**Artificial Intelligence Course**

**Assignment 02**



**Dr.Hashim Yasin**

Student Name: Abdul Salam & Taha Raja

Roll No: 18F-0326 & 18F-0137 Section: CS(6B)

Date performed: 28/04/2021

# Question no 01:

1. **Data Structure Used:**

To store nodes we used 2D array or list in a list so each row is in a list and there is a parent list to store all lists, for traverse we used 3D arrays or list to maintain backtrack option in stack 2D list is maintained for visited list and for up, right and diagonal move 2D list is maintained. Heuristic values are also stored in the form of 2D array or list.

Heuristic valued are calculated manually by Manhattan distance. Greedy-BFS traverse by considering heuristic values where A\* and IDA\* use both path cost and heuristic values.

1. **Searching Algorithm Best for this Problem:**

For given problem greedy-BFS algorithm is best as it has low cost as compare to A\* and IDA\*, G-BFS also reached its destination without using backtrack or stack where A\* and IDA\* used back track option when they struck in obstacles, in short G-BFS reached it goal in low cost with out interrupting with obstacles.

1. **Instructions:**

Heuristic values are calculated manually by using Manhattan formula, to run a program input file path should be entered through code (hard coded), all functions or searches are calling by hardcode, to change iterations in IDA\* number of iterations should be changed manually from code.

# Heuristic Values Screenshot:

# **Searches screenshot:**

# G-BFS:

# A\*:

# IDA\*:

# Python Code :

import copy  
  
class Grid:  
  
 #class elements  
 col = 0  
 row = 0  
 startRow = 0  
 startCol = 0  
 goalRow = 0  
 goalCol = 0  
 grid = 0  
  
 def insertGrid(self):  
 f = open("grid.txt", "r")  
 res = []  
  
 #first Line total cordinates  
 st = f.readline()  
 for i in st.split():  
 if i.isdigit():  
 res.append(int(i))  
 self.col = res.pop(0)  
 self.row = res.pop(0)  
  
 #second Line Starting cordinates  
 st = f.readline()  
 for i in st.split():  
 if i.isdigit():  
 res.append(int(i))  
 self.startCol = res.pop(0)  
 self.startRow = res.pop(0)  
  
 #third line  
 st = f.readline()  
 for i in st.split():  
 if i.isdigit():  
 res.append(int(i))  
 self.goalCol = res.pop(0)  
 self.goalRow = res.pop(0)  
  
 #fill grid  
 self.grid = [[1 for i in range(self.col)] for j in range(self.row)]  
 rowCounter = self.row  
 colCounter = 0  
 for st in f:  
 #print(st)  
 for i in st.split():  
 if i.isdigit():  
 res.append(int(i))  
 for x in res:  
 self.grid[rowCounter - 1][colCounter] = x  
 colCounter += 1  
 colCounter = 0  
 rowCounter -=1  
 res.clear()  
  
  
 print("Col = " + str(self.col))  
 print("row = " + str(self.row))  
 print("S-Col = " + str(self.startCol))  
 print("S-row = " + str(self.startRow))  
 print("G-Col = " + str(self.goalCol))  
 print("G-row = " + str(self.goalRow))  
  
def shortestHuer(listo):  
 index = 0  
 counter = 0  
 shortest = listo[0][2]  
 for x in listo:  
 if shortest > x[2]:  
 shortest = x[2]  
 index = counter  
 counter +=1  
 return index  
def assignHeuristic(obj):  
  
 tRow = obj.row  
 tCol = obj.col  
 gRow = obj.goalRow  
 gCol = obj.goalCol  
 tGrid = [[-1 for i in range(tCol)] for j in range(tRow)]  
  
 i=0  
 j=0  
 counter = 0  
 while i - gRow < 0 or i - gCol < 0:  
  
 i = counter  
 j = 0 - counter  
 x = 0  
 while x <= counter:  
 y = 0 - x  
 if gCol + x < tCol and gRow + i < tRow:  
 tGrid[gRow + i][gCol+x] = counter  
 if gCol + y < tCol and gRow + i < tRow:  
 tGrid[gRow + i][gCol+y] = counter  
 if gCol + x < tCol and gRow + j < tRow and tGrid[gRow + j][gCol + x] == -1:  
 tGrid[gRow + j][gCol + x] = counter  
 if gCol + y < tCol and gRow + j < tRow and tGrid[gRow + j][gCol + y] == -1:  
 tGrid[gRow + j][gCol + y] = counter  
  
 if gCol + i < tCol and gRow + x < tRow:  
 tGrid[gRow + x][gCol + i] = counter  
 if gCol + i < tCol and gRow + y < tRow:  
 tGrid[gRow + y][gCol + i] = counter  
 if gCol + j < tCol and gRow + x < tRow and tGrid[gRow + x][gCol + j] == -1:  
 tGrid[gRow + x][gCol + j] = counter  
 if gCol + j < tCol and gRow + y < tRow and tGrid[gRow + y][gCol + j] == -1:  
 tGrid[gRow + y][gCol + j] = counter  
  
 x += 1  
  
 counter +=1  
  
 print("Heuristic Values by Manhattan distance")  
 temp = copy.deepcopy(tGrid)  
 temp[obj.startRow][obj.startCol] = 'S'  
 temp[obj.goalRow][obj.goalCol] = 'G'  
 temp.reverse()  
 for x in temp:  
 print(x)  
 print("-------------------------------------------------------\n\n\n")  
 return tGrid  
  
  
def bestFirstSearch(obj, hue):  
 bGrid = copy.deepcopy(obj.grid)  
 brow = obj.row  
 bcol = obj.col  
 cRow = obj.startRow  
 cCol = obj.startCol  
 gRow = obj.goalRow  
 gCol = obj.goalCol  
  
 stack = []  
 visited = []  
 count = 0  
 cost = 0  
 bGrid[cRow][cCol] = 'S'  
  
 print("\n\n :Greedy Best First Search:")  
 print("Path :\nS",end=" ")  
  
 search = True  
 found = False  
 if (cRow == gRow and cCol == gCol):  
 search = False  
 found = True  
  
 up = [0,0,-1]  
 right = [0,0,-1]  
 diagonally = [0,0,-1]  
 while(search):  
 queue = []  
  
 if(cRow + 1 < brow):  
 up[0] = cRow + 1  
 up[1] = cCol  
 up[2] = hue[cRow + 1][cCol] #cost  
 queue.append(up)  
 if (cCol + 1 < bcol):  
 right[0] = cRow  
 right[1] = cCol + 1  
 right[2] = hue[cRow][cCol + 1] # cost  
 queue.append(right)  
 if (cRow + 1 < brow and cCol + 1 < bcol):  
 diagonally[0] = cRow + 1  
 diagonally[1] = cCol + 1  
 diagonally[2] = hue[cRow + 1][cCol + 1] # cost  
 queue.append(diagonally)  
  
 next = []  
 for x in queue:  
 if x not in visited and x[2] != -1:  
 if bGrid[x[0]][x[1]] == 0:  
 next.append(x)  
  
 if len(next) != 0:  
 stack.append(next)  
  
 if (len(stack) == 0):  
 search = False  
 found = False  
 break  
  
 move = stack.pop()  
 shortest = shortestHuer(move)  
 cRow = move[shortest][0]  
 cCol = move[shortest][1]  
 cost += shortest+1 #as in our case 1 cost for up 2 cost for right and 3 cost for diagnoally  
  
 print("->" + str(cRow) + "," + str(cCol), end=" ")  
 bGrid[cRow][cCol] = '#'  
 visited.append(move.pop(shortest))  
 if len(move) != 0:  
 stack.append(move)  
  
  
 if (cRow == gRow and cCol == gCol):  
 print("-> G", end=" ")  
 bGrid[cRow][cCol] = 'G'  
 search = False  
 found = True  
  
 up = [0, 0, -1]  
 right = [0, 0, -1]  
 diagonally = [0, 0, -1]  
 count +=1  
 if count%10==0:  
 print(" ")  
 #for i in bGrid:  
 # for j in i:  
 # print(str(j) + " ", end=" ")  
 # print(" ")  
 #num = input("Enter number :")  
  
  
 if found:  
 print("\n\nPath to goal Found")  
 print("Total cost to achive goal is " + str(cost))  
 else:  
 print("\n\n Path to goal not Found")  
  
 bGrid.reverse()  
 for i in bGrid:  
 for j in i:  
 print(str(j) + " " , end=" ")  
 print(" ")  
  
 print("\n-----------------------------------------------\n\n")  
  
def aStarSearch(obj, hue):  
 bGrid = copy.deepcopy(obj.grid)  
 brow = obj.row  
 bcol = obj.col  
 cRow = obj.startRow  
 cCol = obj.startCol  
 gRow = obj.goalRow  
 gCol = obj.goalCol  
  
 stack = []  
 visited = []  
 count = 0  
 cost = 0  
 bGrid[cRow][cCol] = 'S'  
  
 print("\n\n\t :A\* Search:")  
 print("Path :\nS", end=" ")  
  
 search = True  
 found = False  
 if (cRow == gRow and cCol == gCol):  
 search = False  
 found = True  
  
 up = [0, 0, -1]  
 right = [0, 0, -1]  
 diagonally = [0, 0, -1]  
 while (search):  
 queue = []  
  
 if (cRow + 1 < brow):  
 up[0] = cRow + 1  
 up[1] = cCol  
 up[2] = 1 + hue[cRow + 1][cCol] # cost  
 queue.append(up)  
 if (cCol + 1 < bcol):  
 right[0] = cRow  
 right[1] = cCol + 1  
 right[2] = 2 + hue[cRow][cCol + 1] # cost  
 queue.append(right)  
 if (cRow + 1 < brow and cCol + 1 < bcol):  
 diagonally[0] = cRow + 1  
 diagonally[1] = cCol + 1  
 diagonally[2] = 3 + hue[cRow + 1][cCol + 1] # cost  
 queue.append(diagonally)  
  
 next = []  
 for x in queue:  
 if x not in visited and x[2] != -1:  
 if bGrid[x[0]][x[1]] == 0:  
 next.append(x)  
  
 if len(next) != 0:  
 stack.append(next)  
  
 if (len(stack) == 0):  
 search = False  
 found = False  
 break  
  
 move = stack.pop()  
 shortest = shortestHuer(move)  
 cRow = move[shortest][0]  
 cCol = move[shortest][1]  
 cost += shortest + 1 # as in our case 1 cost for up 2 cost for right and 3 cost for diagnoally  
 print("->" + str(cRow) + "," + str(cCol), end=" ")  
 bGrid[cRow][cCol] = '#'  
 visited.append(move.pop(shortest))  
 if len(move) != 0:  
 stack.append(move)  
  
 if (cRow == gRow and cCol == gCol):  
 print("-> G", end=" ")  
 bGrid[cRow][cCol] = 'G'  
 search = False  
 found = True  
  
 up = [0, 0, -1]  
 right = [0, 0, -1]  
 diagonally = [0, 0, -1]  
 count += 1  
 if count % 10 == 0:  
 print(" ")  
 #for i in bGrid:  
 # for j in i:  
 # print(str(j) + " ", end=" ")  
 # print(" ")  
 #num = input("Enter number :")  
  
  
 if found:  
 print("\nPath to goal Found")  
 print("Total cost to achive goal is " + str(cost))  
 else:  
 print("\n\n Path to goal not Found")  
  
 bGrid.reverse()  
 for i in bGrid:  
 for j in i:  
 print(str(j) + " ", end=" ")  
 print(" ")  
 print("\n-----------------------------------------------\n\n")  
  
def iterativeDeepningSearch(obj, hue, itration):  
 bGrid = copy.deepcopy(obj.grid)  
 brow = obj.row  
 bcol = obj.col  
 cRow = obj.startRow  
 cCol = obj.startCol  
 gRow = obj.goalRow  
 gCol = obj.goalCol  
  
 stack = []  
 visited = []  
 count = 0  
 cost = 0  
 path = []  
 print("\n\n\t :IDA\* Search:")  
 search = True  
 found = False  
 if (cRow == gRow and cCol == gCol):  
 search = False  
 found = True  
  
 itrate = 0  
 iRow = cRow  
 iCol = cCol  
 while (itrate < itration and not found):  
  
 print("Itration No = " + str(itrate))  
 search = True  
 bGrid = []  
 bGrid = copy.deepcopy(obj.grid)  
 cRow = iRow  
 cCol = iCol  
 bGrid[cRow][cCol] = 'S'  
 bGrid[gRow][gCol] = 'G'  
 stack = []  
 visited = []  
 cost = 0  
 path = []  
  
 while (search):  
 up = [0, 0, -1]  
 right = [0, 0, -1]  
 diagonally = [0, 0, -1]  
 queue = []  
  
 if (cRow + 1 < brow and cRow + 1 < iRow + itrate):  
 up[0] = cRow + 1  
 up[1] = cCol  
 up[2] = 1 + hue[cRow + 1][cCol] # cost  
 queue.append(up)  
 if (cCol + 1 < bcol and cCol + 1 < iCol + itrate):  
 right[0] = cRow  
 right[1] = cCol + 1  
 right[2] = 2 + hue[cRow][cCol + 1] # cost  
 queue.append(right)  
 if (cRow + 1 < brow and cCol + 1 < bcol and cRow + 1 < iRow + itrate and cCol + 1 < iCol + itrate):  
 diagonally[0] = cRow + 1  
 diagonally[1] = cCol + 1  
 diagonally[2] = 3 + hue[cRow + 1][cCol + 1] # cost  
 queue.append(diagonally)  
  
 next = []  
 for x in queue:  
 if x not in visited and x[2] != -1:  
 if bGrid[x[0]][x[1]] == 0 or bGrid[x[0]][x[1]] == 'G':  
 next.append(x)  
  
 if len(next) != 0:  
 stack.append(next)  
  
 if (len(stack) == 0):  
 search = False  
 found = False  
 break  
  
 move = stack.pop()  
 shortest = shortestHuer(move)  
 cRow = move[shortest][0]  
 cCol = move[shortest][1]  
 cost += shortest + 1 # as in our case 1 cost for up 2 cost for right and 3 cost for diagnoally  
 temp = [cRow,cCol]  
 path.append(temp)  
 bGrid[cRow][cCol] = '#'  
 visited.append(move.pop(shortest))  
  
 if len(move) != 0:  
 stack.append(move)  
  
 if (cRow == gRow and cCol == gCol):  
 bGrid[cRow][cCol] = 'G'  
 search = False  
 found = True  
  
 #for i in bGrid:  
 # for j in i:  
 # print(str(j) + " ", end=" ")  
 # print(" ")  
 #print("==============================================")  
 #num = input("Enter number :")  
  
  
 itrate +=1  
  
  
 print("Path:\nS", end=" ")  
 for p in path:  
 print("->" + str(p[0]) + "," + str(p[1]), end=" ")  
 count += 1  
 if count%10 == 0:  
 print(" ")  
  
  
 if found:  
 print("G")  
 print("\n\n Path to goal Found in " + str(itrate-1) + "th itration")  
 print("Total cost to achive goal is " + str(cost))  
 else:  
 print("\n\n Path to goal not Found")  
  
 bGrid.reverse()  
 for i in bGrid:  
 for j in i:  
 print(str(j) + " ", end=" ")  
 print(" ")  
 print("\n-----------------------------------------------\n\n")  
  
  
  
# Press the green button in the gutter to run the script.  
if \_\_name\_\_ == '\_\_main\_\_':  
  
  
 obj = Grid()  
 obj.insertGrid()  
 heuristicGrid = assignHeuristic(obj)  
  
 bestFirstSearch(obj, heuristicGrid)  
 aStarSearch(obj,heuristicGrid)  
 iterativeDeepningSearch(obj, heuristicGrid, 15)