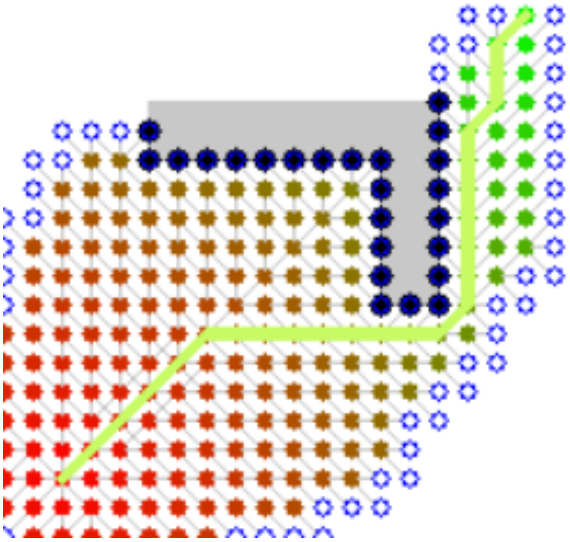
EAI 320 - Intelligent systems

Practical 2 - A* Search



[2]

Author:
V Thalla

Student number:
16053525

February 22, 2018

Contents

1 Introduction

This practical introduces the **Uniform Cost Search, Greedy Best Search and A* Search** to the students of EAI 320. There are several applications for these algorithms, like in Video games where the computer AI has to try to search for the player character. As such, the importance of such algorithms is high while learning fundamentals of Artificial Intelligence.
These search algorithms are required to be designed independently per individual and implemented in Python. There is a maze environment given with clearly stated starting and ending points. The aim is to navigate through the maze, while avoiding the obstacles, and find the most optimal path such the final route has the lowest distance possible. This search is implemented in all the above mentioned manners and compared.

2 Problem Statement

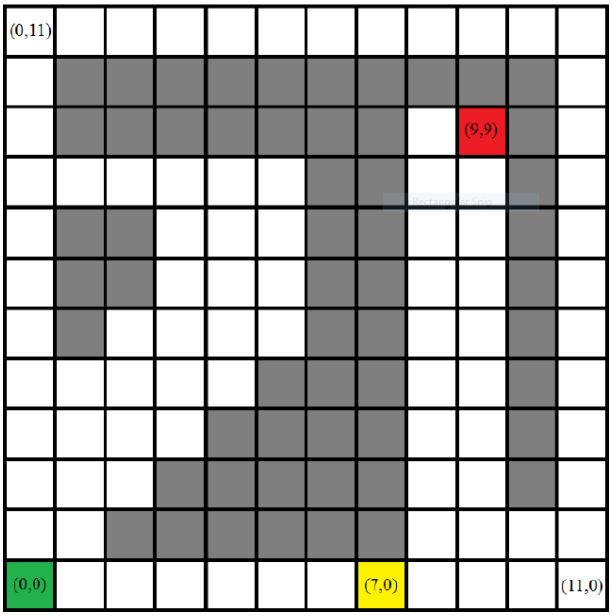


Figure 1.1: Maze Environment for searching

The above maze environment was given as the problem. The starting point([0,0]) and ending point([9,9]) are highlighted in green and red respectively. The Obstacles, which can be viewed as walls are coloured in grey. This maze has to be processed by Uniform Cost search, Greedy Best search and A* search algorithms.

The maze is used as an input for the Searches through the 2D array below:

```
[[0,0,0,0,0,0,0,0,0,0,0,0,0],
[0,0,0,0,0,1,1,1,0,1,1,0],
[0,1,0,0,0,0,1,1,0,1,1,0],
[0,1,1,0,0,0,0,0,0,1,1,0],
[0,1,1,1,0,0,0,0,0,1,1,0],
[0,1,1,1,1,0,0,0,0,1,1,0],
[0,1,1,1,1,1,1,1,1,1,1,0],
[0,1,1,1,1,1,1,1,1,1,1,0],
[0,0,0,0,0,0,0,0,0,0,1,0],
[0,0,0,0,0,0,0,0,0,0,1,0],
[0,0,1,1,1,1,1,1,1,1,1,0],
[0,0,0,0,0,0,0,0,0,0,0,0]]
```

1 = wall
0 = open block
The initial task is to implement these three algorithms through Python code and then represent the ending point, solution path, expanded nodes, visited nodes, walls and starting point in a 2D plot for comparison. The number of nodes expanded also needs to be provided in order to measure the performance of each search in the context of this problem.
There is an additional task of turning the yellow block([7, 0]), an open block in the initial task, into a wall and comparing the new solutions to the initial solutions in the same way as the initial task.

3 Methodology

Each Search instance takes a maze 2D array as an input, clones it inside searches and works on the clone to find a solution and plots the solution on the clone.
All the Searches follow a very simple Algorithm[1]:

```
input: point start, point end
begin
root = new Node(start)
queue = new Priority Queue
Add root to queue with cost function as priority
// eg. Uniform cost priority = 0
While the queue is not empty
    Current = pop the maximum priority element from the queue
    Add Current to a closet set(list)
    if Current.coordinates == end:
        print the path and exit
    for i in Current.children:
        Insert i into the queue if it is not in the closed set, with the cost
            function as priority
end
```

It also uses a simple **explore** function to determine all neighbours (node's children) to add to the open set. This was implemented such that the children were already checked against the closed set for redundancies that would result in infinite repetition loops. So the combination of the two algorithms builds a tree of nodes that is explored till the solution is found.

3.1 Uniform Cost Search:

Uniform Cost Search is an uninformed search that expands radially with a path cost that accumulates as it expands further towards the goal. This results in an order in the queue such that each of the neighbour to an expanded node is traversed before moving further out, unless the neighbour is a wall. Here, the accumulated path distance of the current node is used as the cost function in the above Search algorithm.

3.2 Greedy Best Search:

Greedy Best Search is an informed search that expands knowingly towards the goal with the euclidean distance decreasing as it progresses. It disregards the other neighbours that could be expanded if there is a clear cut shorter route to the goal. However, once it reaches an obstacle in the path, it tries to go around, while satisfying the shortest euclidean distance criterion. Here, the euclidean distance from the current node to the goal is used as the cost function in the above Search algorithm.

3.3 A* Search:

A* Search is a mix of both the Uniform cost and Greedy Best searches. Here, the accumulated path distance of the current node, combined with it's euclidean distance from the goal is used as the cost function in the above Search algorithm. This combines the optimal path-finding method of Uniform Cost with the spear-like direction expansion to find the route in a more informed way than Uniform Cost while being less aggressive than Greedy Best.

There is embedded code inside the above general search function that marks all the expanded nodes. Since this is a task-specification and not a real problem to solve, it has been omitted in the Search algorithm pseudocode above.
The final tree is then traversed backwards from the goal node, through the node parents, till the starting point is reached. The visited nodes, from the open set, are marked and the whole solution is printed as the result. The amount of nodes counted in the tree is also printed, as determined by a Breadth-First search of the tree.
The maze given in the problem is run through all the Search algorithms twice: once with [7,0] as an open block and once with [7,0] as a wall. The two solutions are then compared.

4 Results

4.1 Question 1d - Results with [7,0] as an open block

```
CONSOLE RESULTS:
Uniform Cost Search: 162 nodes
Frontier = 4 nodes
| - - - - - - - - - - - - - - |
| S X X X X X X X X | 0 0 |
| * X X X X | | | X | | 0 |
| * | X X X X | | X | | 0 |
| * | | X X X X X X | | 0 |
| * | | | X X X X X | | 0 |
| * | | | | X X X X | | 0 |
| * | | | | | | | | | 0 |
| * | | | | | | | | | 0 |
| X * * * * * * * X X | 0 |
| X X X X X X X X * G | 0 |
| X X | | | | | | | | 0 |
| X X X X X X X V 0 0 0 0 |
| - - - - - - - - - - - - - - |

Greedy Best Search: 131 nodes
Frontier = 36 nodes
| - - - - - - - - - - - - - - |
| S V X X X X X X X | 0 0 |
| V * X X X | | | X | | 0 |
| * | X X X X | | X | | 0 |
| * | | X X X X X X | | 0 |
| * | | | X X X X X | | 0 |
| * | | | | X X X X | | 0 |
| * | | | | | | | | | 0 |
| * | | | | | | | | | 0 |
| V * V V V V V V V V | 0 |
| V V * * * * * * * G | 0 |
| 0 V | | | | | | | | | 0 |
| 0 0 0 0 0 0 0 0 0 0 0 0 0 |
| - - - - - - - - - - - - - - |
```

```
A* Search: 138 nodes
Frontier = 32 nodes
| - - - - - - - - - - - - - - - |
| S X X X X X X X V | 0 0 |
| * X X X X | | | X | | 0 |
| * | X X X X | | X | | 0 |
| * | | X X X X X X | | 0 |
| * | | | X X X X X | | 0 |
| * | | | | X X X X | | 0 |
| * | | | | | | | | | 0 |
| * | | | | | | | | | 0 |
| V * * * * * * * * V | 0 |
| V V V X X X V V X G | 0 |
| 0 0 | | | | | | | | 0 |
| 0 0 0 0 0 0 0 0 0 0 0 0 0 |
| - - - - - - - - - - - - - - - |
```

4.2 Question 2d - Results with [7,0] as a wall

```
CONSOLE RESULTS:
Uniform Cost Search: 179 nodes
Frontier = 3 nodes
| - - - - - - - - - - - - - - - |
| S * * * * * * * * * * X |
| X X X X X | | | X | | * |
| X | X X X X | | X | | * |
| X | | X X X X X X | | * |
| X | | | X X X X X | | * |
| X | | | | X X X X | | * |
| X | | | | | | | | | * |
| | | | | | | | | | | * |
| X X X X X X X X X V | * |
| X X * * * * * * * G | * |
| X * | | | | | | | | | * |
| X X * * * * * * * * * X |
| - - - - - - - - - - - - - - - |
```

```
Greedy Best Search: 132 nodes
Frontier = 49 nodes
| - - - - - - - - - - - - - - - |
| S V V 0 0 0 0 V X * * V |
| V * V V V | | | * | | * |
| V | * V X X | | * | | * |
| 0 | | * X X X X * | | * |
| 0 | | | * X X X * | | * |
| 0 | | | | * * * X | | * |
| 0 | | | | | | | | | | * |
| | | | | | | | | | | | * |
| 0 V V V V V V V V V | * |
| V V * * * * * * * G | * |
| V * | | | | | | | | | * |
| V V * * * * * * * * * X |
| - - - - - - - - - - - - - - - |

A* Search: 160 nodes
Frontier = 29 nodes
| - - - - - - - - - - - - - - - |
| S * * * * * * * * * * X |
| X X X X X | | | X | | * |
| X | X X X X | | X | | * |
| X | | X X X X X X | | * |
| X | | | X X X X X | | * |
| X | | | | X X X X | | * |
| X | | | | | | | | | | * |
| | | | | | | | | | | | * |
| 0 V V V V V V V V V | * |
| V V * * * * * * * G | * |
| V * | | | | | | | | | * |
| V X * * * * * * * * * X |
| - - - - - - - - - - - - - - - |
```

4.3 Results with [7,0] and [0, 9] as a wall

```
CONSOLE RESULTS:
Uniform Cost Search: 95 nodes
Frontier = 0 nodes
No Uniform Cost Search solution found
| - - - - - - - - - - - - - - |
| S X X X X X X X X | 0 0 |
| X X X X X | | | X | | 0 |
| X | X X X X | | X | | 0 |
| X | | X X X X X X | | 0 |
| X | | | X X X X X | | 0 |
| X | | | | X X X X | | 0 |
| X | | | | | | | | | 0 |
| | | | | | | | | | | 0 |
| 0 0 0 0 0 0 0 0 0 0 0 | 0 |
| 0 0 0 0 0 0 0 0 0 0 G | 0 |
| 0 0 | | | | | | | | 0 |
| 0 0 0 0 0 0 0 0 0 0 0 0 0 |
| - - - - - - - - - - - - - - |

Greedy Best Search: 95 nodes
Frontier = 0 nodes
No Greedy Best Search solution found
| - - - - - - - - - - - - - - |
| S X X X X X X X X | 0 0 |
| X X X X X | | | X | | 0 |
| X | X X X X | | X | | 0 |
| X | | X X X X X X | | 0 |
| X | | | X X X X X | | 0 |
| X | | | | X X X X | | 0 |
| X | | | | | | | | | 0 |
| | | | | | | | | | | 0 |
| 0 0 0 0 0 0 0 0 0 0 0 | 0 |
| 0 0 0 0 0 0 0 0 0 0 G | 0 |
| 0 0 | | | | | | | | 0 |
| 0 0 0 0 0 0 0 0 0 0 0 0 0 |
| - - - - - - - - - - - - - - |
```

```
A* Search: 95 nodes
Frontier = 0 nodes
No A* Search solution found
| - - - - - - - - - - - - - - - |
| S X X X X X X X X | 0 0 |
| X X X X X | | | X | | 0 |
| X | X X X X | | X | | 0 |
| X | | X X X X X X | | 0 |
| X | | | X X X X X | | 0 |
| X | | | | X X X X | | 0 |
| X | | | | | | | | | 0 |
| | | | | | | | | | | 0 |
| 0 0 0 0 0 0 0 0 0 0 0 | 0 |
| 0 0 0 0 0 0 0 0 0 0 G | 0 |
| 0 0 | | | | | | | | | 0 |
| 0 0 0 0 0 0 0 0 0 0 0 0 0 |
| - - - - - - - - - - - - - - - |
```

Key:

- * = Path blocks
- — = wall blocks
- 0 = Open blocks
- V = Visited blocks
- X = Expanded blocks

Table 1.1: 4.1 Results - Question 1d

Search Type	Number of Branches	Frontier nodes
Uniform Cost Search	162	4
Greedy Best Search	131	36
A* Search	138	32

Table 1.2: 4.2 Results - Question 2d

Search Type	Number of Branches	Frontier nodes
Uniform Cost Search	179	3
Greedy Best Search	132	49
A* Search	160	29

Table 1.3: 4.3 Results - Failure

Search Type	Number of Branches	Frontier nodes
Uniform Cost Search	179	3
Greedy Best Search	132	49
A* Search	160	29

5 Discussion

5.1 Question 1e - Shortest path discussion with [7, 0] as an open block

The Greedy Best search goes in the right direction in terms reaching the goal optimally but is hindered by the walls and tries to go around and find a different way to the goal, which eventually ends up being inefficient and doesn't satisfy the optimal path criterion. The Uniform Cost search and the A* star search seem to find the shortest path correctly, albeit with different processing time. The Uniform cost search seems to always be able to reach the goal correctly due to it's radial expansion method so, regardless of the obstacles, it will always find the solution as long as it exists. The A* star search basically acts a directed Uniform Cost search in that it doesn't waste time by trying everything while expanding. It cleverly uses the Greedy Best's given direction in combination with the Uniform cost accumulated distance to make a decision on which node to pick next so it's not as brute forced as Uniform cost.

5.2 Question 2e - Shortest path discussion with [7, 0] as a wall

Again, Uniform Cost search and A* search find the most optimal path while there isn't a change in Greedy Best search. Since Greedy Best was already taking the "go around" way, the change in [7,0] had no effect on this search method. A* star turns out to be the best here again but both Uniform cost and A* searches have the same optimal path.

5.3 Completeness

All the searched are implemented such that if there is a solution, they will definitely find it. The only case where they would fail is if they are boxed in with no way to the solution(for example, if both [7,0] and [0, 9] are walls). In this case, they mark all the visited nodes which are the same in all cases, which vouches for the completeness of the algorithms (they will search till the end before concluding that the solution does not exist).

5.4 Time performance

All the methods will be compared for time performance with the number of branches in the search tree as the criterion for comparison. As we can see from Table 1.1 and Table 1.2, Greedy Best is always the fastest(131 nodes for Question 1d and 132 nodes for Question 2d) since it doesn't waste time on nodes that might not be on the path. Uniform Cost always takes the longest(162 nodes for Question 1d and 179 nodes for Question 2d). Again, this is due to it's brute force radial expansion method. A* turns out be the second fastest(138 nodes for Question 1d and 160 nodes for Question 2d) due to it's approach of using best of both so it's speed tends to be higher than Uniform cost but generally lower than Greedy Best.

5.5 Space performance

All the methods will be compared for space performance with the number of nodes in the Frontier of the search tree. Uniform Cost seems to be the lowest in terms of squandering memory(4 nodes for Question 1d and 3 nodes for Question 2d) as it's expansion rate is low. The total memory used is also the least in both cases as seen from Table 1.1 and Table 1.2. A* seems to have a mediocre amount of memory wastage(32 nodes for Question 1d and 29 nodes for Question 2d) and Greedy Best has the highest number of frontier nodes(36 nodes for Question 1d and 49 nodes for Question 2d). Greedy Best will generally have a lot of nodes on the frontier due to the fact that it disregards the expandable nodes if they are not closest by euclidean distance. Since this directed approach is also present in A* search, it tends to have much more frontier nodes than Uniform Cost but generally lower than Greedy Best. This means that Greedy Best can cause memory overflows if the problem is scaled up to higher order of magnitudes, taking up resources that should be available for other purposes. On the other hand, A* and Uniform Cost are more scalable in terms of memory space.

5.6 Optimality

Uniform cost search will provide the most optimal path properly in all situations but the amount of time it takes is very high. Therefore this should be used in a search with little to no information of the end goal, which needs to be discovered. A small environment is also optimal. Greedy Best search should be used when the end goal is known there is a time constraint. It favours an environment with a relatively straightforward solution and little to no obstacles in the final solution path. This will yield a fast, mostly straight lined solution. A* search should be used when a situation where the end goal is known but with a lot of obstacles and complex routes on the solution path. This will lead the A* to be the fastest solution in an environment where both the uninformed and informed search approaches can be used. This is perfect for enemy AI in video games since these agents have to find the player character in the shortest time possible, while keeping memory usage in check, so resources can be used efficiently in a "frame-rate sensitive" context.

6 Conclusion

As can be seen through the results, A* turns out to be the intermediate best in both situations, setting a middle ground in the memory used and time to get to the result. It's superior in most general case applications where an informed search can be employed. However, as discussed earlier, Uniform Cost and Greedy Best searches have their applications depending on the problem at hand.

Bibliography

[1] Siddharth Agrawal. *ARTIFICIAL INTELLIGENCE–UNIFORM COST SEARCH(UCS)*. 2015. URL: https://algorithmicthoughts.wordpress.com/2012/12/15/artificial-intelligence-uniform-cost-searchucs/.

[2] Subhrajit Bhattacharya. *Astar progress animation*. 2011. URL: https://en.wikipedia.org/wiki/A*_search_algorithm#/media/File:Astar_progress_animation.gif.

7 Appendix: Python code

```
"""
Created on Fri Feb 16 18:04:05 2018

@author: Vishal Thalla(16053525)
"""
"""
TODO:
    adapt Node class for this practical
    create Uniform cost function:**DONE**
    create Greedy Best search:**DONE**
    create A* search: **DONE**
FIXME:**DONE**
    make maze use the practical version
    adapt Explore for practical version maze
FIXME:**DONE**
    fix the greedy search endless loop
    fix the deviation if possible
    fix the branch counting problem
"""

import queue
import math

class Node(object): #needed for new style class
    def __init__(self, coords, parent, dist):
        self.x = coords[0]
        self.y = coords[1]
        self.parent = parent
        self.pathDistance = dist #The cost
        self.children = [] #neighbours to expand into

    def addChild(self, child = []):
        child[0].parent = self
        self.children.append(child[0])
    def __lt__(self, other):
        return self.pathDistance < other.pathDistance

class AStarNode(Node):
    def __init__(self, coords, parent, pathDistance, euclidean):
        self.x = coords[0]
        self.y = coords[1]
        self.parent = parent
        self.pathDistance = pathDistance
        self.euclidean = euclidean #extra field to since A* combines h(n) and
            g(n)
        self.children = []

    def __lt__(self, other):
```

```
        return self.pathDistance + self.euclidean < other.pathDistance +
            other.euclidean

class Tree(object):
    """
    maze is a 2D array
    Will offer no checking for correct input
    """
    def __init__(self, maze):
        self.explored = []
        self.maze = maze #Shallow copy but fine for this practical
        #Potential porblem if the maze array passed in is edited outside

    """
    returns deep copy clone of self.maze
    """
    def mazeClone(self):
        clone = [[0 for i in range(len(self.maze))] for j in
            range(len(self.maze[0]))]
        for i in range(len(self.maze)):
            for j in range(len(self.maze[i])):
                if (self.maze[i][j] == 1):
                    clone[i][j] = "|" #| = wall
        return clone

    """
    conveniently returns the boolean of whether given coords exist in
        self.explored which is the closet set
    """
    def isExplored(self, coords):
        out  = False
        i = 0
        while (i < len(self.explored) and out == False):
            if (self.explored[i][0] == coords[0] and self.explored[i][1] ==
                coords[1]):
                out = True
            i += 1
        return out
    """
    returns straight line distance from p1 to p2
    """
    def getSLD(self, p1, p2):
        return math.sqrt(math.pow(p2[0]-p1[0], 2) + math.pow(p2[1]-p1[1], 2))

    """
    returns number of nodes in the tree
    """
    def DFScount(self, node):
        rest = [i for i in node.children]
        count  = 1
```

```
    for i in range(len(rest)):
        count += self.DFScount(rest[i])
    return count

def UniformCostSearch(self, start, end):
    return self.search(1, start, end)

def GreedyBestSearch(self, start, end):
    return self.search(2, start, end)

def AStarSearch(self, start, end):
    return self.search(3, start, end)

"""
conveniently creates the practical's result plot in a maze clone
"""
def printPath(self, node, end, viewed, solution, found):

    solution[end[0]][end[1]] = "G" #G = goal
    if found == True and (node.x != end[0] or node.y != end[1]):
        solution[node.x][node.y] = "*"
    curr = node.parent

    while (curr != None):
        if (curr.parent == None):
            solution[curr.x][curr.y] = "S" #S = start
        elif found == True:
            solution[curr.x][curr.y] = "*" #* = path
        curr = curr.parent

    while (not viewed.empty()):
        vnode = viewed.get()
        if (solution[vnode.x][vnode.y] == 0):
            solution[vnode.x][vnode.y] = "V"#V = visited

    self.printMaze(solution)

    self.explored = []

"""
prints a given maze(2D array) with contents intact
"""
def printMaze(self, maze):
    out = "| "
    for i in range(len(maze[0])):
        out+= "- "
    out += "|\n"
    for x in range(len(maze)):
        out += "| "
        for y in range(len(maze[x])):
```

```
            out += str(maze[x][y]) + " "
        out += "|\n"
    out += "| "
    for j in range(len(maze[0])):
        out+= "- "
    out += "|\n"
    print(out)

"""
Implements all the searches in 1 function
Method:
    case 1: Uniform cost
    case 2: Greedy Best search
    case 3: A* search

"""
def search(self, method, start, end):
    if method < 1 or method > 3:
        print("Invalid Search method @ func search")
        return
    root = None
    title = ""
    if method == 1:
        root = Node(start, None, float(0))
        title = "Uniform Cost Search"
    elif method == 2:
        root = Node(start, None, float(self.getSLD(start, end)))
        title = "Greedy Best Search"
    elif method == 3:
        root = AStarNode(start, None, float(0), float(self.getSLD(start,
            end)))
        title = "A* Search"

    unvisited = queue.PriorityQueue() #open set or frontier
    self.explored = [] #closed set
    solution = self.mazeClone()
    unvisited.put(root)
    while (not unvisited.empty()):
        curr = unvisited.get()
        if (self.isExplored([curr.x, curr.y])):
            continue; # skip cycle if this block has already been expanded
        solution[curr.x][curr.y] = "X"
        self.explored.append([curr.x, curr.y])

        if (curr.x == end[0] and curr.y == end[1]):
            print(title+ ": " + str(self.DFScount(root)-1) + " nodes")
            print("Frontier = " + str(unvisited.qsize()) + " nodes")
            return self.printPath(curr, end, unvisited, solution, True)
        children = self.explore(curr) # all neighbour coordinates that
            can be expanded into
```

```
        for i in range(len(children)):
            if method == 1:
                curr.addChild([Node(children[i], None, curr.pathDistance +
                    float(self.getSLD([curr.x, curr.y], children[i])))])
            elif method == 2:
                curr.addChild([Node(children[i], None,
                    float(self.getSLD(children[i], end)))])
            elif method == 3:
                curr.addChild([AStarNode(children[i], None,
                    float(curr.pathDistance + self.getSLD([curr.x, curr.y],
                    children[i])), float(self.getSLD(children[i], end)))])
        for j in range(len(children)):
            unvisited.put(curr.children[j])

    print(title+ ": " + str(self.DFScount(root)-1) + " nodes")
    print("Frontier = " + str(unvisited.qsize()) + " nodes")
    print("No " + title + " solution found")
    return self.printPath(curr, end, unvisited, solution, False)

"""
returns coords of all expandable nodes
add in clockwise fashion from up
"""
def explore(self, node):
    x = node.x
    y = node.y
    out = []

    if (y < len(self.maze[0])-1):
        if (not self.isExplored([x, y+1])) and self.maze[x][y+1] ==
            0:#check up
            out.append([x, y+1])
        if x < len(self.maze)-1 and (not self.isExplored([x+1, y+1])) and
            self.maze[x+1][y+1] == 0:#check diag up, right
            out.append([x+1, y+1])

    if (x < len(self.maze)-1 and self.maze[x+1][y] == 0 and not
        self.isExplored([x+1, y])):#check right
        out.append([x+1, y])

    if (y > 0):
        if x < len(self.maze)-1 and (not self.isExplored([x+1, y-1])) and
            self.maze[x+1][y-1] == 0:#check diag down, right
            out.append([x+1, y-1])
        if (not self.isExplored([x, y-1])) and self.maze[x][y-1] ==
            0:#check down
            out.append([x, y-1])
        if x > 0 and (not self.isExplored([x-1, y-1])) and
            self.maze[x-1][y-1] == 0:#check diag down, left
            out.append([x-1, y-1])
```

```
        if (x > 0 and self.maze[x-1][y] == 0 and not self.isExplored([x-1,
            y])):#check left
            out.append([x-1, y])

        if (y < len(self.maze[0])-1):
            if x > 0 and (not self.isExplored([x-1, y+1])) and
                self.maze[x-1][y+1] == 0:#check diag up, left
                out.append([x-1, y+1])
        return out


maze = [[0,0,0,0,0,0,0,0,0,0,0,0],
        [0,0,0,0,0,1,1,1,0,1,1,0],
        [0,1,0,0,0,0,1,1,0,1,1,0],
        [0,1,1,0,0,0,0,0,0,1,1,0],
        [0,1,1,1,0,0,0,0,0,1,1,0],
        [0,1,1,1,1,0,0,0,0,1,1,0],
        [0,1,1,1,1,1,1,1,1,1,1,0],
        [0,1,1,1,1,1,1,1,1,1,1,0],
        [0,0,0,0,0,0,0,0,0,0,1,0],
        [0,0,0,0,0,0,0,0,0,0,1,0],
        [0,0,1,1,1,1,1,1,1,1,1,0],
        [0,0,0,0,0,0,0,0,0,0,0,0]]

tree = Tree(maze)
tree.UniformCostSearch([0, 0], [9, 9])
tree.GreedyBestSearch([0, 0], [9, 9])
tree.AStarSearch([0, 0], [9, 9])
```