Programming Project 1

CS146 Data Structures and Algorithms

Benjamin Garcia Section 02

Part A: Data Shuffling

Task to be Completed:

In this part of the project, we are asked to create a project that takes a text file as an input and use the Fisher-Yates shuffle algorithm to shuffle all lines in the file and output the shuffled result to a new text file. The first line of the text file is formatted with a "%" character followed by the number of distinct numbers in each column and then the number of data lines in the file. The shuffle should not include this first line in the output file. The algorithm used is to start at the last line and then swap it with a randomly selected element from the beginning of the data set up to the line needing to be shuffled (including that line). Once this is done, move to the next line up and perform this technique all the way to the first element.

Solution:

The way I accomplished this task is by creating a Buffered Reader object to read the input file and a Print Writer object to output the shuffled text to the new text document. I then use a Scanner to scan through just the first line to get the 3rd integer which is the number of data lines there are in the input file. Using that number, I created an array with that same amount of space. After, I used the Buffered Reader object to read each line after that and add it to the array. Now, I call the shuffle algorithm on the array to shuffle array elements and then use the Print Writer object to output each array element as a single line to the newly created output file.

Cases to Consider:

In this part of the project, there were not many different cases to consider. The only odd case I can think of is if the first line did not give the number of elements. The way I would get around this is by creating an ArrayList instead of an Array. But it is given in the instructions that the passed in file will be laid out that way with the first line giving information about the data in the text file.

Problems I Ran Into:

One problem I ran into was when shuffling, the instructions said to make the random number generated inclusive of the line being shuffled. When looking on the Java documentation for the Random class, it said the method nextInt(int bound) generated the next pseudorandom integer between 0 (inclusive) and the specified value (exclusive). But, when I made it inclusive by adding 1, the output file did not match. After removing the 1, it matched the target file, but this meant the "random" int being generated is not inclusive of the element being shuffled.

Run Time:

The time I calculated for reading from the input file was always close to 14ms, to shuffle the elements, it was close to 2ms, and creating the final output file took around 9ms.

Applications:

Applications for this shuffle include any task that would require randomizing any number of elements. One example would be a card game that requires a shuffle of 52 cards to give everyone playing a fair advantage. Another example would be if a teacher wanted to create a randomly generated seating chart. By inputting a list of names, the algorithm will output a shuffled list that will mix the students' to random seat numbers.

Part B: Circular Linked List Game

Task to be Completed:

In this part of the project, we are asked to implement a game in which there are a certain number of prisoners lined up and starting at the first, the King of an ancient land would count a certain number of prisoners and eliminate the prisoner he stops on. He keeps doing this with the same count each time until there is only one prisoner remaining who "wins" and is then granted freedom. During each count, if the King reaches the end, he loops back to the front of the line and continues counting.

Solution:

The way I accomplished this task is by creating a doubly Linked List. This meant that each node, or in this case, prisoner, had a pointer to the previous and the next. This made it easier to traverse the Linked List and to remove when needed. To insert the nodes, I created the head node and set that to id 1. Then, I created a "previous" node and set its initial value to be a pointer to the head. Then, I made a loop that started at 2 and went up to the amount that was supposed to be inserted. I assign the "previous.next" to be the new node with id of whatever the "i" value is on the for loop and set the new node's previous to be the current "previous". Then, increment "previous" to point to the new node. At the end when all elements are inserted, I set the next of the current element to the head and the head's previous to the current element, so it creates a loop. When playing the actual game, I set the start position at the head and then create a loop to move to the next node until the "i" in the for loop goes over the step value passed in. Then it will remove the node by reassigning the pointers of the node previous and next of the node to jump over the node to be removed. Then it will set the new node to start counting from as the next node in the loop. It will keep doing this until one node is left. The remaining node's id is then set to the winner.

Cases to Consider:

The cases I had to consider was if the insert method was called with a negative number in which case, I made it so that nothing would happen. I also considered if the step passed in was negative or if the play method was called when the size was less than or equal to 0. If these cases happened, it would print a line in the console and not run anything else.

Problems I Ran Into:

The problem I ran into was that at first, I did not know how to create the Linked List with a given number of nodes. I knew it had to be some sort of loop, but I did not know how to keep track of the current node to link to a newly made node. I finally figured out that I had to first create a node before the loop and set its initial value as a pointer to the head. Then, I could assign the next of it and keep

track of it that way. To remove a node was not difficult because I had created a doubly Linked List, so I just had to move the pointers from it's previous and next nodes.

Applications:

An application of this would be if one object wanted to be selected in a group. It would be easier to simply generate a single random number, but this way turns the selection into more of a game since it eliminates objects one by one. It would also be better because if someone knew the step count, they could calculate where to be to win the game.