
8
0 1 1 0 0 0 0 0
1 0 1 0 0 0 0 0
1 1 0 1 1 0 0 0
0 0 1 0 1 0 0 0
0 0 1 1 0 1 0 0
0 0 0 0 1 0 1 1
0 0 0 0 0 1 0 0
0 0 0 0 0 1 0 0
```

A number called a *seed*
(the data type is *long*)

Number of nodes

Adjacency Matrix

The corresponding graph looks like this:



After reading in the seed, use *StdRandom.setSeed(seed)* to prepare the seed to use in StdRandom.

Then, call *StdRandom.uniform()* for each vertex (make sure to go IN ORDER from vertex 0 to the last vertex). If *StdRandom.uniform()* is less than OR equal to 0.5, then delete the vertex.

Here is a pseudocode you can use:

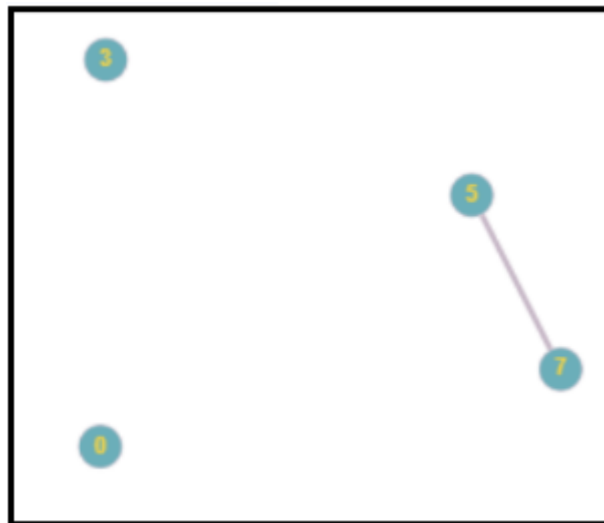
```
Read in the seed as a long int
StdRandom.setSeed(seed);
for ( all vertices, go from vertex 0 to the final vertex ) {
    if ( StdRandom.uniform() <= 0.5 ) {
        delete vertex;
    }
}
```

In theory, the seed ensures that Autolab generates the same random numbers as you when calling `StdRandom.uniform()`. The `StdRandom` class uses the seed to perform the `uniform()` method. In the pseudocode, the 0.5 is chosen since each person has a 50% chance of disappearing.

Also, the choice of deleting when the random number is less than or equal to 0.5 is an arbitrary decision, but you must follow it to ensure consistency between your results and Autolab's results. The same applies for the reason behind iterating from vertex 0 to the last vertex and not in any other order.

Here is more information regarding seeds if you are interested: [Seeds Stack Overflow](#).

The specific seed value causes vertices 1, 2, 4, and 6 to be removed. As such, the corresponding graph looks like this:



This graph is not connected. As such, the output would thus be false.

The output file would then look like this:

```
false
```

Preconditions:

- The graph is guaranteed to be undirected.
- The *input* graph is guaranteed to be connected.

Task: Given an adjacency matrix, use a `random()` function to remove half of the vertices and write into the output file a boolean (true or false) indicating if the graph is still connected.

After a long day of grief, anguish, and diligence, Thanos sits in the Garden and watches the sun rise on a *seemingly* grateful universe, and smiles.

- [Avengers: Infinity War – A Poignant Ending \(Thanos Wins\)](#)



The end.

Helpful Java Classes

The following are some data structures which are automatically imported with “**java.util.***” that can help make your code cleaner and more efficient. You are free to not use any of these, and you are also free to use any other class under “java.util.*” as you see fit. I will not be covering every single method for these data structures, just some useful ones for this assignment. You can find more information about how to use these classes online.

- **ArrayList** is an ordered array-like structure with no size limit, as it automatically resizes
 - You can initialize an empty ArrayList named “name” which holds objects of type “Type” with `ArrayList name = new ArrayList();`
 - For example, an ArrayList of integers named “arrList” is initialized with `ArrayList arrList = new ArrayList<>();`
 - You can add a new element of type “Type” to the end of your ArrayList in average case $O(1)$ time with `name.add(newElement);`
 - You can get the element at some index of your ArrayList in $O(1)$ time with `name.get(index);`

- You can set some index to some new element in $O(1)$ time with `name.set(index, newElement)`;
 - You can check if the `ArrayList` contains some element (returns a boolean) in $O(n)$ time with `name.contains(element)`
- **HashMap** is an unordered data structure which stores and retrieves key value pairs
 - You can initialize an empty `HashMap` named “name” that maps objects of type “Key” to objects of type “Value” with `HashMap<Key, Value> name = new HashMap<>()`;
 - For example, a `HashMap` named “map” which maps strings to integers is initialized with `HashMap<String, Integer> map = new HashMap<>()`;
 - You can add a new key value pair, or update an existing key with a new value in average case $O(1)$ time with `name.put(key, value)`;
 - You can check if the `HashMap` contains some key in average case $O(1)$ time (returns a boolean) with `name.containsKey(key)`
 - You can check the value of some key in the `HashMap` in average case $O(1)$ time with `name.get(key)`
 - You can iterate over all the keys in the `HashMap` with for (`Key key : name.keySet()`) where `Key` is the type of keys in the `HashMap`.
- **HashSet** is an unordered data structure which only stores keys
 - You can initialize an empty `HashSet` named “name” that stores objects of type “Key” with `HashSet name = new HashSet<>()`;
 - For example, a `HashSet` named “set” which stores strings is initialized with `HashSet set = new HashSet<>()`;
 - You can add a new key to the hash set in average case $O(1)$ time with `name.add(key)`
 - You can check if a key exists in the hash set in average case $O(1)$ time (returns a boolean) with `name.contains(key)`
 - You can remove a key in average case $O(1)$ time with `name.remove(key)`
 - You can iterate over all the keys in the `HashSet` with for (`Key key : name`) where `Key` is the type of keys in the `HashSet`.

VSCode Extensions

You can install VSCode extension packs for Java. Take a look at [this tutorial](#). We suggest:

- [Extension Pack for Java](#)
- [Project Manager for Java](#)
- [Debugger for Java](#)

Importing VSCode Project

1. Download the zip file from [Autolab Attachments](#).
2. Unzip the file by double clicking it.
3. Open VSCode
 - Import the folder **InfinityWar** to a workspace through **File > Open Folder**

Executing and Debugging

- You can run your program through VSCode or you can use the Terminal to compile and execute. We suggest running through VSCode because it will give you the option to debug.
- [How to debug your code](#)
- If you choose the Terminal, from InfinityWar directory/folder:
 - to compile: **javac -d bin src/avengers/*.java**
 - to execute ForgeStormBreaker: **java -cp bin avengers.ForgeStormBreaker forgestormbreaker.in forgestormbreaker.out**

Zippping the directory for submission (**READ THIS before submitting**)

- Be careful when zipping the directory for submission.
- Autolab is expecting the exact directory organization we provided to you.
- Autolab is expecting the zip file to be named **infinitywar.zip**
- All files that you have written **MUST** be in the **InfinityWar/src/avengers** directory.

To zip the **InfinityWar** directory navigate to the parent directory:

- **zip -r infinitywar.zip InfinityWar**

Inspect the zip by listing the files in zip without uncompressing it:

- **unzip -l infinitywar.zip**

Your zip file must have the following structure:

```

((base) $ unzip -l infinitywar.zip
Archive:  infinitywar.zip
  Length      Date    Time    Name
-----
0         11-18-2022 18:47    InfinityWar/
0         11-16-2022 19:02    InfinityWar/bin/
0         11-18-2022 18:47    InfinityWar/bin/avengers/
708       11-16-2022 19:02    InfinityWar/README.md
0         11-16-2022 19:02    InfinityWar/lib/
0         11-16-2022 19:02    InfinityWar/.vscode/
154       11-16-2022 19:02    InfinityWar/.vscode/settings.json
0         11-16-2022 19:02    InfinityWar/src/
0         11-16-2022 19:02    InfinityWar/src/avengers/
2505      11-18-2022 18:42    InfinityWar/src/avengers/LocateTitan.java
2199      11-18-2022 18:42    InfinityWar/src/avengers/MindStoneNeighborNeurons.java
9570      11-18-2022 18:42    InfinityWar/src/avengers/StdOut.java
24012     11-18-2022 18:42    InfinityWar/src/avengers/StdRandom.java
27420     11-18-2022 18:42    InfinityWar/src/avengers/StdIn.java
2449      11-18-2022 18:42    InfinityWar/src/avengers/PredictThanosSnap.java
2448      11-18-2022 18:42    InfinityWar/src/avengers/ForgeStormBreaker.java
2584      11-18-2022 18:42    InfinityWar/src/avengers/UseTimeStone.java
137       11-18-2022 18:46    InfinityWar/mindstoneneighborneurons.in
25        11-18-2022 18:46    InfinityWar/forgestormbreaker.in
183       11-18-2022 18:47    InfinityWar/usetimestone.in
242       11-18-2022 18:46    InfinityWar/mindstoneneighborneurons1.in
155       11-18-2022 18:46    InfinityWar/locatetitan1.in
91        11-18-2022 18:47    InfinityWar/usetimestone1.in
834       11-18-2022 18:47    InfinityWar/predictthanosnap2.in
121       11-18-2022 18:46    InfinityWar/locatetitan.in
32        11-18-2022 18:46    InfinityWar/forgestormbreaker1.in
1844      11-18-2022 18:47    InfinityWar/predictthanosnap3.in
190       11-18-2022 18:46    InfinityWar/forgestormbreaker3.in
295       11-18-2022 18:46    InfinityWar/mindstoneneighborneurons3.in
118       11-18-2022 18:46    InfinityWar/locatetitan3.in
149       11-18-2022 18:47    InfinityWar/predictthanosnap.in
279       11-18-2022 18:47    InfinityWar/usetimestone3.in
392       11-18-2022 18:46    InfinityWar/mindstoneneighborneurons2.in
360       11-18-2022 18:46    InfinityWar/locatetitan2.in
120       11-18-2022 18:47    InfinityWar/usetimestone2.in
91        11-18-2022 18:47    InfinityWar/predictthanosnap1.in
32        11-18-2022 18:46    InfinityWar/forgestormbreaker2.in
-----
79739                                37 files

```

Before submission

Collaboration policy. Read our collaboration policy [here](#).

Submitting the assignment.

You will have to submit a zip file. See previous section on how to zip the directory.

Submit *infinitywar.zip* separately via the web submission system called Autolab. To do this, click the *Assignments* link from the course website; click the *Submit* link for that assignment.

Getting help

If anything is unclear, don't hesitate to drop by office hours or post a question on Piazza.

- Find instructors and head TAs office hours [here](#)
- Find tutors office hours on Canvas -> Tutoring -> RU CATS
- In addition to office hours we have the **CAVE** (Collaborative Academic Versatile Environment), a community space staffed with lab assistants which are undergraduate students further along the CS major to answer questions.

[Connect with Rutgers](#)

[Rutgers Home](#)

[Rutgers Today](#)

[myRutgers](#)

[Academic Calendar](#)

[Calendar of Events](#)

[SAS Events](#)

[Explore SAS](#)

[Departments & Degree-Granting Programs](#)

[Other Instructional Programs](#)

[Majors & Minors](#)

[Research Programs, Centers, & Institutes](#)

[International Programs](#)

[Division of Life Sciences](#)

[Explore CS](#)

[We are Hiring!](#)

[Research](#)

[News](#)

[Events](#)

[Resources](#)

[Search CS](#)

[Home](#)

[Back to Top](#)