Marcellus Hunt

2/15/18

362 Project 1

Project Report

For planning my project, I came up with an idea to have a Tour class that would use the linked list and stack class in aiding to run the Tour function which would do the Warnsdoff heuristic and exhaustive procedure for the tour while pushing the necessary information to the stack and the linked list. So let me first analyze my Tour class.

The first of the global variables in my Tour class is the ‘int moveNum’ that will replace the -1 on the board with what is the move number for the spot. The next is ‘string decision’ that will be used as part of a looping mechanism to ask the user what to enter for intial positions for the board to store in the linked list. This will be seen later in the Tour class. The next two variables in the global scope are my arrays ‘moveX’ and ‘moveY’. These arrays each hold the 8 possible moves that can be made by a knight on the board. The last global scope variables Next is on to the functions that help build the tour mechanism. The first function I created in that class was a bool limit function that passed in two integers into its parameters. This function return true if the coordinates (x,y) stayed within the board. It returned true if these two integers were greater than or equal to 0 and also less than my “DIMENSION” variable which is 8. The next function in that class was the bool isEmpty function that passed in two integers and an integer array. This is crucial for when moving to a spot on the board that it doesn’t move off the board an only moves to a spot that hasn’t ben moved too yet. So this function will return true if the last function (limit) to make sure that it stays on the board and then checks if that spot is less than ‘0’ making it valid to move too. The next series of functions will help build the Warnsdoff heuristic that will be used for the first 32 moves.

The purpose of the Warnsdoff algorithm is to move a unvisited spot that is in proximity to the current position with minimum number of unvisited next to that current position. The first function that’s a part of the Warnsdoff algorithm is my emptyNextTo that returns the number of squares that the knight has not yet moved yet that are in proximity to its current position. This function makes a call to the isEmpty and uses the 8 possible moves from my global arrays moveX[] and moveY[]. This is iterated 8 times to return all the possible moves from the spot. This function will be used in the ‘bool nenxtMove function. This function will move the knight from spot to spot using the Warnsdoff heuristic; its parameters take an int array and two int pointers. It starts with trying all 8 adjacent moves from a random spot next to the current position. Then it finds the that spot within proximity with minimum degree or minimum number of unvisited adjacent. If the next spot was not found(equal -1 still) then it returned false. Otherwise store the pointer variables added by the move arrays with the minDegIdx value inside those move arrays. Then have that position hold the moveNum then increment it. Then have the int board array hold the position and push the board to the stack and returns true. The next function ‘warnsdoffFunct’ with two ints and one int array in parameters. This function iterates the bool ‘nextMove’ Function 32 times by putting it inside a for loop. If nextMove = 0, or can’t find next make, return false. The next function will be the main function to run the whole tour. This function is called bool ‘tourFunct’ that takes no parameters. First I declare an array call knight that is set to size of 64. Next I will use two for statements to set all positions in the array to -1. In the program I must ask the user to enter intial positions to be stored in a linked list node then to find tour for those intial positions. So, I create two variables intialX and intialY and they both will hold the value of zero. I use a looping mechanism to ask the questions on while ‘decicison’ = “Y”, then ask the user what intialX and intialY are for intial positions and store them in variables x and y to be used to call the warnsdoffFunct to get the first 32 moves. Before I do that though I store these values inputed by the user in the linked list then I traverse through the node to display the data in all the nodes.The next procedure to run the last 32 moves was the exhaustive procedure that uses backtrack if no available moves are possible. I use a looping mechanism for the back track so while moveNum is from 32 to 63 then I do if else statements that check the available moves. If the move is available then x and y those values that were check in the isEmpty function in the if statement. Then have you global board array hold that position and push that board to the stack. After that I have position hold the moveNum so the order displays on the board and then increment moveNum by 1. When it reaches the else, its suppose to pop that last board state from the stack and use it to find another way to finish the tour but my program always seems to leave a few -1s or sometimes just one without covering it. Then moveNum decrements by -1. After that while I print the board using two for statements while the loop variables are less than ‘DIMENSION’(8). Lastly, I will analyze the data structures used in this project.

First, I will start with the stack. The first function that stack contains that allows me to store information in the stack is the bool ‘push’ that allows an array to be pushed to it. I set the size to 64 so I can hold that many elements that can allow for backtracking when necessary. If the element is greater or equal to the value of 64 then it will return false and output an error message; else it returns true and adds the element to the stack array; The next function that the stack contains is int ‘pop’. This function returns the element at the top of the stack; unless stack element is less than 0, then print out that the stack has negative numbers of elements in it and return 0.

My other data structure implemented was the linked list node structure. I use the struct keyword to construct the list. The variables in the list are int ‘data’ and node ‘\*next’. The private nodes are pointers head and tail. My first function creates a void addNode method that takes two int variables in its parameters. It creates a temporary node that points the data being entered. If the head of the list is empty then the holds that data else the tail holds the temp pointing to a new node. The last function is the void ‘displayList()’. This function traverses through the list until the temporary pointner points to a NULL value then it will print out the data.

In this project I have learned how to manage data and also manipulate it using arrays and using linked lists.