Assignment 1

Dr. Ilias Kotsireas

CP468  
Group 10

Adnan Awad  
169028425

Hubert Bao  
169077248

Zakariya Boukhezza  
169024296

Ebrahim Ghani  
169072923

Ali Syed Muhammed  
169020955

# Table of Contents

[Table of Contents](#_joaoqsandzmi)

[Problem Formulation](#_riovbuyp32ks)

[State](#_ddyp50h0v6mx)

[Initial state](#_tv9vg6qx3q5y)

[Actions](#_y811dj1pl7wg)

[Transition model](#_gnyu8lttkj6m)

[Goal test](#_wj5yakatjxt)

[Path cost](#_fhym4ccchofo)

[GitHub Repository](#_psio7hq6vwlp)

[Explanation](#_5r2zpds67rt)

[H3](#_jg1yz06k7f84)

[Milestone 1](#_5pythgcthalg)

[Milestone 2](#_zgicjvbmu0s8)

[Result](#_7vh8am2c8nj6)

[Analysis](#_fcanxxegj2au)

[Python List Implementation](#_hgezwv8qjbyk)

[Heapq Implementation](#_as62juzdg2od)

[Comment](#_n5cpnl2h20im)

[Appendix](#_igjx3ul3nj5j)

[Code](#_9pdfz97mxspp)

[Actions.py](#_5mjsw75hso5u)

[Analytics.py](#_9gcb4awqlmeq)

[AstarSearch.py](#_iwyroklwub9c)

[Generator.py](#_ycw5341po2p)

[Heuristics.py](#_uo1bsqhd227o)

[Node.py](#_xwogvq3awlot)

[Problem.py](#_rpzojq19wkng)

[main.py](#_h0jy52okn1ch)

[Results](#_7fn0lr24zb9u)

[8\_Puzzle](#_efaa62t44ur)

[15\_Puzzle](#_k2b4n4falw0)

[24\_Puzzle](#_wwgaihxweog6)

[References](#_c313odlirqn9)

# 

# Problem Formulation

## State

s: [] \* 9

Represent the 8-puzzle, such that 8/9 tiles must have number 1-8, and 1 blank tile

## Initial state

si = any: s

## Actions

a: {U, D, L, R}

Represent up, down, left, right respectively

## Transition model

Let xs: s = current state

Let xa: a = current action

U(xs): move tile below blank space up

D(xs): move tile above blank space down

L(xs): move tile right of blank space left

R(xs): move tile left of blank space right

Result(xs, xa): s with tiles moved accordingly

## Goal test

sg = [1,2,3,4,5,6,7,8,nil]

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

## Path cost

Let xy: s = result state

Cost(xs, xa, xy) = 1

# GitHub Repository

See here: [https://github.com/TheChroniclerr/CP468/](https://github.com/TheChroniclerr/CP468)

# Explanation

## H3

Our h3 was implemented using Euclidean distance. It is similar to Manhattan distance, except it calculates the actual geometric distance by using the Pythagorean theorem between the current state coordinates and goal state coordinates and sums them up. This is slightly worse than Manhattan and a lot better than Misplaced Tiles. Manhattan **dominates** Euclidean as Euclidean represents the direct distance, and Manhattan represents the components of that distance, so Manhattan will yield larger h(n) most of the time by the rule of Triangular Inequality.

## Milestone 1

Initially, the code was implemented using list and list.sort() (see [here](https://github.com/TheChroniclerr/CP468/blob/94279d59f3e07d98b202e0ddcd5e1f4ba61d5638/a1/src/AstarSearch.py)). It was extremely problematic as it is too slow and could not run in most cases. It was also implemented before timeouts, so it was impossible to determine whether the endless running was due to a bug or actual timeout.

To prove that the implementation was correct, we did some performance tests and calculations. In a 10-second execution of that version, 3865 nodes were expanded, that is a rate of 386.5 nodes/second.

Note that without backtracking, even cases that are not that deep (i.e. have many steps for the optimal solution) can potentially expand a huge amount of nodes. This is due to the heuristics—including Manhattan—not finding the optimal expansion, and numerous suboptimal nodes expanded into dead ends, wasting computation power.

In which case, the largest 8-puzzle can expand up to nodes..

Although by remaining at a fixed computation rate, it can complete any 8-puzzle in up to 8 minutes, in practice it takes a lot longer. That is because the size of the frontier becomes larger and larger, and sorting takes longer and longer after each iteration. At one point, the program just runs into a choking point where it essentially “stops” running. We were able to verify this by leaving one process running for over 30 minutes, where instead of finishing, only 40k nodes were expanded.

This becomes problematic as for most instances, it seems the code either finishes running in <1 second or never finishes running - it is hard to tell whether it is because the instance is taking too long to run or if it is just bugged. At that point, we needed a **proof of concept** to verify that our code indeed still works for medium-sized expansions, confirming our theory that the unending execution is not actually a bug, but just an instance that requires an overly large number of expansions.

After many runs of the program. We found a good instance: [1, 5, 4, 2, None, 3, 6, 8, 7]. It expanded ~10k nodes for Euclidean and Manhattan and ~15k for Misplaced Tiles. This takes 2–3 seconds to run. This instance was a milestone for us, as it increased our confidence that our code was indeed not bugged.

386.5/second for cases, that would take approximately . So the deepest would take about 15–17 minutes, whereas the shallowest of (ideally) 24 steps would take 0 seconds.

## Milestone 2

The performance of our code is drastically improved with the utilization of “heapq” for sorting f(n)—or we could’ve simply chosen to not use Python, but oh well. The reason being “heapq” uses hashes that drastically improves the average case time complexity, as we will talk about later in section “Analysis”. Here, let’s calculate the performance of our most recent version shared in “Code”.

After a brief testing, in 2 seconds, it produced 61773, 49462, 48723 node expansions for Misplaced Tiles, Manhattan, and Euclidean respectively. Rounding it to 50k, we get 25k nodes/second. 25000 nodes/second / 386.5 nodes/second ~65. In other words, **heapq is 65x faster than list sort**. In 10 seconds, heapq can expand 250k nodes - that is enough to fully complete the 8-puzzle.

However, even this also hits a limit for larger puzzles, as evident from the TIMEOUT indicated in our CSV file results.

Note here, we also implemented a timeout function after becoming aware that there can be instances where even heapq will take forever.

Our next milestone is reached where for a generateRandom(5, 70, 70) (i.e. 24-puzzle generated by randomly scrambling 70 times) and a timeout of 5 seconds, only 1/100 instance failed (surpassing timeout period): [1, 3, 8, 15, 5, 6, 2, 19, 14, 10, 11, 7, 4, 18, 20, 16, 12, 13, None, 9, 21, 17, 22, 23, 24].

However, heapq is not the limit. The performance can be further improved by reducing other bottlenecks in the code, such as deepcopy(). Lastly, coding in a faster language such as C can result in even better performance.

## Result

Our result is generated by a timeout of 5 seconds for scrambling 70 times, 100 times, and 70 times for 8, 15, and 24 puzzle respectively. These results are then stored in a CSV file.

The “TIMEOUT” indicates a timeout, where the code was not able to attain a result in 5 seconds. Of course, I did not notate ># here because it should be self-evident. Given 25k nodes/second from our calculation before, then 5\*25k = 125k. Anything with a timeout likely reached at least >125k node expansions. This would be quite common for Misplaced Tiles.

The “step” in our result is calculated from getting the total amount of states passed from initial to goal state. This is actually 1 more than the path cost, as it does not take 1 step for the initial state to get to itself. For our result, we had it with +1.

Analyzing our result, we can see that all of our steps are odd. That makes sense, as our generator value in main.py actually scrambles 70, 100, 70 times respectively where the scramble does not shift back to the previous state. The parity of the moves should be the same as the parity of the steps. However, since in our case we have +1, then the parity would be opposite.

The difference between the three heuristics is negligible at the start, but grows over time. Misplaced Tiles seem to have as much as over 10x more nodes expanded by Manhattan. On the other hand, the difference between Manhattan and Euclidean is much smaller, where the difference only becomes more prominent reaching the 4th digit.

Going up to 15-puzzle, we saw that timeout starts occurring for Misplaced Tiles, starting at depth 27; whereas it starts at depth 37 for Euclidean; and for Manhattan, it went all the way to 43, and still hasn’t timed out.

The disparity between nodes expanded for the same depth seems to become much greater for larger puzzle sizes. This means the heuristic becomes much less accurate and therefore has worse predictability for larger puzzles.

In the 24-puzzle for much larger values, the advantage of Manhattan became much more apparent. Take [1, 7, 2, 4, 5, 6, 3, 14, 8, 10, 16, 13, 12, 9, 18, 21, 11, 24, None, 15, 22, 23, 17, 20, 19], it has 291, 1919, 165250 nodes expanded for Manhattan, Euclidean, and Misplaced Tiles respectively. In which case, Manhattan expanded 6.5x less node than Euclidean and 568x less node than Misplaced Tiles.

# Analysis

## Python List Implementation

This is the time complexity calculated for the implementation described in “Milestone 1”.

Here, we use a Python list to implement the queue.

1. Node with optimal f(n) popped from queue
2. Node Expanded
3. For each expanded node
   1. Solve for f(n)
   2. Insert back to queue
4. Sort queue by f(n)
5. Repeat back to 1.

Let s be the state space (all reachable states). For a\*a-puzzle, the state space is a!/2.

Let P be the size of the puzzle. O(P) can be considered O(1) since puzzle have fixed sizes.

Node expansion (2.) - O(1)

* Find valid actions - find blank space O(P) + simple algebraic calculations O(1) = O(1)
* Shift - = O(1)

f(n) calculation (3a.) - O(1)

* g(n) - path so far + action cost (in this case 1) = O(1)
* h(n) - O(P) = loop through tiles = O(1)
* g(n) + h(n) - simple addition = O(1)

Queue operations (Python list)

* Insert (3b.) - = O(1)
* Pop (1.) - pop from front of queue have to shift the whole data structure = O(s)
* Sort queue (4.) - sort in general takes O(n log n). In this problem, it sorts a frontier queue that can have up to the size of state space = O(s log s)

Loop (5.) - looping through everything described above O(s) \* the sort operation that is bottleneck inside the loop O(s log s) = **O(s2 log s)**

## Heapq Implementation

Heapq improves performance as it is an ordered data structure. Instead of sorting, it amortizes the cost by inserting into and retrieving from an ordered structure via search.

1. Node with optimal f(n) popped from heapq
2. Node Expanded
3. For each expanded node
   1. Solve for f(n)
   2. Insert back to heapq, auto sort
4. Repeat back to 1.

Heapq operations

* Retrieve - binary search O(log s) + retrieval O(1) = O(log s)
* Insert - binary search for insertion spot O(log s) + insert O(1) = O(log s)

Loop (5.) - looping through is again O(s) \* amortized bottleneck O(log s) = **O(s log s)**

## Comment

O(s log s) < O(s2 log s)

Here, we see that implementing heapq improved the Time Complexity significantly. This is evident from our performance calculation, resulting in a 65x improved rate.

This is important because by running the code ourselves, we can see an actual physical difference in running time between these implementations, demonstrating that theoretical concepts such as Time Complexity can have real and tangible differences.

# Appendix

## Code

We have created 8 py files to work with. They are as follows:

### Actions.py

The code reads as follows:

| **import** math **from** copy **import** deepcopy **from** typing **import** Literal  ActionsType = Literal["U", "D", "L", "R"]  **class** **Actions**:  **def** **\_\_init\_\_**(self, s: list) -> **None**:  """Class constructor.   Args:  s (list): State to perform action to.  """  **assert** s **and** isinstance(s, list) **and** math.sqrt(len(s)).is\_integer(), "Invalid puzzle size."    self.s = s  self.width: int = int(math.sqrt(len(s))) *# get the n of n-puzzle (n by n)*  self.blankPos: int | **None** = self.\_findBlankTileIndex() *# index of blank tile*  **assert** self.blankPos != -1, "Invalid puzzle, no blank tile."  self.validActions: list[str] = self.\_findValidActions() *# list of valid actions*    **def** **result**(self, a: ActionsType) -> list | **None**:  """Transition Model - Alternative method of action call using ActionType keys.   Args:  a (ActionsType): A valid type of action.   Returns:  list | None: Return list if move successful, else None.  """  actionsMap: dict = {  "U": self.up,  "D": self.down,  "L": self.left,  "R": self.right  }   **return** actionsMap[a]()    **def** **up**(self) -> list | **None**:  """Action - Move the tile below the blank tile upwards.   Returns:  list | None: Return list if move successful, else None.  """  **if** "U" **in** self.validActions:  newS: list = deepcopy(self.s)  self.\_shiftTile(newS, self.blankPos + self.width)  **return** newS  **return** **None**    **def** **down**(self) -> list | **None**:  """Action - Move the tile above the blank tile downward.   Returns:  list | None: Return list if move successful, else None.  """  **if** "D" **in** self.validActions:  newS: list = deepcopy(self.s)  self.\_shiftTile(newS, self.blankPos - self.width)  **return** newS  **return** **None**    **def** **left**(self) -> list | **None**:  """Action - Move the tile right of the blank tile leftward.   Returns:  list | None: Return list if move successful, else None.  """  **if** "L" **in** self.validActions:  newS: list = deepcopy(self.s)  self.\_shiftTile(newS, self.blankPos + 1)  **return** newS  **return** **None**    **def** **right**(self) -> list | **None**:  """Action - Move the tile left of the blank tile rightward.   Returns:  list | None: Return list if move successful, else None.  """  **if** "R" **in** self.validActions:  newS: list = deepcopy(self.s)  self.\_shiftTile(newS, self.blankPos - 1)  **return** newS  **return** **None**   **def** **\_shiftTile**(self, newS: list, srcPos: int) -> list:  """Shift the tile in the desired direction.  Shift from a copy of the puzzle.   Args:  newS (list): A copy to shift from.  srcPos (int): Index of tile to be shifted.   Returns:  list: Final state.  """  newS[self.blankPos], newS[srcPos] = newS[srcPos], **None**  **return** newS   *# Configuration functions*  **def** **\_findBlankTileIndex**(self) -> int:  """Find the blank tile in the n-puzzle.   Returns:  int: Index of the blank tile.  """  **for** i, tile **in** enumerate(self.s):  **if** tile **is** **None**:  **return** i  **return** -1   **def** **\_findValidActions**(self) -> list:  """Check if it is possible to move up/down/left/right.   Returns:  list: All valid actions.  """  validActions: list = [];    **if** 0 <= self.blankPos + self.width < len(self.s): validActions.append("U") *# check bottom*  **if** 0 <= self.blankPos - self.width < len(self.s): validActions.append("D") *# check top*  **if** (self.blankPos % self.width) != 0: validActions.append("R") *# check left*  **if** (self.blankPos % self.width) != (self.width - 1): validActions.append("L") *# check right*    **return** validActions |
| --- |

### Analytics.py

The code reads as follows:

| **import** os **import** csv **from** Problem **import** Problem *# type hint*  DEFAULT\_DIR: str = "a1/output"  FILENAMES: dict = {  9: "/8\_Puzzle.csv",  16: "/15\_Puzzle.csv",  25: "/24\_Puzzle.csv" }  CSV\_HEADER: list = [  "initial\_state", "h1\_steps", "h1\_expanded", "h2\_steps", "h2\_expanded", "h3\_steps", "h3\_expanded" ]  **class** **Analytics**:  **def** **\_\_init\_\_**(self, problem: Problem):  self.initialState: str = str(problem.initialState)  *# self.filePointer # points to location in CSV*  self.fileDir: str = DEFAULT\_DIR + FILENAMES[len(problem.initialState)]  self.data: list[dict] = self.\_loadExistingCSV() *# a copy of whole CSV data to modify and overwrite back to*  self.changes: dict = { *# record data to be added/changed, key - column name, value - stored data*  f"{problem.hTag}\_steps": 0,  f"{problem.hTag}\_expanded": 0  }    **def** **recordSteps**(self, steps: int) -> **None**:  **for** key **in** self.changes:  **if** "steps" **in** key:  self.changes[key] = steps   **def** **incrementNodesExpanded**(self) -> **None**:  **for** key **in** self.changes:  **if** "expanded" **in** key:  self.changes[key] += 1    **def** **writeCSV**(self) -> **None**:  *# find record if already exists*  record = **None**  **for** row **in** self.data:  **if** row["initial\_state"] == self.initialState:  record = row  **break**    *# create record if not exist, append to data*  **if** record **is** **None**:  record = self.\_newRecord()  self.data.append(record)    *# apply changes to data (using record) and overwrite CSV*  **for** columnName, newValue **in** self.changes.items():  record[columnName] = newValue  self.\_overwrite()    **def** **\_newRecord**(self) -> dict:  record: dict = {  "initial\_state": self.initialState  }  **for** h **in** ["h1", "h2", "h3"]:  record[f"{h}\_steps"] = "TIMEOUT"  record[f"{h}\_expanded"] = "TIMEOUT"   **return** record    **def** **\_loadExistingCSV**(self) -> list:  **if** **not** os.path.exists(self.fileDir):  **with** open(self.fileDir, 'w', newline='') **as** f:  writer = csv.DictWriter(f, fieldnames=CSV\_HEADER)  writer.writeheader()  **return** []    **with** open(self.fileDir, 'r', newline='') **as** f:  reader = csv.DictReader(f)  **return** list(reader)    **def** **\_overwrite**(self) -> **None**:  **with** open(self.fileDir, 'w', newline='') **as** f:  writer = csv.DictWriter(f, fieldnames=CSV\_HEADER)  writer.writeheader()  writer.writerows(self.data) |
| --- |

### AstarSearch.py

The code reads as follows:

| **import** heapq **import** Problem **from** Node **import** Node **from** Analytics **import** Analytics **from** Actions **import** Actions *# type hint*  **def** **AstarSearch**(problem: Problem) -> Node | **None**:  """Using A\* search algorithm to solve the problem instances as graphs.   Args:  problem (Problem): The problem instance.   Returns:  Node: The node of the goal state.  """  rootNode: Node = Node(problem.initialState, **None**, **None**, 0)  frontierHeap: list[tuple[float, Node]] = [(problem.h(rootNode.state), rootNode)] *# use heapq for performance*  visited: set = set() *# hash-set to track visited nodes*  visited.add(tuple(problem.initialState)) *# convert list to hashable tuple*    analytics: Analytics = Analytics(problem)  **while** frontierHeap:  \_, currNode = heapq.heappop(frontierHeap)  analytics.incrementNodesExpanded()    *# check if current state reached goal state*  **if** problem.reachGoal(currNode.state):  analytics.recordSteps(len(Node.getAncestors(currNode)))  analytics.writeCSV()  **return** currNode    currActions: Actions = problem.setAction(currNode.state)  **for** actionName **in** currActions.validActions:  *# find new state*  newState: list | **None** = currActions.result(actionName)  **if** newState **is** **None**:   **continue**  *# skip already visited node*  **if** tuple(newState) **in** visited:   **continue**  visited.add(tuple(newState))  *# compute new node data, add to heap*  newNode: Node = Node(newState, currNode, actionName, problem.getPathCost() + currNode.pathCost)  f\_n: int = newNode.pathCost + problem.h(newState) *# f(n)*  heapq.heappush(frontierHeap, (f\_n, newNode)) *# sorts by first element (f\_n) automatically*    **return** **None** |
| --- |

### Generator.py

The code reads as follows:

| **import** random  blank = **None**   **def** **isSolvable**(state, n):  a = [x **for** x **in** state **if** x != blank]  inversions = 0   **for** i **in** range(len(a)):  **for** j **in** range(i + 1, len(a)):  **if** a[i] > a[j]:  inversions += 1   **if** n % 2 == 1:  **return** inversions % 2 == 0   **else**:  blank\_i = state.index(blank)  blank\_distance\_top = blank\_i // n - 1  blank\_distance\_bottom = n - blank\_distance\_top  **return** (inversions + blank\_distance\_bottom) % 2 == 0   **def** **allMoves**(state, n):  blank\_i = state.index(**None**)  r, c = divmod(blank\_i, n)  moves = []   **if** r > 0:  moves.append(-n) *# up*  **if** r < n - 1:  moves.append(n) *# down*  **if** c > 0:  moves.append(-1) *# left*  **if** c < n - 1:  moves.append(1) *# right*    **return** moves   **def** **doMove**(state, move):  new\_state = state.copy()  blank\_i = new\_state.index(**None**)  target\_index = blank\_i + move    new\_state[blank\_i], new\_state[target\_index] = (  new\_state[target\_index],  new\_state[blank\_i],  )    **return** new\_state   **def** **generateRandom**(n: int, min: int, max: int) -> list:  """Generate a random nxn-Puzzle in a []\*n list.   Args:  n (int): The size of the puzzle; nxn  min (int): The minimum shifts from original puzzle  max (int): The maximum shifts from original puzzle   Returns:  list: The generated puzzle  """  rand = random.randint(min, max)  state = [i **for** i **in** range(1, n \* n)] + [blank]   **for** i **in** range(rand):  moves = allMoves(state, n)  move = random.choice(moves)  state = doMove(state, move)   **return** state   *# #testing* *# n=3* *# min=20* *# max=40* *# state = [i for i in range(1, n\*n)] + [blank]* *# print("3x3:",isSolvable(state,n))* *# state = [i for i in range(1, 7)] + [8] + [7] + [blank]* *# print("3x3:",isSolvable(state,n))* *# state=generateRandom(n,min,max)* *# print("3x3:",isSolvable(state,n))* *# print(state)* *# n=4* *# state = [i for i in range(1, n\*n)] + [blank]* *# print("4x4:",isSolvable(state,n))* *# state = [i for i in range(1, 14)] + [15] + [14] + [blank]* *# print("4x4:",isSolvable(state,n))* *# state=generateRandom(n,min,max)* *# print("4x4:",isSolvable(state,n))* *# print(state)* *# n=5* *# state = [i for i in range(1, n\*n)] + [blank]* *# print("5x5:",isSolvable(state,n))* *# state = [i for i in range(1, 23)] + [24] + [23] + [blank]* *# print("5x5:",isSolvable(state,n))* *# state=generateRandom(n,min,max)* *# print("5x5:",isSolvable(state,n))* *# print(state)* |
| --- |

### Heuristics.py

The code reads as follows:

| **import** math **from** typing **import** Literal, Callable  Type = Literal["h1", "h2", "h3"]  **def** **h1**(s: list, g: list) -> int:  """Heuristic - estimate true cost from current state to goal state via number of misplaced tiles.   Args:  s (list): Current state.  g (list): Goal state.   Returns:  int: Number of misplaced tiles.  """  *# current state must be a list, and list must have size = n\*n, for n-puzzle*  **assert** type(s) == list **and** math.sqrt(len(s)).is\_integer() , "Invalid heuristic."   misplacedTiles: int = 0   **for** i **in** range(0, len(s)):  misplacedTiles += 1 **if** s[i] != g[i] **else** 0    **return** misplacedTiles  **def** **h2**(s: list, g: list) -> int:  """Heuristic - estimate true cost from current state to goal state via total Manhattan distance.  Manhattan distance is the sum of the distances of the tiles from their goal positions.   Args:  s (list): Current state.  g (list): Goal state.   Returns:  int: total Manhattan distance  """  **return** int(\_findSumOfDists(s, g, **lambda** currRow, currCol, goalRow, goalCol: *# Manhattan distance formula*  abs(currRow - goalRow) + abs(currCol - goalCol)  ))  **def** **h3**(s: list, g: list) -> float:  """Heuristic - estimate true cost from current state to goal state via total Euclidean distance.  Euclidean distance is the distance between two points (current state & goal state in this case) in Euclidean space.   Args:  s (list): Current state.  g (list): Goal state.   Returns:  int: Euclidean distance  """  **return** \_findSumOfDists(s, g, **lambda** currRow, currCol, goalRow, goalCol: *# Euclidean distance formula*  math.dist((currRow, currCol), (goalRow, goalCol))  )  **def** **\_findSumOfDists**(s: list, g: list, getDist: Callable) -> int | float:  """Auxillary function, find the sum of distances by the type of distance calculation used.   Args:  s (list): Current state.  g (list): Goal state.   Returns:  int: Total distance.  """  **assert** type(s) == list **and** math.sqrt(len(s)).is\_integer() , "Invalid heuristic."   totalDistance: int = 0  width: int = int(math.sqrt(len(s))) *# get the n of n-puzzle (n by n)*  indexMap: list = \_createIndexMap(g)   **for** currPos, tileNum **in** enumerate(s):  **if** tileNum **is** **None**: **continue**  goalPos: int = indexMap[tileNum]  *# find the x/y position of s and g, then calculate the Manhattan distance*  currRow, currCol = divmod(currPos, width)  goalRow, goalCol = divmod(goalPos, width)  totalDistance += getDist(currRow, currCol, goalRow, goalCol)    **return** totalDistance  **def** **\_createIndexMap**(s: list) -> list:  """Auxillary function, map the tile number to its goal index.  Let list key represent the tile number, and list value represent the goal index.   Args:  s (list): Goal state.   Returns:  list: Index map.  """  indexMap: list = [**None**] \* (len(s) + 1) *# index position 0 is not used, since tile number starts on 1*    **for** goalPos, tileNumb **in** enumerate(s):  **if** tileNumb **is** **None**: **continue**  indexMap[tileNumb] = goalPos *# goalPos start at 0*    **return** indexMap  Function: dict[str, Callable] = {  "h1": h1,  "h2": h2,  "h3": h3 } |
| --- |

### Node.py

The code reads as follows:

| **class** **Node**:  **def** **\_\_init\_\_**(self, state: any, parent: any, action: str | None, pathCost: int = 1) -> **None**:  """Node constructor   Args:  state (any): Current state.  parent (any): Parent node.  action (str | None): Action parent node took to reach current state, None for root node.  pathCost (int, optional): g(n) - the cost from initial state to current state. Defaults to 1.  """  self.state = state  self.parent = parent  self.action = action  self.pathCost = pathCost *# g(n)*    **def** **\_\_str\_\_**(self) -> str:  **return** f"Node(state={self.state}, action={self.action}, pathCost={self.pathCost})"    **def** **\_\_lt\_\_**(self, other):  **return** self.pathCost < other.pathCost    **def** **getAncestors**(self) -> list:  """Retrieves all direct ancestors of current node.   Returns:  list: A list of direct ancestors ordered from ancestors to descendants.  """  ancestors: list = [self]  **while** ancestors[-1].parent:  ancestors.append(ancestors[-1].parent)    **return** ancestors[::-1] *# slice notation - reverse list* |
| --- |

### Problem.py

The code reads as follows:

| **import** Heuristics **from** Actions **import** Actions **from** typing **import** Callable  **class** **Problem**:  **def** **\_\_init\_\_**(self, initialState: list, goalState: list, heuristic: Heuristics.Type):  """Class constructor.   Args:  initialState (list): The starting state.  goalState (list): The final state to reach.  heuristic (Heuristics.Type): The type of heuristic used to approximate cost.  """  self.initialState = initialState  self.goalState = goalState  self.hTag = heuristic   **def** **h**(self, s: list) -> int | float:  """Gate function that forces Heuristics function to be called in Problem instance.  Find the heuristic value for the current state and pre-defined goal state.   Args:  s (list): Current state.   Returns:  int: estimated cost from current state to goal state.  """  **return** Heuristics.Function[self.hTag](s, self.goalState)   **def** **setAction**(self, s: list) -> Actions:  """Gate function that forces Actions instantiation from Problem instance.  Create an Actions instance for current state.   Args:  s (list): Current state.   Returns:  Actions: Actions instance.  """  **return** Actions(s)   **def** **reachGoal**(self, s: list) -> bool:  """Goal Test - Determines whether a given state is the goal state.   Args:  s (list): Current state.   Returns:  bool: True if goal state is reached, else False.  """  **return** s == self.goalState   **def** **getPathCost**(self, sX: list = None, a: Callable | None = None, nY: list = None) -> int:  """Path Cost - Assigns a numeric cost to each path (action) from previous to current state.  For this specific problem, the action cost is always 1.   Args:  sX (list, optional): Initial state. Defaults to None.  a (function | None, optional): Action function. Defaults to None.  nY (list, optional): Final state. Defaults to None.   Returns:  int: 1  """  **return** 1 |
| --- |

### main.py

The code reads as follows:

| **import** multiprocessing **import** Generator **from** Problem **import** Problem **from** AstarSearch **import** AstarSearch **from** typing **import** Callable **from** Node **import** Node *# type hint*  *# Default values* TIMEOUT = 5 *# seconds* GENERATIONS = 100 *# amount generated*  \_8\_PUZZLE\_GOAL\_STATE = [1, 2, 3, 4, 5, 6, 7, 8, **None**] *# 3x3* \_15\_PUZZLE\_GOAL\_STATE = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, **None**] *# 4x4* \_24\_PUZZLE\_GOAL\_STATE = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, **None**] *# 5x5*  **def** **runWithTimeout**(timeout\_: int, func: Callable, args=()):  **with** multiprocessing.Pool(1) **as** pool:  result = pool.apply\_async(func, args)  **try**:  **return** result.get(timeout=timeout\_)  **except** multiprocessing.TimeoutError:  *# print("Timeout reached. Terminating process.")*  pool.terminate() *# force stop safely*  pool.join()  **return** **None**  **if** \_\_name\_\_ == "\_\_main\_\_":  *# comment one of the for loop out to not generate*  *# change GENERATIONS in range for different number of problem instances to solve*    *# !there can be duplicate generations, double check CSV for 100 unique initial states.*  **for** i **in** range(0, GENERATIONS): *# 8-puzzle*  initialState: list = Generator.generateRandom(3, 70, 70) *# Generate puzzle*  print("ini: " + str(initialState))  **for** heuristic **in** ["h1", "h2", "h3"]:  problem: Problem = Problem(initialState, \_8\_PUZZLE\_GOAL\_STATE, heuristic)  result: Node|**None** = runWithTimeout(TIMEOUT, AstarSearch, (problem,))  **if** result:  print(heuristic + ": Success")  **else**:  print(heuristic + ": Timeout")    **for** i **in** range(0, GENERATIONS): *# 15-puzzle*  initialState: list = Generator.generateRandom(4, 100, 100) *# Generate puzzle*  print("ini: " + str(initialState))  **for** heuristic **in** ["h1", "h2", "h3"]:  problem: Problem = Problem(initialState, \_15\_PUZZLE\_GOAL\_STATE, heuristic)  result: Node|**None** = runWithTimeout(TIMEOUT, AstarSearch, (problem,))  **if** result:  print(heuristic + ": Success")  **else**:  print(heuristic + ": Timeout")    **for** i **in** range(0, GENERATIONS): *# 24-puzzle*  initialState: list = Generator.generateRandom(5, 70, 70) *# Generate puzzle*  print("ini: " + str(initialState))  **for** heuristic **in** ["h1", "h2", "h3"]:  problem: Problem = Problem(initialState, \_24\_PUZZLE\_GOAL\_STATE, heuristic)  result: Node|**None** = runWithTimeout(TIMEOUT, AstarSearch, (problem,))  **if** result:  print(heuristic + ": Success")  **else**:  print(heuristic + ": Timeout")  *# # SINGLE TEST CASE* *# problem = Problem([1, 5, 4, 2, None, 3, 6, 8, 7], \_8\_PUZZLE\_GOAL\_STATE, "h3")* *# print(problem.hTag)* *# goalNode: Node = AstarSearch(problem) # Run A\* search on the problem instance* *# print(goalNode)*  *# FIXED - closed set* *# failed cases: [6, 4, 2, 1, None, 5, 7, 8, 3]*  *# [5, 4, None, 7, 6, 1, 8, 2, 3]*  *# MILESTONE - proof of concept case, convincing me that code is not bugged for larger cases* *# suceeded case: [1, 5, 4, 2, None, 3, 6, 8, 7]*  *# FIXED - heapq better perf (if it's way too large, then it simply takes forever to run, not a bug)* *# failed cases: [5, 4, 3, 7, 6, 1, 2, 8, 13, 11, 12, 15, 14, 10, 9, None]*  *# MILETONE - the only 1/100 24-puzzle that failed for generateRandom(5, 70, 70)* *# failed case: [1, 3, 8, 15, 5, 6, 2, 19, 14, 10, 11, 7, 4, 18, 20, 16, 12, 13, None, 9, 21, 17, 22, 23, 24]* |
| --- |

## Results

The outputs were neatly formatted as CSV files. They are adapted into typeset tables as below:

### 8\_Puzzle

| initial\_state | h1\_steps | h1\_expanded | h2\_steps | h2\_expanded | h3\_steps | h3\_expanded |
| --- | --- | --- | --- | --- | --- | --- |
| [2, 4, 3, 7, None, 1, 6, 5, 8] | 17 | 727 | 17 | 124 | 17 | 160 |
| [7, 4, 1, 2, 3, 5, 8, 6, None] | 17 | 475 | 17 | 63 | 17 | 71 |
| [None, 4, 6, 3, 1, 2, 7, 5, 8] | 13 | 91 | 13 | 28 | 13 | 29 |
| [6, 4, 2, 1, 5, 3, 7, 8, None] | 13 | 99 | 13 | 29 | 13 | 30 |
| [2, 8, 3, 5, 6, 7, None, 1, 4] | 19 | 1280 | 19 | 121 | 19 | 163 |
| [5, 3, None, 2, 1, 8, 4, 6, 7] | 15 | 198 | 15 | 43 | 15 | 46 |
| [1, 2, 3, 4, None, 6, 7, 5, 8] | 3 | 3 | 3 | 3 | 3 | 3 |
| [1, 5, 6, 7, None, 4, 8, 3, 2] | 19 | 1868 | 19 | 342 | 19 | 405 |
| [2, 3, 6, 7, 4, 8, 5, 1, None] | 15 | 238 | 15 | 52 | 15 | 61 |
| [1, 6, 2, 3, None, 7, 8, 4, 5] | 21 | 4615 | 21 | 625 | 21 | 832 |
| [1, 5, 4, 7, 8, 2, None, 6, 3] | 17 | 534 | 17 | 89 | 17 | 103 |
| [4, 5, None, 7, 6, 1, 8, 3, 2] | 17 | 499 | 17 | 69 | 17 | 91 |
| [5, 3, None, 7, 2, 6, 8, 4, 1] | 21 | 3586 | 21 | 498 | 21 | 650 |
| [None, 1, 5, 4, 3, 6, 7, 2, 8] | 11 | 45 | 11 | 23 | 11 | 23 |
| [None, 2, 5, 4, 6, 1, 7, 8, 3] | 19 | 1422 | 19 | 362 | 19 | 403 |
| [None, 3, 6, 1, 4, 5, 7, 8, 2] | 13 | 111 | 13 | 42 | 13 | 44 |
| [None, 1, 3, 4, 5, 6, 7, 8, 2] | 13 | 134 | 13 | 62 | 13 | 63 |
| [None, 1, 5, 4, 3, 2, 7, 8, 6] | 9 | 21 | 9 | 16 | 9 | 16 |
| [7, 4, 3, 8, 1, 6, None, 5, 2] | 17 | 551 | 17 | 101 | 17 | 113 |
| [1, 2, 6, 3, 5, 7, 8, 4, None] | 19 | 1414 | 19 | 288 | 19 | 335 |
| [1, 2, 3, 6, 7, 4, 5, 8, None] | 17 | 633 | 17 | 159 | 17 | 181 |
| [8, 6, None, 3, 2, 1, 5, 4, 7] | 25 | 18494 | 25 | 1435 | 25 | 2199 |
| [None, 4, 2, 5, 1, 3, 7, 8, 6] | 9 | 17 | 9 | 12 | 9 | 12 |
| [6, 2, 4, 1, 3, 5, 7, 8, None] | 23 | 8632 | 23 | 1432 | 23 | 1772 |
| [1, 8, 3, 5, 2, 6, 4, 7, None] | 17 | 657 | 17 | 199 | 17 | 212 |
| [1, 5, 2, 4, None, 3, 7, 8, 6] | 5 | 5 | 5 | 5 | 5 | 5 |
| [None, 6, 2, 1, 3, 8, 7, 5, 4] | 17 | 582 | 17 | 91 | 17 | 126 |
| [1, 8, None, 5, 3, 2, 7, 4, 6] | 19 | 1618 | 19 | 376 | 19 | 439 |
| [7, 1, 3, 2, 8, 4, 5, 6, None] | 15 | 208 | 15 | 35 | 15 | 44 |
| [2, 1, 5, 3, None, 7, 8, 6, 4] | 23 | 10291 | 23 | 1079 | 23 | 1512 |
| [5, 6, 2, 4, None, 3, 1, 7, 8] | 19 | 1908 | 19 | 455 | 19 | 501 |
| [2, 8, 3, 1, 6, 4, 5, 7, None] | 21 | 3636 | 21 | 580 | 21 | 720 |
| [None, 7, 6, 1, 3, 4, 5, 2, 8] | 19 | 1368 | 19 | 136 | 19 | 194 |
| [1, 3, 6, 4, None, 8, 2, 5, 7] | 15 | 295 | 15 | 79 | 15 | 87 |
| [4, 8, 3, 1, 6, 2, 7, 5, None] | 21 | 3856 | 21 | 766 | 21 | 925 |
| [1, 8, None, 7, 4, 2, 6, 5, 3] | 15 | 210 | 15 | 24 | 15 | 31 |
| [8, 2, None, 4, 1, 3, 7, 5, 6] | 17 | 651 | 17 | 188 | 17 | 209 |
| [None, 5, 2, 6, 1, 4, 7, 3, 8] | 21 | 3575 | 21 | 445 | 21 | 604 |
| [6, 3, 2, 1, None, 8, 5, 4, 7] | 17 | 677 | 17 | 64 | 17 | 93 |
| [4, 8, 1, 7, None, 3, 5, 6, 2] | 17 | 669 | 17 | 84 | 17 | 102 |
| [7, 4, 1, 3, None, 6, 2, 5, 8] | 21 | 3976 | 21 | 443 | 21 | 584 |
| [6, 1, 2, 7, None, 3, 4, 5, 8] | 17 | 801 | 17 | 169 | 17 | 198 |
| [1, 4, 5, 2, 6, 8, 7, 3, None] | 23 | 9041 | 23 | 1555 | 23 | 1936 |
| [1, 6, None, 4, 2, 5, 7, 3, 8] | 15 | 280 | 15 | 109 | 15 | 112 |
| [5, 2, None, 8, 7, 6, 4, 3, 1] | 23 | 8112 | 23 | 824 | 23 | 1157 |
| [None, 1, 2, 4, 5, 3, 7, 8, 6] | 5 | 5 | 5 | 5 | 5 | 5 |
| [2, 1, 3, 4, 7, 6, 5, 8, None] | 19 | 1654 | 19 | 507 | 19 | 545 |
| [3, 1, 6, 2, None, 4, 7, 5, 8] | 13 | 115 | 13 | 39 | 13 | 39 |
| [3, 1, 5, 2, None, 6, 7, 4, 8] | 15 | 294 | 15 | 73 | 15 | 78 |
| [4, 7, 2, 1, 3, 6, None, 5, 8] | 19 | 1374 | 19 | 288 | 19 | 320 |
| [6, 1, 8, 4, None, 7, 2, 5, 3] | 25 | 22382 | 25 | 1496 | 25 | 2578 |
| [8, 1, 7, 4, 5, 2, 3, 6, None] | 25 | 17635 | 25 | 955 | 25 | 1850 |
| [1, 5, 3, 7, None, 2, 4, 8, 6] | 19 | 2144 | 19 | 670 | 19 | 726 |
| [3, 8, 4, 1, 5, 2, None, 7, 6] | 21 | 3323 | 21 | 488 | 21 | 590 |
| [4, 1, 5, 3, None, 8, 7, 6, 2] | 15 | 274 | 15 | 53 | 15 | 65 |
| [None, 3, 6, 4, 2, 7, 1, 8, 5] | 21 | 3430 | 21 | 640 | 21 | 738 |
| [2, 4, 5, 3, 1, 6, 7, 8, None] | 17 | 556 | 17 | 134 | 17 | 154 |
| [2, 8, 1, 3, 6, 5, 4, 7, None] | 21 | 3264 | 21 | 401 | 21 | 500 |
| [4, 2, 6, 8, None, 7, 3, 5, 1] | 23 | 9632 | 23 | 827 | 23 | 1242 |
| [1, 2, 3, 8, None, 4, 6, 7, 5] | 13 | 139 | 13 | 43 | 13 | 44 |
| [1, 2, 3, 8, 7, 5, 4, 6, None] | 9 | 22 | 9 | 11 | 9 | 12 |
| [4, 5, 1, 6, 8, 2, None, 7, 3] | 17 | 513 | 17 | 88 | 17 | 98 |
| [4, 2, 6, 3, 5, 7, None, 1, 8] | 23 | 8089 | 23 | 1238 | 23 | 1506 |
| [None, 3, 6, 1, 4, 2, 7, 5, 8] | 9 | 17 | 9 | 11 | 9 | 11 |
| [None, 5, 3, 2, 7, 4, 1, 8, 6] | 17 | 591 | 17 | 141 | 17 | 155 |
| [4, 6, 3, 1, 2, 8, 7, 5, None] | 19 | 1675 | 19 | 446 | 19 | 495 |
| [4, 1, 8, 2, 7, 5, None, 3, 6] | 21 | 3343 | 21 | 298 | 21 | 457 |
| [None, 4, 2, 1, 8, 3, 5, 6, 7] | 21 | 3727 | 21 | 721 | 21 | 846 |
| [4, 6, 3, 7, 2, 5, 1, 8, None] | 21 | 3772 | 21 | 682 | 21 | 838 |
| [4, 3, 1, 8, None, 2, 7, 6, 5] | 15 | 289 | 15 | 61 | 15 | 67 |
| [7, 4, 1, 8, 5, 6, 3, 2, None] | 23 | 7408 | 23 | 688 | 23 | 953 |
| [2, 4, None, 1, 5, 8, 7, 6, 3] | 17 | 573 | 17 | 127 | 17 | 146 |
| [5, 8, 4, 1, None, 6, 7, 3, 2] | 19 | 1706 | 19 | 182 | 19 | 260 |
| [None, 1, 3, 6, 2, 8, 5, 4, 7] | 13 | 90 | 13 | 18 | 13 | 19 |
| [1, 8, 2, 4, 6, 3, 7, 5, None] | 11 | 50 | 11 | 24 | 11 | 24 |
| [1, 2, 3, 5, None, 6, 4, 7, 8] | 5 | 5 | 5 | 5 | 5 | 5 |
| [3, 8, None, 1, 6, 5, 4, 2, 7] | 19 | 1296 | 19 | 176 | 19 | 210 |
| [None, 1, 5, 3, 2, 8, 4, 7, 6] | 17 | 604 | 17 | 147 | 17 | 170 |
| [1, 5, 4, 7, 8, 2, 6, 3, None] | 19 | 1282 | 19 | 154 | 19 | 204 |
| [1, 3, 6, 4, 8, 5, None, 7, 2] | 13 | 117 | 13 | 48 | 13 | 50 |
| [4, 3, 1, 7, None, 2, 8, 5, 6] | 13 | 108 | 13 | 43 | 13 | 43 |
| [7, 5, 1, 8, 2, 3, 4, 6, None] | 19 | 1295 | 19 | 201 | 19 | 224 |
| [None, 5, 6, 2, 7, 3, 4, 1, 8] | 21 | 3721 | 21 | 750 | 21 | 882 |
| [6, 4, 2, 7, 5, 3, 8, 1, None] | 21 | 3372 | 21 | 376 | 21 | 525 |
| [5, 4, 2, 1, 6, 8, None, 7, 3] | 15 | 211 | 15 | 47 | 15 | 52 |
| [1, 4, 3, 5, None, 7, 8, 2, 6] | 19 | 1969 | 19 | 398 | 19 | 482 |
| [5, 8, 4, 1, 6, 2, None, 7, 3] | 17 | 475 | 17 | 29 | 17 | 44 |
| [4, 8, 1, 7, 3, 5, None, 6, 2] | 19 | 1285 | 19 | 105 | 19 | 146 |
| [2, 8, 3, 1, 7, 6, 5, 4, None] | 17 | 632 | 17 | 132 | 17 | 152 |
| [1, 2, None, 4, 6, 8, 7, 3, 5] | 13 | 112 | 13 | 42 | 13 | 48 |
| [1, 6, 3, 7, 4, 5, None, 2, 8] | 15 | 278 | 15 | 101 | 15 | 106 |
| [2, 5, 3, 8, 1, 6, 4, 7, None] | 19 | 1561 | 19 | 397 | 19 | 448 |
| [None, 5, 6, 4, 3, 1, 7, 2, 8] | 17 | 592 | 17 | 111 | 17 | 138 |
| [3, 5, 6, 1, None, 2, 4, 7, 8] | 13 | 115 | 13 | 42 | 13 | 42 |
| [None, 3, 1, 7, 4, 2, 8, 6, 5] | 19 | 1322 | 19 | 251 | 19 | 288 |
| [1, 5, 2, 4, 6, 7, None, 3, 8] | 19 | 1402 | 19 | 213 | 19 | 285 |
| [None, 1, 8, 4, 6, 2, 7, 5, 3] | 15 | 237 | 15 | 58 | 15 | 62 |
| [5, 3, 7, 8, None, 2, 4, 1, 6] | 21 | 4226 | 21 | 370 | 21 | 567 |
| [2, 4, None, 5, 1, 3, 8, 7, 6] | 19 | 1553 | 19 | 340 | 19 | 408 |
| [3, 2, 5, 4, None, 6, 1, 8, 7] | 21 | 4175 | 21 | 727 | 21 | 814 |

## 

### 15\_Puzzle

| initial\_state | h1\_steps | h1\_expanded | h2\_steps | h2\_expanded | h3\_steps | h3\_expanded |
| --- | --- | --- | --- | --- | --- | --- |
| [3, 4, 7, 11, 5, 2, 12, 8, 1, 6, None, 15, 9, 13, 14, 10] | TIMEOUT | TIMEOUT | 33 | 19067 | 33 | 44456 |
| [10, 9, 2, 3, 7, 1, 6, 4, 13, 5, 11, 8, 14, None, 15, 12] | 25 | 18490 | 25 | 372 | 25 | 662 |
| [1, 8, 7, 3, 5, 2, 15, None, 10, 6, 12, 4, 9, 13, 11, 14] | 25 | 26079 | 25 | 826 | 25 | 1148 |
| [1, 6, None, 8, 5, 7, 2, 3, 9, 14, 12, 10, 13, 11, 15, 4] | 25 | 51560 | 25 | 1340 | 25 | 1962 |
| [13, 1, 2, 3, 5, 7, 14, 4, 6, 11, None, 8, 10, 9, 15, 12] | 25 | 23249 | 25 | 317 | 25 | 655 |
| [1, 3, None, 8, 5, 15, 11, 4, 13, 2, 6, 7, 10, 9, 14, 12] | 25 | 23420 | 25 | 367 | 25 | 801 |
| [2, 6, 3, 4, 5, 1, 13, 8, 10, 7, None, 11, 14, 9, 15, 12] | 21 | 4154 | 21 | 178 | 21 | 367 |
| [5, 7, None, 4, 9, 1, 3, 8, 6, 11, 2, 14, 13, 10, 15, 12] | 23 | 11563 | 23 | 421 | 23 | 846 |
| [2, 4, None, 3, 1, 6, 15, 8, 5, 14, 12, 7, 13, 9, 10, 11] | TIMEOUT | TIMEOUT | 29 | 6207 | 29 | 12774 |
| [1, 3, 7, 4, 5, 2, 14, 8, 6, 9, 11, 12, 13, None, 10, 15] | 23 | 21507 | 23 | 2166 | 23 | 2985 |
| [4, 7, 8, 3, 1, 5, 2, 12, 9, 11, 6, 15, 13, 14, 10, None] | TIMEOUT | TIMEOUT | 31 | 11144 | 31 | 21917 |
| [1, 13, 2, 3, 6, None, 7, 4, 5, 9, 14, 8, 10, 11, 12, 15] | 27 | 112165 | 27 | 1782 | 27 | 4302 |
| [None, 6, 3, 4, 5, 1, 8, 12, 13, 2, 11, 7, 10, 9, 14, 15] | 23 | 5476 | 23 | 432 | 23 | 602 |
| [6, 7, 8, 3, 2, 9, 4, None, 1, 10, 5, 11, 13, 14, 15, 12] | 27 | 95436 | 27 | 2678 | 27 | 4436 |
| [1, 6, None, 8, 5, 11, 4, 2, 9, 7, 3, 12, 13, 10, 14, 15] | 17 | 347 | 17 | 40 | 17 | 46 |
| [6, 1, 2, 3, 5, 10, 8, 4, 9, 14, 15, 11, 13, 7, 12, None] | 27 | 109670 | 27 | 5123 | 27 | 8357 |
| [9, 5, 2, 4, 6, 3, 11, None, 1, 8, 12, 7, 13, 10, 14, 15] | TIMEOUT | TIMEOUT | 29 | 6285 | 29 | 10341 |
| [2, 4, None, 3, 1, 9, 6, 11, 5, 7, 8, 14, 13, 10, 15, 12] | TIMEOUT | TIMEOUT | 31 | 21616 | 31 | 44163 |
| [1, 3, 4, 8, 2, 6, 7, 12, 5, 13, 14, 11, 9, None, 10, 15] | 19 | 477 | 19 | 110 | 19 | 125 |
| [3, 5, None, 4, 6, 2, 7, 8, 9, 10, 1, 12, 13, 14, 11, 15] | 25 | 60211 | 25 | 2943 | 25 | 4780 |
| [1, 2, 3, 8, 5, 6, 12, 4, 9, 11, None, 15, 13, 10, 7, 14] | 23 | 27252 | 23 | 2567 | 23 | 3191 |
| [2, 6, 5, 4, 9, 1, 3, 7, None, 14, 11, 10, 13, 15, 12, 8] | 23 | 6664 | 23 | 201 | 23 | 303 |
| [1, 2, 3, 4, 5, 6, 11, 7, 13, 9, 12, 14, 15, None, 10, 8] | 19 | 1857 | 19 | 150 | 19 | 174 |
| [1, 2, None, 4, 6, 9, 7, 12, 10, 13, 3, 11, 5, 14, 15, 8] | TIMEOUT | TIMEOUT | 31 | 34692 | 31 | 55187 |
| [None, 8, 2, 6, 4, 9, 10, 3, 1, 5, 15, 7, 13, 14, 12, 11] | TIMEOUT | TIMEOUT | 37 | 42740 | 37 | 140291 |
| [1, 2, 3, 4, 13, 10, 12, None, 14, 9, 7, 6, 5, 8, 11, 15] | TIMEOUT | TIMEOUT | 31 | 10251 | 31 | 26369 |
| [2, 6, 1, 4, 9, 10, 5, 3, 13, 7, 11, 8, 14, None, 15, 12] | TIMEOUT | TIMEOUT | 29 | 16916 | 29 | 25081 |
| [5, 1, 4, 8, 2, 6, 7, 3, 9, 10, 11, 12, 13, 14, 15, None] | 17 | 363 | 17 | 150 | 17 | 155 |
| [1, 15, 2, 3, 9, 6, 7, 8, 5, 14, 11, 4, 13, None, 10, 12] | TIMEOUT | TIMEOUT | 33 | 108934 | TIMEOUT | TIMEOUT |
| [2, 1, 6, 4, 13, None, 10, 3, 14, 12, 7, 9, 15, 5, 11, 8] | TIMEOUT | TIMEOUT | 39 | 139881 | TIMEOUT | TIMEOUT |
| [1, 3, 4, 6, 10, None, 2, 5, 14, 15, 7, 8, 13, 9, 11, 12] | TIMEOUT | TIMEOUT | 31 | 7342 | 33 | 71558 |
| [5, 2, 7, 3, 9, 6, 11, 4, 1, 8, None, 12, 13, 10, 14, 15] | 23 | 11731 | 23 | 1134 | 23 | 1312 |
| [6, 11, 1, 12, 2, None, 4, 15, 13, 8, 9, 7, 10, 5, 3, 14] | TIMEOUT | TIMEOUT | 43 | 176430 | TIMEOUT | TIMEOUT |
| [2, 10, 8, 3, 5, 1, 9, 4, None, 11, 6, 12, 13, 7, 14, 15] | 25 | 22358 | 25 | 311 | 25 | 635 |
| [None, 2, 8, 3, 5, 6, 1, 4, 9, 10, 7, 11, 14, 15, 13, 12] | TIMEOUT | TIMEOUT | 29 | 10592 | 29 | 15607 |
| [2, 3, 6, 4, 10, 9, 5, None, 1, 7, 11, 8, 13, 14, 15, 12] | 19 | 906 | 19 | 36 | 19 | 49 |
| [9, 4, 5, 11, 2, None, 1, 3, 14, 10, 8, 7, 6, 13, 15, 12] | TIMEOUT | TIMEOUT | 31 | 592 | 31 | 2850 |
| [5, 1, 7, 6, 3, 11, 2, 4, 13, 14, None, 8, 10, 9, 15, 12] | TIMEOUT | TIMEOUT | 29 | 1795 | 29 | 5982 |
| [5, 1, 7, 2, 9, 3, 10, 4, 6, 14, 8, 15, 13, 11, 12, None] | 25 | 13456 | 25 | 325 | 25 | 567 |
| [1, 11, None, 3, 5, 15, 2, 4, 9, 6, 10, 7, 13, 14, 8, 12] | 25 | 64024 | 25 | 1701 | 25 | 3129 |
| [3, 11, None, 4, 6, 2, 1, 8, 9, 7, 14, 15, 10, 5, 12, 13] | TIMEOUT | TIMEOUT | 35 | 12335 | 35 | 43615 |
| [None, 10, 4, 8, 2, 13, 6, 7, 1, 5, 15, 3, 9, 14, 11, 12] | TIMEOUT | TIMEOUT | 35 | 70662 | 35 | 152968 |
| [5, 1, 3, 4, 2, None, 7, 9, 13, 10, 11, 12, 14, 15, 6, 8] | TIMEOUT | TIMEOUT | 33 | 69311 | 33 | 141044 |
| [2, 1, 4, 8, 6, None, 3, 12, 7, 5, 10, 11, 9, 13, 14, 15] | 25 | 24799 | 25 | 1821 | 25 | 2296 |
| [1, 3, None, 4, 5, 2, 7, 15, 9, 14, 8, 10, 13, 12, 6, 11] | 21 | 4901 | 21 | 67 | 21 | 197 |
| [1, 2, 6, 4, 5, 12, 11, 8, None, 9, 7, 15, 13, 3, 10, 14] | TIMEOUT | TIMEOUT | 29 | 10896 | 29 | 21873 |
| [2, 6, 4, 8, 1, 9, 3, 12, None, 13, 11, 7, 14, 10, 5, 15] | 25 | 15134 | 25 | 335 | 25 | 696 |
| [1, 14, 6, 8, 5, None, 3, 10, 9, 7, 4, 2, 13, 15, 12, 11] | TIMEOUT | TIMEOUT | 29 | 2083 | 29 | 5975 |
| [7, 2, 3, 4, 9, 5, 6, 8, 10, 11, None, 1, 13, 14, 15, 12] | TIMEOUT | TIMEOUT | 29 | 16242 | 29 | 36697 |
| [1, 5, 2, 4, 9, 6, 3, 10, 13, 12, 7, 11, 15, None, 14, 8] | TIMEOUT | TIMEOUT | 29 | 5962 | 29 | 10529 |
| [6, 1, 3, 4, 10, None, 7, 8, 2, 13, 15, 11, 5, 9, 14, 12] | 19 | 715 | 19 | 44 | 19 | 60 |
| [2, 7, 10, 3, 1, 9, 4, 8, 5, 6, None, 12, 13, 14, 11, 15] | 23 | 12422 | 23 | 932 | 23 | 1295 |
| [10, 8, 4, 3, 13, 1, 5, None, 9, 15, 6, 2, 14, 7, 12, 11] | TIMEOUT | TIMEOUT | 39 | 35381 | TIMEOUT | TIMEOUT |
| [1, 2, 3, 4, 6, 8, 11, 12, 5, 9, 15, 14, 13, 10, 7, None] | 19 | 1456 | 19 | 131 | 19 | 149 |
| [1, 2, 3, 4, 9, 6, 10, None, 11, 14, 8, 7, 13, 12, 5, 15] | 25 | 54652 | 25 | 1259 | 25 | 2211 |
| [5, 1, 2, 3, 10, 7, 14, 6, 15, 8, 4, 11, 9, 13, 12, None] | TIMEOUT | TIMEOUT | 31 | 962 | 31 | 3417 |
| [2, 4, 3, 12, 1, 9, 8, None, 6, 10, 15, 11, 5, 13, 7, 14] | TIMEOUT | TIMEOUT | 35 | 72894 | 35 | 132285 |
| [2, 1, 10, 4, 3, None, 15, 6, 5, 12, 9, 8, 13, 14, 7, 11] | TIMEOUT | TIMEOUT | 33 | 15550 | 33 | 44940 |
| [5, 1, 2, 3, 9, 6, 7, 4, None, 13, 10, 8, 14, 15, 11, 12] | 15 | 33 | 15 | 23 | 15 | 23 |
| [2, 5, 3, 4, 1, 6, 7, 8, 13, 9, 15, 11, 10, None, 14, 12] | 15 | 108 | 15 | 40 | 15 | 45 |
| [1, 6, 2, 4, 5, None, 10, 8, 7, 11, 3, 12, 9, 13, 14, 15] | 17 | 469 | 17 | 73 | 17 | 83 |
| [1, 6, 4, 2, 10, 3, 11, 8, 5, 13, None, 12, 9, 7, 14, 15] | TIMEOUT | TIMEOUT | 31 | 29652 | 31 | 52615 |
| [6, 5, 3, 7, 9, 1, 4, 15, 13, 12, 2, 8, 14, None, 11, 10] | TIMEOUT | TIMEOUT | 37 | 69566 | TIMEOUT | TIMEOUT |
| [1, 9, 4, 11, 6, None, 7, 3, 5, 14, 2, 15, 13, 10, 12, 8] | TIMEOUT | TIMEOUT | 33 | 18704 | 33 | 59812 |
| [3, 2, 4, 8, 1, None, 14, 6, 9, 5, 11, 12, 10, 13, 15, 7] | TIMEOUT | TIMEOUT | 29 | 4715 | 29 | 10300 |
| [1, 7, 2, 4, 6, 3, 11, 8, 5, 13, None, 14, 9, 12, 15, 10] | 25 | 44914 | 25 | 610 | 25 | 1440 |
| [1, 3, 4, 8, 9, 10, 14, None, 2, 5, 12, 15, 13, 6, 11, 7] | TIMEOUT | TIMEOUT | 31 | 4885 | 31 | 15383 |
| [3, 4, 12, 6, 2, 5, 1, None, 9, 14, 10, 8, 13, 11, 15, 7] | TIMEOUT | TIMEOUT | 31 | 1955 | 31 | 6244 |
| [9, 1, None, 7, 2, 6, 4, 3, 10, 5, 11, 8, 13, 14, 15, 12] | 19 | 684 | 19 | 89 | 19 | 105 |
| [1, 2, 4, 8, 5, 11, 12, 6, 9, 7, None, 15, 13, 10, 3, 14] | 21 | 5802 | 21 | 158 | 21 | 243 |
| [2, 3, None, 7, 5, 6, 10, 4, 14, 1, 15, 8, 9, 13, 12, 11] | 25 | 17996 | 25 | 396 | 25 | 979 |
| [2, 6, 4, 11, 1, 10, 7, 9, None, 13, 8, 3, 14, 5, 15, 12] | TIMEOUT | TIMEOUT | 33 | 9290 | 33 | 24814 |
| [1, 2, 3, 4, 14, 7, 6, None, 9, 10, 5, 12, 13, 8, 11, 15] | TIMEOUT | TIMEOUT | 31 | 34765 | 31 | 78533 |
| [1, 2, 7, 5, 9, 12, 3, 4, 10, 14, 15, 6, 13, None, 8, 11] | TIMEOUT | TIMEOUT | 35 | 47106 | 35 | 155430 |
| [1, 2, 6, 4, 5, 13, 7, 3, 9, 11, 12, 8, 10, None, 14, 15] | TIMEOUT | TIMEOUT | 29 | 23218 | 29 | 43596 |
| [2, 8, 3, 4, 5, None, 12, 10, 7, 1, 6, 15, 14, 9, 13, 11] | TIMEOUT | TIMEOUT | 31 | 2028 | 31 | 8115 |
| [5, 2, 8, 3, 6, 1, 14, 4, 9, 13, 10, 11, 7, None, 15, 12] | TIMEOUT | TIMEOUT | 31 | 11402 | 31 | 28589 |
| [1, 2, 3, 12, 9, 5, 4, 8, 10, 7, None, 6, 13, 14, 11, 15] | 23 | 22330 | 23 | 1280 | 23 | 1816 |
| [5, 1, None, 3, 2, 9, 10, 4, 13, 7, 12, 6, 14, 11, 8, 15] | 29 | 199703 | 29 | 1347 | 29 | 3384 |
| [5, 1, 4, 8, 3, 7, 10, None, 14, 12, 13, 6, 9, 2, 11, 15] | TIMEOUT | TIMEOUT | 33 | 1510 | 33 | 7614 |
| [1, 6, 8, 2, 9, 7, 4, None, 10, 5, 11, 3, 13, 14, 15, 12] | 23 | 13121 | 23 | 908 | 23 | 1259 |
| [1, 7, None, 8, 5, 6, 4, 2, 9, 13, 3, 12, 10, 11, 14, 15] | 27 | 115309 | 27 | 2840 | 27 | 5912 |
| [2, 3, None, 4, 1, 5, 7, 15, 6, 13, 12, 10, 9, 8, 14, 11] | TIMEOUT | TIMEOUT | 33 | 17442 | 33 | 54588 |
| [2, 12, 3, 4, 1, 14, 6, 7, 5, 13, 11, 8, 9, 10, 15, None] | TIMEOUT | TIMEOUT | 29 | 6118 | 29 | 11706 |
| [2, 3, None, 4, 1, 7, 11, 8, 5, 9, 6, 12, 13, 10, 14, 15] | 13 | 24 | 13 | 16 | 13 | 16 |
| [1, 2, 3, 4, 5, 6, 11, 7, 13, 8, None, 12, 14, 9, 15, 10] | 17 | 922 | 17 | 105 | 17 | 140 |
| [1, 2, 15, 3, 7, None, 4, 8, 12, 5, 14, 11, 6, 9, 10, 13] | TIMEOUT | TIMEOUT | 37 | 38841 | 37 | 108033 |
| [5, 1, 4, 7, 9, 6, 3, 14, 13, 11, 12, 8, 10, None, 15, 2] | TIMEOUT | TIMEOUT | 35 | 31126 | 35 | 114375 |
| [5, 1, 2, 4, 6, 7, 10, 8, 13, 9, 3, 11, 14, None, 15, 12] | 17 | 208 | 17 | 43 | 17 | 43 |
| [1, 6, 2, 4, 5, 14, 7, None, 11, 10, 8, 15, 12, 13, 9, 3] | TIMEOUT | TIMEOUT | 37 | 54010 | 37 | 181232 |
| [1, 2, 4, 8, 5, 6, 3, 12, 11, 7, None, 15, 9, 13, 10, 14] | 17 | 250 | 17 | 29 | 17 | 29 |
| [1, 7, 10, 2, 5, 12, 6, 8, None, 13, 4, 3, 14, 9, 11, 15] | TIMEOUT | TIMEOUT | 31 | 4386 | 31 | 10445 |
| [None, 1, 2, 3, 6, 10, 7, 4, 9, 5, 13, 8, 14, 11, 15, 12] | 23 | 6724 | 23 | 477 | 23 | 703 |
| [6, 1, 4, 15, 10, None, 2, 3, 9, 12, 13, 7, 5, 14, 8, 11] | TIMEOUT | TIMEOUT | 39 | 85327 | TIMEOUT | TIMEOUT |
| [5, 1, 2, 4, 7, None, 8, 14, 9, 6, 15, 3, 11, 10, 13, 12] | TIMEOUT | TIMEOUT | 31 | 5257 | 31 | 14745 |
| [2, 6, 4, 7, 9, 14, 10, 3, 1, 13, 5, 8, 15, 12, 11, None] | TIMEOUT | TIMEOUT | 35 | 9010 | 35 | 26894 |
| [5, 4, 2, 12, 13, 8, 7, 1, None, 6, 3, 15, 14, 11, 9, 10] | TIMEOUT | TIMEOUT | 41 | 73814 | TIMEOUT | TIMEOUT |
| [1, 3, None, 4, 10, 2, 9, 8, 6, 5, 7, 11, 13, 14, 15, 12] | 17 | 597 | 17 | 99 | 17 | 129 |
| [1, 14, 4, 8, 5, None, 3, 11, 10, 6, 7, 12, 9, 2, 13, 15] | TIMEOUT | TIMEOUT | 31 | 17286 | 31 | 27884 |
| [1, 2, 12, 3, 6, 9, 7, 4, 13, 5, None, 8, 11, 14, 10, 15] | TIMEOUT | TIMEOUT | 29 | 6723 | 29 | 15205 |

## 

### 24\_Puzzle

| initial\_state | h1\_steps | h1\_expanded | h2\_steps | h2\_expanded | h3\_steps | h3\_expanded |
| --- | --- | --- | --- | --- | --- | --- |
| [6, 1, 4, 5, 10, 7, 12, 3, 8, 15, None, 11, 2, 14, 9, 16, 17, 13, 18, 20, 21, 22, 23, 19, 24] | 25 | 3984 | 25 | 316 | 25 | 398 |
| [1, 2, 3, 4, 5, 6, 12, 7, 8, 10, 17, 16, 13, 18, 15, 21, 11, 23, 9, 19, None, 22, 24, 14, 20] | 23 | 2275 | 23 | 154 | 23 | 174 |
| [1, 2, None, 4, 5, 6, 8, 3, 9, 10, 11, 7, 13, 14, 15, 16, 12, 18, 19, 20, 21, 17, 22, 23, 24] | 9 | 9 | 9 | 9 | 9 | 9 |
| [2, 8, 3, 4, 5, 1, 7, 13, 19, 9, 6, 17, 12, 24, 10, 11, 16, 22, None, 14, 21, 23, 18, 20, 15] | 31 | 107671 | 31 | 400 | 31 | 1139 |
| [1, 2, 3, 4, 5, 11, 6, 8, 9, 10, 16, 7, 12, 24, 15, 22, 21, 14, 13, 19, None, 17, 18, 23, 20] | 21 | 551 | 21 | 44 | 21 | 49 |
| [6, 1, 3, 4, 5, 16, 2, 7, 9, 10, 17, 11, 8, 14, 15, 21, 22, 19, 23, 20, 18, 13, None, 12, 24] | 27 | 18041 | 27 | 50 | 27 | 153 |
| [1, 7, 2, 4, 5, 6, 12, 3, 8, 10, 21, 11, 19, 18, 14, 22, 16, 9, 13, 15, None, 23, 17, 24, 20] | 29 | 35168 | 29 | 299 | 29 | 688 |
| [1, 8, 3, 10, 4, 6, 2, 7, None, 5, 17, 12, 13, 14, 15, 11, 16, 9, 18, 19, 21, 22, 23, 24, 20] | 27 | 37629 | 27 | 1578 | 27 | 2848 |
| [2, 7, 3, 4, 5, 6, 1, 8, 9, 10, 18, 16, 13, 14, 15, 11, 12, 23, 17, 20, 21, 22, None, 19, 24] | 25 | 25367 | 25 | 1215 | 25 | 1867 |
| [1, 14, 4, 13, 5, 7, 3, 2, None, 9, 6, 11, 12, 8, 10, 16, 17, 18, 19, 15, 21, 22, 23, 24, 20] | 23 | 4244 | 23 | 84 | 23 | 163 |
| [None, 3, 2, 5, 10, 1, 7, 4, 9, 15, 6, 12, 8, 14, 19, 11, 22, 17, 13, 18, 16, 21, 23, 24, 20] | 27 | 3667 | 27 | 275 | 27 | 345 |
| [2, 6, 3, 4, 5, 1, 12, 7, 8, 10, 11, 13, 19, 9, 15, 16, None, 17, 14, 20, 21, 22, 18, 23, 24] | 17 | 79 | 17 | 28 | 17 | 28 |
| [6, 1, 2, 3, 4, 8, None, 12, 9, 5, 11, 7, 14, 20, 10, 16, 18, 13, 15, 19, 21, 17, 22, 23, 24] | 25 | 772 | 25 | 50 | 25 | 64 |
| [1, 2, 3, 4, 9, 6, 7, 8, 10, 5, 11, 17, None, 14, 15, 16, 13, 12, 19, 20, 21, 22, 18, 23, 24] | 19 | 1406 | 19 | 319 | 19 | 372 |
| [2, 3, 10, 8, 5, 1, 7, 12, 4, 13, 6, 11, None, 19, 9, 16, 14, 18, 24, 15, 21, 17, 22, 23, 20] | TIMEOUT | TIMEOUT | 33 | 1621 | 33 | 5127 |
| [1, 7, 2, 4, 5, 6, 3, 8, 9, 10, 11, 13, None, 18, 15, 16, 12, 14, 17, 20, 21, 22, 23, 19, 24] | 17 | 396 | 17 | 86 | 17 | 88 |
| [1, 2, 3, 5, 10, 6, 7, 9, 15, 4, 11, 12, 8, 20, None, 16, 17, 14, 19, 24, 21, 22, 13, 18, 23] | 21 | 1739 | 21 | 35 | 21 | 64 |
| [1, 7, 2, 4, 5, 6, 3, 14, 8, 10, 16, 13, 12, 9, 18, 21, 11, 24, None, 15, 22, 23, 17, 20, 19] | 31 | 165250 | 31 | 291 | 31 | 1919 |
| [1, 2, None, 3, 5, 6, 7, 13, 4, 9, 11, 12, 14, 8, 20, 16, 17, 18, 15, 10, 21, 22, 23, 19, 24] | 17 | 326 | 17 | 71 | 17 | 72 |
| [7, 6, 2, 3, 5, 11, None, 8, 4, 10, 12, 1, 13, 9, 14, 21, 17, 16, 20, 15, 22, 23, 18, 19, 24] | 27 | 3845 | 27 | 222 | 27 | 280 |
| [7, 12, 2, 3, 5, 1, 11, 8, 4, 10, None, 6, 13, 9, 15, 16, 17, 18, 14, 19, 21, 22, 23, 24, 20] | 17 | 65 | 17 | 24 | 17 | 24 |
| [1, 7, 2, 4, 5, 6, 9, 12, 14, 10, 11, 3, 8, 15, 19, 16, 17, 23, 13, 20, 21, 22, None, 18, 24] | 25 | 20656 | 25 | 279 | 25 | 668 |
| [1, 2, 3, 10, 4, 6, None, 8, 9, 5, 11, 7, 13, 14, 15, 16, 12, 24, 23, 19, 21, 17, 22, 18, 20] | 21 | 1385 | 21 | 186 | 21 | 264 |
| [1, 7, 2, 5, 10, 18, None, 8, 9, 3, 6, 11, 4, 13, 15, 16, 12, 17, 14, 20, 21, 22, 23, 19, 24] | 27 | 35326 | 27 | 425 | 27 | 922 |
| [1, 2, 3, 4, 5, 6, 7, 8, None, 19, 11, 12, 14, 10, 9, 17, 18, 13, 20, 15, 16, 21, 22, 23, 24] | 19 | 295 | 19 | 55 | 19 | 55 |
| [1, 3, 8, 4, 5, 6, 2, 9, None, 10, 16, 11, 7, 14, 19, 12, 13, 24, 15, 20, 21, 22, 17, 18, 23] | 29 | 121046 | 29 | 496 | 29 | 2256 |
| [1, 2, 3, 4, 5, 12, 22, 6, 9, 10, 13, 21, 8, 14, 15, 7, None, 11, 18, 19, 17, 16, 23, 24, 20] | TIMEOUT | TIMEOUT | 35 | 33029 | 35 | 96329 |
| [2, 3, 13, 4, 9, 1, 12, 6, 10, 5, 21, 11, 7, 14, 15, 17, None, 8, 24, 19, 22, 16, 18, 23, 20] | TIMEOUT | TIMEOUT | 37 | 14183 | 37 | 30780 |
| [1, 2, 3, 4, 5, 11, 6, 13, 8, 10, 16, 7, 9, 15, None, 12, 14, 18, 19, 20, 21, 17, 22, 23, 24] | 23 | 2684 | 23 | 278 | 23 | 349 |
| [1, 7, 2, 4, 5, 11, 6, 13, 8, 10, 3, 12, 18, 9, 14, 16, None, 17, 20, 15, 21, 22, 23, 19, 24] | 21 | 782 | 21 | 75 | 21 | 83 |
| [11, 1, 8, 4, 5, 7, 6, 2, 9, 10, 13, 3, 14, 15, None, 16, 12, 17, 24, 19, 21, 22, 18, 23, 20] | 27 | 15712 | 27 | 312 | 27 | 379 |
| [2, 3, 9, 4, 10, 1, None, 7, 5, 15, 6, 12, 8, 13, 14, 11, 16, 17, 23, 19, 21, 22, 24, 18, 20] | 27 | 4635 | 27 | 227 | 27 | 427 |
| [2, 3, 7, 4, 5, 1, 6, 19, 8, 9, 11, 17, 14, 18, None, 16, 13, 12, 24, 10, 21, 22, 23, 20, 15] | 29 | 53468 | 29 | 312 | 29 | 791 |
| [2, 6, 3, 4, 5, 1, 7, 8, 9, 10, None, 16, 13, 14, 15, 12, 11, 22, 17, 20, 21, 23, 24, 19, 18] | 27 | 64578 | 27 | 1813 | 27 | 3235 |
| [1, 2, 3, 4, 5, 13, 6, 8, 9, 10, 7, 12, 18, 15, None, 16, 22, 19, 24, 20, 21, 11, 17, 14, 23] | TIMEOUT | TIMEOUT | 33 | 16062 | 33 | 49172 |
| [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 12, 17, 13, 14, 15, 16, 11, 18, 19, 20, 21, 22, None, 23, 24] | 13 | 137 | 13 | 71 | 13 | 71 |
| [2, 3, 4, 15, 5, 1, 6, 8, 14, 9, 22, 7, None, 18, 10, 12, 11, 17, 19, 13, 16, 21, 23, 24, 20] | TIMEOUT | TIMEOUT | 37 | 23525 | 37 | 85101 |
| [1, 2, None, 3, 5, 6, 12, 7, 4, 10, 11, 18, 22, 8, 15, 17, 21, 14, 9, 23, 16, 20, 24, 13, 19] | TIMEOUT | TIMEOUT | 37 | 3998 | 37 | 29062 |
| [1, 2, 4, 5, 10, 6, 7, 3, 9, 19, 11, 20, 8, 22, None, 16, 14, 13, 12, 18, 21, 17, 23, 24, 15] | TIMEOUT | TIMEOUT | 35 | 4828 | 35 | 19263 |
| [1, 2, 3, 5, 9, 6, 7, 8, 10, 14, 12, 13, 17, 4, None, 11, 16, 18, 20, 15, 21, 22, 23, 19, 24] | 23 | 3042 | 23 | 273 | 23 | 349 |
| [2, 9, 6, 4, 5, 1, 7, 3, 13, 10, None, 11, 12, 8, 15, 17, 18, 19, 14, 20, 16, 21, 22, 23, 24] | 27 | 10829 | 27 | 160 | 27 | 325 |
| [1, 7, 2, 4, 5, 11, 6, 3, 8, 10, 16, 13, 14, 9, 15, 21, 12, 17, 18, 19, None, 22, 23, 24, 20] | 17 | 17 | 17 | 17 | 17 | 17 |
| [1, 2, 3, 10, 4, 6, 7, 8, 9, 5, 11, 13, 18, 14, 15, 16, 12, 19, 20, 24, 21, 17, 22, 23, None] | 19 | 440 | 19 | 118 | 19 | 123 |
| [2, 7, 3, 4, 5, 1, 11, 8, 9, 10, None, 6, 13, 14, 15, 16, 12, 17, 23, 19, 21, 22, 24, 18, 20] | 17 | 108 | 17 | 30 | 17 | 33 |
| [6, 2, 8, 3, 5, 11, 12, 1, 4, 10, 16, 13, 7, 9, 14, 17, None, 18, 19, 15, 21, 22, 23, 24, 20] | 25 | 11623 | 25 | 1004 | 25 | 1420 |
| [None, 2, 3, 4, 5, 1, 6, 11, 7, 9, 13, 12, 14, 8, 10, 22, 17, 18, 19, 15, 16, 21, 23, 24, 20] | 29 | 81908 | 29 | 2363 | 29 | 4082 |
| [1, 2, 3, 4, 5, 6, 12, 7, 9, 10, 11, 8, 19, 23, 13, 16, 17, 24, 15, 14, 21, 22, None, 18, 20] | 23 | 11466 | 23 | 322 | 23 | 561 |
| [None, 2, 3, 8, 5, 1, 6, 4, 15, 10, 12, 7, 13, 18, 19, 11, 21, 17, 23, 9, 22, 16, 24, 14, 20] | TIMEOUT | TIMEOUT | 37 | 7787 | 37 | 47086 |
| [7, 2, 3, 4, 5, 1, 6, 9, None, 10, 11, 13, 8, 14, 15, 16, 12, 18, 19, 20, 21, 17, 22, 23, 24] | 17 | 217 | 17 | 101 | 17 | 101 |
| [1, 7, 2, 4, 5, 17, 6, 14, 13, 9, 12, 8, 3, 19, 10, 11, 22, 18, None, 15, 16, 21, 23, 24, 20] | 27 | 6339 | 27 | 168 | 27 | 247 |
| [1, 3, 7, 10, 4, 11, 6, 2, 8, 15, 16, 13, None, 5, 9, 21, 17, 12, 14, 20, 22, 18, 23, 19, 24] | TIMEOUT | TIMEOUT | 33 | 1706 | 33 | 7071 |
| [1, 7, 3, 4, 5, 6, 11, 9, 18, 10, None, 2, 13, 8, 15, 21, 12, 19, 14, 20, 17, 16, 22, 23, 24] | TIMEOUT | TIMEOUT | 33 | 15458 | 33 | 39378 |
| [11, 3, None, 4, 5, 1, 6, 12, 9, 10, 16, 2, 8, 14, 15, 17, 13, 7, 19, 20, 21, 22, 18, 23, 24] | 27 | 51280 | 27 | 1093 | 27 | 1849 |
| [1, 7, 2, 3, 5, 6, 12, 8, 4, 10, None, 17, 13, 9, 14, 11, 22, 16, 20, 15, 23, 21, 18, 19, 24] | 25 | 2367 | 25 | 188 | 25 | 190 |
| [1, 7, 2, 3, 4, 12, 9, 8, 10, 5, 6, 11, 13, 14, 15, 16, 17, 23, 18, 20, 21, 22, None, 19, 24] | 21 | 325 | 21 | 62 | 21 | 63 |
| [7, 12, 2, 3, 9, 6, 1, 8, 5, 4, None, 11, 15, 14, 10, 16, 13, 23, 18, 20, 21, 17, 22, 19, 24] | 31 | 80558 | 31 | 398 | 31 | 1410 |
| [1, 2, 4, 5, 14, 6, 7, 3, 9, 10, 18, 12, 8, 15, None, 11, 16, 17, 19, 20, 21, 22, 13, 23, 24] | TIMEOUT | TIMEOUT | 29 | 3661 | 29 | 6943 |
| [1, 3, None, 8, 4, 6, 2, 14, 9, 5, 11, 7, 18, 13, 10, 16, 12, 17, 19, 15, 21, 22, 23, 24, 20] | 19 | 288 | 19 | 86 | 19 | 90 |
| [1, 5, 2, 3, 9, 7, 14, 12, 4, 8, 6, 11, 18, 19, 10, 16, 17, 15, None, 13, 21, 22, 23, 24, 20] | TIMEOUT | TIMEOUT | 39 | 59525 | TIMEOUT | TIMEOUT |
| [7, 1, 3, 4, 5, 2, 12, 8, 9, 10, 6, 17, 19, 13, 15, 11, 18, 14, None, 20, 16, 21, 22, 23, 24] | 21 | 308 | 21 | 50 | 21 | 76 |
| [1, 2, 3, 4, None, 6, 8, 17, 10, 5, 11, 7, 12, 9, 13, 16, 22, 18, 24, 14, 21, 23, 20, 19, 15] | 27 | 38183 | 27 | 199 | 27 | 506 |
| [1, 2, 8, 3, 5, 6, 7, 13, 4, 14, None, 11, 18, 10, 9, 17, 12, 19, 24, 15, 16, 21, 22, 23, 20] | 23 | 417 | 23 | 32 | 23 | 69 |
| [1, 2, 3, 4, 5, 6, 8, 10, None, 15, 11, 7, 20, 17, 14, 16, 12, 13, 23, 9, 21, 22, 19, 18, 24] | TIMEOUT | TIMEOUT | 31 | 5355 | 31 | 19662 |
| [1, 7, 2, 4, 5, 6, 12, 3, 9, 10, 11, 8, 13, 15, 20, 16, 17, 18, None, 24, 21, 22, 23, 14, 19] | 17 | 248 | 17 | 54 | 17 | 57 |
| [1, 2, 3, 4, 5, 6, 12, 7, 9, 10, 11, 8, 13, 15, 20, 21, None, 16, 19, 24, 22, 14, 18, 17, 23] | TIMEOUT | TIMEOUT | 29 | 2047 | 29 | 4833 |
| [1, 2, 8, 3, 5, 6, 12, 7, None, 9, 11, 14, 13, 4, 15, 16, 24, 22, 23, 10, 21, 17, 18, 20, 19] | TIMEOUT | TIMEOUT | 31 | 1002 | 31 | 3848 |
| [1, 2, 3, 4, 5, 6, None, 7, 8, 9, 11, 12, 13, 14, 10, 21, 17, 18, 19, 15, 22, 16, 23, 24, 20] | 15 | 102 | 15 | 58 | 15 | 58 |
| [1, 8, 2, 9, 4, 6, 7, 3, 15, 5, None, 16, 11, 14, 10, 12, 17, 13, 20, 24, 21, 22, 18, 23, 19] | 31 | 117001 | 31 | 1810 | 31 | 4861 |
| [1, 2, 3, 4, 5, 6, 9, 7, 18, 10, 17, 8, None, 12, 20, 11, 16, 14, 15, 13, 21, 22, 23, 19, 24] | 27 | 78574 | 27 | 333 | 27 | 1061 |
| [1, 2, None, 10, 4, 6, 7, 3, 9, 5, 17, 12, 8, 14, 15, 11, 16, 13, 18, 19, 21, 22, 23, 24, 20] | 19 | 439 | 19 | 127 | 19 | 135 |
| [1, 2, 3, 4, 5, 6, 7, 8, 15, 9, 12, 17, 23, 13, 10, 11, 21, 14, 18, 19, 16, 22, None, 24, 20] | 25 | 13450 | 25 | 577 | 25 | 926 |
| [6, 1, 3, 4, 5, 2, None, 7, 10, 15, 11, 12, 8, 13, 9, 16, 17, 18, 14, 20, 21, 22, 23, 19, 24] | 15 | 52 | 15 | 31 | 15 | 31 |
| [1, 2, 4, 5, 10, 6, 3, 12, None, 8, 16, 7, 13, 9, 14, 21, 11, 18, 20, 15, 22, 17, 23, 19, 24] | 27 | 7257 | 27 | 444 | 27 | 687 |
| [1, 7, 2, 5, 9, 6, 3, 13, 8, 4, 11, 12, 16, 14, 10, 21, None, 17, 24, 15, 22, 23, 19, 18, 20] | 31 | 92360 | 31 | 1670 | 31 | 3693 |
| [1, 2, 4, 8, 5, 6, 7, 13, None, 10, 11, 12, 9, 3, 19, 21, 17, 23, 15, 14, 22, 16, 24, 18, 20] | 27 | 27595 | 27 | 662 | 27 | 1539 |
| [1, 2, 3, 4, 5, 6, 7, 9, 10, 15, 11, 12, 8, 18, 14, 17, 21, 23, 13, 24, None, 16, 22, 20, 19] | 23 | 1785 | 23 | 100 | 23 | 207 |
| [1, 3, 7, 2, 5, 6, 13, 9, 4, 10, 11, 19, 18, 8, 15, 16, None, 12, 14, 20, 21, 17, 22, 23, 24] | TIMEOUT | TIMEOUT | 29 | 3740 | 29 | 6839 |
| [6, 2, 8, 3, 4, 11, 7, 1, 9, 5, None, 12, 13, 15, 10, 21, 17, 18, 14, 19, 22, 16, 23, 24, 20] | 27 | 9416 | 27 | 1064 | 27 | 1379 |
| [1, 2, 3, 4, 5, 6, 7, 8, 10, 19, 11, 13, 17, 18, None, 16, 12, 9, 14, 20, 21, 22, 23, 15, 24] | 25 | 66922 | 25 | 658 | 25 | 1975 |
| [None, 3, 8, 4, 5, 2, 7, 13, 9, 10, 1, 6, 22, 14, 15, 12, 11, 17, 19, 20, 16, 21, 18, 23, 24] | 23 | 752 | 23 | 62 | 23 | 72 |
| [1, 2, 3, 5, 9, 6, 7, 14, 4, 10, 11, 12, 8, 13, None, 16, 17, 24, 19, 15, 21, 22, 18, 23, 20] | 19 | 804 | 19 | 108 | 19 | 139 |
| [6, 1, 4, 9, 5, 16, 2, 3, None, 10, 7, 13, 8, 14, 15, 21, 11, 23, 18, 20, 17, 12, 22, 19, 24] | 25 | 988 | 25 | 39 | 25 | 43 |
| [1, 2, 8, 3, 5, 6, 7, 4, 9, 10, 11, 12, 13, 14, 15, 21, 17, 18, 24, 19, 22, 16, None, 23, 20] | 21 | 2170 | 21 | 518 | 21 | 587 |
| [1, 2, 3, 4, 5, 6, None, 7, 9, 10, 11, 12, 17, 18, 13, 16, 23, 8, 15, 24, 21, 22, 19, 14, 20] | TIMEOUT | TIMEOUT | 29 | 4080 | 29 | 10668 |
| [3, 7, 8, 4, 5, 6, 1, 2, 9, 10, 11, 12, 13, 14, 15, 16, 22, 17, None, 20, 21, 23, 19, 18, 24] | 25 | 21603 | 25 | 986 | 25 | 1556 |
| [2, 6, 4, 10, None, 11, 1, 7, 5, 8, 12, 13, 9, 3, 14, 16, 17, 18, 20, 15, 21, 22, 23, 19, 24] | 27 | 5387 | 27 | 63 | 27 | 163 |
| [1, 3, 7, 4, 5, 6, 2, 8, 15, 9, 11, 17, 12, 13, 10, 21, None, 22, 14, 24, 23, 16, 18, 20, 19] | 29 | 46214 | 29 | 529 | 29 | 1757 |
| [2, 7, 3, 4, 5, 1, 12, 8, 9, 10, 6, 17, 13, 15, 20, 11, 18, 19, 24, 23, 16, 21, 22, 14, None] | 23 | 810 | 23 | 73 | 23 | 92 |
| [1, 2, 3, 4, 5, 6, 7, 8, 14, 9, 11, 12, None, 13, 10, 16, 17, 19, 24, 15, 21, 22, 18, 20, 23] | 15 | 186 | 15 | 39 | 15 | 39 |
| [1, 7, 2, 4, 5, 6, 12, 3, 8, 9, None, 11, 13, 14, 10, 18, 22, 17, 24, 15, 16, 21, 23, 20, 19] | 27 | 15974 | 27 | 879 | 27 | 1281 |
| [1, 2, 3, 4, 5, 6, 7, 13, 8, 9, 17, 16, 12, 18, 10, 21, 11, 15, 20, 24, 19, 23, 22, 14, None] | TIMEOUT | TIMEOUT | 33 | 979 | 33 | 4498 |
| [1, 8, 3, 5, 9, 6, 2, 14, 7, 4, 11, 13, 12, 19, 10, 16, 21, 18, None, 20, 22, 17, 23, 15, 24] | TIMEOUT | TIMEOUT | 37 | 52953 | TIMEOUT | TIMEOUT |
| [1, 2, 8, 3, 5, 6, 7, 13, None, 9, 11, 12, 18, 4, 14, 21, 16, 22, 15, 10, 23, 17, 19, 20, 24] | 27 | 11120 | 27 | 205 | 27 | 303 |
| [1, 4, 10, 5, 15, 6, 3, 9, 14, 12, 11, 8, 2, 7, 19, 17, 13, 22, 18, 20, 16, 21, None, 23, 24] | TIMEOUT | TIMEOUT | 43 | 42413 | TIMEOUT | TIMEOUT |
| [1, 11, None, 2, 5, 7, 3, 8, 4, 10, 6, 12, 15, 9, 17, 16, 22, 13, 24, 14, 21, 23, 18, 20, 19] | TIMEOUT | TIMEOUT | 37 | 14709 | 37 | 60484 |
| [6, 1, 2, 4, 5, 7, 12, 3, 9, 10, None, 11, 8, 13, 15, 17, 14, 18, 24, 19, 16, 21, 22, 23, 20] | 23 | 673 | 23 | 95 | 23 | 95 |
| [1, 2, 3, 4, 5, 6, 7, 9, 14, 10, 12, 13, None, 24, 19, 11, 17, 15, 8, 20, 16, 21, 18, 22, 23] | 25 | 15906 | 25 | 153 | 25 | 240 |
| [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 21, 16, 22, 19, 20, 17, 23, None, 18, 24] | 19 | 3023 | 19 | 757 | 19 | 848 |
| [12, 1, 2, 3, 5, 6, 7, 9, 4, 10, 21, 11, 8, 14, 15, 22, 17, 16, 18, 19, None, 23, 13, 24, 20] | 29 | 27033 | 29 | 603 | 29 | 838 |
| [1, 2, 3, 4, 5, 6, 13, 7, 9, 10, 11, 18, None, 12, 24, 17, 8, 14, 20, 23, 16, 21, 15, 22, 19] | TIMEOUT | TIMEOUT | 37 | 14385 | 37 | 93174 |

# References

Kotsireas, I. (2025). Informed Search Algorithms Heuristics [Chapter 4]. In *m4-heuristics* (p. 28).

Kotsireas, I. (2025). Solving Problems by Searching [Chapter 3]. In *m3-search* (p. 17, p. 31).