Uninformed and Informed Search

February 11, 2026

1 M3 Project

2 Justin Stutler

In this project, you will compare the performance of uninformed and informed search algorithms for the 4-sided dominoes problem. You will be given N2 tiles (N = {2,3,4}) and will be asked to arrange them in an NxN grid in a way that adjacent tiles have the same number in their neighboring sides. The tiles cannot be rotated. See the example below for N=3.

An initial version of the code with the problem specification (below) and a report template (at the bottom) are available in this notebook. Deliverables are the final code (non-functioning code is worth 0 points) and the comparison report.

Solve the task above using: - one uninformed search algorithm of your choice (20pts) - one informed search algorithm of your choice (20pts)

For your solution, describe the: - search state space (10pts) - successor function (10pts) - heuristic function for the informed search (10pts)

Run each algorithm with at least 10 different initial states for each value of N, and compare the performance of the chosen algorithms for different puzzle sizes in terms of: - average number of expanded states (15pts) - success rate (15pts)

You are free to set a maximum number of expanded states for each algorithm, and return failure in case this number is reached.

3 Implementation

3.1 Domino Code

[1]:
```
import random
```

[2]:
```
class Domino():
    """
    Implementation of a 4-sided domino tile.

    Methods
    -------
    is_above(other)
        Checks of the tile is above the other tile.
```

```
    is_under(other)
        Checks of the tile is under the other tile.
    is_on_the_left_of(other)
        Checks of the tile is on the left of the other tile.
    is_on_the_right_of(other)
        Checks of the tile is on the right of the other tile.
    """
    def __init__(self, top: int, right: int, bottom: int, left: int):
        assert isinstance(top, int) and isinstance(right, int) and␣
 ↪isinstance(bottom, int) and isinstance(left, int), "Invalid tile value!"
        self.top = top
        self.right = right
        self.bottom = bottom
        self.left = left

    def is_above(self, other):
        assert isinstance(other, Domino), "Invalid tile type!"
        return self.bottom == other.top

    def is_under(self, other):
        assert isinstance(other, Domino), "Invalid tile type!"
        return self.top == other.bottom

    def is_on_the_left_of(self, other):
        assert isinstance(other, Domino), "Invalid tile type!"
        return self.right == other.left

    def is_on_the_right_of(self, other):
        assert isinstance(other, Domino), "Invalid tile type!"
        return self.left == other.right
```

[3]:
```
"""
This implementation is provided as a starting point. Feel free to change it as␣
 ↪needed.
"""
class FourDominoes():
    """
    Implementation of the 4-sided dominoes puzzle.

    Methods
    -------
    show()
        Visualize the current state.
    move(action)
        Apply an action to the current state.
    successor(state)
        Finds the list of successors for a given state.
```

```
    goal_test(state)
        Checks if the current state is a goal state.
    """

    def __init__(self, N, state=None):
        """
        Parameters
        ----------
        N
            Grid size (puzzle contains N^2 tiles)
        state
            Initial state configuration. If None is provided, a random one is␣
↪created.
        """
        assert N >= 2 and N <= 4, "Invalid grid size!"
        self.N = N

        if state is not None:
            assert len(state) == self.N, "Invalid state size!"
            for row in state:
                assert len(row) == self.N, "Invalid state size!"
                for tile in row:
                    assert isinstance(tile, Domino), "Invalid state type!"
            self.state = state
        else:
            self.state = self.__get_random_state()

    def __get_random_state(self):
        """
        Generates a random puzzle configuration (grid of dominoes)

        Return
        ----------
        tuple
            A tuple describing a unique puzzle configuration.
        """
        temp = []
        for i in range(self.N):
            for j in range(self.N):
                domino = Domino(
                    random.randint(1,9) if i == 0 else temp[(i-1)*self.N+j].
↪bottom, # top
                    random.randint(1,9),                                    ␣
↪    # right
                    random.randint(1,9),                                    ␣
↪    # bottom
```

```
                    random.randint(1,9) if j == 0 else temp[i*self.N+j-1].right␣
↪    #left
                )
                temp.append(domino)
        random.shuffle(temp) # if you comment this line, the state will be a␣
↪final state (solution)
        return tuple(tuple(temp[i*self.N:(i+1)*self.N]) for i in range(self.N))

    def show(self):
        """
        Prints the current state.
        """
        #line
        print(' ', end='')
        for i in range(self.N):
            print('  ', end=' ' if i < self.N-1 else ' \n')
        for i in range(self.N):
            # first line of tile
            print(' ', end='')
            for j in range(self.N):
                print(' {} '.format(self.state[i][j].top), end='' if j < self.
↪N-1 else '\n')
            # second line of tile
            print(' ', end='')
            for j in range(self.N):
                print('{}{} '.format(self.state[i][j].left, self.state[i][j].
↪right), end='' if j < self.N-1 else '\n')
            # third line of tile
            print(' ', end='')
            for j in range(self.N):
                print(' {} '.format(self.state[i][j].bottom), end='' if j <␣
↪self.N-1 else '\n')
            # line
            print(' ' if i < self.N-1 else ' ', end='')
            for j in range(self.N):
                print('  ', end=' ' if i < self.N-1 and j < self.N-1 else ' ' if␣
↪i == self.N-1 and j < self.N-1 else ' \n' if i < self.N-1 else ' \n')

    def move(self, action):
        """
        Uses a given action to update the current state of the puzzle. Assumes␣
↪the action is valid.

        Parameters
        ----------
        action
```

```
            Tuple with coordinates of two tiles to be swapped ((row1,col1),␣
↪(row2,col2))
        """
        assert len(action) == 2 and all(len(coord) == 2 for coord in action)␣
↪and all(isinstance(x, int) and x >= 0 and x < self.N for coord in action for␣
↪x in coord), "Invalid action!"
        (r1, c1), (r2, c2) = action
        temp = [list(row) for row in self.state]
        temp[r1][c1], temp[r2][c2] = temp[r2][c2], temp[r1][c1]
        self.state = tuple(tuple(x) for x in temp)

    def successor(self, state):
        """
        Finds the list of successors for a given state.

        Parameters
        ----------
        state
            A tuple describing a unique puzzle configuration.

        Returns
        -------
        list
            A list of pairs (action,state) with all states that can be reached␣
↪from the given state with a single action.
        """

        successors = []
        for i in range(self.N*self.N):
            r1 = i//self.N
            c1 = i%self.N
            for j in range(i+1, self.N*self.N):
                r2 = j//self.N
                c2 = j%self.N
                action = ((r1,c1), (r2,c2))
                copy = FourDominoes(self.N, state)
                copy.move(action)
                successors.append((action,copy.state))
        return successors

    def goal_test(self, state):
        """
        Checks if the given state is a goal state.

        Parameters
        ----------
        state
```

```
            A tuple describing a unique puzzle configuration.

        Returns
        -------
        bool
             True if the given state is a goal state, and False otherwise.
        """

        for i in range(self.N):
            for j in range(self.N):
                if i > 0 and not state[i][j].is_under(state[i-1][j]):
                    return False
                if j > 0 and not state[i][j].is_on_the_right_of(state[i][j-1]):
                    return False
        return True
```

3.2 Test Code

[4]:
```
for N in range(2,5):
    print('-----------------')
    print('{}x{} grid of tiles'.format(N,N))
    print('-----------------')

    x = FourDominoes(N)
    x.show()

    print('This state has {} successors!'.format(len(x.successor(x.state))))
    if(x.goal_test(x.state)):
        print('This state is a goal state!')
    else:
        print('This state is not a goal state!')

    x.move(((0,0),(1,1)))
    x.show()

    print('This state has {} successors!'.format(len(x.successor(x.state))))
    if(x.goal_test(x.state)):
        print('This state is a goal state!')
    else:
        print('This state is not a goal state!')

    print()
```

```
-----------------
2x2 grid of tiles
-----------------

 4  4
```

```
 9 3 4 2
  9  5

  9  9
 3 4 3 7
  5  4

This state has 6 successors!
This state is not a goal state!

  9  4
 3 7 4 2
  4  5

  9  4
 3 4 9 3
  5  9

This state has 6 successors!
This state is not a goal state!

-----------------
3x3 grid of tiles
-----------------

  2  2  9
 6 3 3 6 3 3
  9  5  4

  7  5  9
 6 3 5 2 3 3
  1  7  1

  4  7  1
 6 9 9 2 2 6
  1  2  9

This state has 36 successors!
This state is not a goal state!

  5  2  9
 5 2 3 6 3 3
  7  5  4

  7  2  9
 6 3 6 3 3 3
  1  9  1
```

```
 4  7  1
6 9 9 2 2 6
 1  2  9

This state has 36 successors!
This state is not a goal state!

-----------------
4x4 grid of tiles
-----------------

 9  1  5  1
1 2 9 9 2 9 6 1
 9  1  6  9

 2  7  2  6
3 9 4 4 5 3 1 4
 5  3  2  7

 7  1  9  2
2 2 7 6 2 1 5 5
 4  9  1  9

 2  1  3  9
5 2 9 1 2 8 4 5
 1  7  4  1

This state has 120 successors!
This state is not a goal state!

 7  1  5  1
4 4 9 9 2 9 6 1
 3  1  6  9

 2  9  2  6
3 9 1 2 5 3 1 4
 5  9  2  7

 7  1  9  2
2 2 7 6 2 1 5 5
 4  9  1  9

 2  1  3  9
5 2 9 1 2 8 4 5
 1  7  4  1

This state has 120 successors!
This state is not a goal state!
```

3.3 Depth-Limited Depth First Search (DFS) Deepening Code

[5]:
```
from abc import ABCMeta, abstractmethod
import random
import math
```

[6]:
```
class DepthLimitedDFS():
    """
    Depth-limited DFS implementation

    Methods
    -------
    search(state, limit)
        Runs DFS starting from a given start state, and returns the list of␣
 ↪actions to reach a goal state and the number of expanded states,
        but does not let the search go beyond the limit number of steps.
    successor(state)
        Finds the list of successors for a given state.
    goal_test(state)
        Checks if the current state is a goal state.
    """

    __metaclass__ = ABCMeta

    def search(self, state, limit, max_nodes_allowed, expanded_nodes_so_far):
        num_expanded_nodes = 0
        total_successors_generated = 0

        states_to_expand = [(state, [])] # [state, path of actions]

        visited_states = set() # track states visited

        while states_to_expand:
            if expanded_nodes_so_far + num_expanded_nodes >= max_nodes_allowed:␣
 ↪# if max_nodes limit is reached, terminate
                return None, num_expanded_nodes, total_successors_generated

            current_state, path = states_to_expand.pop()

            if self.goal_test(current_state):
                return path, num_expanded_nodes, total_successors_generated

            visited_states.add(current_state) # Add state to visited set to␣
 ↪prevent cycles or repeats
```

```
            if len(path) < limit: # Only expand if the path is less than the␣
↪current depth limit
                num_expanded_nodes += 1 # track nodes expanded
                successors = self.successor(current_state)
                total_successors_generated += len(successors)

                for action, child_state in reversed(successors): # Add valid␣
↪successors to the stack in reverse order
                    if child_state not in visited_states:
                        new_path = path + [action]
                        states_to_expand.append((child_state, new_path))

        return None, num_expanded_nodes, total_successors_generated # return if␣
↪no solution is found

    @abstractmethod
    def successor(self, state):
        """
        Finds the list of successors for a given state.

        Parameters
        ----------
        state
            A tuple describing a unique world configuration.

        Returns
        -------
        list
            A list of pairs (action,state) with all states that can be reached␣
↪from the given state with a single action.
        """
        pass

    @abstractmethod
    def goal_test(self, state):
        """
        Checks if the current state is a goal state.

        Parameters
        ----------
        state
            A tuple describing a unique world configuration.

        Returns
        -------
        bool
              True if the given state is a goal state, and False otherwise.
```

```
        """
        pass
```

4 Iterative Deepening Implemented Code

[7]:
```
class IterativeDeepening(DepthLimitedDFS):
    """
    Iterative Deepening implementation

    Methods
    -------
    search(state, limit)
        Runs Iterative Deepening starting from a given start state, and returns␣
↪the list of actions to reach a goal state and the number of expanded states.
    successor(state)
        Finds the list of successors for a given state.
    goal_test(state)
        Checks if the current state is a goal state.
    """

    __metaclass__ = ABCMeta

    def search(self, state, max_expanded_nodes_allowed):
        if max_expanded_nodes_allowed <= 0: # if invalid limit is given return
            return None, 0, 0

        total_expanded_nodes = 0
        total_successors_generated = 0
        last_expanded_nodes = -1
        limit = 0 # depth-limit not expanded node limit

        while True:
            solution, nodes_in_run, successors_in_run = \
                super(IterativeDeepening, self).search(state, limit,␣
↪max_expanded_nodes_allowed, total_expanded_nodes)

            total_expanded_nodes += nodes_in_run # track expanded nodes

            total_successors_generated += successors_in_run # track total␣
↪successors

            if total_expanded_nodes >= max_expanded_nodes_allowed: # terminate␣
↪if expanded nodes limit is reached
                return None, total_expanded_nodes, total_successors_generated

            if solution is not None:
```

```
                return solution, total_expanded_nodes, total_successors_generated # return if solution if found

            if nodes_in_run == last_expanded_nodes: # do not repeat nodes
                break

            last_expanded_nodes = nodes_in_run
            limit += 1 # increment depth

        return None, total_expanded_nodes, total_successors_generated # return none if no solution if found

    @abstractmethod
    def successor(self, state):
        """
        Finds the list of successors for a given state.

        Parameters
        ----------
        state
            A tuple describing a unique world configuration.

        Returns
        -------
        list
            A list of pairs (action,state) with all states that can be reached from the given state with a single action.
        """
        pass

    @abstractmethod
    def goal_test(self, state):
        """
        Checks if the current state is a goal state.

        Parameters
        ----------
        state
            A tuple describing a unique world configuration.

        Returns
        -------
        bool
             True if the given state is a goal state, and False otherwise.
        """
        pass
```

5 Iterative Deepening Implemented Code

[8]:
```
class IDImplemented(FourDominoes, IterativeDeepening):
    """
    IDImplemented is a class that inherits the methods from the FourDominoes␣
 ↪class as well as Depth-Limited DFS and Iterative Deepening classes.
    This class implements the Iterative Deepening algorithm on the FourDominoes␣
 ↪Puzzle.
    """
    def __init__(self, N, state=None):
        """
        Initializes the puzzle.

        Parameters
        ----------
        N
            The grid size (the puzzle will have N^2 tiles).
        state
            An optional initial state. If None, a random one is created.
        """
        # Initialize the FourDominoes part of the class
        super().__init__(N, state)
```

6 Test Code

[9]:
```
for N in range(2,5): # test each configuration: 2x2, 3x3, ,4x4
    print('Configuration:', N,"x",N)

    n = 10 # test n=10 initial states
    avgExpanded=0 # tracks average number of expanded nodes per configuration
    goals=0 # tracks goal states achieved per configuration

    for i in range(0,n): # for n examples
        num_moves=0
        avgS=0
        totS=0

        x = IDImplemented(N) # instanciate x as IDImplemented object

        while x.goal_test(x.state): # ensures puzzle does not start in a goal␣
 ↪state
            x = IDImplemented(N)

        print("Initial State:")
        x.show() # show initial state
```

```
        print('This state has {} successors!'.format(len(x.successor(x.
↪state)))) # calculate # successors # prints intial state successors
        n=100000
        solution, expanded_states, total_successors = x.search(x.state,␣
↪max_expanded_nodes_allowed=n) # Run ID with limit to prevent infinite run␣
↪time

        print(f"Search complete.\nStates expanded: {expanded_states}") # print␣
↪# expanded states it took to complete the search
        print('Total Successors: ', total_successors) # print total_successors

        avg_branching_factor = 0
        if expanded_states > 0:
            avg_branching_factor = total_successors / expanded_states #␣
↪calculate avg branching factor
        print('Average Branching Factor: ', avg_branching_factor) # print avg␣
↪branching factor

        # Check if a solution was found and display the result
        if solution:
            num_moves=len(solution)
            print("Solution found in", num_moves, "moves!")
            # print("Sequence of moves:", solution) DO NOT Display steps of␣
↪solution unless you want thousands of steps printed

            for move in solution: # Apply the moves to the initial puzzle to␣
↪see the final state
                x.move(move)

            print("Final State:") # show final state
            x.show()
            goals += 1
        else:
            print("No solution found within the search limit.") # print if no␣
↪solution is found
      ␣
↪print('-------------------------------------------------------------------------------------
↪# seperate each puzzle
        avgExpanded += expanded_states
    avgExpanded = avgExpanded / 10
    print('Average Number of Expanded States: ', avgExpanded)
    print('Goal States Achieved: ', goals)
    print('Fails: ', 10-goals)
  ␣
↪print('-------------------------------------------------------------------------------------
↪# seperate puzzles by N value
```

```
Configuration: 2 x 2
Initial State:

 8  4
7 1 9 2
 4  5

 5  5
4 9 3 7
 5  5

This state has 6 successors!
Search complete.
States expanded: 3
Total Successors:  18
Average Branching Factor:  6.0
Solution found in 2 moves!
Final State:

 5  8
3 7 7 1
 5  4

 5  4
4 9 9 2
 5  5

--------------------------------------------------------------------------------
--------------------
Initial State:

 2  7
5 4 4 1
 7  2

 5  4
6 4 4 5
 4  8

This state has 6 successors!
Search complete.
States expanded: 4
Total Successors:  24
Average Branching Factor:  6.0
Solution found in 2 moves!
Final State:

 5  7
```

```
 6 4 4 1
  4  2

  4  2
 4 5 5 4
  8  7

--------------------------------------------------------------------------------
--------------------
Initial State:

  4  8
 3 8 5 6
  8  5

  8  1
 8 6 6 8
  1  3

This state has 6 successors!
Search complete.
States expanded: 1
Total Successors:  6
Average Branching Factor:  6.0
Solution found in 1 moves!
Final State:

  4  8
 3 8 8 6
  8  1

  8  1
 5 6 6 8
  5  3

--------------------------------------------------------------------------------
--------------------
Initial State:

  4  9
 8 2 6 5
  9  9

  2  8
 2 3 5 5
  8  3

This state has 6 successors!
```

```
Search complete.
States expanded: 1
Total Successors:  6
Average Branching Factor:  6.0
Solution found in 1 moves!
Final State:

 4  2
8 2 2 3
 9  8

 9  8
6 5 5 5
 9  3

--------------------------------------------------------------------------------
--------------------
Initial State:

 8  3
4 5 7 2
 3  5

 1  3
2 9 5 2
 6  1

This state has 6 successors!
Search complete.
States expanded: 6
Total Successors:  36
Average Branching Factor:  6.0
Solution found in 2 moves!
Final State:

 8  3
4 5 5 2
 3  1

 3  1
7 2 2 9
 5  6

--------------------------------------------------------------------------------
--------------------
Initial State:

 9  3
```

```
4 6 6 5
 1  6

 6  6
6 3 3 3
 9  3

This state has 6 successors!
Search complete.
States expanded: 4
Total Successors:  24
Average Branching Factor:  6.0
Solution found in 2 moves!
Final State:

 6  6
6 3 3 3
 9  3

 9  3
4 6 6 5
 1  6

--------------------------------------------------------------------------------
--------------------
Initial State:

 8  7
5 7 2 4
 3  8

 6  8
2 5 1 2
 8  6

This state has 6 successors!
Search complete.
States expanded: 1
Total Successors:  6
Average Branching Factor:  6.0
Solution found in 1 moves!
Final State:

 8  7
1 2 2 4
 6  8

 6  8
```

```
 2 5 5 7
  8  3

--------------------------------------------------------------------------------
--------------------
Initial State:

  7  2
 3 3 7 6
  2  7

  4  8
 3 4 6 1
  8  2

This state has 6 successors!
Search complete.
States expanded: 1
Total Successors:  6
Average Branching Factor:  6.0
Solution found in 1 moves!
Final State:

  7  4
 3 3 3 4
  2  8

  2  8
 7 6 6 1
  7  2

--------------------------------------------------------------------------------
--------------------
Initial State:

  8  7
 2 4 1 1
  9  8

  4  7
 1 4 4 8
  7  6

This state has 6 successors!
Search complete.
States expanded: 3
Total Successors:  18
Average Branching Factor:  6.0
```

```
Solution found in 2 moves!
Final State:

 7  4
1 1 1 4
 8  7

 8  7
2 4 4 8
 9  6

--------------------------------------------------------------------------------
--------------------
Initial State:

 5  6
9 9 9 3
 1  1

 9  2
2 6 6 9
 5  6

This state has 6 successors!
Search complete.
States expanded: 4
Total Successors:  24
Average Branching Factor:  6.0
Solution found in 2 moves!
Final State:

 9  2
2 6 6 9
 5  6

 5  6
9 9 9 3
 1  1

--------------------------------------------------------------------------------
--------------------
Average Number of Expanded States:  2.8
Goal States Achieved:  10
Fails:  0
--------------------------------------------------------------------------------
--------------------
Configuration: 3 x 3
Initial State:
```

```
 1  1  1
1 7 7 9 4 1
 5  9  4

 5  5  2
1 2 8 4 1 9
 1  2  2

 1  2  4
6 7 7 6 9 1
 8  2  1

This state has 36 successors!
Search complete.
States expanded: 40422
Total Successors:  1455192
Average Branching Factor:  36.0
Solution found in 8 moves!
Final State:

 5  1  1
8 4 4 1 1 7
 2  4  5

 2  4  5
1 9 9 1 1 2
 2  1  1

 2  1  1
7 6 6 7 7 9
 2  8  9

--------------------------------------------------------------------------------
--------------------
Initial State:

 4  7  4
6 6 6 5 1 7
 2  1  3

 2  8  1
7 5 2 3 8 1
 9  1  8

 9  1  3
3 4 5 6 3 3
 9  4  1
```

```
This state has 36 successors!
Search complete.
States expanded: 73820
Total Successors:  2657520
Average Branching Factor:  36.0
Solution found in 9 moves!
Final State:

 7  1  4
6 5 5 6 6 6
 1  4  2

 1  4  2
8 1 1 7 7 5
 8  3  9

 8  3  9
2 3 3 3 3 4
 1  1  9

--------------------------------------------------------------------------------
--------------------
Initial State:

 6  5  8
7 7 8 9 3 8
 1  1  6

 6  1  9
3 4 4 7 9 9
 7  1  6

 1  1  7
3 5 5 5 4 3
 4  8  9

This state has 36 successors!
Search complete.
States expanded: 31974
Total Successors:  1151064
Average Branching Factor:  36.0
Solution found in 8 moves!
Final State:

 8  5  9
3 8 8 9 9 9
 6  1  6
```

```
 6  1  6
3 4 4 7 7 7
 7  1  1

 7  1  1
4 3 3 5 5 5
 9  4  8

--------------------------------------------------------------------------------
--------------------
Initial State:

 5  1  7
9 7 7 2 7 7
 7  7  3

 2  9  6
9 7 1 2 3 9
 1  2  2

 4  3  4
7 8 2 1 7 1
 9  9  4

This state has 36 successors!
Search complete.
States expanded: 27244
Total Successors:  980784
Average Branching Factor:  36.0
Solution found in 7 moves!
Final State:

 6  5  4
3 9 9 7 7 1
 2  7  4

 2  7  4
9 7 7 7 7 8
 1  3  9

 1  3  9
7 2 2 1 1 2
 7  9  2

--------------------------------------------------------------------------------
--------------------
Initial State:
```

```
 9  3  5
8 6 6 7 6 8
 3  8  9

 1  5  3
3 6 6 4 1 8
 9  3  1

 3  9  8
1 3 9 1 8 2
 5  8  3

This state has 36 successors!
Search complete.
States expanded: 16600
Total Successors:  597600
Average Branching Factor:  36.0
Solution found in 7 moves!
Final State:

 3  1  5
1 3 3 6 6 4
 5  9  3

 5  9  3
6 8 8 6 6 7
 9  3  8

 9  3  8
9 1 1 8 8 2
 8  1  3

--------------------------------------------------------------------------------
--------------------
Initial State:

 4  9  6
1 6 3 5 2 4
 2  7  8

 8  2  7
2 3 8 2 6 2
 1  8  2

 6  1  2
5 5 6 3 4 8
 6  4  2
```

```
This state has 36 successors!
Search complete.
States expanded: 18169
Total Successors:  654084
Average Branching Factor:  36.0
Solution found in 7 moves!
Final State:

 1  9  6
6 3 3 5 5 5
 4  7  6

 4  7  6
1 6 6 2 2 4
 2  2  8

 2  2  8
4 8 8 2 2 3
 2  8  1

--------------------------------------------------------------------------------
--------------------
Initial State:

 1  4  3
9 2 2 9 6 8
 2  2  1

 4  1  4
9 7 1 6 6 6
 3  4  4

 8  8  3
4 1 8 5 8 6
 1  4  8

This state has 36 successors!
Search complete.
States expanded: 22842
Total Successors:  822312
Average Branching Factor:  36.0
Solution found in 7 moves!
Final State:

 3  3  8
8 6 6 8 8 5
 8  1  4
```

```
 8  1  4
4 1 1 6 6 6
 1  4  4

 1  4  4
9 2 2 9 9 7
 2  2  3

--------------------------------------------------------------------------------
--------------------
Initial State:

 3  1  8
6 7 8 5 7 6
 6  9  5

 6  9  7
8 8 7 8 3 4
 1  9  3

 2  9  9
8 7 4 8 6 5
 7  8  1

This state has 36 successors!
Search complete.
States expanded: 100000
Total Successors:  3600000
Average Branching Factor:  36.0
No solution found within the search limit.
--------------------------------------------------------------------------------
--------------------
Initial State:

 8  6  2
9 4 8 3 4 8
 7  4  9

 4  3  9
4 7 4 6 8 4
 3  2  5

 1  7  9
6 8 2 4 3 8
 6  9  8

This state has 36 successors!
```

```
Search complete.
States expanded: 77777
Total Successors:  2799972
Average Branching Factor:  36.0
Solution found in 9 moves!
Final State:

 8  3  1
9 4 4 6 6 8
 7  2  6

 7  2  6
2 4 4 8 8 3
 9  9  4

 9  9  4
3 8 8 4 4 7
 8  5  3

--------------------------------------------------------------------------------
--------------------
Initial State:

 2  6  6
2 9 5 4 8 3
 4  2  6

 4  3  6
5 3 3 2 7 2
 4  3  9

 5  3  2
8 5 2 5 1 8
 3  6  6

This state has 36 successors!
Search complete.
States expanded: 32814
Total Successors:  1181304
Average Branching Factor:  36.0
Solution found in 8 moves!
Final State:

 2  5  6
1 8 8 5 5 4
 6  3  2

 6  3  2
```

```
 8 3 3 2 2 9
  6  3  4

  6  3  4
 7 2 2 5 5 3
  9  6  4

--------------------------------------------------------------------------------
--------------------
Average Number of Expanded States:  44166.2
Goal States Achieved:  9
Fails:  1
--------------------------------------------------------------------------------
--------------------
Configuration: 4 x 4
Initial State:

  8  3  7  9
 6 1 7 7 7 7 7 8
  1  3  2  7

  1  2  8  6
 6 5 5 7 8 7 5 7
  6  3  8  9

  2  2  5  4
 7 5 1 2 2 9 9 5
  6  2  4  3

  4  1  7  7
 7 5 7 5 4 6 9 6
  2  5  7  4

This state has 120 successors!
Search complete.
States expanded: 100000
Total Successors:  12000000
Average Branching Factor:  120.0
No solution found within the search limit.
--------------------------------------------------------------------------------
--------------------
Initial State:

  6  3  6  8
 4 3 1 1 8 1 5 5
  5  6  7  3

  2  4  9  7
```

```
8 9 9 8 5 9 2 8
 6  3  7  6

 7  5  3  7
1 5 6 2 3 9 9 3
 6  7  7  9

 9  7  6  7
7 2 9 7 2 8 9 2
 4  2  1  6

This state has 120 successors!
Search complete.
States expanded: 100000
Total Successors:  12000000
Average Branching Factor:  120.0
No solution found within the search limit.
--------------------------------------------------------------------------------
--------------------
Initial State:

 9  9  9  5
5 4 5 7 5 7 1 6
 9  3  9  4

 8  1  6  7
7 2 7 6 6 5 7 7
 3  9  6  6

 4  6  9  9
6 1 8 7 8 8 7 8
 2  6  5  9

 3  2  4  6
4 2 8 7 6 9 9 5
 8  3  3  7

This state has 120 successors!
Search complete.
States expanded: 100000
Total Successors:  12000000
Average Branching Factor:  120.0
No solution found within the search limit.
--------------------------------------------------------------------------------
--------------------
Initial State:

 5  6  7  8
```

```
5 6 2 1 1 2 1 3
 6  1  2  3

 1  9  5  5
4 8 2 2 2 5 8 2
 6  5  9  9

 3  7  7  2
5 1 3 8 5 1 8 9
 9  7  5  9

 2  2  9  9
3 5 5 4 5 3 8 5
 5  9  1  2

This state has 120 successors!
Search complete.
States expanded: 100000
Total Successors:  12000000
Average Branching Factor:  120.0
No solution found within the search limit.
--------------------------------------------------------------------------------
--------------------
Initial State:

 3  2  7  2
5 4 9 5 9 9 2 9
 2  2  1  5

 3  4  2  5
1 5 7 5 5 3 9 5
 2  2  6  3

 4  7  7  3
5 6 5 4 3 2 7 8
 4  7  5  5

 2  2  5  5
2 7 4 7 6 5 4 2
 5  4  7  7

This state has 120 successors!
Search complete.
States expanded: 100000
Total Successors:  12000000
Average Branching Factor:  120.0
No solution found within the search limit.
--------------------------------------------------------------------------------
```

```
--------------------
Initial State:

 9  5  4  1
2 2 3 5 1 4 5 8
 4  8  8  9

 1  8  3  7
7 8 4 7 6 2 8 9
 6  6  5  4

 4  6  4  7
2 9 6 1 8 5 5 7
 5  5  3  4

 5  8  6  4
8 4 5 2 9 7 2 6
 4  8  5  1

This state has 120 successors!
Search complete.
States expanded: 100000
Total Successors:  12000000
Average Branching Factor:  120.0
No solution found within the search limit.
--------------------------------------------------------------------------------
--------------------
Initial State:

 7  9  9  2
1 6 9 4 5 4 6 6
 2  8  6  5

 6  5  6  3
6 4 4 1 7 4 8 9
 1  4  5  2

 7  2  9  6
7 7 6 3 1 7 6 8
 3  9  6  6

 4  6  6  5
6 5 4 8 2 6 9 6
 7  2  9  4

This state has 120 successors!
Search complete.
States expanded: 100000
```

```
Total Successors:  12000000
Average Branching Factor:  120.0
No solution found within the search limit.
--------------------------------------------------------------------------------
--------------------
Initial State:

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

 8  4  6  1
8 5 4 7 1 5 4 8
 6  6  8  3

 5  6  9  7
8 2 5 4 7 1 2 4
 9  4  8  4

 3  7  3  3
8 2 5 1 5 6 7 7
 5  8  3  8

This state has 120 successors!
Search complete.
States expanded: 100000
Total Successors:  12000000
Average Branching Factor:  120.0
No solution found within the search limit.
--------------------------------------------------------------------------------
--------------------
Initial State:

 1  9  9  1
9 6 2 5 4 9 1 9
 7  5  3  9

 3  3  9  2
7 2 6 9 4 6 6 2
 9  3  1  3

 7  6  2  9
5 4 7 5 9 7 5 3
 7  9  9  2

 9  9  3  7
7 1 2 4 4 1 4 2
 7  1  2  1
```

```
This state has 120 successors!
Search complete.
States expanded: 100000
Total Successors:  12000000
Average Branching Factor:  120.0
No solution found within the search limit.
--------------------------------------------------------------------------------
--------------------
Initial State:

 3  7  3  3
6 2 3 3 3 3 1 2
 1  4  6  4

 4  6  3  9
6 2 2 7 7 9 9 4
 1  7  3  8

 6  4  6  5
4 3 2 4 4 1 2 6
 3  5  6  6

 8  7  9  6
9 9 9 3 4 2 1 6
 6  3  7  5

This state has 120 successors!
Search complete.
States expanded: 100000
Total Successors:  12000000
Average Branching Factor:  120.0
No solution found within the search limit.
--------------------------------------------------------------------------------
--------------------
Average Number of Expanded States:  100000.0
Goal States Achieved:  0
Fails:  10
--------------------------------------------------------------------------------
--------------------
```

7 A * Code

[10]:
```
from abc import ABCMeta, abstractmethod
from queue import PriorityQueue
import random
import math
```

[11]:
```
class Astar():
    """
    A* implementation

    Methods
    -------
    search(state)
        Runs A* starting from a given start state, and returns the list of␣
↪actions to reach a goal state and the number of expanded states.
    successor(state)
        Finds the list of successors for a given state.
    goal_test(state)
        Checks if the current state is a goal state.
    """

    __metaclass__ = ABCMeta

    def search(self, state, limit=10000):
        """
        Runs A* starting from a given start state.
        """
        num_expanded_states = 0
        expanded_states = []
        visited_states = set()
        total_successors_generated = 0    # tracks total successors generated


        states_to_expand = PriorityQueue()

        initial_heuristic = self.heuristic(state) # calculate heuristic for␣
↪inital state
        # Format: (f(n), g(n), random_val(used to break ties), state,␣
↪parent_idx, action
        states_to_expand.put((initial_heuristic, 0, random.random(),state, -1,␣
↪None))

        while not states_to_expand.empty():
            if limit <= 0: # if limit is reached, terminate
                return None, num_expanded_states, total_successors_generated #␣
↪return if limit is reached

            total_cost, path_cost, _, current_state, parent, action =␣
↪states_to_expand.get()

            if current_state in visited_states: # Ensures nodes are not visited␣
↪repeatedly
```

```
                continue

            current_successors = self.successor(current_state) # generate␣
↪successors for current state
            total_successors_generated += len(current_successors) # add␣
↪successors to total

            num_expanded_states += 1 # update count of expanded nodes
            visited_states.add(current_state) # update set of visted states
            expanded_states.append((parent, action)) # update list so we can␣
↪move through solution

            if self.goal_test(current_state): # return solution if found
                solution = []
                while parent != -1:
                    solution.append(action)
                    parent, action = expanded_states[parent]
                solution.reverse()
                return solution, num_expanded_states, total_successors_generated

            for action, child, act_cost, heur_cost in current_successors:
                new_total_cost = heur_cost + path_cost + act_cost # update f(n)
                new_path_cost = path_cost + act_cost # update g(n)

                states_to_expand.put((new_total_cost, new_path_cost, random.
↪random(), child, num_expanded_states - 1, action))

            limit -= 1 # decrement limit

        return None, num_expanded_states, total_successors_generated # return␣
↪if no solution is found

    def successor(self, state):
        """
        Finds the list of successors for a given state for A* search.

        Returns:
        list
            A list of tuples (action, child_state, action_cost, heuristic_cost)
        """
        successors = []
        N = len(state)

        # Iterate through all unique pairs of tiles to swap
        for i in range(N*N):
            r1, c1 = i // N, i % N
            for j in range(i + 1, N*N):
```

```
                r2, c2 = j // N, j % N

                action = ((r1, c1), (r2, c2))

                # Create a mutable copy of the state to perform the swap
                temp_state_list = [list(row) for row in state]
                temp_state_list[r1][c1], temp_state_list[r2][c2] =␣
↪temp_state_list[r2][c2], temp_state_list[r1][c1]

                child_state = tuple(tuple(row) for row in temp_state_list) #␣
↪convert back to tuple

                action_cost = 1 # Each swap has a cost of 1
                heuristic_cost = self.heuristic(child_state) # finds mismatches␣
↪in current state

                successors.append((action, child_state, action_cost,␣
↪heuristic_cost))

        return successors

    @abstractmethod
    def goal_test(self, state):
        """
        Checks if the current state is a goal state.

        Parameters
        ----------
        state
            A tuple describing a unique world configuration.

        Returns
        -------
        bool
             True if the given state is a goal state, and False otherwise.
        """
        pass
```

8 Implement A* Code

[12]:
```
class AstarImplemented(FourDominoes, Astar):
    """
    Combines the FourDominoes problem with the A* search algorithm.
    """
    def __init__(self, N, state=None):
        # init FourDominoes class
```

```
        super().__init__(N, state)

    def heuristic(self, state):
        """
        Calculates the heuristic value (h(n)) for a state.
        The value is the total number of mismatched adjacent tile sides.
        """
        mismatches = 0
        N = len(state)

        for i in range(N):
            for j in range(N):
                # Check for a vertical mismatch with the tile below
                if i < (N - 1) and not state[i][j].is_above(state[i+1][j]):
                    mismatches += 1
                # Check for a horizontal mismatch with the tile to the right
                if j < (N - 1) and not state[i][j].
↪is_on_the_left_of(state[i][j+1]):
                    mismatches += 1
        return mismatches

    def successor(self, state):
        """
        Generates all possible successor states by swapping two tiles.
        Returns a list of (action, child_state, action_cost, heuristic_cost).
        """
        successors = [] # List to store all generated successors
        N = len(state)

        for i in range(N*N): # Iterate through each tile to be the first in a␣
↪swap
            r1, c1 = i // N, i % N # Convert 1D index to 2D coordinates

            for j in range(i + 1, N*N): # Iterate through the remaining tiles␣
↪to be the second in a swap
                r2, c2 = j // N, j % N # Convert 1D index to 2D coordinates

                action = ((r1, c1), (r2, c2)) # Define the swap action

                temp_state_list = [list(row) for row in state] # Convert tuple␣
↪to list to allow mutation
                temp_state_list[r1][c1], temp_state_list[r2][c2] =␣
↪temp_state_list[r2][c2], temp_state_list[r1][c1] # Perform the tile swap
                child_state = tuple(tuple(row) for row in temp_state_list) #␣
↪Convert list back to a hashable tuple
```

```
                successors.append((action, child_state, 1, self.
↪heuristic(child_state))) # append [action, state, action_cost=1, h(n)]

        return successors
```

9 Test Code

[13]:
```
for N in range(2,5): # test each configuration: 2x2, 3x3, ,4x4
    print('Configuration:', N,"x",N)

    n = 10 # test n=10 initial states
    avgExpanded=0 # tracks average number of expanded nodes per configuration
    goals=0 # tracks goal states achieved per configuration

    for i in range(0,n): # for n examples
        num_moves=0
        avgS=0
        totS=0

        x = AstarImplemented(N) # instanciate x as A*Implemented object

        while x.goal_test(x.state): # ensures puzzle does not start in a goal␣
↪state
            x = AstarImplemented(N)

        print("Initial State:")
        x.show() # show initial state

        print('This state has {} successors!'.format(len(x.successor(x.
↪state)))) # calculate # successors # prints intial state successors
        limit=100000
        solution, expanded_states, total_successors = x.search(x.state,␣
↪limit=limit) # Run A* with limit to prevent infinite run time

        print(f"Search complete.\nStates expanded: {expanded_states}") # print␣
↪# expanded states it took to complete the search
        print('Total Successors: ', total_successors) # print total_successors

        avg_branching_factor = 0
        if expanded_states > 0:
            avg_branching_factor = total_successors / expanded_states #␣
↪calculate avg branching factor
        print('Average Branching Factor: ', avg_branching_factor) # print avg␣
↪branching factor

        # Check if a solution was found and display the result
```

```
        if solution:
            num_moves=len(solution)
            print("Solution found in", num_moves, "moves!")
            # print("Sequence of moves:", solution) DO NOT Display steps of␣
↪solution unless you want thousands of steps printed

            for move in solution: # Apply the moves to the initial puzzle to␣
↪see the final state
                x.move(move)

            print("Final State:") # show final state
            x.show()
            goals += 1
        else:
            print("No solution found within the search limit.") # print if no␣
↪solution is found
      ␣
↪print('-------------------------------------------------------------------------------------
↪# seperate each puzzle
        avgExpanded += expanded_states
    avgExpanded = avgExpanded / 10
    print('Average Number of Expanded States: ', avgExpanded)
    print('Goal States Achieved: ', goals)
    print('Fails: ', 10-goals)
  ␣
↪print('-------------------------------------------------------------------------------------
↪# seperate puzzles by N value
```

```
Configuration: 2 x 2
Initial State:

 7  9
4 1 6 6
 9  7

 9  1
6 9 1 7
 1  8

This state has 6 successors!
Search complete.
States expanded: 4
Total Successors:  24
Average Branching Factor:  6.0
Solution found in 2 moves!
Final State:
```

```
 9  9
6 6 6 9
 7  1

 7  1
4 1 1 7
 9  8

--------------------------------------------------------------------------------
--------------------
Initial State:

 1  6
6 5 1 1
 4  7

 4  5
1 4 8 6
 1  6

This state has 6 successors!
Search complete.
States expanded: 8
Total Successors:  48
Average Branching Factor:  6.0
Solution found in 3 moves!
Final State:

 5  1
8 6 6 5
 6  4

 6  4
1 1 1 4
 7  1

--------------------------------------------------------------------------------
--------------------
Initial State:

 3  3
3 1 6 5
 2  2

 2  1
1 3 5 3
 7  3
```

```
This state has 6 successors!
Search complete.
States expanded: 3
Total Successors:  18
Average Branching Factor:  6.0
Solution found in 2 moves!
Final State:

 3  1
6 5 5 3
 2  3

 2  3
1 3 3 1
 7  2

--------------------------------------------------------------------------------
--------------------
Initial State:

 6  1
1 8 5 9
 4  4

 4  2
8 3 9 5
 1  6

This state has 6 successors!
Search complete.
States expanded: 5
Total Successors:  30
Average Branching Factor:  6.0
Solution found in 2 moves!
Final State:

 2  1
9 5 5 9
 6  4

 6  4
1 8 8 3
 4  1

--------------------------------------------------------------------------------
--------------------
Initial State:
```

```
 2  2
1 7 6 4
 2  9

 3  3
4 8 7 3
 1  3

This state has 6 successors!
Search complete.
States expanded: 4
Total Successors:  24
Average Branching Factor:  6.0
Solution found in 2 moves!
Final State:

 2  3
1 7 7 3
 2  3

 2  3
6 4 4 8
 9  1

--------------------------------------------------------------------------------
--------------------
Initial State:

 4  8
3 4 4 3
 5  7

 7  5
9 8 8 1
 9  6

This state has 6 successors!
Search complete.
States expanded: 2
Total Successors:  12
Average Branching Factor:  6.0
Solution found in 1 moves!
Final State:

 8  4
4 3 3 4
 7  5
```

```
 7  5
9 8 8 1
 9  6

--------------------------------------------------------------------------------
--------------------
Initial State:

 1  5
4 3 6 4
 5  4

 6  8
4 4 3 6
 9  6

This state has 6 successors!
Search complete.
States expanded: 4
Total Successors:  24
Average Branching Factor:  6.0
Solution found in 2 moves!
Final State:

 1  8
4 3 3 6
 5  6

 5  6
6 4 4 4
 4  9

--------------------------------------------------------------------------------
--------------------
Initial State:

 1  9
2 4 3 4
 9  7

 2  5
4 7 4 4
 1  2

This state has 6 successors!
Search complete.
States expanded: 3
Total Successors:  18
```

```
Average Branching Factor:  6.0
Solution found in 2 moves!
Final State:

 1  5
2 4 4 4
 9  2

 9  2
3 4 4 7
 7  1

--------------------------------------------------------------------------------
--------------------
Initial State:

 8  1
8 5 7 8
 5  8

 8  5
8 4 4 6
 2  4

This state has 6 successors!
Search complete.
States expanded: 2
Total Successors:  12
Average Branching Factor:  6.0
Solution found in 1 moves!
Final State:

 1  8
7 8 8 5
 8  5

 8  5
8 4 4 6
 2  4

--------------------------------------------------------------------------------
--------------------
Initial State:

 9  6
6 8 5 3
 3  9
```

```
 4  9
8 6 9 5
 3  4

This state has 6 successors!
Search complete.
States expanded: 2
Total Successors:  12
Average Branching Factor:  6.0
Solution found in 1 moves!
Final State:

 9  6
9 5 5 3
 4  9

 4  9
8 6 6 8
 3  3

--------------------------------------------------------------------------------
--------------------
Average Number of Expanded States:  3.7
Goal States Achieved:  10
Fails:  0
--------------------------------------------------------------------------------
--------------------
Configuration: 3 x 3
Initial State:

 4  7  6
7 6 1 8 4 2
 6  7  4

 2  1  9
9 5 2 6 6 2
 9  7  2

 2  4  6
2 1 5 9 9 9
 4  1  4

This state has 36 successors!
Search complete.
States expanded: 529
Total Successors:  19044
Average Branching Factor:  36.0
Solution found in 8 moves!
```

```
Final State:

 6  2  4
9 9 9 5 5 9
 4  9  1

 4  9  1
7 6 6 2 2 6
 6  2  7

 6  2  7
4 2 2 1 1 8
 4  4  7

--------------------------------------------------------------------------------
--------------------
Initial State:

 6  7  1
8 4 2 1 3 6
 1  7  7

 9  1  2
4 5 6 3 3 5
 2  8  3

 7  6  8
7 8 1 2 2 2
 1  6  9

This state has 36 successors!
Search complete.
States expanded: 258
Total Successors:  9288
Average Branching Factor:  36.0
Solution found in 7 moves!
Final State:

 7  6  8
2 1 1 2 2 2
 7  6  9

 7  6  9
7 8 8 4 4 5
 1  1  2

 1  1  2
3 6 6 3 3 5
```

```
 7  8  3

--------------------------------------------------------------------------------
--------------------
Initial State:

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

 5  7  3
1 8 7 3 5 7
 4  9  5

 9  4  7
7 1 9 8 8 9
 7  9  6

This state has 36 successors!
Search complete.
States expanded: 407
Total Successors:  14652
Average Branching Factor:  36.0
Solution found in 7 moves!
Final State:

 7  4  2
8 9 9 8 8 5
 6  9  5

 6  9  5
4 7 7 1 1 8
 3  7  4

 3  7  4
5 7 7 3 3 5
 5  9  1

--------------------------------------------------------------------------------
--------------------
Initial State:

 9  2  7
8 2 7 5 9 3
 6  7  9

 6  5  8
5 7 9 1 1 8
```

```
 4  2  6

 4  6  9
3 2 7 8 2 7
 4  9  2

This state has 36 successors!
Search complete.
States expanded: 22
Total Successors:  792
Average Branching Factor:  36.0
Solution found in 7 moves!
Final State:

 5  8  9
9 1 1 8 8 2
 2  6  6

 2  6  6
7 5 5 7 7 8
 7  4  9

 7  4  9
9 3 3 2 2 7
 9  4  2

--------------------------------------------------------------------------------
--------------------
Initial State:

 1  2  6
1 4 2 7 7 4
 7  1  6

 4  6  7
1 1 4 2 8 9
 1  4  3

 7  1  7
9 1 8 1 7 7
 9  7  1

This state has 36 successors!
Search complete.
States expanded: 33
Total Successors:  1188
Average Branching Factor:  36.0
Solution found in 7 moves!
```

```
Final State:

 2  7  6
2 7 7 7 7 4
 1  1  6

 1  1  6
8 1 1 4 4 2
 7  7  4

 7  7  4
8 9 9 1 1 1
 3  9  1

--------------------------------------------------------------------------------
--------------------
Initial State:

 6  1  2
6 4 9 8 8 4
 1  3  2

 6  9  6
8 6 8 7 4 9
 7  1  2

 2  3  7
1 9 7 1 3 9
 4  2  9

This state has 36 successors!
Search complete.
States expanded: 15
Total Successors:  540
Average Branching Factor:  36.0
Solution found in 6 moves!
Final State:

 6  6  6
8 6 6 4 4 9
 7  1  2

 7  1  2
3 9 9 8 8 4
 9  3  2

 9  3  2
8 7 7 1 1 9
```

```
 1  2  4

--------------------------------------------------------------------------------
--------------------
Initial State:

 5  5  4
8 3 9 2 4 2
 4  8  6

 4  8  5
8 4 7 8 2 8
 5  5  8

 9  6  3
4 9 8 7 2 7
 6  4  4

This state has 36 successors!
Search complete.
States expanded: 58
Total Successors:  2088
Average Branching Factor:  36.0
Solution found in 5 moves!
Final State:

 9  5  3
4 9 9 2 2 7
 6  8  4

 6  8  4
8 7 7 8 8 4
 4  5  5

 4  5  5
4 2 2 8 8 3
 6  8  4

--------------------------------------------------------------------------------
--------------------
Initial State:

 4  2  9
2 5 1 2 2 3
 5  4  4

 5  4  7
5 7 8 2 9 5
```

```
 6  7  5

 9  5  6
5 1 2 9 3 8
 6  3  5

This state has 36 successors!
Search complete.
States expanded: 7
Total Successors:  252
Average Branching Factor:  36.0
Solution found in 4 moves!
Final State:

 9  2  9
5 1 1 2 2 3
 6  4  4

 6  4  4
3 8 8 2 2 5
 5  7  5

 5  7  5
2 9 9 5 5 7
 3  5  6

--------------------------------------------------------------------------------
--------------------
Initial State:

 4  9  6
7 4 6 8 9 5
 1  6  5

 7  2  5
5 7 5 6 6 7
 4  9  6

 9  9  3
1 9 2 6 8 9
 9  2  7

This state has 36 successors!
Search complete.
States expanded: 282
Total Successors:  10152
Average Branching Factor:  36.0
Solution found in 8 moves!
```

```
Final State:

 2  9  3
5 6 6 8 8 9
 9  6  7

 9  6  7
1 9 9 5 5 7
 9  5  4

 9  5  4
2 6 6 7 7 4
 2  6  1

--------------------------------------------------------------------------------
--------------------
Initial State:

 6  7  7
3 9 9 8 2 8
 6  9  3

 1  2  3
2 3 9 1 5 9
 6  7  6

 9  1  3
8 9 9 4 9 5
 3  2  1

This state has 36 successors!
Search complete.
States expanded: 674
Total Successors:  24264
Average Branching Factor:  36.0
Solution found in 8 moves!
Final State:

 7  9  1
2 8 8 9 9 4
 3  3  2

 3  3  2
9 5 5 9 9 1
 1  6  7

 1  6  7
2 3 3 9 9 8
```

```
 6  6  9

--------------------------------------------------------------------------------
--------------------
Average Number of Expanded States:  228.5
Goal States Achieved:  10
Fails:  0
--------------------------------------------------------------------------------
--------------------
Configuration: 4 x 4
Initial State:

 9  6  8  4
5 6 2 4 8 8 7 8
 8  9  1  9

 1  5  9  2
6 3 4 2 9 7 8 2
 9  8  8  3

 8  4  1  5
8 6 2 8 3 9 8 7
 1  5  5  2

 3  2  7  1
8 9 6 5 9 8 4 6
 4  8  4  1

This state has 120 successors!
Search complete.
States expanded: 12046
Total Successors:  1445520
Average Branching Factor:  120.0
Solution found in 13 moves!
Final State:

 1  7  2  6
3 9 9 8 8 2 2 4
 5  4  3  9

 5  4  3  9
4 2 2 8 8 9 9 7
 8  5  4  8

 8  5  4  8
8 8 8 7 7 8 8 6
 1  2  9  1
```

```
 1  2  9  1
4 6 6 5 5 6 6 3
 1  8  8  9

--------------------------------------------------------------------------------
--------------------
Initial State:

 8  5  4  9
8 9 9 2 1 7 2 8
 1  8  1  2

 4  1  8  7
8 3 1 1 9 8 8 8
 9  7  4  1

 3  1  8  7
2 8 7 2 8 1 8 2
 4  8  3  1

 4  1  5  9
4 8 3 2 7 4 2 2
 9  8  5  7

This state has 120 successors!
Search complete.
States expanded: 847
Total Successors:  101640
Average Branching Factor:  120.0
Solution found in 12 moves!
Final State:

 5  4  8  1
7 4 4 8 8 1 1 1
 5  9  3  7

 5  9  3  7
9 2 2 2 2 8 8 2
 8  7  4  1

 8  7  4  1
9 8 8 8 8 3 3 2
 4  1  9  8

 4  1  9  8
1 7 7 2 2 8 8 9
 1  8  2  1
```

```
--------------------------------------------------------------------------------
--------------------
Initial State:

 6  5  5  2
3 9 1 8 8 1 7 8
 3  6  9  5

 8  8  4  4
2 9 9 4 1 3 3 1
 7  1  3  9

 3  5  1  7
9 7 4 4 7 3 2 7
 3  5  4  4

 5  9  6  9
3 7 7 3 4 1 9 9
 5  4  2  8

This state has 120 successors!
Search complete.
States expanded: 23
Total Successors:  2760
Average Branching Factor:  120.0
Solution found in 11 moves!
Final State:

 8  8  5  6
2 9 9 4 4 4 4 1
 7  1  5  2

 7  1  5  2
2 7 7 3 3 7 7 8
 4  4  5  5

 4  4  5  5
1 3 3 1 1 8 8 1
 3  9  6  9

 3  9  6  9
9 7 7 3 3 9 9 9
 3  4  3  8

--------------------------------------------------------------------------------
--------------------
Initial State:
```

```
 6  2  7  6
9 6 5 9 5 7 6 4
 7  9  1  4

 9  4  1  2
1 4 9 5 4 7 7 7
 9  2  1  2

 2  5  2  9
5 4 4 3 4 8 8 5
 7  8  7  9

 1  1  7  9
3 5 7 3 5 9 3 3
 7  5  1  2

This state has 120 successors!
Search complete.
States expanded: 7043
Total Successors:  845160
Average Branching Factor:  120.0
Solution found in 13 moves!
Final State:

 2  6  6  2
5 9 9 6 6 4 4 8
 9  7  4  7

 9  7  4  7
8 5 5 9 9 5 5 7
 9  1  2  1

 9  1  2  1
1 4 4 7 7 7 7 3
 9  1  2  5

 9  1  2  5
3 3 3 5 5 4 4 3
 2  7  7  8

--------------------------------------------------------------------------------
--------------------
Initial State:

 7  3  9  3
8 2 5 3 2 2 4 4
 1  7  3  3
```

```
 4  4  8  7
3 4 7 3 3 2 2 8
 9  3  7  8

 2  3  9  8
4 2 5 8 2 4 2 9
 1  3  5  9

 1  9  3  4
4 2 8 9 9 4 2 2
 8  8  4  9

This state has 120 successors!
Search complete.
States expanded: 14797
Total Successors:  1775640
Average Branching Factor:  120.0
Solution found in 13 moves!
Final State:

 4  4  3  2
7 3 3 4 4 4 4 2
 3  9  3  1

 3  9  3  1
5 8 8 9 9 4 4 2
 3  8  4  8

 3  8  4  8
5 3 3 2 2 2 2 9
 7  7  9  9

 7  7  9  9
2 8 8 2 2 2 2 4
 8  1  3  5

--------------------------------------------------------------------------------
--------------------
Initial State:

 2  5  7  9
9 2 8 6 2 5 4 9
 4  9  2  4

 1  7  2  4
9 6 4 8 9 7 1 4
 3  3  5  5
```

```
 7  3  5  4
1 8 6 4 2 9 5 9
 6  1  5  7

 6  5  6  3
2 1 7 5 6 6 8 2
 1  4  7  9

This state has 120 successors!
Search complete.
States expanded: 15650
Total Successors:  1878000
Average Branching Factor:  120.0
Solution found in 14 moves!
Final State:

 5  2  6  7
2 9 9 2 2 1 1 8
 5  4  1  6

 5  4  1  6
7 5 5 9 9 6 6 6
 4  7  3  7

 4  7  3  7
1 4 4 8 8 2 2 5
 5  3  9  2

 5  3  9  2
8 6 6 4 4 9 9 7
 9  1  4  5

--------------------------------------------------------------------------------
--------------------
Initial State:

 1  8  2  8
3 3 5 1 3 4 6 9
 9  9  8  8

 9  5  7  3
6 3 9 1 4 6 8 3
 8  8  2  4

 9  9  8  2
1 2 8 8 2 5 1 2
 7  9  7  3
```

```
 4  8  4  8
3 1 2 2 3 4 2 2
 6  1  8  4

This state has 120 successors!
Search complete.
States expanded: 32187
Total Successors:  3862440
Average Branching Factor:  120.0
Solution found in 16 moves!
Final State:

 2  7  8  5
3 4 4 6 6 9 9 1
 8  2  8  8

 8  2  8  8
5 1 1 2 2 2 2 2
 9  3  1  4

 9  3  1  4
8 8 8 3 3 3 3 4
 9  4  9  8

 9  4  9  8
6 3 3 1 1 2 2 5
 8  6  7  7

--------------------------------------------------------------------------------
--------------------
Initial State:

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

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

 5  2  6  1
4 9 7 2 3 1 2 2
 5  3  4  3

 5  5  4  3
8 8 1 6 9 5 8 4
 6  1  1  6
```

```
This state has 120 successors!
Search complete.
States expanded: 20092
Total Successors:  2411040
Average Branching Factor:  120.0
Solution found in 14 moves!
Final State:

 5  2  2  5
8 8 8 7 7 2 2 1
 6  1  3  5

 6  1  3  5
2 4 4 8 8 4 4 9
 6  3  6  5

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

 4  7  4  1
9 5 5 1 1 2 2 2
 1  2  6  3

--------------------------------------------------------------------------------
--------------------
Initial State:

 7  4  8  9
1 1 6 6 9 1 2 8
 6  7  8  8

 2  2  8  2
3 2 8 4 3 4 1 1
 7  8  2  2

 8  6  4  8
4 3 7 9 1 3 1 3
 4  4  4  8

 8  8  2  4
6 3 4 8 3 3 3 7
 1  8  1  4

This state has 120 successors!
Search complete.
States expanded: 188
Total Successors:  22560
```

```
Average Branching Factor:  120.0
Solution found in 12 moves!
Final State:

 2  9  2  8
3 2 2 8 8 4 4 8
 7  8  8  8

 7  8  8  8
1 1 1 3 3 4 4 3
 6  8  2  4

 6  8  2  4
7 9 9 1 1 1 1 3
 4  8  2  4

 4  8  2  4
6 6 6 3 3 3 3 7
 7  1  1  4

--------------------------------------------------------------------------------
--------------------
Initial State:

 4  1  5  7
1 1 6 6 8 8 6 4
 5  5  7  1

 1  8  7  9
1 7 5 6 2 6 2 1
 4  4  8  7

 4  3  4  8
8 7 7 9 7 9 9 5
 4  4  7  1

 1  4  7  4
4 2 6 6 6 8 7 9
 3  8  1  4

This state has 120 successors!
Search complete.
States expanded: 58678
Total Successors:  7041360
Average Branching Factor:  120.0
Solution found in 13 moves!
Final State:
```

```
 9  4  1  4
2 1 1 1 1 7 7 9
 7  5  4  4

 7  5  4  4
6 8 8 8 8 7 7 9
 1  7  4  7

 1  7  4  7
4 2 2 6 6 6 6 4
 3  8  8  1

 3  8  8  1
7 9 9 5 5 6 6 6
 4  1  4  5

--------------------------------------------------------------------------------
--------------------
Average Number of Expanded States:  16155.1
Goal States Achieved:  10
Fails:  0
--------------------------------------------------------------------------------
--------------------
```

10 Report

10.1 Solution description

Uninformed Search Algorithm: Iterative Deepening
Iterative Deepening is a Depth First Search (DFS) algorithm with a depth-limit that will increase each time all nodes have been explored within the depth-limit. The algorithm explores a tree by expanding the deepest unexpanded node first up to a depth-limit, essentially exploring as far as possible along each branch until it hits the depth-limit before backtracking. Then once it has explored all of the nodes of the tree up to the depth-limit, it will increase the depth-limit until it has found an answer or reached a cut off. For this experiment, I gave a cutoff of 100,000 nodes which means the algorithm will explore 100,000 nodes before terminating and determining it cannot find an answer.

Informed Search Algorithm: A Star
A Star is an algorithim that expands the node with the lowest f(n) where f(n) = g(n) + h(n).
g(n) represents the cost from the root node to the current node. In this case, the cost is repesented by actions which have a cost of 1. Each action will be counted towards g(n) which represents the number of actions to get to a state from the original state.
h(n) represents the estimated cost from the current node to the goal node. In this case, h(n) uses the number of mismatched sides to estimate how many actions are needed to solve the puzzle.
A* is an informed search algorithm due to its use of h(n) to help guide its search.

10.1.1 Search space

The search space consists of all possible arrangements of the domino tiles on the grid.
State: A state is represented as an N x N grid containing N^2 domino tiles. Each dominio is a square containing a number on each side.
State Space: The total number of states is (N^2)! 2x2 grid: 4! = 24 states
3x3 grid: 9! = 362,880 states
4x4 grid: 16! 2.09 x 10^13 states
Start State: The search begins from a randomly shuffled arrangement of the tiles.
Goal State: The goal is a state where every side of a tile touching another side of a tile contains the same numeric value.

10.1.2 Successor function

The successor function generates all states that are reachable from the current state through a single action. In any given NxN configuration for the dominoes puzzle, the number of successors can be calculated as the number of combinations of 2 tiles from N^2, which is (N^2 * (N^2 - 1))/2
2x2 grid: 2^2 * (2^2 -1))/2 = 6 successors
3x3 grid: 3^2 * (3^2 - 1))/2 = 36 successors
4x4 grid: 4^2 * (4^2 - 1))/2 = 120 successors

In the dominoes puzzle all states of a given configuration have the same possible number of moves, so the number of successors for any given state is the same, making the average number of successors the same as the number of successors from any given state.

10.1.3 Heuristic function

The heuristic function utilized in this project is the following: h(n) = # of mismatched(different numeric values) touching sides within the domino puzzle. This heuristic was derived from the logic of a puzzle with 0 mismatched sides being equivalent to the goal state while a puzzle with a higher number of mismatched sides is increasingly further away from the goal state(The lower h(n) the better).

Ties: When 2 nodes contain the same f(n) value, the node with the lower g(n) will be chosen. If both nodes have the same f(n) and g(n) value, the random function is used to generate a random value for each node, and the node with the lowest random value is chosen.

The heuristic is not admissible, as the number of mismatched sides can be an overestimation on how many actions are necessary to solve the puzzle. For example, the puzzle may only require 2 tiles to be swapped (1 move) to be solved, but the number of mismatched sides is larger than 1.

Uninformed search algorithms like Iterative Deepening do not use a heuristic function to predict the best path towards the goal, instead they explore paths in a specficly defined manner. ID defines the manner of exploration as explore the deepest node.

10.2 Experimental results

Results are presented in the table below. Each search was repeated 10 times for each implemented method. The maximum number of expanded states per run was set to 100000.
Avg shows the average number of expanded states
Goal shows the number of times the goal was reached in each case

Fail #1 shows the number of times the search ended without reaching the goal
Fail #2 shows the number of times the search ended because the maximum number of expanded states was reached.

Although Iterative Deepening and A* successfully found the goal state in the 2x2 puzzle in all test cases, Iterative Deepening expanded fewer nodes on average.

The performance of Iterative Deepening decreased to finding the goal state in 9 cases while reaching the limit for expanded nodes in one case. A* continued to reach the goal state in all 10 test cases in the 3x3 puzzle utilizing significantly fewer expanded nodes on average.

Iterative Deepening failed to reach a goal state in the 4x4 configuration reaching the limit for expanded nodes each time while A* continued to reach every goal state.

Due to the heuristic utilized in the A* algorithm not being admissible, the A* algorithm will not be guarenteed to find the optimal solution, but it does outperform Iterative Deepening by prioritizing minimizing mismatches.

In all cases where Iterative Deepening failed, it failed by reaching the number of allowed expanded nodes. A* did not fail a single time.

Puzzle size	Uninformed		Iterative Deepening		Informed		A*	
	Avg	Goal	Fail #1	Fail #2	Avg	Goal	Fail #1	Fail #2
2x2	2.8	10	0	0	3.7	10	0	0
3x3	44166.2	9	0	1	228.5	10	0	0
4x4	100000	0	0	10	16155.1	10	0	0

[ ]: