Jaiwin Shah Manan Shah

2019130058, 2019130059

TE Comps

Batch C

# Experiment – 2

**Aim:** To implement the fire extinguisher system using BFS and DFS.

**Theory:**

- **Depth-First-Search (DFS)**:
  1) DFS always expands DEPTH-FIRST the deepest node in the current frontier of thesearch tree.
  2) The search proceeds immediately to the deepest level of the search tree, wherethe nodes have no successors.
  3) DFS uses a LIFO queue.
  4) Visits children before siblings.

- **Breadth-First-Search (BFS):**
  1) BFS is a simple strategy in which the root node is expanded first, then all the successors of the root node are expanded next, then their successors, and so on.
  2) All the nodes are expanded at a given depth in the search tree before any nodesat the next level are expanded.
  3) BFS uses a FIFO queue.
  4) Visits siblings before children

**Code**:

```
from tkinter import *
from PIL import Image
from time import *
import timeit
import random
# import dictionary for graph
from collections import defaultdict
# queue using queue module
from queue import Queue

import sys
import gc
#garbage collection
gc.collect()

# function for adding edge to graph
```

```
graph = defaultdict(list)
def addEdge(graph,u,v):
    graph[u].append(v)

# definition of function
def generate_edges(graph):
    edges = []

    # for each node in graph
    for node in graph:

        # for each neighbour node of a single node
        for neighbour in graph[node]:

            # if edge exists then append
            edges.append((node, neighbour))
    return edges

# direction includes right,left,up,down,suck,nop
def dirs(vacpos,act,flag): #flag 0 dont move vac
    global visited_pieces
    if act == 'right':
        if (flag==1):
            vac_pos[1] = vac_pos[1] + 1
        else:
            return vacpos[0]+str(int(vacpos[1]) + 1)
    elif act == 'left':
        if (flag==1):
            vac_pos[1] = vac_pos[1] - 1
        else:
            return vacpos[0]+str(int(vacpos[1]) - 1)
    elif act == 'up':
        if (flag==1):
            vac_pos[0] = vac_pos[0] - 1
        else:
            return str(int(vacpos[0])-1)+vacpos[1]

    elif act == 'down':
        if (flag==1):
            vac_pos[0] = vac_pos[0] + 1
        else:
            return str(int(vacpos[0])+1)+vacpos[1]

    if (flag==1):
        visited_pieces = visited_pieces + 1

def actuator(act):
    global cleaned_pieces
```

```
    if act == 'suck':
        maps[vac_pos[0]-1][vac_pos[1]-1] = 0
        cleaned_pieces = cleaned_pieces + 1

# '0' means clean & '1' means dirty
def initmap(row,col):
    global dirty_pieces
    global maps
    global visited
    for j in range(row):
        tmp = []
        tmpv = []
        for i in range(col):
            rand = random.randint(0,1)
            if rand == 1 :
                dirty_pieces = dirty_pieces + 1
            tmp.append(rand)               #initializing pieces with random
cleanliness
            tmpv.append(0)
        visited.append(tmpv)
        maps.append(tmp)

    for i in range(row):
        tmp = []
        for j in range(col):
                if maps[i][j] == 0 :
                    tmp.append(Label(image = tile_clean ))
                else:
                    tmp.append(Label(image = tile_dirty ))

        lab1.append(tmp)

    for i in range(row):
            for j in range(col):
                lab1[i][j].grid(row=i,column=j)

def allowance(pos):
    if rows == 1 and cols != 1:    #horizontal movement
        allow = ['right','left']
    elif rows != 1 and cols == 1:  #vertical movement
        allow = ['up','down']
    elif rows == 1 and cols == 1:  #nop
        allow = []
    else:                          #two dimentional movement
        if pos[0] == '1' and pos[1] == '1':
            allow = ['right','down']
        elif pos[0] == '1' and pos[1] == str(cols):
            allow = ['left','down']
```

```
        elif pos[0] == str(rows) and pos[1] == str(cols):
            allow = ['left','up']
        elif pos[0] == str(rows) and pos[1] == '1':
            allow = ['right','up']
        elif pos[0] == '1' and (pos[1]!= '1' or pos[1]!=str(cols)):
            allow = ['right','left','down']
        elif pos[0] == str(rows) and (pos[1]!= '1' or pos[1]!= str(cols)) :
            allow = ['right','left','up']
        elif pos[1] == '1' and (pos[0]!= '1' or pos[1]!= str(rows)):
            allow = ['right','up','down']
        elif pos[1] == str(cols) and (pos[0]!= '1' or pos[1]!= str(rows)):
            allow = ['left','up','down']
        else:
            allow = ['right','left','up','down']
    return allow

#main
#initializing
maps = []
tmp_maps = []
visited = []
visited_pieces = 0
dirty_pieces = 0
cleaned_pieces = 0

cb = 0  #default radio button value non
rows =  8                #Row number
cols = 8                #Column number
vac_pos = [2,2]           #Current Cursor
vac_str = str(vac_pos[0]) + str(vac_pos[1])

mem_cost=1  #set default memory cost

mygui = Tk()
mygui.title("BFS & DFS Algorithms in AI Vacuum Cleaner World")
mygui.geometry("650x480+100+100")
ll = Label(mygui,text="Please check one of the algorithms.")
ll.grid(row = 0,column=cols)

lab1 = []

tile_clean = PhotoImage(file="Src\\tile2.png")
tile_dirty = PhotoImage(file="Src\\tile1.png")
done_icon = PhotoImage(file="Src\done_icon.png")

initmap(rows,cols)
mygui.update()
```

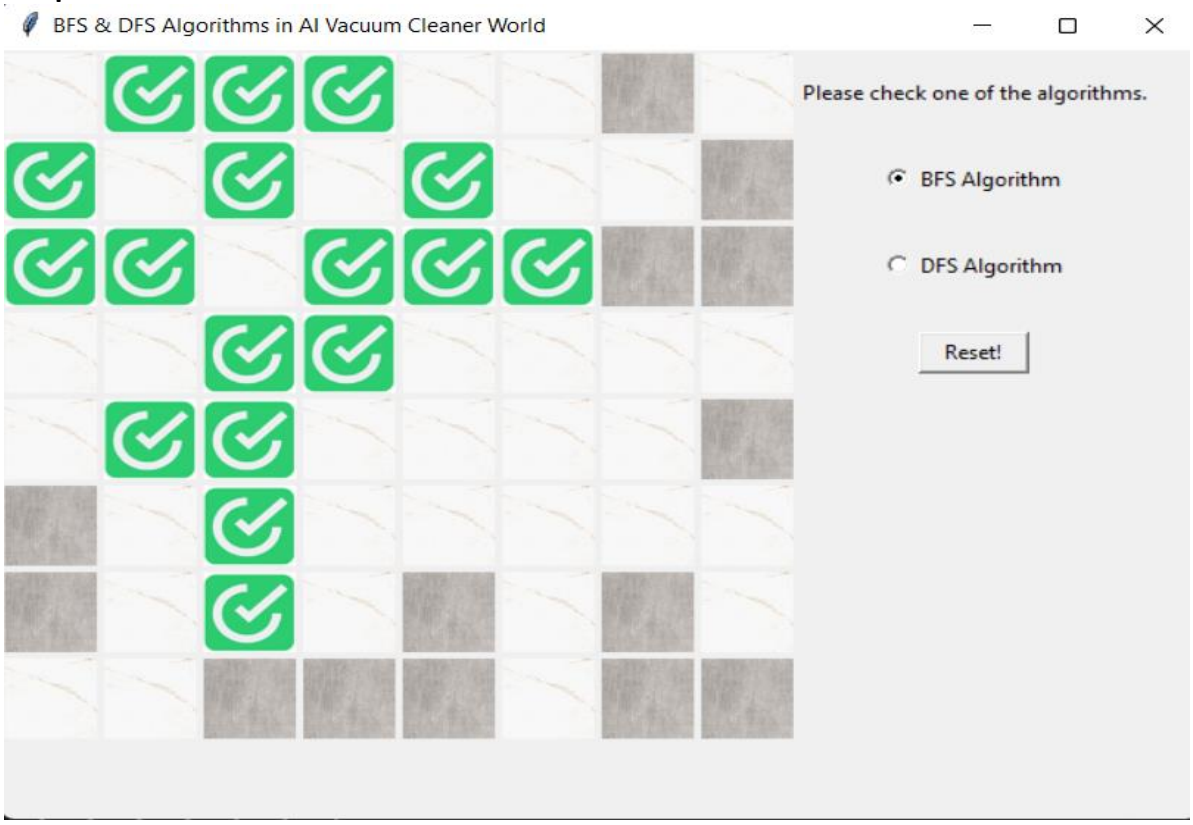
```
tmp_maps = maps[:]

print('-------------------------------------')
print('Current Cursor Location :' , vac_pos)
print('-------------------------------------')
print('env. before cleaning:')
#showing the whole env.
for i in range (rows):
    print (maps[i])
print('-------------------------------------')
#print('Steps and directions:')
#checking pieces
# Initializing a queue
qs = Queue(maxsize = 100000)
qd = Queue(maxsize = 100000)

#print(allowance(vac_str))
sys.setrecursionlimit(10**6)

def rbfs(vacpos,fro_d):
    def find_if_exist(frd):
        for p in allow:
            if frd == p:
                return True
        return False
    global visited_pieces
    global visited
    global cleaned_pieces
    global dirty_pieces
    global mem_cost

    if (cleaned_pieces==dirty_pieces):
        print('-------------------------------------')
        print('Environment Total Pieces :' , rows*cols)
        print('visited_pieces : ', visited_pieces)
        print('dirty_pieces : ', dirty_pieces)
        print('cleaned_pieces : ', cleaned_pieces)
        print('Memory Cost in BFS : ', mem_cost)
        #print('The Agent\'s Score : ', cleaned_pieces/visited_pieces)
        print('-------------------------------------')
        #showing the whole env. after cleaning
        print('env. after cleaning:')
        for i in range (rows):
            print (maps[i])
        return
    allow = allowance(vacpos)

    if (fro_d == 'right'):
```

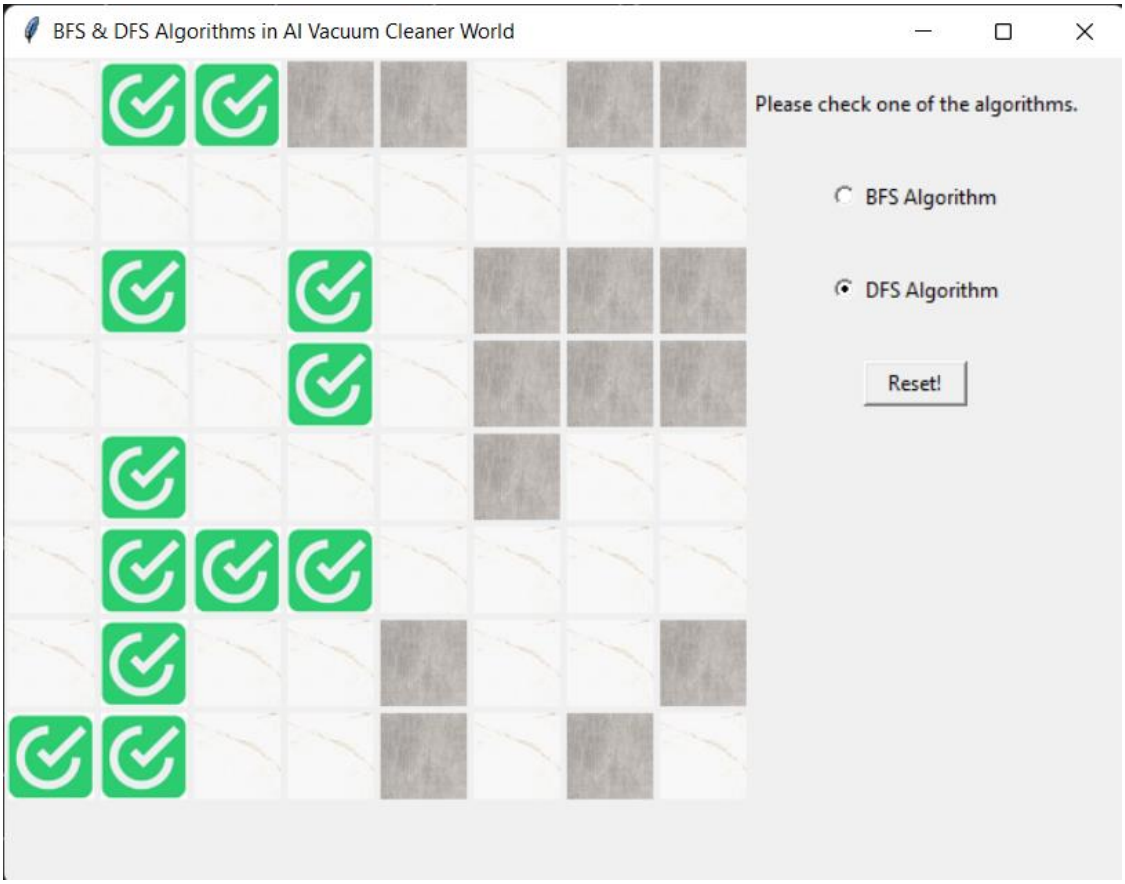
```
        frd = 'left'
    elif (fro_d == 'left'):
        frd = 'right'
    elif (fro_d == 'up'):
        frd = 'down'
    elif (fro_d == 'down'):
        frd = 'up'
    if (fro_d!='None'):
        if (find_if_exist(frd) == True):
            allow.remove(frd)

    if (visited[int(vacpos[0])-1][int(vacpos[1])-1]==0):
            m=0
            for m in range(len(allow)):
                    addEdge(graph,vacpos,dirs(vacpos,allow[m],0))
                    qs.put(dirs(vacpos,allow[m],0))
                    qd.put(allow[m])
                    mem_cost = mem_cost +1

    visited_pieces = visited_pieces + 1
    visited[int(vacpos[0])-1][int(vacpos[1])-1] =  1
    if maps[int(vacpos[0])-1][int(vacpos[1])-1] == 1:
        maps[int(vacpos[0])-1][int(vacpos[1])-1] = 0
        lab1[int(vacpos[0])-1][int(vacpos[1])-1].config(image = done_icon)
        mygui.update()
        sleep(0.5)
        cleaned_pieces = cleaned_pieces + 1

    news = qs.get()
    newd = qd.get()

    #print(news,'\n')
    rbfs(news,newd)

stacks = []
stackd = []
def rdfs(vacpos,fro_d):
    def find_if_exist(frd):
        for p in allow:
            if frd == p:
                return True
        return False
    global visited_pieces
    global cleaned_pieces
    global dirty_pieces
    global mem_cost
    global visited
```

```
    if (cleaned_pieces==dirty_pieces):
        print('-------------------------------------')
        print('Environment Total Pieces :' , rows*cols)
        print('visited_pieces : ', visited_pieces)
        print('dirty_pieces : ', dirty_pieces)
        print('cleaned_pieces : ', cleaned_pieces)
        print('Memory Cost in DFS : ', mem_cost)
        #print('The Agent\'s Score : ', cleaned_pieces/visited_pieces)
        print('-------------------------------------')
        #showing the whole env. after cleaning
        print('env. after cleaning:')
        for i in range (rows):
            print (maps[i])

        return
    allow = allowance(vacpos)
    if (fro_d == 'right'):
        frd = 'left'
    elif (fro_d == 'left'):
        frd = 'right'
    elif (fro_d == 'up'):
        frd = 'down'
    elif (fro_d == 'down'):
        frd = 'up'
    if (fro_d!='None'):
        if (find_if_exist(frd) == True):
            allow.remove(frd)

    if (visited[int(vacpos[0])-1][int(vacpos[1])-1]==0):
            m=0
            for m in range(len(allow)):
                addEdge(graph,vacpos,dirs(vacpos,allow[m],0))
                stacks.append(dirs(vacpos,allow[m],0))
                mem_cost = mem_cost +1
                stackd.append(allow[m])

    visited_pieces = visited_pieces + 1
    visited[int(vacpos[0])-1][int(vacpos[1])-1] = 1
    if maps[int(vacpos[0])-1][int(vacpos[1])-1] == 1:
        maps[int(vacpos[0])-1][int(vacpos[1])-1] = 0
        lab1[int(vacpos[0])-1][int(vacpos[1])-1].config(image = done_icon)
        mygui.update()
        sleep(0.5)

        cleaned_pieces = cleaned_pieces + 1

    news = stacks.pop()
    newd = stackd.pop()
```

```
    #print(news,'\n')
    rdfs(news,newd)
def check():
        global cb
        cb = v.get()
        if (cb == 1):
                qs.put(vac_str)
                tic = timeit.default_timer()
                rbfs(vac_str,'None')
                toc = timeit.default_timer()
                print("Timier: ",toc - tic - (cleaned_pieces)*0.5,"Seconds")
                sys.exit("Done!")
        elif(cb == 2):
                stacks.append(vac_str)
                rdfs(vac_str,'None')
                sys.exit("Done!")
v = IntVar()
Radiobutton(mygui,text="BFS Algorithm",padx=10,variable = v,value
=1,command=check).grid(row = 1,column=cols)
Radiobutton(mygui,text="DFS Algorithm",padx=10,variable = v,value
=2,command=check).grid(row = 2,column=cols)

Button(mygui,text="Reset!",padx=10,command=check).grid(row = 3,column=cols)
mygui.mainloop()
```

**Output:**






**Conclusion**:

DFS searches until it reaches the leaves of the tree or the last node of the graph that has no further successors, whereas BFS investigates all the nodes of a specific frontier before moving on to the next frontier. BFS is complete, whereas DFS isn't. BFS is better for nodes that are closer to the source and DFS is better for nodes that are further away from the source. Hence, we learnt to implement both BFS & DFS.