N-Puzzle

[Problem Definition 2](#_Toc25483376)

[General Idea 2](#_Toc25483377)

[8 Puzzle 2](#_Toc25483378)

[Special Cases 3](#_Toc25483379)

[General N Puzzle 3](#_Toc25483380)

[Main Goals in Details 3](#_Toc25483381)

[Project Requirements 4](#_Toc25483382)

[Required Implementation 4](#_Toc25483383)

[Input 5](#_Toc25483384)

[Output 5](#_Toc25483385)

[Test Cases 5](#_Toc25483386)

[Deliverables 6](#_Toc25483387)

[Implementation (60%) 6](#_Toc25483388)

[Document (40%) 6](#_Toc25483389)

[Allowed Codes 6](#_Toc25483390)

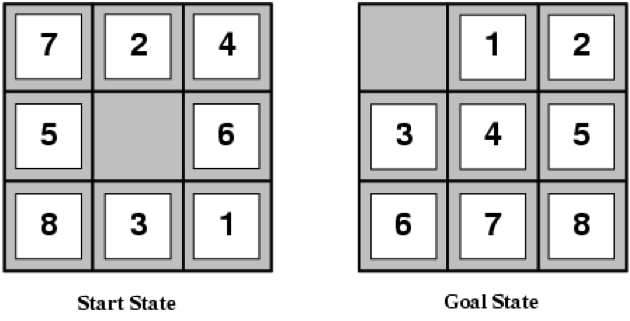
[Milestones 6](#_Toc25483391)

[BONUSES 7](#_Toc25483392)

# Problem Definition

The N-Puzzle is known in finite versions such as the 8-puzzle (a 3x3 board) and the 15-puzzle (a 4x4 board), and with various names like "sliding block", "tile puzzle", etc. The N-Puzzle is a board game for a single player. It consists of (N) numbered squared tiles in random order, and one blank space ("a missing tile").

The objective of the puzzle is to rearrange the tiles in order by making sliding moves that use the empty space, using the fewest moves. Moves of the puzzle are made by sliding an adjacent tile into the empty space. Only tiles that are horizontally or vertically adjacent to the blank space (not diagonally adjacent) may be moved.

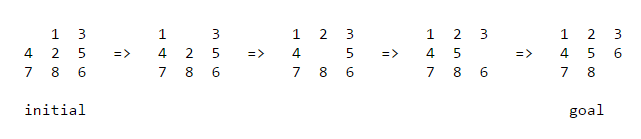


The N-puzzle is a classical problem for modelling algorithms involving heuristics. Commonly used heuristics for this problem include counting the number of misplaced tiles [Hamming Distance] or finding the sum of the taxicab distances [Manhattan distance] between each block and its position in the goal configuration.

# General Idea

## 8 Puzzle

The 8-puzzle problem is a puzzle popularized by Sam Loyd in the 1870s. It is played on a 3-by-3 grid with 8 square blocks labeled 1 through 8 and a blank square. Your goal is to rearrange the blocks so that they are in order. 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 position (left) to the goal position (right).



## Special Cases

Not all initial board positions can lead to the goal state as the following example:

|  |  |  |
| --- | --- | --- |
| 1 | 2 | 3 |
| 4 | **5** | **6** |
| 8 | **7** |  |

This board has NO feasible solution possible. So, you should work on detecting and reporting such situations too.

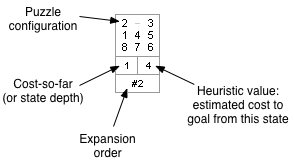
## General N Puzzle

The *N*-puzzle (N = 8, 15...) is a classical problem for modelling [algorithms](https://en.wikipedia.org/wiki/Algorithm) involving [heuristics](https://en.wikipedia.org/wiki/Heuristic_(computer_science)). Commonly used heuristics for this problem include counting the number of misplaced tiles (Hamming Distance) OR finding the sum of the [Manhattan distances](https://en.wikipedia.org/wiki/Taxicab_distance) between each block and its position in the goal configuration. Note that both are admissible, i.e. they never overestimate the number of moves left, which ensures optimality for certain search algorithms such as A\*. A\* is one of the informed search algorithms that can be used in such problems to get the optimal solution.

# Main Goals in Details

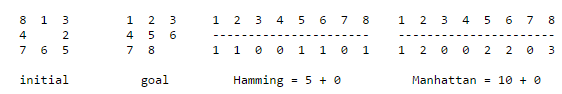
Your task is **to read in a file** containing an N puzzle with (N) numbered tiles and one blank space – representing an initial state. To solve the following puzzle:

1. **First, detect whether the board is solvable or not.** If not solvable, then no feasible solution for the given puzzle.
2. **If solvable,** then your objective is to get the shortest path to get the final solution from the initial board using A\* as a search algorithm as the following:
3. Convert the given puzzle into tree while applying the A\* search algorithm on it to get the least number of movements that can be needed while the searching process to get the goal state. Where each node in the search tree represents an arrangement of tiles (one board state), for example the initial given state will be the graph’s root. A typical example for the state is shown below:



1. The success of this approach hinges on the choice of **priority function for a state (Heuristic value)**. You will search for the goal based on the following two priority functions:
2. **Hamming priority function**: The number of blocks in the wrong position, plus the number of moves made so far to get to the state. Intuitively, a state with a small number of blocks in the wrong position is close to the goal state, and we prefer a state that has been reached using a small number of moves.
3. **Manhattan priority function**: The sum of the distances (sum of the vertical and horizontal distance) from the blocks to their goal positions, plus the number of moves made so far to get to the state.

For example, the Hamming and Manhattan priorities of the initial state below are 5 and 10, respectively.



* Hint: Since you will search for the goal upon some criteria [priority functions], **priority queue** will be a suitable data structure in this case to use for EFFICIENT implementation.

# Project Requirements

## Required Implementation

|  |  |
| --- | --- |
| **Requirement** | **Performance** |
| 1. Read in a file containing an N board with (N – 1) numbered tiles and one blank space – representing an initial state. |  |
| 1. **Determine** whether a given state is solvable or not? If not solvable**,** then no feasible solution for the given board and EXIT. | **Time:** should be **bounded by O(S2)**, S is the puzzle size |
| 1. **IF SOLVABLE, apply A\* search algorithm** on N-puzzle to construct graph/tree till you get the goal state with the shortest path from the initial state with the lowest number of moves. | **Time:** **bounded by O(E log(V))**, E is the total number of moves and V is the number of states till reaching to the solution |
| 1. **Print a STEP by STEP** movements occur in the A\* algorithms till you reach the final solvable board. |  |

## Input

**Puzzle file:**

***First line is a number indicating the size*** of a puzzle (N).

*T****he description is just a list of the tiles in their initial positions***, with the rows listed from top to bottom, and the tiles listed from left to right within a row, where the tiles are represented by numbers 1 to N2 - 1, plus '0' which is equivalent to the blank space.

Here is an abbreviated example:

**Puzzle1.txt:**

**3**

**0 1 3**

**4 2 5**

**7 8 6**

## Output

Print out:

1. Statement that indicates if the given board is solvable or not.
2. If solvable, display the board at each movement till the final solution to simulate the shortest sequence of moves till you solve the puzzle.
3. The fewest number of moves required at the end to re-arrange the puzzle.

Here is an abbreviated example:

**Solvable.**

**# steps = 4**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | 3 |  | 1 | 0 | 3 |  | 1 | 2 | 3 |  | 1 | 2 | 3 |  | 1 | 2 | 3 |
| 4 | **2** | **5** | **4** | **2** | **5** | **4** | **0** | **5** | **4** | **5** | **0** | **4** | **5** | **6** |
| 7 | **8** | **6** | **7** | **8** | **6** | **7** | **8** | **6** | **7** | **8** | **6** | **7** | **8** | **0** |

**Goal**

**Initial**

## Test Cases

* Sample Cases (Solvable & Unsolvable)
* Complete Cases

# Deliverables

## Implementation (60%)

1. **Read** in a file containing an N board with (N – 1) numbered tiles and one blank space – representing an initial state.
2. **Determine** whether a given state is solvable or not?
3. **Implement A\*** search algorithm on N-puzzle by constructing the graph/tree and the priority queue according to the following two different priority functions
   1. Hamming Distance
   2. Manhattan Distance
4. **Print a STEP by STEP** movements occur in the A\* algorithms till you reach the final solvable board.

## Document (40%)

1. Entire Source code.
2. Detailed analysis of your code.
3. Comparison between the two distances (Hamming vs. Manhattan) over the "**Complete Test**" cases:
   1. Min number of moves
   2. Execution time

## Allowed Codes

* Priority Queue Code [**MUST UNDERSTAND ITS CODE AND ANALYZE CORRECTLY**]

# Milestones

|  |  |  |
| --- | --- | --- |
|  | **Deliverables** | **Due to** |
| **Milestone1** | 1. **Read** and parse the input file 2. R**epresent** the N-puzzle in form of graph/tree 3. **Implement A\*** search algorithm using **Hamming Distance** 4. STEP by STEP display for the 3x3 case 5. **Modify A\*** search algorithm to use **Manhattan Distance** 6. Find a formula to indicate whether any given game state is solvable or not. 7. Documentation II 8. **Output:**    * + Min number of moves      + Solvable or not + min number of moves + STEP by STEP display for 3x3 case | **(START of Week Before Practical Exam)** |
| **For Milestone1:**   * + **MUST** deliver the required tasks and **ENSURE** it’s worked correctly   + **MUST** deliver in your scheduled time (TO BE ANNOUNCED) | | |

# BONUSES

1. Apply **different shortest path algorithm** (BFS or DFS) to search for the goal state. Show the difference between it and A\* you used to solve the unordered board in terms of the number of movements done by each and how they were be done (simulate the solution step by step).
2. **Simulation user friendly GUI** which allows you to rewind the search one step (movement) at a time over a generic N-puzzle. (e.g. generate (NxN) matrix of picture boxes and swap them according to each movement to simulate the solution step by step).

<https://www.geeksforgeeks.org/check-instance-8-puzzle-solvable/>

<https://www.geeksforgeeks.org/8-puzzle-problem-using-branch-and-bound/>

<https://algorithmsinsight.wordpress.com/graph-theory-2/a-star-in-general/implementing-a-star-to-solve-n-puzzle/>