Assignment 1 – SEARCHING FOR SOLUTIONS

Due date: 18/11/2022 [Friday]
[This assignment must be completed as a team of 3]
CORE ASSESSMENT
Worth: 8%

Summary

Given problem is a programming and making report. The problem is based on forward search engine that supports multiple search strategies and uses this implementation to find solutions for a solving N tile puzzle problem. You need to implement tree-based search algorithms which include BREATH FIRST, DEPTH FIRST, BEST FIRST SEARCH, A* Search, and Hill Climbing. Next step is to create a report based on the search implementation to solve the puzzle.

Implementation

You can implement the programming using either C/ Python for problem 1. You must gain permission before using anything else. Assignment work will be tested on a standard Microsoft Windows XP operating system.

Search Algorithms

PROBLEM – SOLVING N TILE PUZZLE PROBLEM

You are given a row containing 2N tiles arranged in 2N + 1 spaces. There are N Red tiles (R), N Green tiles (G), and a single empty space. The tiles are initially in an arbitrary ordering. Your goal is to arrange the tiles such that all green tiles are positioned to the left of the red ones, and one red tile is in the rightmost position. The goal position of the empty space is not specified. Tiles can be moved to the empty space when the empty space is at most N cells away. Hence there are at most 2N legal moves from each state. The cost of each move is the distance between the tile and the empty space to which it moves (1 to N). "Circular" moves (i.e. moves wrapping from one side of the row to the other) are not permitted.

To illustrate the tile domain, consider the trivial case where $N = 2$.

Initial state may look like:

G	R	–	R	G

There are 4 successors to the initial state:

G	R	–	R	G
G	R	R	–	G
G	R	G	R	–
–	R	G	R	G

There are 4 goal states:

G	G	R	-	R
G	G	-	R	R
G	-	G	R	R
-	G	G	R	R

IMPLEMENTATION PROCEDURE

- Your implementation should read in an arbitrary starting state for tile problems containing up to 21 positions (10 RED, 10 GREEN, and 1 space). Each input file contains a single line specifying the initial state using R to represent a RED tile, G to represent GREEN, and x to represent the space. For example, one possible starting state for a 7-position problem is the string:

  GRRGXGR

  You should use one operator, move x, to specify which tile to move into the current empty position. x is the position of the card (starting from left to right). Note that you only need specify which card to move since it will always move into the single empty position. In the absence of any other heuristic function that determines which tile to move, all legal moves should be considered in increasing order (i.e. from 0 to 2N + 1; move 1 occurs before move 3).

- Implement the solutions to solve the puzzle using:

  1) BREATH FIRST SEARCH TREE
  2) DEPTH FIRST SEARCH TREE.
  3) BEST FIRST SEARCH
  4) A * SEARCH
  5) HILL CLIMBING

PROGRAMMING INSTRUCTIONS

In this programming, you will have to make decisions about how to solve puzzle problems, states, nodes, etc. You need to implement the following search algorithms:

Search Type	Description	Method
Depth-first search	Select one option, try it, go back when there are no more options	DFS
Breadth-first search	Expand all options one level at a time	BFS
Best-first (F=g+h)	Take the best (weighted score), okay to go back	BEST
A* SEARCH	Priority queue sorted according to the heuristic function	ASEARCH
Hill climbing(Using H function only)	Take the best (weighted score), no going back	HC

SAMPLE OUTPUT SHOULD LIKE THIS: **(NOT CORRECT ANSWER)**

Your program should print out the **initial state** followed by each **move operation** selected and its successor state (i.e. both the operator path and the state path).

At the end, it should:

- **Print the cost or the total number of positions that the blank had to move**
- **The number of nodes expanded -**not just the ones to the solution path.

For example, given an initial state **RGxRG,** your program might print the following lines:

```
GRXRG
MOVE 5 GRGRX
MOVE 4 GRGXR
MOVE 2 GXGRR

5
20
```

NUMBER NODE EXPANDED is 20. By this you have completed in solving the puzzle.

SEARCH METHODS:

Your program must support four different search strategies: breadth-first search (BFS), depth first search (DFS), Best first Search , A* (A-star) & Hill Climbing. Each of these search strategies should be selectable at runtime by using a command-line keyword. Note that you should be implementing this using a single function, not three different functions i.e.one for each search strategy.

NOTE: use a next-state queue containing states that have been generated but not-yet visited as your primary data structure. In such a data structure, BFS implies a FIFO discipline, DFS implies a LIFO or "stack" discipline, BEST first implies the search with a heuristic that attempts to predict how close the end of a path is to a solution, so that paths which are judged to be closer to a solution are extended first., hill climbing implies the best score of the node, & A* implies a priority queue sorted according to the heuristic function. Also note that you will need a means of avoiding loops by checking whether or not a state has already been visited. A* combines the cumulative cost so far g(n) with a guess about the cost of getting to the solution from the current state h(n), using the evaluation function f(n)= g(n)+h(n).Define g(n) and h(n) such that g(successor(n)) ≥ g(n) and h(n) is an admissible heuristic (you can't use h = 0).

General Pseudo code for Searching

The following is the **basic outline** for the various search algorithms (some steps need to be modified depending on the specifics of the search being used).

```
OPEN = { startNode }      // Nodes under consideration.
CLOSED = { }         // Nodes that have been expanded.

While OPEN is not empty
{
Remove the first item from OPEN. Call this item X.

If goalState?(X) return the solution found.

// Expand node X if it isn't a goal state.
Add X to CLOSED.

Generate the immediate neighbors (i.e., children) of X.
Eliminate those children already in OPEN or CLOSED.
Based on the search strategy, insert the remaining
children into OPEN.
}
Return FAILURE // Failed if OPEN exhausted w/o a goal found.
```

GENERAL INSTRUCTIONS

The programming part consists of two directories (one for each problem) and a readme file. The directories should be named PB-1. For problem 1, name your programming file as DFS.c/py, BFS.cc/java, Best.cc/java, AStar.cc/java & Hill.cc/java. Use the below template for the readme file that is provided on Assignments. READ ME File should be a text file, not an MS-Word document, or any formatted file. It does not have any extensions. It is not a directory, or is not in any other directory.

Once you are finished developing your code, copy all necessary source files including header files, library files, into the respective directories. Write general, modular code. Document/comment your code, so it can easily read and understand it. Use meaningful variable names.

REPORT

Write a report that presents the searching solutions for Solving a tile puzzle by BFS, DFS , BFS heuristic search by A* algorithm & Hill Climbing.

- Explain the problem formulation for the search strategies in solving the puzzle
- Brief the search algorithms used.
- How your group implemented them?
- Need to highlight your individual contributions in above problems.

Content in the Report

- Brief the search algorithms used?
- Explain how the program was implemented (Include flowcharts). Briefly explain the difference in implementation.
- Conclude with a discussion about the best type of search algorithm you would use for this type of chosen problem. Include thoughts about how you could improve performance
- Add in Individual contribution in every stage of solving the problem.

Report Structure

- Include Cover Page and table of content
- Limit the report to 10 pages excluding cover page and table content
- Font style: Times New Roman, Size: 12, Single Spaced column
- Cite and acknowledge sources correctly
- Make your Report to save in Word or PDF.

Readme.txt file

You must include a single readme.txt file with your work with the following details:

- **Student Details:** Your full student names, ROLL NO and programs.
- **Features/Bugs/Missing:** Include a list of the features you have implemented. Clearly state if a required feature has not been implemented. Failure to do this will result in penalties. Include a list of any known bugs.
- **Acknowledgements/Resources:** Include in your readme.txt file a list of the resources you have used to create your work. A simple list of URLs is not enough. Include with each entry a basic description of how the person or website assisted you in your work.
- **Notes:** Anything else you want to tell the marker, such as how to plot the graphical interface of your program, and something particular about your implementation.

Marking Scheme & Submission

You must submit your Softcopy on the Google Class Room. Create a single zip file for your code and a working version of your program. Hard Copy of the report must be submitted at M239 Assignment box. You should name your source code and the zipped file as XXXXX_Ass1; where XXXXX is your team name.

Marking Scheme

REQUIREMENTS	MARKS
PROBLEM 1& 2	
BREATH FIRST	3
DEPTH FIRST	3
BEST FIRST	3
HILL CLIMBING	3
A * SEARCH	3
INPUT/OUTPUTS COMMAND LINE PARAMETERS OUTPUT TRACE FILE SOLUTION PRINTED TO SCREEN PROPERLY	3
RESULT	2
READABLITY OD CODE	1
READ ME FILE	2
REPORT	5
INDIVIDUAL CONTRIBUTION	3
DEMO	2
TOTAL	**30**