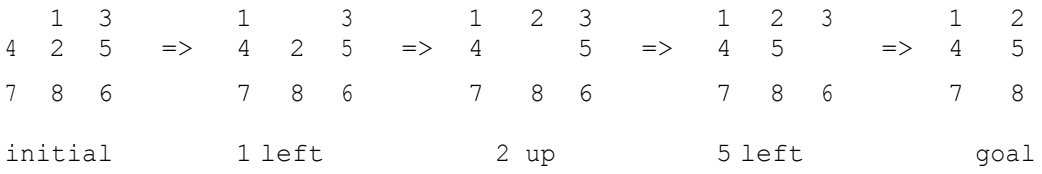
CSE 402: Offline Assignment 2

Solve *n* Puzzle

Write a program to solve the *n*-puzzle problem using the A* search algorithm.

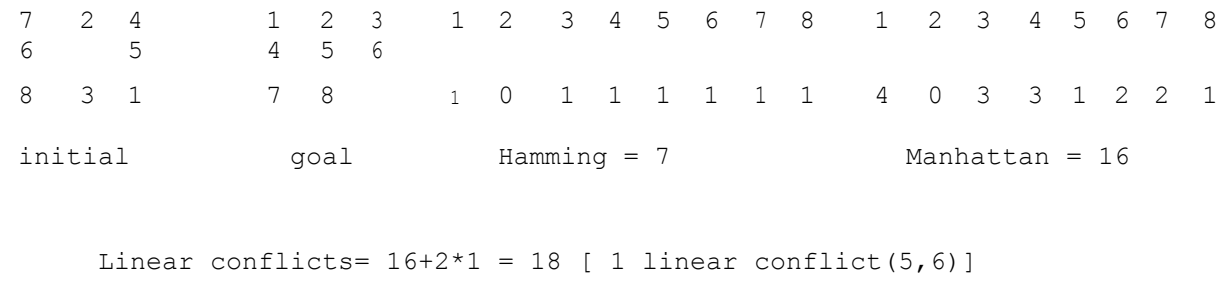
The problem. The *n*-puzzle is a puzzle invented and popularized by Noyes Palmer Chapman in the 1870s. It is played on a *k* x *k* grid with *k*^2-1 (=n) square blocks labeled 1 through n and a blank square. Your goal is to rearrange the blocks so that they are in order, using as few moves as possible. You are permitted to slide blocks horizontally or vertically into the blank square. The following shows a sequence of legal moves from an *initial board* (left) to the *goal board* (right).



Best-first search. Now, we describe a solution to the problem that illustrates a general artificial intelligence methodology known as the A* search algorithm. We define a *search node* of the game to be a board, the number of moves made to reach the board, and the previous search node. First, insert the initial search node (the initial board, 0 moves, and a null previous search node) into a priority queue (open list). Then delete from the priority queue the search node with the minimum priority, insert the node into the closed list, and insert onto the priority queue all neighboring search nodes which are not in the closed list. Repeat this procedure until the search node dequeued corresponds to a goal board. The success of this approach hinges on the choice of *priority function* for a search node. In A* search algorithm, the priority function $f(n)$ represents an estimated cost of path from initial state to the goal state though node n. $f(n)$ is calculated as $f(n)=g(n)+h(n)$, where $g(n)$ is the cost to reach node n from the initial state and $h(n)$ is an estimated cost from node n to goal state. The function $h(n)$ is often called a heuristic. Finding optimal path by A* search algorithm depends of the properties of the heuristic used. In this assignment we will consider two priority functions:

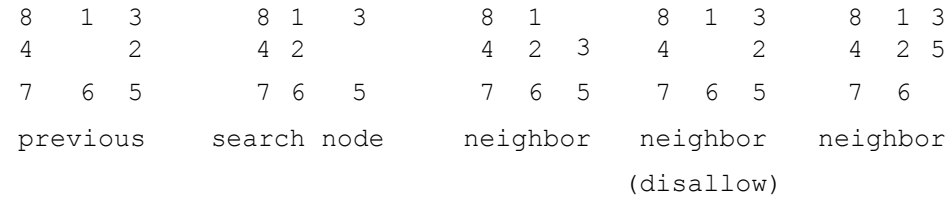
- ***Hamming distance****:* The number of blocks in the wrong position (not including the blank.
- ***Manhattan distance****:* The sum of the Manhattan distances (sum of the vertical and horizontal distance) from the blocks to their goal positions.
- ***Linear Conflict***: Two tiles t_j and t_k are in a linear conflict if t_j and t_k are the same line, the goal positions of t_j and t_k are both in that line, t_j is to the right of t_k, and goal position of t_j is to the left of the goal position of t_k. Here line indicated both rows and columns. The linear conflict heuristic is calculated as *Manhattan distance* + 2*(*Linear conflicts*).

For example, the values of Hamming, Manhattan and Linear conflict of the initial search node below are 7, 16 and 18, respectively. Remember that, to calculate the priority function of the open list, you need to add the cost to reach the current node with the value of heuristic.

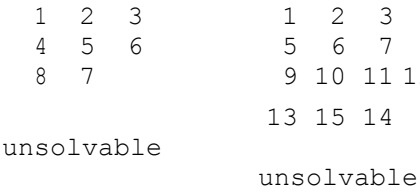


Linear conflicts= 16+2*1 = 18 [1 linear conflict(5,6)]

We make a key observation: To solve the puzzle from a given search node on the priority queue, the total number of moves we need to make (including those already made) is at least its priority, using either the Hamming or Manhattan priority function. (For Hamming priority, this is true because each block that is out of place must move at least once to reach its goal position. For Manhattan priority, this is true because each block must move its Manhattan distance from its goal position. For Linear conflict this is true because each conflicting pair causes at least two more moves. Note that we do not count the blank square when computing the Hamming or Manhattan priorities.) Consequently, when the goal board is dequeued, we have discovered not only a sequence of moves from the initial board to the goal board, but one that makes the fewest number of moves. (Challenge for the mathematically inclined: prove this fact.)

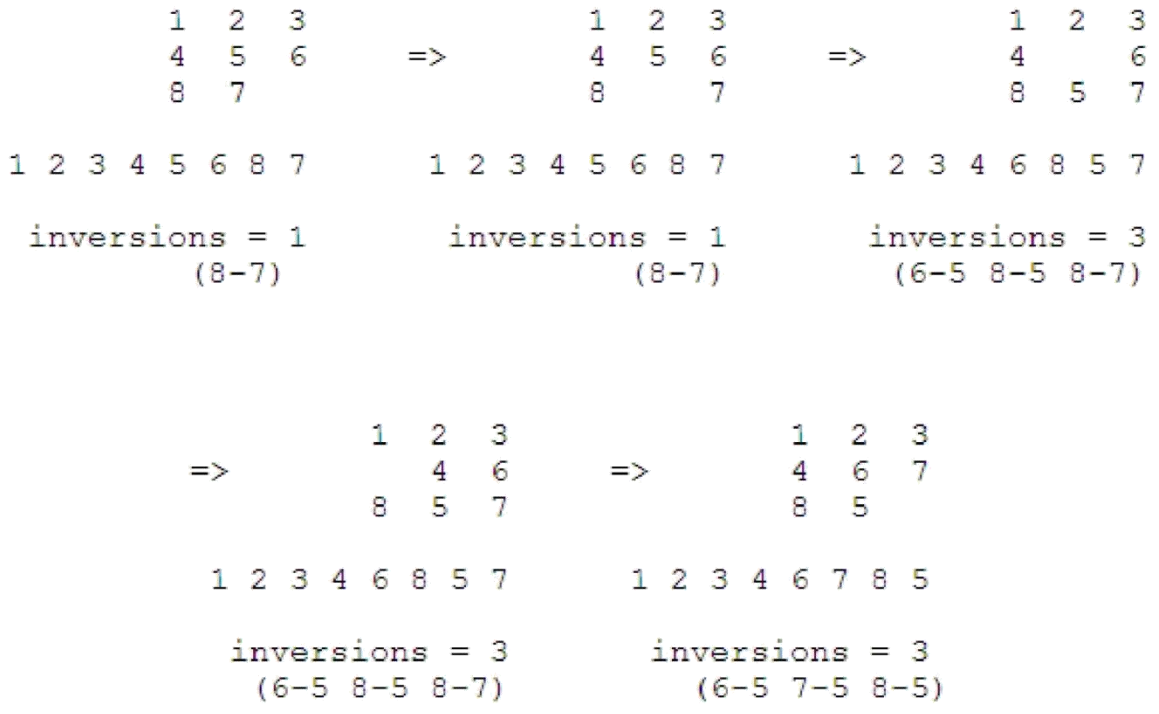


Detecting unsolvable puzzles. Not all initial boards can lead to the goal board by a sequence of legal moves, including the two below:



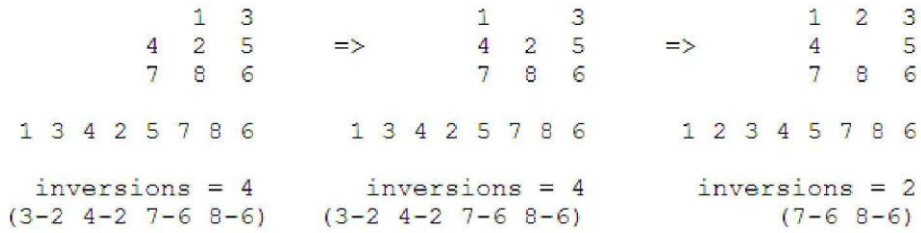
To detect such situations, use the fact that boards are divided into two equivalence classes with respect to reachability: (i) those that lead to the goal board and (ii) those that cannot lead to the goal board. Moreover, we can identify in which equivalence class a board belongs *without* attempting to solve it.

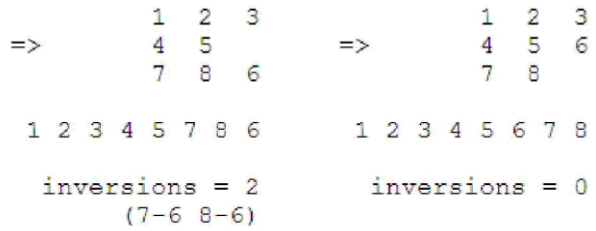
- *Odd board size.* Given a board, an *inversion* is any pair of blocks *i* and *j* where $i < j$ but *i* appears after *j* when considering the board in row-major order (row 0, followed by row 1, and so forth).



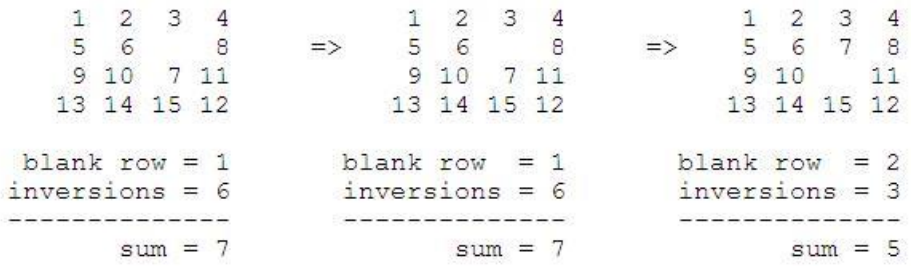
If the board size *N* is an odd integer, then each legal move changes the number of inversions by an even number. Thus, if a board has an odd number of inversions, then it *cannot* lead to the goal board by a sequence of legal moves because the goal board has an even number of inversions (zero).

The converse is also true: if a board has an even number of inversions, then it *can* lead to the goal board by a sequence of legal moves.

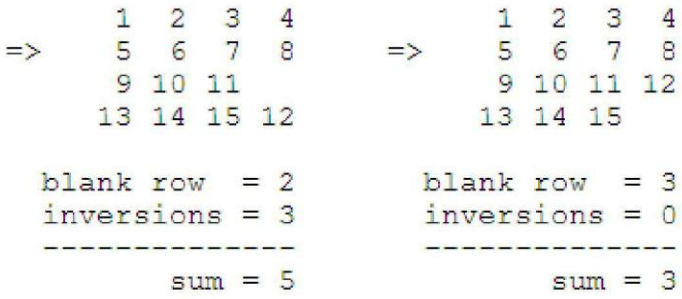




- *Even board size.* If the board size *N* is an even integer, then the parity of the number of inversions is not invariant. However, the parity of the number of inversions *plus* the row of the blank square is invariant: each legal move changes this sum by an even number. If this sum is even, then it *cannot* lead to the goal board by a sequence of legal moves; if this sum is odd, then it *can* lead to the goal board by a sequence of legal moves.



.



Tasks:

1. Implement A* search algorithm using the three mentioned heuristics.
2. Show the optimal cost to reach the goal state and the steps.
3. For different heuristics, find number of explored nodes and expanded nodes.
4. Experiment for n=8, 15.

Deadline: December 2, 2018, 11:50 pm.

Do not copy code from other sources.