ai-lab-04

March 7, 2023

1 Example 01

1.1 Implement a graph with Breadth First Search

[1]:
```
graph = {'5' : ['3','7'],'3' : ['2', '4'],'7' : ['8'],'2' : [],'4' : ['8'],'8' :
 ↪ [] }
# List for visited nodes.
visited = []

#Initialize a queue
queue = []

#function for BFS
def bfs(visited, graph, node):
    visited.append(node)
    queue.append(node)
    # Creating loop to visit each node
    while queue:
        m = queue.pop(0)
        print (m,end=" ")

        for neighbour in graph[m]:
            if neighbour not in visited:
                visited.append(neighbour)
                queue.append(neighbour)
print("Following is the Breadth-First Search")
bfs(visited, graph, '5')
```

```
Following is the Breadth-First Search
5 3 7 2 4 8
```

2 Example 02

2.1 Create the following graph and find the Minimum cost from node 0 to node 6 with Uniform-cost Search algorithm

[2]:

```python
# create the graph
graph,cost = [[] for i in range(8)],{}
# add edge
graph[0].append(1)
graph[0].append(3)
graph[3].append(1)
graph[3].append(6)
graph[3].append(4)
graph[1].append(6)
graph[4].append(2)
graph[4].append(5)
graph[2].append(1)
graph[5].append(2)
graph[5].append(6)
graph[6].append(4)
# add the cost
cost[(0, 1)] = 2
cost[(0, 3)] = 5
cost[(1, 6)] = 1
cost[(3, 1)] = 5
cost[(3, 6)] = 6
cost[(3, 4)] = 2
cost[(2, 1)] = 4
cost[(4, 2)] = 4
cost[(4, 5)] = 3
cost[(5, 2)] = 6
cost[(5, 6)] = 3
cost[(6, 4)] = 7
# goal state
goal = []
# set the goal
# there can be multiple goal states
goal.append(6)
# get the answer
answer = uniform_cost_search(goal, 0)
# print the answer
print("Minimum cost from 0 to 6 is = ",answer[0])
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
~\AppData\Local\Temp\ipykernel_20872\362816828.py in <module>
     33 goal.append(6)
     34 # get the answer
```

```
---> 35 answer = uniform_cost_search(goal, 0)
     36 # print the answer
     37 print("Minimum cost from 0 to 6 is = ",answer[0])

NameError: name 'uniform_cost_search' is not defined
```

[29]:

```python
def uniform_cost_search(goal, start):
    # minimum cost upto
    # goal state from starting
    global graph,cost
    answer = []
    # create a priority queue
    queue = []
    # set the answer vector to max value
    for i in range(len(goal)):
        answer.append(10**8)
    # insert the starting index
    queue.append([0, start])
    # map to store visited node
    visited = {}
    # count
    count = 0
    # while the queue is not empty
    while (len(queue) > 0):
        # get the top element of the
        queue = sorted(queue)
        p = queue[-1]
        # pop the element
        del queue[-1]
        # get the original value
        p[0] *= -1
        # check if the element is part of
        # the goal list
        if (p[1] in goal):
            # get the position
            index = goal.index(p[1])
            # if a new goal is reached
            if (answer[index] == 10**8):
                count += 1
            # if the cost is less
            if (answer[index] > p[0]):
                answer[index] = p[0]
            # pop the element
            del queue[-1]
            queue = sorted(queue)
            if (count == len(goal)):
                return answer
```

```
        # check for the non visited nodes
        # which are adjacent to present node
        if (p[1] not in visited):
            for i in range(len(graph[p[1]])):
            # value is multiplied by -1 so that
            # least priority is at the top
                queue.append( [(p[0] + cost[(p[1], graph[p[1]][i])])* -1,␣
↪graph[p[1]][i]])
        # mark as visited
        visited[p[1]] = 1
    return answer
```

3 Example 03

3.1 Perform Breadth first traversal on a Binary Search Tree and print the elements traversal.

[36]:
```
class Node(object):
    def __init__(self, value,left=None, right=None):
        self.value = value
        self.left = left
        self.right = right
class BinarySearchTree(object):
    def __init__(self, value):
        self.root = Node(value)
    def insert(self, value):
        current = self.root
        while current:
            if value > current.value:
                if current.right is None:
                    current.right = Node(value)
                    break
                else:
                    current = current.right
            else:
                if current.left is None:
                    current.left = Node(value)
                    break
                else:
                    current = current.left
    def Breadth_first_search(self,root):
        """In BFS the Node Values at each level of the Tree are traversed␣
↪before going to next level"""
        visited = []
        if root:
            visited.append(root)
```

```python
            print (root.value)
        current = root
        while current :
            if current.left:
                print(current.left.value)
                visited.append(current.left)
            if current.right:
                print( current.right.value)
                visited.append(current.right)
            visited.pop(0)
            if not visited:
                break
            current = visited[0]
t = BinarySearchTree(100)
t.insert(12)
t.insert(92)
t.insert(112)
t.insert(123)
t.insert(2)
t.insert(11)
t.insert(52)
t.insert(3)
t.insert(66)
t.insert(10)
print( "Output of Breadth First search is ")
t.Breadth_first_search(t.root)
```

```
Output of Breadth First search is
100
12
112
2
92
123
11
52
3
66
10
```

4 Example 04

4.1 In the following class "Graph", implement Iterative Deepening Depth Search methods. Find if the target node = 6, is reachable from source node = 0, given max depth = 3. Also, Implement Depth Limited Search.

[54]:
```
from collections import defaultdict
# This class represents a directed graph using adjacency list representation
class Graph:
    def __init__ (self,vertices):
        # No. of vertices
        self.V = vertices
        # default dictionary to store graph
        self.graph = defaultdict(list)
    # function to add an edge to graph
    def addEdge(self,u,v):
        self.graph[u].append(v)
    def IDDFS(self,src, target, maxDepth):
        # Repeatedly depth-limit search till the
        # maximum depth
        for i in range(maxDepth):
            if (self.IDDFS(src, target, i)):
                return True
        return False
#Create a graph given in the above diagram
g = Graph (7);
g.addEdge(0, 1)
g.addEdge(0, 2)
g.addEdge(1, 3)
g.addEdge(1, 4)
g.addEdge(2, 5)
g.addEdge(2, 6)
target = 6; maxDepth = 3; src = 0
if g.IDDFS(src, target, maxDepth) == True:
    print ("Target is reachable from source within max depth")
else :
    print ("Target is NOT reachable from source within max depth")
```

```
Target is NOT reachable from source within max depth
```

5 Question 01

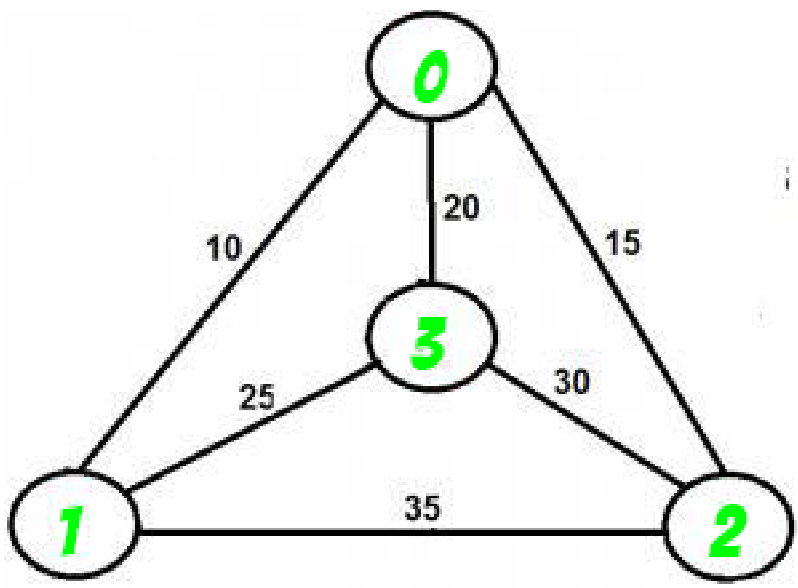
5.1 Traveling Salesman Problem:

5.1.1 Given a set of cities and distances between every pair of cities, the problem is to find the shortest possible route that visits every city exactly once and returns to the starting point. Like any problem, which can be optimized, there must be a cost function. In the context of TSP, total distance traveled must be reduced as much as possible.

5.1.2 Consider the below matrix representing the distances (Cost) between the cities. Find the shortest possible route that visits every city exactly once and returns to the starting point.

[57]:
```
from IPython.display import Image
Image(filename=r'C:\Users\Bilal\Desktop\AI Lab 04\Q2.png')
```

[57]:



[ ]:
```
from sys import maxsize
from itertools import permutations

## create the graph
graph,cost = [[] for i in range(4)],{}
vertices = 4
# add edge and weigths
graph[0].append(0)
graph[0].append(10)
graph[0].append(15)
graph[0].append(20)
```

```
graph[1].append(10)
graph[1].append(0)
graph[1].append(35)
graph[1].append(25)

graph[2].append(15)
graph[2].append(35)
graph[2].append(0)
graph[2].append(30)

graph[3].append(20)
graph[3].append(25)
graph[3].append(30)
graph[3].append(0)

# Printing Cost
# for i in range(4):
#     for j in range(4):
#         if(i!=j):
#             print("vertex: ",i," to ","vertex:",j,"Cost:",cost[i,j])

def travellingSalesmanProblem (graph, source):

    # Storing all vertices apart from source vertices
    vertex = []
    for i in range(vertices):
        if i!=source:
            vertex.append(i)

    min_path=maxsize
    next_permutation=permutations(vertex)
    for i in next_permutation:

        current_pathweight =0

        k=source
        for j in i:
            current_pathweight+=graph[k][j]
            k=j

        current_pathweight+=graph[k][s]

        min_path = min (min_path, current_pathweight)

    return min_path
```

```
print("The graph given in the question is as follows:")
print(graph)
source =0
print("The weight of the MST for the given graph␣
↪is",travellingSalesmanProblem(graph,source),"\b.")
```

6 Question 02

6.1 Implement DFS on graph and tree.

6.2 DFS on graph

[55]:
```
from collections import defaultdict

class Graph:

    def __init__(self):
        self.graph = defaultdict(list)

    def addEdge(self, u, v):
        self.graph[u].append(v)

    def DFSUtil(self, v, visited):

        visited.add(v)
        print(v, end=' ')

        for neighbour in self.graph[v]:
            if neighbour not in visited:
                self.DFSUtil(neighbour, visited)

    def DFS(self, v):

        visited = set()

        self.DFSUtil(v, visited)

g = Graph()
g.addEdge(0, 1)
g.addEdge(0, 2)
g.addEdge(1, 2)
g.addEdge(2, 0)
g.addEdge(2, 3)
g.addEdge(3, 3)
g.addEdge(0,4)
g.addEdge(1,4)
g.addEdge(1,3)
```

```
print("Following is DFS from (starting from vertex 0)")
g.DFS(0)
```

```
Following is DFS from (starting from vertex 0)
0 1 2 3 4
```

6.3 DFS on tree

[84]:
```
# Creating a tree node

class Node:
    def __init__(self, data):
      self.left = None
      self.right = None
      self.data = data
    def PrintTree(self):
      print(self.data)



    def insert(self, data):
# Compare the new value with the parent node
        if self.data:
         if data < self.data:
            if self.left is None:
               self.left = Node(data)
            else:
               self.left.insert(data)
         elif data > self.data:
               if self.right is None:
                  self.right = Node(data)
               else:
                  self.right.insert(data)
        else:
         self.data = data
    def PrintTree(self):
        if self.left:
            self.left.PrintTree()
        print(self.data),
        if self.right:
            self.right.PrintTree()

def printPostorder(root):

    if root:
```

```
        # First recur on left child
        printPostorder(root.left)

        # the recur on right child
        printPostorder(root.right)

        # now print the data of node
        print(root.data)


root = Node(12)
root.insert(6)
root.insert(14)
root.insert(3)
root.insert(99)
root.insert(10)
root.insert(4)
root.insert(72)

print("\nPrinting DFS")
printPostorder(root)
```

```
Printing DFS
4
3
10
6
72
99
14
12
```

7 Question 03

7.1 Write a program to solve the 8-puzzle problem using the DFS and BFS search algorithm

[ ]:
```
import sys
import numpy as np


class Node:
        def __init__(self, state, parent, action):
                self.state = state
                self.parent = parent
```

```
                self.action = action


class StackFrontier:
        def __init__(self):
                self.frontier = []

        def add(self, node):
                self.frontier.append(node)

        def contains_state(self, state):
                return any((node.state[0] == state[0]).all() for node in self.
 ↪frontier)

        def empty(self):
                return len(self.frontier) == 0

        def remove(self):
                if self.empty():
                        raise Exception("Empty Frontier")
                else:
                        node = self.frontier[-1]
                        self.frontier = self.frontier[:-1]
                        return node


class QueueFrontier(StackFrontier):
        def remove(self):
                if self.empty():
                        raise Exception("Empty Frontier")
                else:
                        node = self.frontier[0]
                        self.frontier = self.frontier[1:]
                        return node


class Puzzle:
        def __init__(self, start, startIndex, goal, goalIndex):
                self.start = [start, startIndex]
                self.goal = [goal, goalIndex]
                self.solution = None

        def neighbors(self, state):
                mat, (row, col) = state
                results = []

                if row > 0:
```

```
            mat1 = np.copy(mat)
            mat1[row][col] = mat1[row - 1][col]
            mat1[row - 1][col] = 0
            results.append(('up', [mat1, (row - 1, col)]))
        if col > 0:
            mat1 = np.copy(mat)
            mat1[row][col] = mat1[row][col - 1]
            mat1[row][col - 1] = 0
            results.append(('left', [mat1, (row, col - 1)]))
        if row < 2:
            mat1 = np.copy(mat)
            mat1[row][col] = mat1[row + 1][col]
            mat1[row + 1][col] = 0
            results.append(('down', [mat1, (row + 1, col)]))
        if col < 2:
            mat1 = np.copy(mat)
            mat1[row][col] = mat1[row][col + 1]
            mat1[row][col + 1] = 0
            results.append(('right', [mat1, (row, col + 1)]))

        return results

    def print(self):
        solution = self.solution if self.solution is not None else None
        print("Start State:\n", self.start[0], "\n")
        print("Goal State:\n",  self.goal[0], "\n")
        print("\nStates Explored: ", self.num_explored, "\n")
        print("Solution:\n ")
        for action, cell in zip(solution[0], solution[1]):
            print("action: ", action, "\n", cell[0], "\n")
        print("Goal Reached!!")

    def does_not_contain_state(self, state):
        for st in self.explored:
            if (st[0] == state[0]).all():
                return False
        return True

    def solve(self):
        self.num_explored = 0

        start = Node(state=self.start, parent=None, action=None)
        frontier = QueueFrontier()
        frontier.add(start)

        self.explored = []
```

```
                while True:
                        if frontier.empty():
                                raise Exception("No solution")

                        node = frontier.remove()
                        self.num_explored += 1

                        if (node.state[0] == self.goal[0]).all():
                                actions = []
                                cells = []
                                while node.parent is not None:
                                        actions.append(node.action)
                                        cells.append(node.state)
                                        node = node.parent
                                actions.reverse()
                                cells.reverse()
                                self.solution = (actions,  cells)
                                return

                        self.explored.append(node.state)

                        for action, state in self.neighbors(node.state):
                                if not frontier.contains_state(state) and self.
 ↪does_not_contain_state(state):
                                        child = Node(state=state, parent=node,␣
 ↪action=action)
                                        frontier.add(child)


start = np.array([[0,1, 2], [6,7,8], [3,4,5]])
goal = np.array([[0,1,2], [3,4,5], [6,7,8]])


startIndex = (1, 1)
goalIndex = (1, 0)


p = Puzzle(start, startIndex, goal, goalIndex)
p.solve()
p.print()
```

[ ]:

[ ]:

[ ]: