Amirtha Ganesh – Network Generation

1) Structure & function of the Program:

This part of the assignment contains 4 files – network.c (containing the helper functions), network.h (header file for helper functions), a common main.c (which contains the ring graph of the village) and makefile.

Description of Ring Graph Network generation (Adjacency Matrix)

- In order to create the ring graph, I created 2 data structure namely **"ringgraph"** to recreate the setting the graph with node name, left side neighbors, right side neighbors and the node status
- The second data structure is "**village**" which stores the **villagers** in form of "**ringgraph**"
- ***Functions:***

1)**"euclidean_mod**"- To find the Euclidean modulus and return positive mod(re) value
2) **"randprob"** - To roll a die and generate a random probability between 0 and 1
3) **"select_random_node"** - To select random node and return the randomly nodes selected.
4**) "check_array" -** to check whether the selected random node is already a neighor.
5) **"make_network "** - to generate ringnetwork for each villager in the village 'v'

Ring Graph Network generation:

- **"pReplaceRandom" :** This a function Parameter which decides the degree of random nodes to the villager. If the "pReplaceRandom" is 0 then there is no random neighbors or else there will be some positive probability for selecting the random neighbor for a villager
- **make_network(pv,neighbor,pReplaceRandom) :** By calling the functions creates the with those three parameters creates a village with ring network graph relationship, where **"pv"** is the pointer to the village **"v"** , **"neighbor**" is the number of neighbors for each villagers and "**pReplaceRandom"** informs random replacement probability.
- If the **pReplaceRandom =** 0 then, the villagers are mapped as ring graph through looping and **offsetting the neighbors appropriately** using "**Euclidean modulus**" or else random probability is found and if it is more than (1- **pReplaceRandom)** than neighbors are selected randomly by calling the function "**select_random_node"** or else neighbors sampling offsetting appropriately.

Athulya Ram – Epidemic Model Simulation

1) Structure & function of the Program:

This part of the assignment contains 4 files – model.c (containing the helper functions), model.h (header file for helper functions), a common main.c (which contains the epidemic model) and makefile.

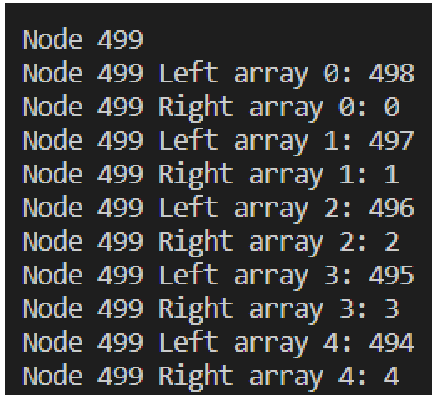
Description of agent-based SIR epidemic model:

- I keep track of the agents (individuals in the population) using a matrix *Agents[numIterations+1][numAgents].* Here each row of the matrix corresponds to an iteration, and each column tracks the status of an individual (with markers: S=0, I=1, R=2).
- I keep track of the total number of susceptible, infected, and recovered individuals in each iteration using the arrays *S_counter[], I_counter[],* and *R_counter[].*
- A random person is chosen to be initially infected. The corresponding matrix entry is set to 1 (for infected).
- Looping through the population using a *for* loop, first, I disconnect edges of a newly infected person according the probability *pDisconnect.* For the first iteration, the initially infected person undergoes this process. For the following iterations, every person who was infected in the previous iteration (*Agents[i-1][j] == 1 && Agents[i-2][j] == 0*) goes through disconnection. This way the effects of disconnection is only seen in the next iteration. When a neighbour is disconnected, their corresponding entries in the "neighbour arrays" of the graph structure is set to –1, so they don't count anymore.
- Next, I loop through all neighbours of an individual and find the total number of infected neighbours (with the help of markers in matrix *Agents[][][]*). As *pInfection* is the probability of infection per contact, the infection rate *inf_rate = pInfection*number of infected neighbours.*
- The status of an individual is determined by checking if a randomly selected number in (0,1) is lesser than a given probability. *S* becomes *I* with the probability *inf_rate* (note that this is zero if an individual has no infected neighbours), and *I* becomes *R* with the probability *pRecovery.*
- This process loops for 100 iterations using a *for* loop, denoting 100 days. After each iteration, the total number of *S, I, R* is stored in corresponding arrays.
- We run the model 10 times using a *for* loop and prints the results for each of these 10 times.

2. Evidence of correct operation

1. Structure of ring graph

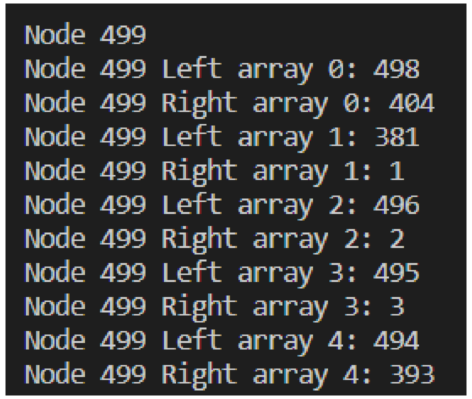
When *pReplaceRandom* and *pDisconnect* is set to 0, we get a ring graph.
Example (with 10 neighbours): Node 499 has neighbours (494, 495, 496, 497, 498, 0, 1, 2, 3, 4). This demonstrates the ring structure.



```
Node 499
Node 499 Left array 0: 498
Node 499 Right array 0: 0
Node 499 Left array 1: 497
Node 499 Right array 1: 1
Node 499 Left array 2: 496
Node 499 Right array 2: 2
Node 499 Left array 3: 495
Node 499 Right array 3: 3
Node 499 Left array 4: 494
Node 499 Right array 4: 4
```

2. Replacing random edges

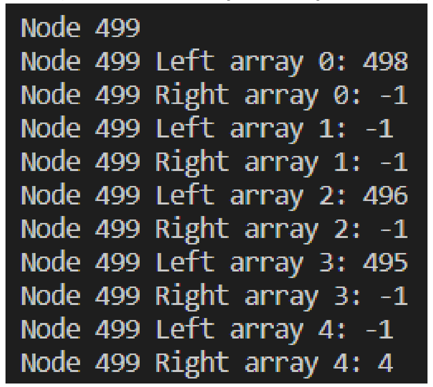
When *pReplaceRandom* is set to 0.25, we get a small-world graph.
Example (with 10 neighbours): We can see that neighbours 0, 497, and 4 (roughly 25%) has been replaced with distant nodes.



```
Node 499
Node 499 Left array 0: 498
Node 499 Right array 0: 404
Node 499 Left array 1: 381
Node 499 Right array 1: 1
Node 499 Left array 2: 496
Node 499 Right array 2: 2
Node 499 Left array 3: 495
Node 499 Right array 3: 3
Node 499 Left array 4: 494
Node 499 Right array 4: 393
```

3. Disconnecting edges after S->I

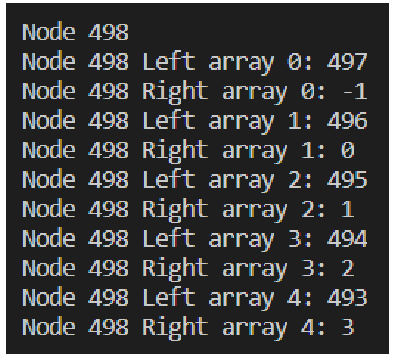
When *pDisconnect* is set to 0.5 and node 499 is infected, some of the edges get disconnected.
Example (with 10 neighbours): We can see that roughly half of the nodes have been disconnected (denoted by entry of –1).



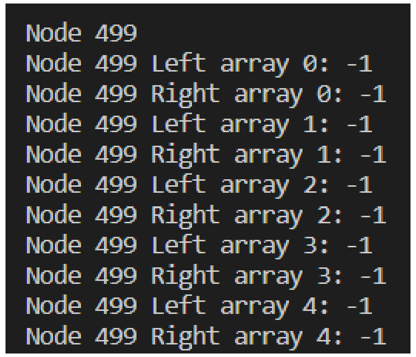
```
Node 499
Node 499 Left array 0: 498
Node 499 Right array 0: -1
Node 499 Left array 1: -1
Node 499 Right array 1: -1
Node 499 Left array 2: 496
Node 499 Right array 2: -1
Node 499 Left array 3: 495
Node 499 Right array 3: -1
Node 499 Left array 4: -1
Node 499 Right array 4: 4
```

4. Evidence that the model works

When *pDisconnect = 1* and node 499 is infected initially, all edges are disconnected and there is no dynamics.

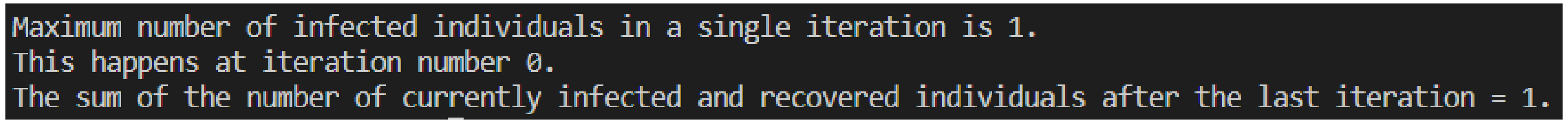


```
Node 498
Node 498 Left array 0: 497
Node 498 Right array 0: -1
Node 498 Left array 1: 496
Node 498 Right array 1: 0
Node 498 Left array 2: 495
Node 498 Right array 2: 1
Node 498 Left array 3: 494
Node 498 Right array 3: 2
Node 498 Left array 4: 493
Node 498 Right array 4: 3
```



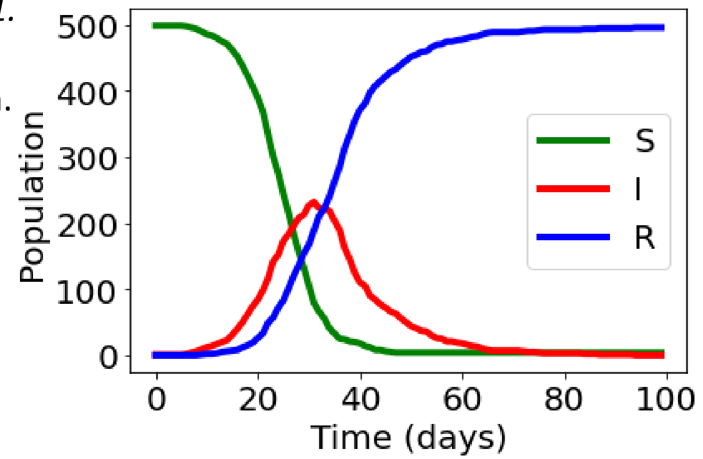
```
Node 499
Node 499 Left array 0: -1
Node 499 Right array 0: -1
Node 499 Left array 1: -1
Node 499 Right array 1: -1
Node 499 Left array 2: -1
Node 499 Right array 2: -1
Node 499 Left array 3: -1
Node 499 Right array 3: -1
Node 499 Left array 4: -1
Node 499 Right array 4: -1
```

Node 499 has been disconnected with all its neighbours.
Node 498 has been disconnected with node 499.



```
Maximum number of infected individuals in a single iteration is 1.
This happens at iteration number 0.
The sum of the number of currently infected and recovered individuals after the last iteration = 1.
```

Results of a single realization of the model, with the basic epidemic parameters descibed in the pdf, *pReplaceRandom = 0.25, pDisconnect = 0.1.*
Simulated in C,
Plotted in python.



3) Results and inference

A. The maximum number of infected individuals in a single iteration.

B. The iteration number at which the maximum number of infected individuals occurs.

C. The sum of the number of currently infected and recovered individuals after the last iteration.

Paramet ers	pReplaceRandom = 0, pDisconnect = 0			pReplaceRandom = 0.25, pDisconnect = 0			pReplaceRandom = 0, pDisconnect = 0.5			pReplaceRandom = 0.25, pDisconnect = 0.5		
Result	**Median**	**Max**	**Min**	**Median**	**Max**	**Min**	**Median**	**Max**	**Min**	**Median**	**Max**	**Min**
A	23.5	31	1	205.5	244	185	6	18	1	4	183	1
B	47.5	92	0	29.5	39	24	12.5	45	0	10	36	0
C	123.5	171	1	493.5	497	488	10.5	71	1	4	481	1

- **Case 1: (pReplaceRandom = 0, pDisconnect = 0)**: This is the dynamics of the epidemic when individuals interact only with their neighbours, but no disconnection occurs. As an individual only interacts with their neighbours, the epidemic does not spread far and less than 50% of the population is affected.
- **Case 2: (pReplaceRandom = 0.25, pDisconnect = 0):** This is the worst-case scenario. Individuals randomly interact with the population, and no disconnection occurs due to the disease. We can see that nearly all of the population is infected.
- **Case 3: (pReplaceRandom = 0, pDisconnect = 0.5):** This is the best-case scenario. Individuals interact only with their neighbours, and once an individual gets infected there is a 50% chance that each of their neighbours stop interacting with them. This is reflected by the low maximum peak and low maximum total cases.
- **Case 4: (pReplaceRandom = 0.25, pDisconnect = 0.5):** Here, the individuals interact randomly with the population, but to counter that, there is a 50% chance that disconnection occurs. So the results are lower than case 2 where there is no disconnection. But we can see that there is very high variability in the results, due to the high chance of disconnection.

To conclude, the parameters *pReplaceRandom* and *pDisconnect* have very important roles in the epidemic. *pReplaceRandom* dictates how well the population is mixed and *pDisconnect* tells us how often an infected person is 'isolated'. To eradicate the epidemic successfully, we need to decrease *pReplaceRandom* and increase *pDisconnect*.

4)Division of labor

Activity/Role	Amirtha Ganesh Pugazhendhi	Athulya Ram
Network Generation (Program development and debugging)	100%	NA
Epidemic Modelling (Program development and debugging)	NA	100%
Documentation and Slide preparation	50%	50%
Approximate man-hours	24	24