Memory Simulation

Josh Levy and Max Day

Introduction Details

Since our group had more experience with Java, we decided to use that for our project. Our project is uploaded to github (https://github.com/jml312/Simulator) with all documents and outputs included. We have not included any other files in Canvas besides this report, but if needed, we can turn in any file necessary. In terms of project involvement, about 95% of the project was done synchronously over Zoom. While the rest of this document will discuss the project, various comments are made in the code for your understanding as well.

Part 1a

For 1a, our simulator created its own MemoryPartition class (MemoryPartition.java). Each partition had an associated int of pages, a double of the probability of the partition being requested (req), a double of probability of the partition being freed (free), Boolean of whether or not it is allocated, and an array of the pages it currently allocates. Each partition also included an array of double probabilities of size n. This was used instead of a matrix to hold the probability of being requested given any j is allocated (metoo). Each size(), req(), free(), and metoo() were all generated randomly. As time is incremented, the simulator has a random uniform chance of picking one of three options to generate a request: request, free, or metoo. Only one of these three will happen every timestep and it always begins with a request. When an option is picked, a random double is generated. This double is compared across the entire array of partitions starting from the first. The first array with its own probability be lower than the random double is assigned to be partitioned. For the request and metoo, the partition must not be allocated. For the free to execute, it must be allocated. If not, it moves to the next one. For metoo(), we chose the probability that corresponded with the last partition that had been allocated. These functions would generate t requests to be performed by the simulation.

Part 2a

The simulator main method then assigned each of the partitions to an array of size b. In this part, n, b, and t could all be changed to give different effects. After each time step, the request that was completed (if any), the degree of fragmentation, and the range of free pages was output. Fragmentation was calculated as the total amount of free pages subtracted by the largest contiguous block of free pages all divided by the total amount of free pages. In the example below, the amount of pages requested by the partition is 3, the range that they are allocated to is 0-2, it begins at time step 1, it is the 19th partition generated, ‘the probability of that partition being requested is shown, and the fragmentation and free page list is displayed.

```
pages requested: 3
Page found at range 0 - 2
@t = 1 | REQUESTING partition 19 | pages =  3 | prob = 0.05
@t = 1 |  fragmentation = 0.0 | pagesList = [3, 999]
```

Instead of first-fit or best-fit, we chose the ith-fit. For a block of i pages, the block can only be allocated at every i pages, meaning that a block of 5 pages could only be allocated starting on page 0, 5, 10, 15, 20 and so on. We believed this approach would be interesting since it is similar to first fit, but would most likely be more fragmented given the small free blocks that could exist between partitions. Since we did not have another list and were only using the array of partitions we generated to be allocated and freed, we did not have any overhead.

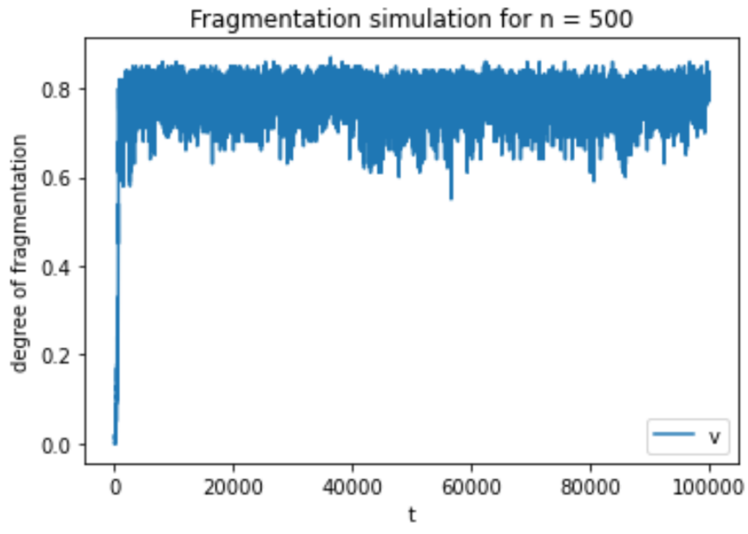
Part 1b

Each chunk of code will be explained individually.

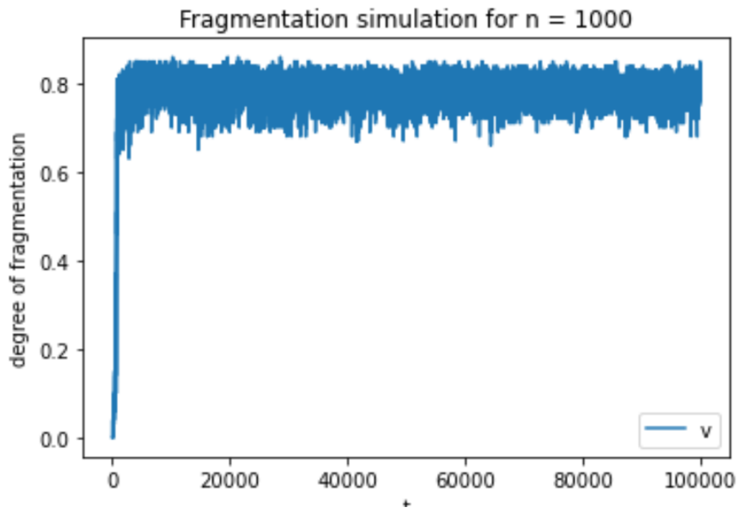
```
pages requested: 3
Page found at range 0 - 2
@t = 0 | REQUESTING partition 1 | pages =  3 | prob = 1.0
@t = 0 |  fragmentation = 0.0 | pagesList = [3, 19]
[1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]

pages requested: 2
Page found at range 4 - 5
@t = 1 | REQUESTING partition 2 | pages =  2 | prob = 1.0
@t = 1 | FREEING partition 2 | pages =  2 | prob = 1.0
pages requested: 2
Page found at range 4 - 5
@t = 1 | partition 2 | pages =  2 | meToo = 1.0
@t = 1 |  fragmentation = 0.07 | pagesList = [3, 3][6, 19]
[1, 1, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]

@t = 2 | FREEING partition 3 | pages =  1 | prob = 1.0
@t = 2 |  fragmentation = 0.07 | pagesList = [3, 3][6, 19]
[1, 1, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]

pages requested: 10
Page found at range 10 - 19
@t = 3 | REQUESTING partition 4 | pages =  10 | prob = 1.0
@t = 3 |  fragmentation = 0.2 | pagesList = [3, 3][6, 9]
```

```
[1, 1, 1, 0, 1, 1, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]

pages requested: 11
No page range available
@t = 4 | REQUESTING partition 5 | pages =  11 | prob = 1.0
@t = 4 |  fragmentation = 0.2 | pagesList = [3, 3][6, 9]
[1, 1, 1, 0, 1, 1, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]

pages requested: 1
Page found at range 3 - 3
@t = 5 | REQUESTING partition 6 | pages =  1 | prob = 1.0
@t = 5 |  fragmentation = 0.0 | pagesList = [6, 9]
[1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]

pages requested: 5
No page range available
@t = 6 | REQUESTING partition 7 | pages =  5 | prob = 1.0
@t = 6 |  fragmentation = 0.0 | pagesList = [6, 9]
[1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]

@t = 7 |  fragmentation = 0.0 | pagesList = [6, 9]
[1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]

@t = 8 |  fragmentation = 0.0 | pagesList = [6, 9]
[1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]
```

Part 2b



We began our simulation with parameters n=500, t=100000, and b =1000. The output is in the same git repository as output1.txt and fragmentation1.csv. In this simulation we see a quick rise in fragmentation as it reaches 0.8 fragmentation by timestep 680. After 36410 timesteps, the simulation hits its height in fragmentation at 0.85. The simulation had an average fragmentation of 0.7752. After 100000 timesteps, the fragmentation ends at 0.79 with 69 free blocks, 41 individual chunks, the largest one being 5 blocks in length. From this, we can infer that since chunks were only freed ⅓ of the time, an equilibrium was reached by the end of around 1000 timesteps, centered around .80. There were multiple blocks that could not be allocated at many points in the simulation.



To test against this, we changed n to be 1000, which can be seen in output2.txt. Fragmentation rose quicker as it reached 0.8 at timestep 555. It reached a height of 0.87 at 6256 timesteps, reaching it much quicker than previous simulation. The simulation had an average fragmentation of 0.7797, slightly higher than the previous. After 100000 timesteps, the simulation ended with a degree of fragmentation of 0.76 with 58 free blocks, 41 individual blocks, and a largest chunk of 6 pages.

This first simulation, while they had similar fragmentation, had many more free blocks with the same amount of individual blocks. It seems that the second simulation climbed faster and hit its maximum much faster, meaning that the increasing the amount of available pages, we might see an equilibrium reached. Raising the number of blocks allowed for there to almost always be a partition that matched the probability and it also made sure that there were always partitions generated for metoo().

Concluding Remarks

The ith-fit algorithm, much like the best fit algorithm, left many tiny holes throughout the block. It did however take less time to allocate and most likely would have had close to the same fragmentation throughout the simulation. While we are still unsure of how efficient the ith-fit algorithm (if at all), more testing could be applied to help decide.