**Practical No 2:**

**Implement Iterative deep depth first search for Romanian map problem.**

from collections import deque

import sys

infinity = float('inf')

**class Node:**

**def \_\_init\_\_(self, state, parent=None, action=None, path\_cost=0):**

self.state = state

self.parent = parent

self.action = action

self.path\_cost = path\_cost

self.depth = 0

if parent:

self.depth = parent.depth + 1

**def \_\_repr\_\_(self):**

return "<Node {}>".format(self.state)

**def expand(self, problem):**

return [self.child\_node(problem, action)

for action in problem.actions(self.state)]

**def child\_node(self, problem, action):**

next\_state = problem.result(self.state, action)

next\_node = Node(next\_state, self, action,

problem.path\_cost(self.path\_cost, self.state,action, next\_state))

return next\_node

**def path(self):**

node, path\_back = self, []

while node:

path\_back.append(node)

node = node.parent

return list(reversed(path\_back))

**class Graph:**

**def \_\_init\_\_(self, graph\_dict=None, directed=True):**

self.graph\_dict = graph\_dict or {}

self.directed = directed

if not directed:

self.make\_undirected()

**def make\_undirected(self):**

for a in list(self.graph\_dict.keys()):

for (b, dist) in self.graph\_dict[a].items():

self.connect1(b, a, dist)

**def connect1(self, A, B, distance):**

self.graph\_dict.setdefault(A, {})[B] = distance

**def get(self, a, b=None):**

links = self.graph\_dict.setdefault(a, {})

if b is None:

return links

else:

return links.get(b)

**def nodes(self):**

s1 = set([k for k in self.graph\_dict.keys()])

s2 = set([k2 for v in self.graph\_dict.values() for k2, v2 in v.items()])

nodes = s1.union(s2)

return list(nodes)

**def UndirectedGraph(graph\_dict=None):**

return Graph(graph\_dict = graph\_dict, directed=False)

**class Problem(object):**

**def \_\_init\_\_(self, initial, goal=None):**

self.initial = initial

self.goal = goal

**def actions(self, state):**

raise NotImplementedError

**def result(self, state, action):**

raise NotImplementedError

**def goal\_test(self, state):**

if isinstance(self.goal, list):

return is\_in(state, self.goal)

else:

return state == self.goal

**def path\_cost(self, c, state1, action, state2):**

return c + 1

**def value(self, state):**

raise NotImplementedError

**class GraphProblem(Problem):**

**def \_\_init\_\_(self, initial, goal, graph):**

Problem.\_\_init\_\_(self, initial, goal)

self.graph = graph

**def actions(self, A):**

return list(self.graph.get(A).keys())

**def result(self, state, action):**

return action

**def path\_cost(self, cost\_so\_far, A, action, B):**

return cost\_so\_far + (self.graph.get(A, B) or infinity)

**def find\_min\_edge(self):**

m = infinity

for d in self.graph.graph\_dict.values():

local\_min = min(d.values())

m = min(m, local\_min)

return m

**def depth\_limited\_search(problem, limit=50):**

**def recursive\_dls(node, problem, limit):**

if problem.goal\_test(node.state):

return node

elif limit == 0:

return 'cutoff'

else:

cutoff\_occurred = False

for child in node.expand(problem):

result = recursive\_dls(child, problem, limit - 1)

if result == 'cutoff':

cutoff\_occurred = True

elif result is not None:

return result

return 'cutoff' if cutoff\_occurred else 'Not found'

return recursive\_dls(Node(problem.initial), problem, limit)

**def iterative\_deepening\_search(problem, limit):**

for depth in range(0,limit):

print("checking with depth :", depth)

result = depth\_limited\_search(problem, depth)

print("result : ", result)

return result

mumbai\_map = UndirectedGraph(dict(

kurla=dict(sion=5,santacruz=2),

sion=dict(kurla=5,bandra=3,dadar=4),

dadar=dict(sion=4,bandra=4),

bandra=dict(santacruz=3,sion=3,dadar=4),

santacruz=dict(bandra=3,kurla=2)))

print("searching from kurla to bandra with level 5...")

mumbai\_problem = GraphProblem('kurla','bandra', mumbai\_map)

print(iterative\_deepening\_search(mumbai\_problem, 5))

print("searching from kurla to bandra with level 2...")

mumbai\_problem = GraphProblem('kurla','bandra', mumbai\_map)

print(iterative\_deepening\_search(mumbai\_problem, 2))

**Output:**

**searching from kurla to bandra with level 5...**

**checking with depth : 0**

**result : cutoff**

**checking with depth : 1**

**result : cutoff**

**checking with depth : 2**

**result : <Node bandra>**

**checking with depth : 3**

**result : <Node bandra>**

**checking with depth : 4**

**result : <Node bandra>**

**<Node bandra>**

**searching from kurla to bandra with level 2...**

**checking with depth : 0**

**result : cutoff**

**checking with depth : 1**

**result : cutoff**

**cutoff**

**Roll No:13,14**