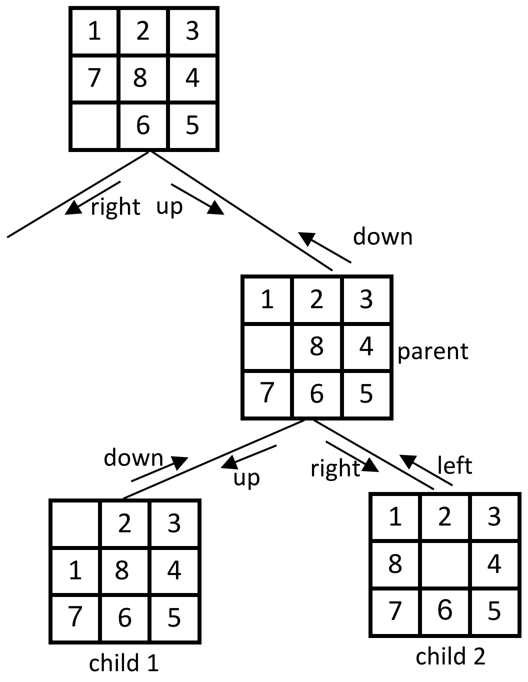
ASSIGNMENT

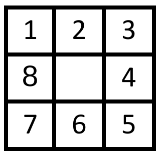
INFROMED SEARCH

PURPOSE: To Understand heuristic search, admissible and consistent heuristics, branching factors, algorithm evaluation, related data structures and algorithm implementation..

In this assignment, you develop graph search algorithms to solve the **8-puzzle problem** that has been discussed in class.

DESCRIPTION: The figure below shows one possible state in an eight puzzle. Suppose for each state the board is represented by a list of 9 elements which indicate the value for each cell of the puzzle [c1 c2 c3 c4 c5 c6 c7 c8 c9]. For example, [1 2 3 7 8 4 0 6 5], represents the board value of the state at the root node in the figure. Note that the number 0 is used to represent the position of the blank tile. Also note that the tiles are placed in the list in **row-major** order. The parent (for the best path back) for a state can also be represented by the 9 element list that shows its board value. We also associate two values with each state, *gval* and *hval*. *hval* is set equal to the heuristic value (*h*) of a state (for example the Manhatan distance from the current state to the goal), and *gval* (*g*) shall be equal to the cost of the path from the start/initial state to the given state (recall that *g* at the start state is zero, and *h* at the goal state is zero).





GOAL

Therefore, the representation for a node should at least include:
(i) A list/array of integers representing the corresponding board.
(ii) A node (or pointer to one) corresponding to the current best parent (null for the start/initial state), and the action for the transition from the parent.
(iii) Two numbers for the *h* value and *g* value.
Your `program` should also have variables (or classes) implementing "`openList`" and "`closedList`" (or equivalently `frontier` and `exploredSet`) corresponding to the two lists maintained by the heuristic search algorithm. The `program` should have a concrete implementation that includes the following:

- **(a)** A method **`expand`**`(aNode)`, which will compute the children of a node and return them as an array of nodes.
- **(b)** A method **`equalState`** that given any two states, will return `true` if states in two nodes are equal, and `false` if the two states are not equal. (States are equal if their board representations are the same/equal, not if they point to the same object.)
- **(c)** A method **`order`**`(openList) (or` **`order`**`(frontier)),` which returns an array containing the current *nodes* on the *frontier/openList* in the ranked order of their promise.
- **(d)** A method **`solve`**`(aNode)` that when called, will return an array of nodes in proper order, with the name of the actions to take at each step, to go from the start state to the final/goal state. This shall represent the solution to the given 8-puzzle problem.

The program of course needs to have additional methods.

Note about the best parent of a node in graph search with inconsistent heuristics: When a heuristic function is not consistent (i.e. not monotone), one node at any time represents the *bestParent* to a given node and this must be kept track of (except for the start/initial node whose parent is null). The *bestParent* value must be maintained and updated by using a *setParent* method. The *setParent* method sets the best parent of a node and its children to correspond to the best path to the node when new cheaper paths to the given node are discovered by the algorithm. This will makes sure that back pointers are updated and maintained correctly.

TASKS:
You need to do the following tasks:
Task 1 (40 pts). Develop a program that will solve the 8-puzzle problems by applying heuristic graph search using the following heuristics, where n is a node in the search. Use a step (transition) cost equal to 1. Recall that h(n) is not the same as f(n) in heuristic graph search (in general). For specifications on the program name, input/output, etc., see additional specifications below.
1. h1(n) = No. of displaced tiles; (do not count the blank tile)
2. h2(n) = The Manhattan/block distance; (do not count the blank tile)
3. h3(n) = No. of displaced tiles + the Manhattan distance.

Task 2 (40 pts). Run your program on at least 5 puzzles in each of the following 3 puzzle sets:
Set 1: Puzzles that need 5 actions in right sequence to solve the puzzle.
Set 2: Puzzles that need 10 actions in right sequence to solve the puzzle.
Set 3: Puzzles that need 15 actions in right sequence to solve the puzzle.

This will solve 15 (5x3) puzzles with three different heuristics. (i) Report the results as described in the additional specifications below. (ii) For each set (problem size), calculate the average branching factor for each heuristic, based on the problems solved. Create a table based on the results with average branching factors for each problem size listed across the rows for each heuristic (columns).

Task 3 (20 pts). Answer the following questions:

1. For each heuristic, state if the heuristic admissible or not? If it is, explain why, and if it is not, provide an example puzzle for which it does not produce the optimal solution.
2. Show/prove that all consistent heuristics are also admissible.

EXTRA CREDIT (10 points): Consider the following heuristic for the 8-puzzle
Maybe you just want to look at Manhattan distance (i.e. the number of blocks away) of the 1, the 2, and the 3 to the locations in which they are supposed to be in the goal state. The heuristic, while less informative than Manhattan distance of all tiles, it is still admissible and consistent. But let's say that you choose an additional group of squares, perhaps 5, 6, and 7. And then the way you calculate the heuristic value at each node is by randomly selecting one of those sets (1,2, and 3) or (5, 6, and 7) and computing their Manhattan distance to their goal locations. This heuristic is still admissible - it can only ever underestimate or match the number of moves needed to get to the goal state. But it is no longer consistent – **Show** that this heuristic is not consistent.
If you decide to do the extra credit, include it with the report file described in the "Turnin" section.

ADDITIONAL SPECIFICATIONS

Your program needs to adhere to the specifications listed here (it will make scoring more uniform).

Program Name And Input/Output: Please name your program "EightPuzzleSolver". If the program is in Java, it will be good to provide Javadoc comments for all class variables and methods in your classes. Your programs will be run with different test cases in Windows with the Eclipse environment. Please make sure that your programs compile and run correctly in this environment.

Input: The “EightPuzzleSolver” program should open and take its input from a text file (.txt) named “inputFile”. “inputFile” contains one or more lines with each line corresponding to initial state for a 8-puzzle problem to be solved. Each line has nine numbers separated by a “,” delimiter between them, for example. “1,2,3,7,8,4,0,6,5”; in which zero indicates the location of the blank tile, and the tiles are listed in the **row-major** order.

Output: Your program should print its output to three files named “OutfileHeuristic1”, “OutfileHeuristic2”, and “OutfileHeuristic3”. The output in each file should show (i) the problem followed by (ii) the number of nodes expanded to find the solution, and then (iii) show the found solution. For the found solution, the sequence of states, the actions to transition from each state to the next, and the *h* and *g* values for each state should be given. Each state and its associated information is shown on one line, and <u>the last element on the line is a number</u> that indicates <u>the iteration at which that state was first reached</u> by the algorithm. Each file includes the output for all sets of problems (3 sets of 5 problems each) solved by that heuristic clearly separated and labelled, for example:

”START OF PROBLEM SET 1”,

    “PROBLEM 1, SET 1”,
    “PROBLEM 2, SET 1”,
    …,
    “END OF PROBLEM SET 1”,

”START OF PROBLEM SET 2”,

    “PROBLEM 1, SET 2”,
    “PROBLEM 2, SET 2”,
    …

Turnin

Your turn in material should include (i) your program file and the input file(s) (ii) your report file (pdf) that includes the items in tasks 2 and 3 (branching factors, answers to questions, etc.), (iii) the three output files (text), OutfileHeuristic1, OutfileHeuristic2, OutfileHeuristic3, that include the problems and their solutions, (iv) a short video showing/explaining your program, running of your program (show your input file(s) and folder) and then opening and examining the output (files).