SLIDER PUZZLE

Informed Search Algorithm

DESCRIPTION OF THE PUZZLE:

In this assignment you will implement algorithm for solving the slider puzzle that has tiles marked 1 to 20 placed in random order on a sliding board. The tiles can be moved around the circular track but cannot be removed from the track. There is a knob attached to the track that when used can reverse the order four of the chosen numbers being rotated using the knob. We will display the board from the position of the knob.

For example the track in the picture below has

17 18 19 15 16 20 1 2 3 5 7 10 11 8 9 6 4 12 13 14

then rotating the knob at 1st position would result in

15 19 18 17 16 20 1 2 3 5 7 10 11 8 9 6 4 12 13 14

For example the track in the picture below has

17 18 19 15 16 20 1 2 3 5 7 10 11 8 9 6 4 **12 13 14**

then rotating the knob at 18th position would result in

14 16 20 1 2 3 5 7 10 11 8 9 6 4 12 13 14 **17 12 13**

The objective of the puzzle to rotate the knob the numbers come in ascending order. The challenge is to solve the puzzle with least number of rotations. Rotation will be counted as 1 points and shifting the knob to the desired position costs nothings.

The quality of the algorithm that you write will be measured by several factors.

- NUM_EXPANDED, The number of nodes expanded –(expanded nodes have their children generated. This indicates the time complexity.)
- FRINGE_MAX, The maximum number of nodes kept in memory (this would be the memory complexity) (this would be maximum size of the fringe list)
- PATH COST (The cost of the path to the goal state.)

You will write three algorithms
1. Uniform cost search (h=0)
2. A* algorithm with a heuristic, namely, number of tiles out of order. With this heuristic both teams must get the same results for NUM_EXPANDED, FRINGE_MAX and PATH_COST
2. Your own heuristic to solve the problem above. This heuristic is entirely up to you.
Your grade will depend on the performance of these three aspects between the teams.

A. *Optimality*: The algorithm that has the minimal PATH_COST is the winner.
B. *Time Complexity*: If both teams have same path cost, then the team with less number for NUM_EXPANDED will be the winner.
C. *Space Complexity*: If PATH_COST and NUM_EXPANDED are same, then the team with smallest FRINGE_MAX is the winner.

Although 20 tiles is the standard game, we will be implementing a flexible board with 5 up to 20 tiles.

The input to the program will be two command line options The size of the board and the initial board. If the word "random" is used, you should start with random board. If another name is given, it is the file name containing the initial state of the board as a list of tiles. The first tile is at the left of the rotating knob.

%java Main 6 random

%java Main 18 input.txt

Then input.txt will contain
18 17 19 15 20 16 1 2 3 5 7 10 11 8 9 6 4 12 13 14

The output from your algorithm would be following:

Initial State of board:
Move made
New State of the board
Move made
New State of the board.
etc until goal state is reached.

You will output the following summary information

- NUM_EXPANDED
- FRINGE_MAX
- PATH COST

Debug Mode:

There will be two levels of debug mode. First level is -d1 and the second level is -d2. If the program is run in the debug mode,

%java Main -d1 6 input.txt
%java Main -d2 6 input1.txt

In the debug level 1, you will print the list of expanded nodes with cost associated with them. Then you will also print the information printed in the non-debug mode.

Expanded node 1 – Heuristic value
Expanded node 2 – Heuristic value
Expanded node 3 – Heuristic value
etc

Followed by:

Initial State of board:
Move made
New State of the board
Move made
New State of the board.
etc until goal state is reached.

- NUM_EXPANDED
- FRINGE_MAX
- PATH COST

In the second level of debug mode -d2, you will print the fringe list with heuristic of each node.

Expanded Node1 -Heuristic value

********** Begin Fringe List **********
Fringe node 1 with state and heuristic
Fringe node 2 with state and heuristic
Fringe node 3 with state and heuristic
Fringe node 4 with state and heuristic
********** End Fringe List **********

Expanded Node 2-Heuristic value
********** Begin Fringe List **********
Fringe node 1 with state and heuristic
Fringe node 2 with state and heuristic
Fringe node 3 with state and heuristic
Fringe node 4 with state and heuristic
Fringe node 5 with state and heuristic
Fringe node 6 with state and heuristic

********** End Fringe List **********

Expanded Node 3-Heuristic value
********** Begin Fringe List **********
Fringe node 1 with state and heuristic
Fringe node 2 with state and heuristic
Fringe node 3 with state and heuristic
Fringe node 4 with state and heuristic
Fringe node 5 with state and heuristic
Fringe node 6 with state and heuristic
Fringe node 7 with state and heuristic
Fringe node 8 with state and heuristic
********** End Fringe List **********

etc
Followed by:

Initial State of board:
Move made
New State of the board
Move made
New State of the board.
Etc until goal state is reached.

- NUM_EXPANDED
- FRINGE_MAX
- DEPTH

Testing the Code:

You must test your code thoroughly with small size boards with several initial configurations to ensure correctness. Then you can run the code on larger size boards.

Creating and submitting demo.txt

Submit a demo.txt file with output of the program as follows.

5 runs with board of size 6. One run should be board with goal state as initial state. Second with two numbers in opposite order (switch 1 and 2). The other three should be randomly selected boards.

5 runs with a board of size 20.
one board with goal state as initial state.
One board as follows
2 1 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 4 3
(this can solved by 1 moves – left shift 2 times and rotate)
Three other randomly generated boards.

Creating a README FILE

Create a README file with description of the problem, algorithm and the heuristic. Also indicate any special observations such as solvable versus unsolvable boards. Any additional observations you have made.

DESIGN OF THE CODE:

You must have following classes
State – containing states and valid transitions from each state
Node – State plus other information needed to run algorithms such as depth, h(n), g(n), parent node etc.
Main- deals with reading the command line argument
AStar – implements A* algorithm
Any other class as you see fit.