Gavin Franklin

2/16/17

Programming Assignment 1

The Knights Tour

Project description:

The basis of this project revolves around the knight chest piece and it’s moving capabilities. A knight, unlike any other chest piece moves in a unique L shape, making it one of the most fascinating and strategic pieces in the game of chess. The implementation of the knights tour is the recreation of the scenario in which the knight piece starts at one position on the chess board and traverses to every square without revisiting any. This is a simple problem to solve for almost any of the other chess pieces, but given the knights strange moving pattern it can require some complex computation.

To complete this project I used two classes, a node class, and a stack class. The node class is used to store user inputs at the beginning of the program and then later used to traverse those inputs to give the proper amount of outputs. The stack class is used to store the position of a move. These positions are then passed to a few functions to determine where the knight should move next on the chest board. These two classes along with the main class are the backbone of my project.

Other data structure in the main class include an, a two dimensional array and of course many variables to help sort through the problem. The single dimension array is used to store instances of stacks. Depending on what position we are in the array depends on what stack values we are dealing with. If we want to backtrack we just go to the previous positon. If we want to move forward then we go to the next positon and deal with that stacks information. The two dimensional array is a 8x8 array that holds integers. These integers will either hold the value of -1 to represent that the knight has yet to move to that positon or a number representing the order in which the knight moved to each position. This is useful in both determining if a move is viable and printing out the final product.

One important thing to mention is my implementation of the “push” and “pop” functionalities of stacks. This is not the traditional method and the stack class itself does not have a push or pop function. Instead a one dimensional array holds 64 instances of a stack class. A variable then keeps track of where in the array the program currently is. If the program backtracks it simply subtracts one from that variable. If the program “pushes” it adds one to the variable then changes that stacks I and J values to the location of the move.

It is possible to do this in a sort of reverse manner. If the program only had one stack instance it could have two arrays in the stack and have a variable that keeps track of where in the arrays you are. Pushing and popping will still be incrementing or de-incrementing that variable. I decided to go with the multi instance method however, because it is less difficult to change and pass values and memory is not very concerning.

At the start of the program the user is prompted to input a row value then a column value. The user is then asked if they would like to add another or not. They then get opportunities to edit their “list” of inputs by inserting, adding or deleting which are all functions of the node class. The start function stores all the user inputs in nodes and makes proper changes to the linked list of nodes when the user decides to edit the list. After the user is satisfied with the list the program enters a while loop until all the starting positons are solved. In this loop it first resets the value of the counter that keeps track of the position in the one dimensional array then it resets the 2d array values to -1 to clear the board in a sense. The current nodes values are then loaded into the first stacks I and j values. Once everything is set up a second while loop is started that calls the heuristic algorithm 31 times or half the board minus 1 for the start. The heuristic algorithm only moves one time. It takes a positions coordinates on the board then checks every possible move to make sure 1. That move is not off the board and 2. That the move has not already been visited. This is all done through a couple functions that check for these conditions. The heuristic then checks all the moves of each of those moves using the same function. The move with the least amount of moves is the one the heuristic decides to move to.

After 32 moves the main function ends the while loop and calls the Stacktry() function which implements backtracking. It is important to note that the heuristic would be sufficient to finish the problem. The stacktry function (or the backtracking function) takes in the same coordinate parameters as the heuristic. Once the backtracking function starts it analyses one move at a time and keeps track of what moves were visited by storing it in the current stack class. If the program finds a valid move, (tested through the same functions in the Heuristic algorithm) it moves there without any algorithm or decision making.

If at any point in time the algorithm fails to find a move it simply goes back to the previous move that it made and continues the same procedure keeping track of the moves that each stack class has already visited. When you move the two dimensional array gets updated by inserting the current move number into the appropriate coordinates. The checked variable in the stack class keeps track of where each position has already checked and gets reset when you move to a new position but stays the same when you go back to a previous positon.

One important thing to note is that the backtracking implementation is much slower than the heuristic implementation. Since the heuristic makes the right move every time the program does not have to keep track of any other information and no backtracking is needed. This is why a heuristic method is desired in almost every situation and backtracking solutions take an exponentially larger amount of time when it comes to larger grids(larger than 8x8) especially when no spots in the gird have already been moved to by a heuristic algorithm.

The backtracking function runs on a while loop so one is not needed in main. Once it has made all the moves the exit condition is met and the program returns to main. The next thing to be run is the print function which uses nested for loops to print out the two dimensional list. This printing accurately shows the order in which the knight traveled starting with zero and ending with sixty three. Once all the values are printed the main function checks to see if all the nodes have been visited. If not then the entire process is started again on the next node.

Finally, once all nodes have been deleted the main function calls the clear function whose job is to clear any instances that were created on the heap. To do this it first deletes the entire stack stored in the one dimensional array. Then it goes through a for loop and reaches every node. A temporary node is used to properly delete all the nodes that way none of them go out of scope.

Comments and Conclusions:

Finding the complexity of this program is a little tricky because it needs to be broken down into certain parts. First we should talk about the nested for loops used to reset or initialize the values of the two dimensional array. Nested for loops have a complexity of n^2. However, since we know what n is we can compute the actual run time for these for loops. N = 8 since the dimensions of the board are 8x8. The for loops complexity is a constant 64\*number of nodes \* 2(for print function as well). This can be ignored because it is a constant or N where n is number of nodes(most likely small).

The heuristic algorithm has a complexity of theta of N. Since the heuristic algorithm makes one move every time and does not implement backtracking because the move it makes is always right we will make only N moves. This can be boiled down to a constant for our program as well were n = 31 (because we use the heuristic for exactly 31 moves), and the complexity is the constant 31\*b where b is the total number of nodes.

The big question in this program is what the complexity of backtracking algorithm is. As previously stated the complexity of backtracking algorithm is much larger than that of the. If we imagine the thirty two moves that are left as an array the best case is the knight lands on the correct path every time making the best case scenario N. Since this implementation includes a array we can assume that the average time spent making a decision is N/2 where n is the number of spaces left leaving us with the equation N/2 + N-x/2 + N-y/2……. Where x and y could be anything. Since these are constants we can ignore them giving us an average run time of N. However, even though the average case is N the worst case for more complex. The program could backtrack many many times resulting in upwards of N! if some spaces were not already visited.

Transcript of session

Insert value for row 0

Insert value for col 0

Would you like to add another 1)yes 2) no 1

Insert value for row 2

Insert value for col 5

Would you like to add another 1)yes 2) no 1

Insert value for row 7

Insert value for col 7

Would you like to add another 1)yes 2) no 1

Insert value for row 5

Insert value for col 6

Would you like to add another 1)yes 2) no 2

Would you like to 1. Insert 2. Add 3. Delete 4. Run the program 4

Output:

(0,0)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

|0||3||6||19||50||37||16||29|

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

|7||20||1||4||17||30||43||36|

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

|2||5||18||49||38||51||28||15|

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

|21||8||59||52||31||42||35||44|

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

|58||63||56||41||48||39||14||27|

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

|9||22||53||60||55||32||45||34|

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

|62||57||24||11||40||47||26||13|

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

|23||10||61||54||25||12||33||46|

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(2,5)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

|45||50||29||14||43||18||1||16|

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

|30||13||44||49||28||15||36||19|

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

|51||46||31||42||35||0||17||2|

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

|12||63||58||53||48||27||20||37|

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

|57||52||47||32||41||34||3||26|

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

|8||11||62||59||54||23||38||21|

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

|61||56||9||6||33||40||25||4|

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

|10||7||60||55||24||5||22||39|

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(7,7)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

|6||9||22||27||48||11||50||29|

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

|23||26||7||10||21||28||47||12|

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

|8||5||24||33||40||49||30||51|

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

|25||60||39||20||31||44||13||46|

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

|4||19||32||59||34||41||52||43|

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

|63||58||61||38||55||36||45||14|

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

|18||3||56||35||16||1||42||53|

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

|57||62||17||2||37||54||15||0|

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(5,6)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

|7||10||23||28||47||12||49||30|

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

|24||27||8||11||22||29||46||13|

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

|9||6||25||34||43||48||31||50|

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

|26||63||42||21||32||55||14||45|

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

|5||20||33||56||35||44||51||38|

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

|60||57||62||41||54||37||0||15|

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

|19||4||59||36||17||2||39||52|

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

|58||61||18||3||40||53||16||1|

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Source Code:

//These include statements are for my headers

#include "Node.h"

#include "Stack.h"

//The following allow me to include output files as well as perform input output.

#include <cstdio>

#include <iostream>

#include <fstream>

#include <sstream>

using namespace std;

//Global variables

stringstream ss;

bool keepgoing = true;

//This variable will keep track of where I am at in my array of stacks.

int counter = 0;

//This array will hold instances of my stack class

Stack\* des = new Stack[64];

//This 2d array will hold the values of the positions I visit and keep track of

//What has not been visited

int posholder [8][8];

//These three global variables are pointers to Nodes that will allow me to move

//Through the linked list.

Node\* headnode = new Node();

Node\* currentnode = headnode;

Node\* tempernode = headnode;

//This keeps track of the total ammount of nodes.

int totalnodes = 1;

//Allows me to output

ofstream Outfile;

//Debugging purposes

int tester;

int stacktester;

//Start function is the main UI of the program.

void Start()

{

int position;

//I had to run this for loop because my initilization of the stacks in my array

//Did not set the checked values.

for(int a = 0; a < 64; a++)

{

des[a].setchecked(0);

}

//These function variables will be useful for keeping track of node values.

bool keepgoing2 = true;

int ival;

int jval;

int answer1;

//This is for initial adding phase

while(keepgoing2 == true)

{

cout << "Enter starting row";

cin >> ival;

cout << "Enter starting col";

cin >> jval;

//Sets i and j values in the node to the users input.

currentnode->seti(ival);

currentnode->setj(jval);

//Loops to keep adding

cout << "Would you like to add another? 1.yes or 2.no.";

cin >> answer1;

if(answer1 == 2)

{

keepgoing2 = false;

}

else

{

//Incriments total nodes and makes a new node.

totalnodes++;

currentnode->setnextnode(new Node());

currentnode = currentnode->getnextnode();

}

}

//Have to reset node

currentnode = headnode;

bool keepgoinga = true;

//This loop if for changes in the user input.

while(keepgoinga == true)

{

currentnode = headnode;

//Outputs all the existing positions

for(int b = 0; b < totalnodes; b++)

{

if(b == totalnodes -1)

{

cout << "(";

cout << currentnode->geti();

cout << ",";

cout << currentnode->getj();

cout << ")";

cout << " ";

}

else

{

cout << "(";

cout << currentnode->geti();

cout << ",";

cout << currentnode->getj();

cout << ")";

cout << " ";

currentnode = currentnode->getnextnode();

}

}

currentnode = headnode;

cout << "Would you like to 1.) insert 2.) add 3.) delete or 4.) start program ";

cin >> answer1;

//For inserting

if(answer1 == 1)

{

cout << "Where would you like to insert";

cin >> position;

if (position <= totalnodes)

{

//Goes to proper node

for(int c = 1; c < position; c++)

{

currentnode = currentnode->getnextnode();

}

totalnodes++;

cout << "new row:";

cin >> ival;

cout << "new col:";

cin >> jval;

currentnode->insertnew(ival,jval);//Calls insert function of Node.

}

}

//For Adding

if(answer1 == 2)

{

//Goes to last node

for(int d = 1; d < totalnodes; d++)

{

currentnode = currentnode->getnextnode();

}

totalnodes++;

cout << "row:";

cin >> ival;

cout << "col";

cin >> jval;

//Just makes a new node at end of list with user input values.

currentnode->setnextnode(new Node(ival, jval));

}

//For deletion

if(answer1 == 3)

{

cout << "Where would you like to delete";

cin >> position;

totalnodes = totalnodes -1;

//Goes to proper node.

for(int e = 1; e < position-1; e++)

{

currentnode = currentnode->getnextnode();

}

//Calls the destroy function of the node.

currentnode->destroy();

}

//This answer will start the program.

if(answer1 == 4)

{

keepgoinga = false;

}

}

}

//This Function Checks to see if a space is movable by returning either a 1 or 0.

//Only checks if the range is off the board. Not if value is already taken.

int movable(int movi,int movj)

{

int rval = 0;

if(movi >= 8)

{

rval = 1;

}

else if(movj >= 8)

{

rval = 1;

}

else if(movi < 0)

{

rval = 1;

}

else if(movj < 0)

{

rval = 1;

}

return rval;

}

int movable2(int mov2i, int mov2j)

{

int rval2;

if(posholder[mov2i][mov2j] != -1)

{

rval2 = 1;

}

return rval2;

}

//This is my backtracking method.

void Stacktry()

{

bool keepgoing4 = true;

//This will make all the moves unlike the huristic algorithm.

while(keepgoing4 == true)

{

//These two values help reduce code clutter

int tempi = des[counter].geti();

int tempj = des[counter].getj();

//For testing positions

int testing = 0;

//Each Stack has a checked variable that hold the "positions" that have been checked.

if(des[counter].getchecked() == 0)

{

//This incraments the checked variable to say don't check this value again.

des[counter].setchecked(1);

//Tests if position can be moved to.

testing = movable(tempi + 2, tempj + 1);

if(testing == 0)

//if(((tempj +1 >= 0 && tempj + 1 < 8 )&& (tempi + 2 >= 0 && tempi + 2 < 8)) && (posholder[tempi+2][tempj+1] == -1));

{

//Checks to see if position was already landed on.

if(posholder[tempi +2][tempj +1] == -1)

{

counter++;

des[counter].seti(tempi + 2);

des[counter].setj(tempj + 1);

des[counter].setchecked(0);

posholder[tempi + 2][tempj + 1] = counter;

}

}

}

//This process is repeated for every position that they could move.

else if(des[counter].getchecked() == 1)

{

des[counter].setchecked(2);

testing = movable(tempi +2, tempj-1);

if(testing == 0)

//if(tempj -1 >= 0 && tempj - 1 < 8 && tempi + 2 >= 0 && tempi + 2 < 8 && posholder[tempi+2][tempj-1] == -1);

{

if(posholder[tempi +2][tempj -1]==-1)

{

counter++;

des[counter].seti(tempi + 2);

des[counter].setj(tempj - 1);

des[counter].setchecked(0);

posholder[tempi + 2][tempj - 1] = counter;

}

}

}

else if(des[counter].getchecked() == 2)

{

des[counter].setchecked(3);

testing = movable(tempi -2, tempj -1);

//if(tempj-1 >= 0 && tempj-1 < 8 && tempi-2 >= 0 && tempi-2 < 8 && posholder[tempi-2][tempj-1] == -1);

if(testing == 0)

{

if(posholder[tempi-2][tempj-1]== -1)

{

counter++;

des[counter].seti(tempi - 2);

des[counter].setj(tempj - 1);

des[counter].setchecked(0);

posholder[tempi - 2][tempj - 1] = counter;

}

}

}

else if(des[counter].getchecked() == 3)

{

des[counter].setchecked(4);

testing = movable(tempi -2, tempj+1);

if(testing == 0)

//if(tempj+1 >= 0 && tempj+1 < 8 && tempi-2 >= 0 && tempi-2 < 8 && posholder[tempi-2][tempj+1] == -1);

{

if(posholder[tempi -2][tempj +1]== -1)

{

counter++;

des[counter].seti(tempi - 2);

des[counter].setj(tempj + 1);

des[counter].setchecked(0);

posholder[tempi - 2][tempj + 1] = counter;

}

}

}

else if(des[counter].getchecked() == 4)

{

des[counter].setchecked(5);

testing = movable(tempi-1, tempj+2);

if(testing == 0)

//if(tempj+2 >= 0 && tempj+2 < 8 && tempi-1 >= 0 && tempi-1 < 8 && posholder[tempi-1][tempj+2] == -1);

{

if(posholder[tempi-1][tempj+2]==-1)

{

counter++;

des[counter].seti(tempi - 1);

des[counter].setj(tempj + 2);

des[counter].setchecked(0);

posholder[tempi - 1][tempj + 2] = counter;

}

}

}

else if(des[counter].getchecked() == 5)

{

des[counter].setchecked(6);

testing = movable(tempi+1, tempj+2);

if(testing == -1)

//if(tempj+2 >= 0 && tempj+2 < 8 && tempi+1 >= 0 && tempi+1 < 8 && posholder[tempi+1][tempj+2] == -1);

{

if(posholder[tempi+1][tempj +2]==-1)

{

counter++;

des[counter].seti(tempi + 1);

des[counter].setj(tempj + 2);

des[counter].setchecked(0);

posholder[tempi + 1][tempj + 2] = counter;

}

}

}

else if(des[counter].getchecked() == 6)

{

testing = movable(tempi+1, tempj -2);

des[counter].setchecked(7);

if(testing == 0)

//if(tempj-2 >= 0 && tempj-2 < 8 && tempi+1 >= 0 && tempi+1 < 8 && posholder[tempi+1][tempj-2] == -1);

{

if(posholder[tempi+1][tempj-2]==-1)

{

counter++;

des[counter].seti(tempi + 1);

des[counter].setj(tempj - 2);

des[counter].setchecked(0);

posholder[tempi + 1][tempj - 2] = counter;

}

}

}

else if(des[counter].getchecked() == 7)

{

des[counter].setchecked(8);

testing = movable(tempi-1,tempj-2);

//if(tempj-2 >= 0 && tempj-2 < 8 && tempi-1 >= 0 && tempi-1 < 8 && posholder[tempi-1][tempj-2] == -1);

{

if(posholder[tempi-1][tempj-2]==-1)

{

counter++;

des[counter].seti(tempi - 1);

des[counter].setj(tempj - 2);

des[counter].setchecked(0);

posholder[tempi - 1][tempj - 2] = counter;

}

}

}

//Finally if a position can't move the array call is pushed back so we're dealing

//With the previous stack or "popping".

else if(des[counter].getchecked()== 8)

{

posholder[tempi][tempj] = -1;

counter = counter -1;

}

//This will test if the counter reaches 63 so the program knows it's done

if(counter == 63)

{

keepgoing4 = false;

}

}

}

//Print function is called for every node and the outfile is open and closed in main.

void Print()

{

//Outfile.open("output.txt");

string stringprint;

//Outputs the current node values

Outfile << "(";

Outfile << currentnode->geti();

Outfile << ",";

Outfile << currentnode->getj();

Outfile << ")";

Outfile << endl;

//nested for loop to get through all values in 2d array

for(int f = 0; f < 8; f++)

{

Outfile << endl;

Outfile << "\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_";

Outfile << endl;

for(int g = 0; g < 8; g++)

{

//I used string stream although I don't think it's required.

Outfile << "|";

ss << posholder[f][g];

ss >> stringprint;

// cout <<"outfiel correct";

// cin >> tester;

// Outfile << "1";

// cout << "outfi

// cin >> tester;

Outfile << stringprint;

ss.clear();

ss.str("");

Outfile << "|";

}

}

Outfile << endl;

Outfile << "\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_";

Outfile << endl;

Outfile << endl;

Outfile << endl;

//Outfile.close();

}

//This function clears the classes off the heap.

void Clear()

{

currentnode = headnode;

//Deletes all the instances of stack

delete[]des;

//Deletes every node.

for(int i = 0; i<totalnodes; i++)

{

if(i == totalnodes -1)

{

delete currentnode;

}

else

{

tempernode = currentnode;

currentnode = currentnode->getnextnode();

delete tempernode;

}

}

}

//This function tests all possible moves form a given position and return how many it can move to

//This is useful in the hurristic algorithm.

int totalmoves(int tmovesi, int tmovesj)

{

int maxmoves = 0;

int totaltest;

//Checks if movable

totaltest = movable(tmovesi+2, tmovesj+1);

//if((tmovesi + 2 < 8) && (tmovesi + 2 >= 0) && (tmovesj + 1 < 8) && (tmovesj + 1 >= 0) && posholder[tmovesi + 2][tmovesj +1] == -1)

if(totaltest == 0)

{

//Checks if moved to.

if(posholder[tmovesi+2][tmovesj+1]==-1)

{

//Incraments total moves.

maxmoves++;

}

}

totaltest = movable(tmovesi+2, tmovesj-1);

//if(tmovesi + 2 < 8 && tmovesi + 2 >= 0 && tmovesj - 1 < 8 && tmovesj - 1 >= 0 && posholder[tmovesi + 2][tmovesj -1] == -1)

if(totaltest == 0)

{

if(posholder[tmovesi +2][tmovesj -1]== -1)

{

maxmoves++;

}

}

totaltest = movable(tmovesi -2, tmovesj+1);

//if(tmovesi - 2 < 8 && tmovesi - 2 >= 0 && tmovesj + 1 < 8 && tmovesj + 1 >= 0 && posholder[tmovesi - 2][tmovesj +1] == -1)

if(totaltest == 0)

{

if(posholder[tmovesi -2][tmovesj+1]==-1)

{

maxmoves++;

}

}

totaltest = movable(tmovesi -2, tmovesj - 1);

if(totaltest == 0)

//if(tmovesi - 2 < 8 && tmovesi - 2 >= 0 && tmovesj - 1 < 8 && tmovesj - 1 >= 0 && posholder[tmovesi - 2][tmovesj -1] == -1)

{

if(posholder[tmovesi-2][tmovesj-1]==-1)

{

maxmoves++;

}

}

totaltest = movable(tmovesi +1, tmovesj + 2);

if(totaltest == 0)

//if(tmovesi + 1 < 8 && tmovesi + 1 >= 0 && tmovesj + 2 < 8 && tmovesj + 2 >= 0 && posholder[tmovesi + 1][tmovesj +2] == -1)

{

if(posholder[tmovesi + 1][tmovesj +2]==-1)

{

maxmoves++;

}

}

totaltest = movable(tmovesi -1, tmovesj + 2);

//if(tmovesi - 1 < 8 && tmovesi - 1 >= 0 && tmovesj + 2 < 8 && tmovesj + 2 >= 0 && posholder[tmovesi - 1][tmovesj +2] == -1)

if(totaltest == 0)

{

if(posholder[tmovesi -1][tmovesj +2]==-1)

{

maxmoves++;

}

}

totaltest = movable(tmovesi +1, tmovesj -2);

if(totaltest == 0)

//if(tmovesi + 1 < 8 && tmovesi + 1 >= 0 && tmovesj - 2 < 8 && tmovesj - 2 >=0 && posholder[tmovesi + 1][tmovesj -2] == -1)

{

if(posholder[tmovesi +1][tmovesi -2]==-1)

{

maxmoves++;

}

}

totaltest = movable(tmovesi -1, tmovesj -2);

if(totaltest == 0)

//if(tmovesi - 1 < 8 && tmovesi - 1 >= 0 && tmovesj - 2 < 8 && tmovesj - 2 >= 0 && posholder[tmovesi - 1][tmovesj -2] == -1)

{

if(posholder[tmovesi-1][tmovesj-2]==-1)

{

maxmoves++;

}

}

return maxmoves;

}

//This is the huristic algorithm.

void Hur(int huri, int hurj)

{

int total;

int movementi= 0;

int movementj = 0;

int minmoves = 8;

int hurtest = 0;

//Check if move is valid

hurtest = movable(huri +2, hurj +1);

if(hurtest == 0)

//if(huri + 2 < 8 && huri + 2 >= 0 && hurj + 1 < 8 && hurj + 1 >= 0)

{

//Checks if it has been taken

if(posholder[huri + 2][hurj +1] == -1)

{

//Finds the total moves it has from there.

total = totalmoves(huri + 2, hurj + 1);

//If it the new lowest set it to that

if(total <= minmoves)

{

minmoves = total;

movementi = huri + 2;

movementj = hurj + 1;

}

}

}

hurtest = movable(huri +2, hurj-1);

if(hurtest == 0)

//if(huri + 2 < 8 && huri + 2 >= 0 && hurj - 1 < 8 && hurj - 1 >= 0 && posholder[huri + 2][hurj -1] == -1)

{

if(posholder[huri+2][hurj-1]==-1)

{

total = totalmoves(huri + 2, hurj - 1);

if(total <= minmoves)

{

minmoves = total;

movementi = huri + 2;

movementj = hurj - 1;

}

}

}

hurtest = movable(huri-2, hurj +1);

if(hurtest == 0)

//if(huri - 2 < 8 && huri - 2 >= 0 && hurj + 1 < 8 && hurj + 1 >= 0&& posholder[huri - 2][hurj +1] == -1)

{

if(posholder[huri -2][hurj+1]==-1)

{

total = totalmoves(huri - 2, hurj + 1);

if(total <= minmoves)

{

minmoves = total;

movementi = huri - 2;

movementj = hurj + 1;

}

}

}

hurtest = movable(huri-2,hurj-1);

if(hurtest == 0)

//if(huri - 2 < 8 && huri - 2 >= 0 && hurj - 1 < 8 && hurj - 1 >= 0 && posholder[huri - 2][hurj -1] == -1)

{

if(posholder[huri-2][hurj-1]==-1)

{

total = totalmoves(huri - 2, hurj - 1);

if(total <= minmoves)

{

minmoves = total;

movementi = huri - 2;

movementj = hurj - 1;

}

}

}

hurtest = movable(huri+1,hurj+2);

if(hurtest == 0)

//if(huri + 1 < 8 && huri + 1 >= 0 && hurj + 2 < 8 && hurj + 2 >= 0&& posholder[huri + 1][hurj +2] == -1)

{

if(posholder[huri +1][hurj +2]==-1)

{

total = totalmoves(huri + 1, hurj + 2);

if(total <= minmoves)

{

minmoves = total;

movementi = huri + 1;

movementj = hurj + 2;

}

}

}

hurtest = movable(huri-1, hurj + 2);

if(hurtest == 0)

//if(huri - 1 < 8 && huri - 1 >= 0 && hurj + 2 < 8 && hurj + 2 >= 0 && posholder[huri - 1][hurj +2] == -1)

{

if(posholder[huri-1][hurj +2] == -1)

{

total = totalmoves(huri - 1, hurj + 2);

if(total <= minmoves)

{

minmoves = total;

movementi = huri - 1;

movementj = hurj + 2;

}

}

}

hurtest = movable(huri +1, hurj -2);

if(hurtest == 0)

//if(huri + 1 < 8 && huri + 1 >= 0 && hurj - 2 < 8 && hurj - 2 >= 0 && posholder[huri + 1][hurj -2] == -1)

{

if(posholder[huri +1][hurj -2]==-1)

{

total = totalmoves(huri + 1, hurj - 2);

if(total <= minmoves)

{

minmoves = total;

movementi = huri + 1;

movementj = hurj - 2;

}

}

}

hurtest = movable(huri-1, hurj-2);

if(hurtest == 0)

//if(huri - 1 < 8 && huri - 1 >= 0 && hurj - 2 < 8 && hurj - 2 >= 0 && posholder[huri - 1][hurj -2] == -1)

{

if(posholder[huri-1][hurj-2]==-1)

{

total = totalmoves(huri - 1, hurj - 2);

if(total <= minmoves)

{

minmoves = total;

movementi = huri - 1;

movementj = hurj - 2;

}

}

}

//Make the lowest move.

counter = counter + 1;

des[counter].seti(movementi);

des[counter].setj(movementj);

posholder[movementi][movementj] = counter;

}

int main()

{

//This is to help see if we're done.

int maincounter = 1;

//opens file

Outfile.open("output.text");

Start();

//Sets current to head

currentnode = headnode;

//This loop goes for as many nodes as there are.

while(keepgoing == true)

{

counter = 0;

//resets the values of 2d array to -1

for(int w = 0; w < 8; w++)

{

for(int x = 0; x < 8; x++)

{

posholder[w][x]= -1;

}

}

//Sets the initial value

des[counter].seti(currentnode->geti());

des[counter].setj(currentnode->getj());

posholder[des[counter].geti()][des[counter].getj()] = counter;

//Runs the huristic 31 times since initial value was just set

while(counter < 33)

{

Hur(des[counter].geti(), des[counter].getj());

}

//Runs the backtracking algorithm.

Stacktry();

Print();

//Ends the entire thing

if(maincounter == totalnodes)

{

keepgoing = false;

}

else

{

maincounter++;

currentnode = currentnode->getnextnode();

}

}

//Deletes instances

Clear();

//Closes outifle

Outfile.close();

}

#ifndef STACK\_H

#define STACK\_H

class Stack

{

public:

Stack();

int getchecked();

int geti();

int getj();

void setchecked(int schecked);

void seti(int si);

void setj(int sj);

private:

int i;

int j;

int checked;

};

#endif // STACK\_H

#ifndef NODE\_H\_INCLUDED

#define NODE\_H\_INCLUDED

class Node

{

public:

Node();

Node(int ipos, int jpos);

Node\* getnextnode();

Node\* gettempnode();

int geti();

int getj();

void seti(int itemp);

void setj(int jtemp);

void setnextnode(Node\* tempnextnode);

void settempnode(Node\* temptempnode);

void destroy();

void add();

void insertnew(int inew, int jnew);

private:

Node\* next;

Node\* temp;

int i;

int j;

};

#endif // NODE\_H\_INCLUDED

#include "Node.h"

using namespace std;

Node::Node(){

}

Node::Node(int itemp, int jtemp)

{

i = itemp;

j = jtemp;

}

void Node::destroy()

{

settempnode(next);

setnextnode(next->getnextnode());

delete temp;

}

void Node::insertnew(int inew, int jnew)

{

temp = next;

setnextnode(new Node(inew, jnew));

next->setnextnode(temp);

}

int Node::geti()

{

return i;

}

int Node::getj()

{

return j;

}

void Node::seti(int itemp)

{

i = itemp;

}

void Node::setj(int jtemp)

{

j = jtemp;

}

Node\* Node::getnextnode()

{

return next;

}

Node\* Node::gettempnode()

{

return temp;

}

void Node::setnextnode(Node\* tempnextnode)

{

next = tempnextnode;

}

void Node::settempnode(Node\* temptempnode)

{

temp = temptempnode;

}

#include "Stack.h"

Stack::Stack()

{

//ctor

checked = 0;

}

int Stack::geti()

{

return i;

}

void Stack::seti(int si)

{

i = si;

}

int Stack::getj()

{

return j;

}

void Stack::setj(int sj)

{

j = sj;

}

int Stack::getchecked()

{

return checked;

}

void Stack::setchecked(int schecked)

{

checked = schecked;

}