605.202: Introduction to Data Structures

Thomas Muamar

Project 3: Scorpion Solitaire

Due Date: Aug 3,2021

Dated Turned In: Aug 2, 2021

Project 3: Scorpion Solitaire

Justification for data structures/reasons for choices:

Starting off with the main input which is reading the text file. The text file is read through a buffered reader. Once the text file has been read to a string deck it will then get used in the class scorpion.java were the inputDeck method reads the input text to be used as a linked list deck. In order to do this the big issue was trying to figure out how to read the only value of 10. In order to do this, I developed an if else statement that would develop a string for both the rank and the suit. As it would go through those it would then be added to the linked list that was developed. Made the node for the linked list within the card class to make it easier to implement into the linked list.

Linked List structure was really important here because it allowed me to iterate through the columns and compare values of in between columns. When removing a card from the column the linked list would also allow an overall better removal process because you didn't have to delete a bunch of empty rows it was just shifting the head node to point to the previous card in that list. The linked list structure also was the best choice here because inserting and deleting the elements is a lot quicker and easier.

My choice for developing the game board was as an array of linked lists that would insert the cards row by row to allow for accessing of each column. Array of linked lists is a useful data structure because it combines a static structure with a dynamic structure. The reason to use an array of linked list is because before hand we know the amount of stuff that is going to be on the board but the only thing, we don't know is the number of nodes that are going to be in each of the columns. So, the linked list allows for better memory when dealing with unknown amounts.

One of the most important aspects of this lab was the computer decisions. The way I decided to do this was when it was traversing through the columns, and it found any possible moves it would then do that move first. If there is a king as it traverses through those values and there is an empty column it will switch that king to the empty columns and what ever cards are underneath. So basically, selects the card and then checks each of the cards at the end of that column to see if it can attach. The last three cards that are left are dealt out when the computer doesn't have any more moves to make.

Once the computer runs out of moves the game will prompt user if they want to play again. If user decides to play again then I decided to take the input text into a list and then shuffle that list. Once that list was shuffled, I would put it back into a string and then run the game with that shuffled deck.

What I Learned:

Through this project I learned of a new solitaire game that is impossible for me to win. I also learned a lot more about linked list structures and how to apply them in different scenarios. As I did here apply the linked list class using the card class as the node to depict the next and previous cards through out the columns of the tableau.

Learned how to read a file that contains a deck of cards. Was help full to read the discussion boards allowing me to separate the rank and suit into different strings instead of how I did it the first time with splitting the data by spaces.

What I might do differently next time/if it was a user played game:

What I might do differently next time is probably use an array to first depict the cards to then shuffle that array and then implement that array into a linked list of cards. It would make it easier to shuffle the deck of cards from the input file and not have to shuffle a linked list of cards for the next games that are to be played.

Something that also could have been done differently if there was more time is to find a more optimal way of searching for the solutions. Maybe trying to find the best solution. I feel like the best solution would be to try and have the first four columns to become empty to allow for more moves. Mostly just figuring out a way to win the game because I couldn't really figure out a way to win the game on the website.

I don't think the structures would differ all too much if the user was able to play the game themselves. Just have to prompt the user of which column the card is in they want to select and then the card they would like to select in that given column. Then once they have the selected card they can then move it to one of the bottom rows were it will make a comparison by the rank and suit.

Issues of Efficiency:

The issues with using a linked list structure are having to access each node sequentially. So, in this lab the problem was that you would have to traverse through each column in order to find out whether there was a matching value to the selected card. Traversing through each column with a linked list would produce a time complexity of O(n). While with an array the searching would be at constant time of O(1). The best part about using the linked list for this solitaire problem is that moving from one to the other isn't as costly as an array because the memory locations are repeated and static in an array.

The best part about using a linked list structure is space isn't that big of a deal. With this project having a linked list structure helps because when you need to move a whole stack of cards under neath another stack of cards the linked list won't have any issue. Unlike an array that has dynamic size that could end up causing an issue if the pile of cards gets too large.