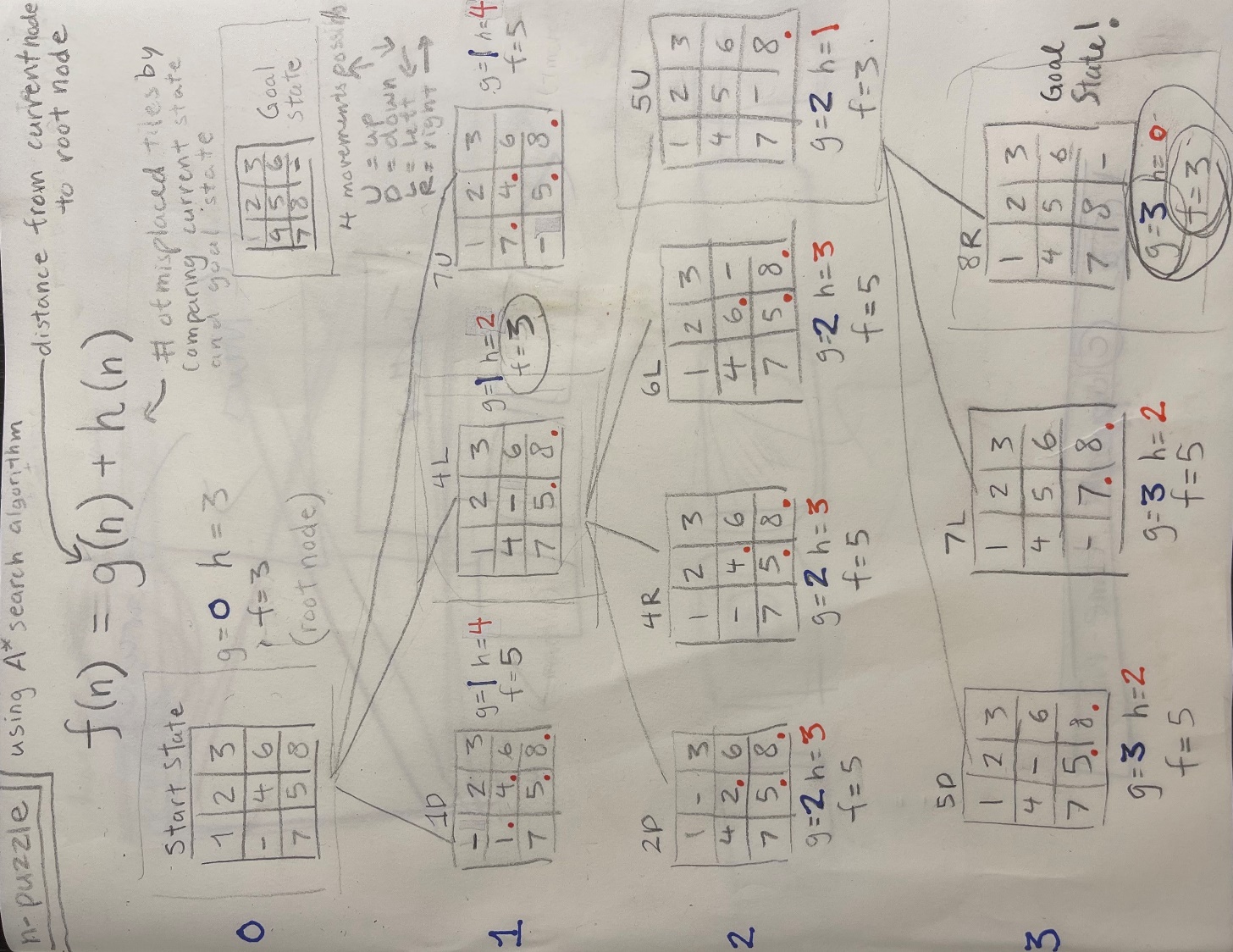
API Documentation

# Approaches to solve the n-puzzle

* Depth First Search (DFS)
  + Recursive technique to explore all possible states for the final state
  + Visits child nodes before adjacent nodes (slower)
* Iterative Deepening Search
  + Increments of maximum level threshold
* Breath First Search (BFS)
  + Iterative approach explores all adjacent positions before moving to next level
* **A\* Algorithm** 
  + May use BFS or DFS
  + Only some moves are explored (faster search for final state)
  + **f(n) = h(n) + g(n)**
    - Heuristic function
    - h(n) is the number of misplaced tiles by comparing current state with goal state
    - g(n) is distance from current state to start state

# Solving the n-puzzle with A\* Algorithm

To check all the possibilities to reach a finish state would take a long time. The A\* algorithm is based on heuristic methods that helps reach the final goal state solution without having to go through all the possibilities. The algorithm is used for efficient pathfinding between nodes. In the case of the n-puzzle it is used to find the quickest way from the goal state to the start state. It is based on g(n) which is the distance from the root node or start state puzzle, and h(n) which is the number of misplaced tiles by comparing the current state to goal state puzzle.



Notes

# Explanation of 3x3 n-puzzle solver

|  |  |
| --- | --- |
| **Code in Python 3.8.5** | **Explanation** |
| class Node: | Firstly, the object ‘node’ is created.  ‘Class’ is used to create objects with attributes and functionalities.  ‘Node’ works as the foundation of the data structure for linked lists and trees. |
| def \_\_init\_\_(self,hvalue,gvalue,fvalue): | ‘Def’ is used create the ‘\_init\_()’ function. This is executed to initiate the object with its properties of ‘self’, ‘hvalue’, ‘gvalue’, and ‘fvalue’. These will be used throughout. |
| self.hvalue = hvalue  self.gvalue = gvalue  self.fvalue = fvalue | Values are assigned to object properties. The ‘self’ variable represents the instance of the object itself. The rest of the values relate back to the A\* algorithm:  f(n) = h(n) + g(n)  This algorithm can be used to solve the n-puzzle. The ‘hvalue’ is the number of misplaced tiles by comparing current state and goal state of the puzzle. The ‘gvalue’ is the distance of the current node from the root node. The ‘fvalue’ is the calculated sum of ‘gvalue’ with the ‘hvalue’. |
| def child(self): | ‘Def’ is used to create a function. The ‘child(self)’ function is created. It is used to create child notes from the parent start state of the puzzle by referencing it’self’. To do this it must first move the blank or ‘-‘ of the n-puzzle. In an n-puzzle there are 4 possible moves: Up, Down, Left, or Right. |
| x,y = self.find(self.hvalue,’-’) | Assigns the values ‘x’ and ‘y’ as what the ‘self.find’ finds. The find() method is used to find the first occurrence of a specified value. Inside the bracket the first parameter value ‘self.hvalue’ is the value to be found. The second parameter value inside the bracket ‘-‘ is where to start the search. |
| move\_blank = [[x,y-1],[x,y+1],[x-1,y],[x+1,y]] | These are position values for moving the ‘-‘ blank space.  ‘x, y-1’ is down  ‘x, y+1’ is up  ‘x-1,y’ is left  ‘x+1,y’ is right  This list is created and called the ‘move\_blank’ name of the list. |
| children = [] | An empty list called ‘children’ is created. |
| for i in move\_blank:  child = self.shuffle(self.hvalue,x,y,i[0],i[1]) | A for loop for the ‘move\_blank’ list is created to execute the following set of statements, once for each item in the list.  shuffle() randomizes items of a list in place.  Within the loop, child is defined as a randomized list of items from the object instance ‘hvalue’ property, the ‘x’ value, the ‘y’ value, and the first and second ‘i’ value. |
| if child is not None:  child\_node = Node(child,self.level+1,0)  children.append(child\_node) | If the child is not none() a.k.a. the condition is not empty, then another child node will be created. This means the ‘gvalue’ grows by 1 because the distance from the current node to root node is one more away.  The new child node is appended the blank ‘children’ list that was created. |
| return children | The for loop will keep looping to create children nodes by shuffling the ‘-‘ through the 4 possible movements until there is none to be created. The appended child nodes created will then be returned as children. |
| def shuffle(self,puz,x1,y1,x2,y2): | ‘shuffle’ is defined with the following parameters called ‘self’, ‘puz’, ‘x1’, ‘y1’, ‘x2’, ‘y2’ |
| if x2 >= 0 and x2 < len(self.hvalue) and y2 >= 0 and y2 < len(self.hvalue): | An if condition is created. If ‘x2’ is greater or equal than 0 and less than the length of ‘self.hvalue’ (number of misplaced tiles) and ‘y2’ is greater or equal to 0 and less than the length of ‘self.value’ then… |
| temp\_puz = []  temp\_puz = self.copy(puz)  temp = temp\_puz[x2][y2]  temp\_puz[x2][y2] = temp\_puz[x1][y1]  temp\_puz[x1][y1] = temp  return temp\_puz | An empty list is created called ‘temp\_puz’ for temporary puzzle. A copy of the temporary form of the puzzle is created.  ‘temp’ is assigned as ‘temp\_puz[x2][y2]’, which is then assigned to ‘temp\_puz[x1],[y1]’, which is then assigned by to ‘temp’  The created ‘temp\_puz’ list is returned. |
| else:  return None | Else ‘None’ is returned if the position value of the blank space is moved to out of the boundaries of the puzzle. |
| def copy(self,start\_state): | The ‘copy’ function is created to create a similar matrix of the given node with previous parameter ‘self’ and new parameter ‘root’. |
| temp = [] | An empty list is created called ‘temp’. |
| for i in start\_state:  t = []  for j in i:  t.append(j)  temp.append(t)  return temp | A nested loop of a for loop within a for loop is created for to execute the following set of statements, once for each item in the list, and to ‘append’ or save an element to the end of the list each time. The list within the list creates a matrix.  For the ‘start\_state’, ‘t’ is assigned a blank list. In the for loop for ‘i’, the ‘j’ element is appended to the end of the ‘t’ list. The ‘t’ list is then appended to the ‘temp’ list.  The resulting ‘temp’ list is returned. This is to create a matrix.  Matrix is created from user input. A Python matrix is a two-dimensional rectangular array of data stored in rows and columns. This matrix will be used to represent the puzzle. |
| def find(self,puz,x): | The ‘find’ function is created to find the position of the blank space of the given puzzle start state with previous parameter ‘self’ and new parameter ‘puz’ and ‘x’. |
| for I in range(0,len(self.data)):  for j in range(0,len(self.data)):  if puz[i][j] == x:  return I,j | A nested loop is used to find the position.  For ‘i’ in range from 0 to the length of the ‘self.hvalue’ (number of misplaced tiles) is the range for ‘j’ from 0 to the length of the ‘self.hvalue’ is if the puzzle indexed values of ‘i’ and ‘j’ are equal to x then return ‘i’ and ‘j’. This is the starting position of the “-“ blank space. |
| class Puzzle: | Secondly, the object ‘Puzzle’ is created.  ‘Class’ is used to create objects with attributes and functionalities.  ‘Node’ works as the foundation of the data structure for linked lists and trees. |
| def \_\_init\_\_(self,size): | ‘Def’ is used create the ‘\_init\_()’ function. This is executed to initiate the object with its properties of ‘self’ and ‘size’. |
| self.n = size  self.open = []  self.closed = [] | Values are assigned to object properties. The ‘self’ variable represents the instance of the object itself. The puzzle is initialized by the specified size. The size is defined as 3 in the following. Two lists are created. The open list is a collection of all generated nodes. The closed list is a collection of all expanded nodes or puzzle variations that were already searched. |
| def accept\_start\_state(self): | ‘Def’ is used create the accept\_start\_state’ function. This accepts the start state from the user. |
| puz = [] | Empty list for ‘puz’ or puzzle is created. |
| for i in range(0,self.n):  temp = input().split(" ")  puz.append(temp)  return puz | Asks user for input, splits it with a space, and appends this input, and returns it as ‘puz’ matrix. |
| def goal\_state(self): | Defines the ‘goal\_state’ or the goal state of the puzzle. |
| puz = [['1', '2', '3'], ['4', '5', '6'], ['7', '8', '-']] | This matrix is the goal state. Some n-puzzle has the ‘-‘ as the [1,1] position, I have it at the end. |
| return puz | Returns this as ‘puz’ matrix. |
| def f(self,start,goal):  return self.h(start.hvalue,goal)+start.gvalue | A\* Algorithm to calculate heuristic value f(x) = h(x) + g(x). |
| def h(self,start,goal):  temp = 0  for i in range(0,self.n):  for j in range(0,self.n):  if start[i][j] != goal[i][j] and start[i][j] != '-':  temp += 1  return temp | Compares the difference between the inputted start state matrix of the user and the defined goal state matrix. Calls back to ‘temp’ from earlier. If the start state does not equal the goal state then temp is assigned 1, otherwise temp is returned as 0. |
| def process(self): | New function for ‘process’ is defined. This process will be run to accept the start and goal state. |
| print("Hello, here is a solver for a 3 x 3 sliding n-puzzle :)")  print("Enter the start state 3 by 3 matrix with numbers ranging from 1 to 8.\n" "Input '-' to represent the blank space.\n" "Put a space between each input. Press the 'enter' key after each row of three inputs.\n" "For example:\n"'1', '2', '3\n''-', '4', '6\n' '7', '5', '8\n') | Print instructions for start state input. |
| start = self.accept\_start\_state() | Defines ‘start’ as input |
| print("Goal state of matrix is\n" '1', '2', '3\n''4', '5', '6\n' '7', '8', '-') | Prints goal state for user to see. |
| goal = self.goal\_state() | Defines ‘goal’ as the goal state matrix listed above. |
| start = Node(start,0,0) | Defines start as node with 0,0 |
| start.fvalue = self.f(start,goal)  self.open.append(start)  print("\n\n") | The ‘start.fvalue’ is assigned to the ‘self.f(start, goal)’ It is appended to ‘start’.  Print a new line. |
| while True:  cur = self.open[0]  print("...next")  for i in cur.hvalue:  for j in i:  print(j,end=" ")  print("") | While matrixes are generating, for each new matrix is generated print “… next”. |
| if(self.h(cur.hvalue,goal) == 0):  break  for i in cur.child():  i.fvalue = self.f(i,goal)  self.open.append(i)  self.closed.append(cur)  del self.open[0] | If statement checks for if hvalue = 0. In the A\* algorithm the h value is 0 for the goal state, so there’s no difference left between start and goal state. It sorts the variations of the puzzle sorted into the two types of lists, closed and open. The self.open variable is deleted. |
| self.open.sort(key = lambda x:x.fvalue,reverse=False) | This sorts the open list based on the f value. The variations of the puzzle from start to goal state has a lower f value than other variations. |
| puz = Puzzle(3) | Defines puzzle as a 3x3 puzzle to solve |
| puz.process() | The definition is run. |
| print("Done!") | Prints ‘Done!’ |

Refences for n-puzzle

Kendall, G., Parkes, A., & Spoerer, K. (2008). A survey of NP-complete puzzles. *ICGA Journal*, *31*(1), 13-34.

Yeo. (1983). The physics of the N-puzzle: a case study in artificial intelligence.

8 puzzle problem using A\* search algorithm in English\ Artificial intelligence tutorial English.

<https://www.youtube.com/watch?v=wJu3IZq1NFs&t=359s>

<https://en.wikipedia.org/wiki/Sliding_puzzle>

<https://brilliant.org/wiki/a-star-search/>

<https://blog.goodaudience.com/solving-8-puzzle-using-a-algorithm-7b509c331288>

<https://www.code-learner.com/how-to-develop-a-digital-n-puzzle-game-with-python/>

<http://www.sfu.ca/~tjd/310summer2019/a1.html>

<https://github.com/andavies/n-puzzle>

<https://github.com/tristanpenman/n-puzzle>

[https://en.wikipedia.org/wiki/A\*\_search\_algorithm](https://en.wikipedia.org/wiki/A*_search_algorithm)

<https://tristanpenman.com/demos/n-puzzle/>

<https://medium.com/swlh/solving-eight-puzzle-problem-and-its-visualization-in-tkinter-8b90f80ba178>

<https://api.slack.com/authentication/migration>

Refences for understanding the programming

<https://www.pythonforbeginners.com/basics/classes-in-python>

<https://stackoverflow.com/questions/625083/what-init-and-self-do-in-python>

<https://www.w3schools.com/python/gloss_python_class_init.asp>

<https://www.tutorialspoint.com/python/python_nodes.htm>

<https://www.w3schools.com/python/ref_string_find.asp>

<https://www.w3schools.com/python/ref_list_append.asp>

<https://www.w3schools.com/python/python_for_loops.asp>

<https://stackoverflow.com/questions/45577114/a-star-algorithm-open-and-closed-lists>

<https://www.pythonforbeginners.com/basics/classes-in-python>

<https://realpython.com/python-command-line-arguments/>