National University of Computer & Emerging Sciences

Karachi Campus

Artificial Intelligence

Programming Assignment #2

Instructions ***Each Problem of 10 points***

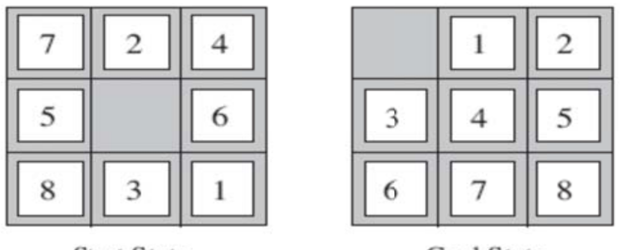
Due Date: The assignment is due by April 10, 2014 till midnight.

Submission: The assignment has to be submitted via slate website submission. You must submit the source code files with proper naming convention for example (Assignment No. 1 Problem No. 1) you should give CS401-K1xxxxx-A1P1.cpp. You should copies all questions for the assignment in a single folder (named as your id e.g. K1xxxxx) and zipped it before uploading to slate.

Sample Input and Output files: The sample input and output files are available from course website at slate. Make sure that you have tested your programs against all the available input files and EXACT output file is produced.

8-Puzzle

The 8-puzzle is a smaller version of the slightly better known 15-puzzle. The puzzle consists of an area divided into a grid, 3 by 3. On each grid square is a tile, expect for one square which remains empty. A tile that is next to the empty grid square can be moved into the empty space, leaving its previous position empty in turn. Tiles are numbered, 1 thru 8 for the 8-puzzle, so that each tile can be uniquely identified. The aim of the puzzle is to achieve a given configuration of tiles from a given (different-initial) configuration by sliding the individual tiles around the grid as described above. For example: The following diagram shows an initial and goal state of the problem.



Start State Goal State

Considering the above start state, there are four possibilities, (1) we can move a 6 to the left (2) we can move 5 to right, (3) we can move 3 up and (4) we can move 2 down. These are the four possible operators <UP, DOWN, LEFT and RIGHT>. You people already have implemented part of the as a task in Assignment No. 1, in this assignment you need to implement and test few of the uninformed and informed method of state space search on 8-puzzle. The input files are the same for all the problems.

Problem 1 8-Puzzle (Uninformed-BFS)

In this problem you need to implement a solution to 8-puzzle by using Breadth First Search (BFS), you are allowed to keep track of visited states (8-puzzle search is a graph search). The solution should print all sequence of moves that you apply to reach to a goal state.

Problem 2 8-Puzzle (Uninformed-Iterative Deepening)

In this problem you need to implement a solution to 8-puzzle by using Iterative Deepening (ID), you are allowed to keep track of visited states. The solution should print all sequence of moves that you apply to reach to a goal state.

Problem 3 8-Puzzle (Informed-Greedy Best First)

In this problem you need to implement a solution to 8-puzzle by using Greedy Best First Search (GBFS), you are allowed to keep track of visited states. The suggested Heuristic is f (h) = number of tiles that are not in the correct place (not counting the blank). The solution should print all sequence of moves that you apply to reach to a goal state.

Problem 4 8-Puzzle (Informed-A*)

In this problem you need to implement a solution to 8-puzzle by using A*, you are allowed to keep track of visited states. The heuristic for A* will be Hamming priority function, which can be defined as The number of blocks in the wrong position, plus the number of moves made so far to get to the search node. Intuitively, a search node with a small number of blocks in the wrong position is close to the goal, and we prefer a search node that has been reached using a small number of moves. The solution should print all sequence of moves that you apply to reach to a goal state.

Problem 5 8-Puzzle (Beyond Classical search-Steepest hill climbing)

In this problem you need to implement a solution to 8-puzzle by using Steepest Hill Climbing (SHC), you are allowed to keep track of visited states (8-puzzle search is a graph search). The solution should print all sequence of moves that you apply to reach to a goal state.

Input/output

The input file contains two occurrence of a 3 x 3 board separate by an empty line. Given as initial and goal state. The output file should contain a sequence of move, one move per line.

Example

A1P3in1.txt	A1P3out1.txt
0 1 3	1 left
4 2 5	2 up
7 8 6	5 left
	6 up
1 2 3	Goal state found
4 5 6	
7 8 0	

<The end>