云南大学数学与统计学院
《算法图论实验》上机实践报告

课程名称：算法图论实验	**年级：**2015 级	**上机实践成绩：**
指导教师：李建平	**姓名：**刘鹏	**专业：**信息与计算科学
上机实践名称：图搜索算法	**学号：**20151910042	**上机实践日期：**2018-12-31
上机实践编号：1	**组号：**	

一、 实验目的

1. 理解广度优先搜索算法的具体步骤；
2. 学会读开源的代码库，并逐步学会使用这些代码库完成扩展性的实验。

二、 实验内容

1. 用形式化伪代码表示图的广度优先搜索算法；
2. 借助开源代码库，完成高质量的广度优先搜索算法编程。

三、 实验平台

Windows 10 Pro 1809；
MacOS Mojave 10.14.2；
Python 2；

四、 算法设计

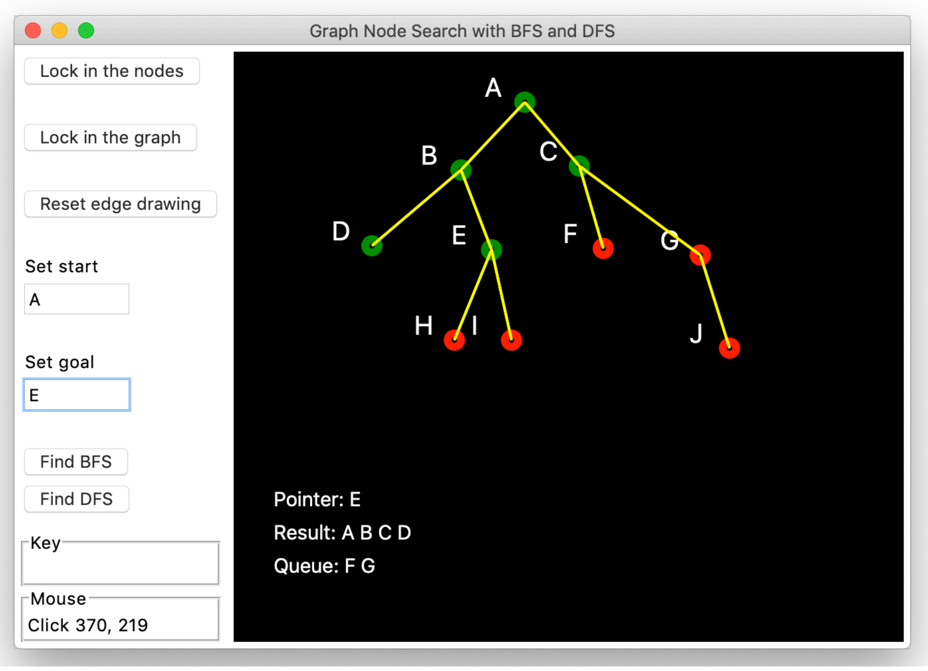
本次理论课上所讲的 Searching 算法在图论中一般被称为广度优先的图遍历算法（Breath First Searching, BFS）。在一定规则下循环地使用这个算法可以对一个图进行遍历，并得到所有的连通子图（连通分支）。这个算法十分重要，它是 Dijkstra 算法以及更一般的 Prim 算法的基础与原型。下面对 Searching 算法（广度优先图遍历算法）进行形式化描述。

```
Algorithm    图的反圈遍历算法，记此算法为SEARCH
Input        图$G = (V, E)$，并假定图$G$是无向图；
             图$G$中的某个起点$v$

Output       自$v_1$出发所有有路可到达的点以及路过的边所构成的诱导子图，记之为$\varepsilon$-CLOSURE =
             SEARCH$(G,\ v_1)$

Begin

Step 1       // 初始化染色
             for each vertex  $u \in (V - \{v\})$:
                $u.\textbf{color} = \textbf{White}$
                $u.\textbf{d} = \infty$
                $u.\boldsymbol{\pi} = \text{NIL}$

Step 2      // 初始化给定的起点
            $v.\textbf{color} = \textbf{Gray}$
            $v.\textbf{d} = 0$
            $v.\boldsymbol{\pi} = \text{NIL}$
            $Q = \emptyset$
            $\text{ENQUEUE}(Q,\ v)$

Step 3      while $Q \neq \emptyset$:
                $u = \text{DEQUEUE}(Q)$
                for each vertex $x \in \boldsymbol{\Phi}_G(u)$:
                    if $x.\textbf{color} == \textbf{White}$:
                        $\text{ENQUEUE}(Q,\ x)$
                        $x.\textbf{color} = \textbf{Gray}$
                        $x.\textbf{d} = u.\textbf{d} + 1$
                        $x.\boldsymbol{\pi} = u$
                $u.\textbf{color} = \textbf{Black}$

End
```

根据这个算法，可以很快写出图的极大连通分支搜索算法SUB-GRAPH(G)、图的连通性判断算法$\text{IS}-\text{CONNECTED}(G)$等。

五、 程序代码

由于染色的时候，黑白色在白底纹的显示屏上很不突出，而把底纹改成其他颜色的话纸质打印有会出现涂抹现象，所以代码中把染色变为了具有高对比度的红绿色。



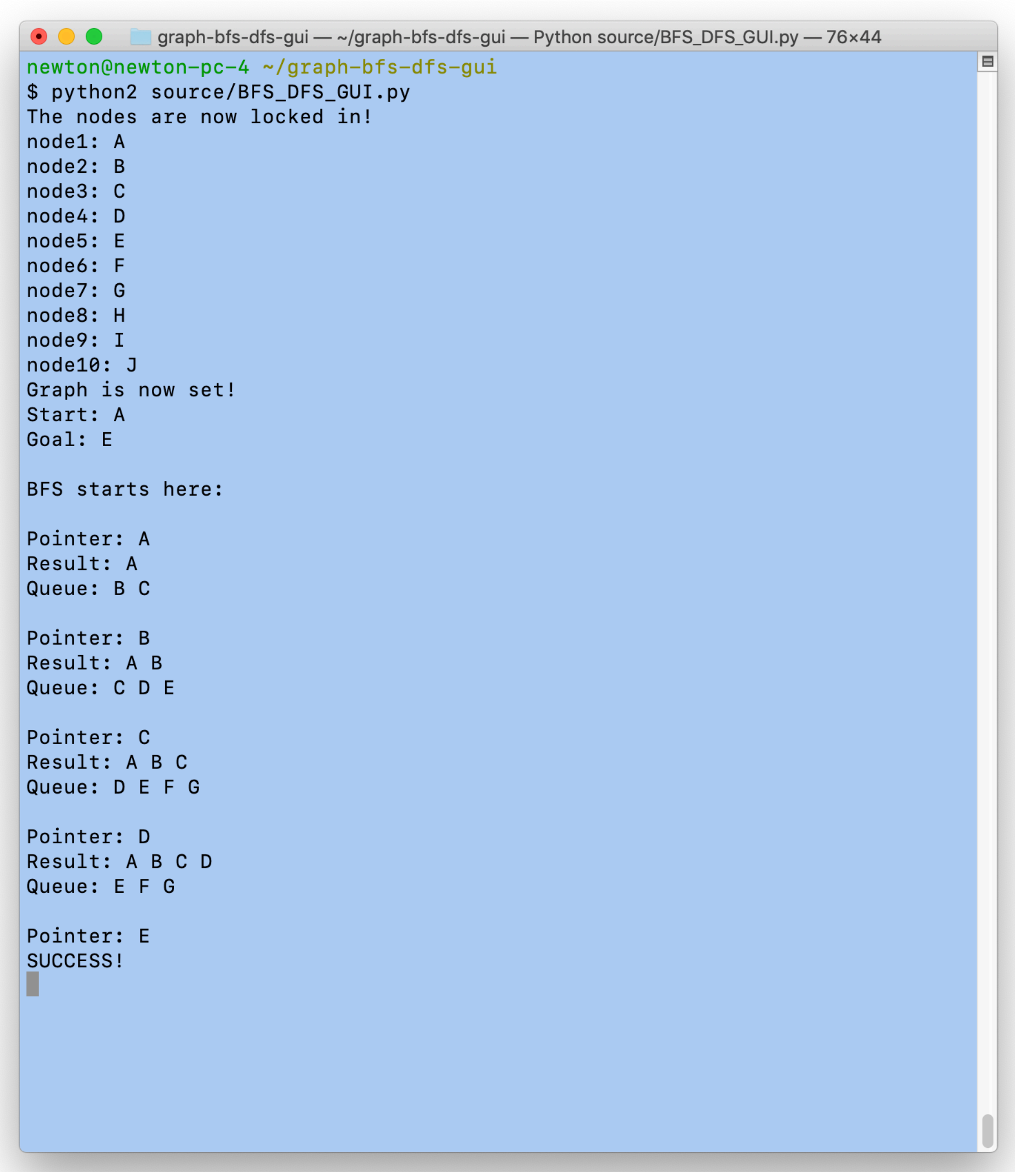
Set start: A

Set goal: E

Mouse: Click 370, 219

```
Pointer: E
Result: A B C D
Queue: F G
```

可以在下图中看出，求 A 到 E 的路时，队列有如命令行中所示的变化。


graph-bfs-dfs-gui — ~/graph-bfs-dfs-gui — Python source/BFS_DFS_GUI.py — 76×44

```
newton@newton-pc-4 ~/graph-bfs-dfs-gui
$ python2 source/BFS_DFS_GUI.py
The nodes are now locked in!
node1: A
node2: B
node3: C
node4: D
node5: E
node6: F
node7: G
node8: H
node9: I
node10: J
Graph is now set!
Start: A
Goal: E

BFS starts here:

Pointer: A
Result: A
Queue: B C

Pointer: B
Result: A B
Queue: C D E

Pointer: C
Result: A B C
Queue: D E F G

Pointer: D
Result: A B C D
Queue: E F G

Pointer: E
SUCCESS!
```

5.1　程序代码

由于 C 语言的 GUI 代码库太复杂，这里采用 Python 的 simpleGUItk 进行编程。核心语言还是 Python 2。

```
"""
    Title: A graph node search program in Python with the application of
           bread first and depth first search algorithms.
    Author: Ryan Gilera
"""


import simpleguitk as simplegui

# Constants
HEIGHT = 400
WIDTH = 500
NODE_SPACE_ALLOWANCE = 20
EDGE_COLOR = "Yellow"
EDGE_SIZE = 2
```

```
NODE_LABEL_COLOR = "White"
NODE_COLOR = "Red"
NODE_MARK_COLOR = "Green"

# Global variables
start = 0
goal = 0
placeNodes = True
setNodesRelation = False
draw_relations = False
draw_mark_relations =  False
setGoal = False
setStart = False
displayResult = False
lock_nodes = False
nodes = []

pos1 = [0,0]
pos2 = [0,0]
pos_lock = False
indx = 0
letter_label_default = '@'
letter_pos = 1
current_node_letters_low = []
current_node_letters_up = []


class Point:
    def __init__(self,pos,node_colour,node_mark_colour):
        self.pos = pos
        self.children = []
        self.radius = 5
        self.colour = node_colour
        self.mark_colour = node_mark_colour
        self.index = 0
        self.is_mark = False
        self.label = '@'

    def draw(self,canvas):
        if self.is_mark == False:
            canvas.draw_circle(self.pos, self.radius, 6, self.colour)
        else:
            canvas.draw_circle(self.pos, self.radius, 6, self.mark_colour)



def mouseclick(pos):
    global pos1, pos2, pos_lock, indx, draw_relations, draw_mark_relations, nodes, indx_mark_color
    global letter_label_default, letter_pos

    # Creates new instance of point(node) if the last position of
    # the mouseclick is not on  top of a previous node
    allow_place_node = True

    if placeNodes == True:
        if nodes: # Checks if the nodes are not empty
            for p, location in enumerate(nodes):
                if ((pos[0] >= (nodes[p].pos[0]-NODE_SPACE_ALLOWANCE) and pos[0] <= (nodes[p].pos[0]+NODE_SPACE_ALLOWANCE)) and
                    (pos[1] >= (nodes[p].pos[1]-NODE_SPACE_ALLOWANCE) and pos[1] <= (nodes[p].pos[1]+NODE_SPACE_ALLOWANCE))):
                    print "Warning: Cannot create node on top of another node!"
                    allow_place_node = False
                    break
            # Creates new instance of Point class if no nodes detected in
            # the vicinity of mouseclick position
            if allow_place_node == True:
                nodes.append(Point(pos,NODE_COLOR,NODE_MARK_COLOR))
                nodes[-1].label = chr(ord(letter_label_default) + letter_pos)
                letter_pos += 1
        # Else creates a node for first time
        else:
            nodes.append(Point(pos,NODE_COLOR,NODE_MARK_COLOR))
            nodes[-1].label = chr(ord(letter_label_default) + letter_pos)
            letter_pos += 1

    # Sets up the edges or links
    if setNodesRelation == True:

        # If mouseclick pos is on top of a current node mark that node
        for i, position in enumerate(nodes):
            if ((pos[0] >= (nodes[i].pos[0]-NODE_SPACE_ALLOWANCE) and pos[0] <= (nodes[i].pos[0]+NODE_SPACE_ALLOWANCE)) and
                (pos[1] >= (nodes[i].pos[1]-NODE_SPACE_ALLOWANCE) and pos[1] <= (nodes[i].pos[1]+NODE_SPACE_ALLOWANCE))):
                if pos_lock == False:
                    pos1[0] = pos[0]
                    pos1[1] = pos[1]

                    indx = i
                    indx_mark_color = i
                    pos_lock = True
                    draw_mark_relations = True
                    break

                else:
                    # If it is the second node that is not the same of
                    # the first marked node, then creates a new relation/edge
                    if i != indx:
                        pos2[0] = pos[0]
                        pos2[1] = pos[1]
                        nodes[indx].children.append(i)
                        nodes[i].children.append(indx)

                        pos_lock = False
                        draw_relations = True
                        draw_mark_relations = False
                        break
                    else:
                        print "Warning: Recursion or self loop is not allowed."


def button_refresh_new_relation():
    global pos_lock, pos1, pos2, nodes, draw_relations, draw_mark_relations

    if lock_nodes == False and setNodesRelation == True:
        pos_lock = False
        draw_mark_relations = False
        draw_relations = False
        pos1[0] = 0
        pos1[1] = 0
        pos2[0] = 0
        pos2[1] = 0

        # This empties the list of children attribute of Point class
        for i, child in enumerate(nodes):
            del nodes[i].children[:]
    else:
        print "Warning: This action is not allowed."


def button_lock_nodes():
    global placeNodes, setNodesRelation, current_node_letters_up, nodes, current_node_letters_low

    # Can only lock nodes if the set-up is right
    # Prevents locking nodes later in the program
    if placeNodes == True and setNodesRelation == False and setStart == False and setGoal == False:
        placeNodes = False
        setNodesRelation = True

        # Fills two new lists of all node labels(letters)
        # for later use in input start and goal
        if nodes:
            for n, obj in enumerate(nodes):
                current_node_letters_up.append(nodes[n].label)

            for let in current_node_letters_up:
                current_node_letters_low.append(let.lower())

        print "The nodes are now locked in!"
    else:
        print "Warning: This action is not allowed."

def button_lock_graph():
    global placeNodes, setNodesRelation, nodes, lock_nodes

    if setNodesRelation is True:
        placeNodes = False
        setNodesRelation = False
        lock_nodes = True

        # Sets the index of nodes list and apply it to each index attribute of Point class
        # for index/element reference only, for later use in BFS and DFS functions
        for d, dot in enumerate(nodes):
            nodes[d].index = d
            print "node"+str(d+1)+":", nodes[d].label

            # This is important
            # This sorts the elements of children attribute list in ascending order
            nodes[d].children.sort()

        print "Graph is now set!"
    else:
        print "Warning: This action is not allowed."


def input_start_handler(start_string):
    global start, nodes, setStart

    setStart = False
    if start_string.isdigit():
        # Allows number as input for starting node
        # 1 for A, 2 for B and so on and so forth
        temp_start = int(start_string) - 1
        for element, num in enumerate(nodes):
            if temp_start == element:

                # Minus one because node label starts at 1 not zero(index)
                start = temp_start
                print "Start:", chr(start+65)
                setStart = True
                break
        if setStart == False:
            print "Warning: This number is outside of the nodes!"
    else:
        # Allows letter as input for starting node
        if start_string in current_node_letters_up:
            start = ord(start_string) - 65
            setStart = True
            print "Start:", chr(start+65)
        else:
            if start_string in current_node_letters_low:
                start = ord(start_string) - 97
                setStart = True
                print "Start:", chr(start+65)
            else:
                print "Warning: Unknown input. Enter a number or the node letter."


def input_goal_handler(goal_string):
    global goal, nodes, setGoal

    setGoal = False
    if goal_string.isdigit():

        # Allows number as input for goal node
        # 1 for A, 2 for B and so on and so forth
        temp_goal = int(goal_string) - 1
        for element, num in enumerate(nodes):
            if temp_goal == element:
                #minus one because node label starts at 1 not zero(index)
                goal = temp_goal
                print "Goal:", chr(goal+65)
                setGoal = True
                break
        if setGoal == False:
            print "Warning: This number is outside of the nodes!"
    else:
        # Allows letter as input for goal node
        if goal_string in current_node_letters_up:
            goal = ord(goal_string) - 65
            setGoal = True
            print "Goal:", chr(goal+65)
        else:
            if goal_string in current_node_letters_low:
                goal = ord(goal_string) - 97
                setGoal = True
                print "Goal:", chr(goal+65)
            else:
                print "Warning: Unknown input. Enter a number or the node letter."


def button_breadth_first_search():
    global nodes, displayResult, result_string, queue_string, pointer_string
    displayResult = False
    pointer_string = ""

    # Resets all nodes markings (color)
    for d, marking_obj in enumerate(nodes):
        nodes[d].is_mark = False

    in_queue_result = False

    if placeNodes == False and setNodesRelation == False and setStart == True and setGoal == True:
        print " "
        print "BFS starts here:"

        # Checks queue if defined,
        # if it is, then go to else and empty the list; otherwise create a new list
        try:
            queue
        except:
            queue = []
        else:
            del queue[:]

        queue.append(nodes[start])
        queue[0].is_mark = True

        try:
            result
        except:
            result = []
        else:
            del result[:]

        while queue:
            pointer = queue[0]
            queue.pop(0)

            pointer.is_mark = True
            print " "
            print "Pointer:", pointer.label

            if pointer.index == goal:
                pointer_string =  "Pointer: " + pointer.label
                result_string = "Result: "
                queue_string = "Queue: "

                for obj in result:
                    result_string += str(obj.label)
                    result_string += " "
                for objt in queue:
                    queue_string += str(objt.label)
                    queue_string += " "

                displayResult = True
                print "SUCCESS!"
                break
            else:
                result.append(pointer)

                for neighbor in pointer.children:
                    in_queue_result = False
                    for i in queue:
                        #print "neighbor:", neighbor+1, "queue:", i.index+1
                        if neighbor == i.index:
                            in_queue_result = True

                    for j in result:
                        #print "neighbor:", neighbor+1, "result:", j.index+1
                        if neighbor == j.index:
                            in_queue_result = True

                    if in_queue_result == False:
                        for objct in nodes:
                            if objct.index == neighbor:
                                queue.append(nodes[objct.index])
            result_string = "Result: "
            queue_string = "Queue: "
            for obj in result:
                result_string += str(obj.label)
                result_string += " "
            print result_string

            for objt in queue:
                queue_string += str(objt.label)
                queue_string += " "
            print queue_string


def button_depth_first_search():
    global nodes, displayResult, result_string, queue_string, pointer_string
    displayResult = False
    pointer_string = ""

# Resets all nodes markings (color)
    for d, marking_obj in enumerate(nodes):
        nodes[d].is_mark = False

    in_queue_result = False

    if placeNodes == False and setNodesRelation == False and setStart == True and setGoal == True:
        print " "
        print "DFS starts here:"

        # Checks queue if defined,
        # if it is, then go to else and empty the list; otherwise create new list
        try:
            queue
        except:
            queue = []
        else:
            del queue[:]

        #print queue
        queue.append(nodes[start])
        queue[0].is_mark = True
        #print "queue:", queue
        try:
            result
        except:
            result = []
        else:
            del result[:]


        try:
            temp_list
        except:
            temp_list = []
        else:
            del temp_list[:]


        while queue:
            pointer = queue[0]
            queue.pop(0)
            #print "pointer is", pointer
            pointer.is_mark = True
            print " "
            print "Pointer:", pointer.label


            if pointer.index == goal:
                pointer_string =  "Pointer: " + pointer.label
                result_string = "Result: "
                queue_string = "Queue: "

                for obj in result:
                    result_string += str(obj.label)
                    result_string += " "
                for objt in queue:
                    queue_string += str(objt.label)
                    queue_string += " "

                displayResult = True
                print "SUCCESS!"
                break
            else:
                result.append(pointer)
                del temp_list[:]

                for neighbor in pointer.children:
                    in_queue_result = False

                    for i in queue:
                        #print "neighbor:", neighbor+1, "queue:", i.index+1
                        if neighbor == i.index:
                            in_queue_result = True
```

```python

                    for j in result:
                        #print "neighbor:", neighbor+1, "result:", j.index+1
                        if neighbor == j.index:
                            in_queue_result = True

                    if in_queue_result == False:
                        for obj in nodes:
                            if obj.index == neighbor:
                                temp_list.append(nodes[obj.index])

                if temp_list:
                    queue[0:0] = temp_list

            result_string = "Result: "
            queue_string = "Queue: "
            for obj in result:
                result_string += str(obj.label)
                result_string += " "
            print result_string

            for objt in queue:
                queue_string += str(objt.label)
                queue_string += " "
            print queue_string


def draw_handler(canvas):
    global result_string, queue_string, pointer_string
    global placeNodes, setNodesRelation, setStart, setGoal, pos1

    # Draws nodes
    if draw_mark_relations == True and setNodesRelation == True:
        canvas.draw_circle(nodes[indx_mark_color].pos, 15, 3, "Yellow", "Black")

    if nodes:
        for i, vertex in enumerate(nodes):
            nodes[i].draw(canvas)
            canvas.draw_text(nodes[i].label, (nodes[i].pos[0]-30, nodes[i].pos[1]), 20, NODE_LABEL_COLOR)

    # Draws edges
    if draw_relations == True:
        for n, point in enumerate(nodes):
            if nodes[n].children:
                for child in nodes[n].children:
                    canvas.draw_line(nodes[n].pos, nodes[child].pos, EDGE_SIZE, EDGE_COLOR)

    # Display results
    if displayResult == True:
        canvas.draw_text(pointer_string, (30, 345), 15, NODE_LABEL_COLOR)
        canvas.draw_text(result_string, (30, 370), 15, NODE_LABEL_COLOR)
        canvas.draw_text(queue_string, (30, 395), 15, NODE_LABEL_COLOR)


# Creates the frame window
frame = simplegui.create_frame("Graph Node Search with BFS and DFS",WIDTH,HEIGHT)

frame.set_mouseclick_handler(mouseclick)
frame.set_draw_handler(draw_handler)

# Button, input and label controls for the frame window
button1 = frame.add_button('Lock in the nodes', button_lock_nodes)
label1 = frame.add_label(' ')

button2 = frame.add_button('Lock in the graph', button_lock_graph)
label2 = frame.add_label(' ')

button3 = frame.add_button('Reset edge drawing', button_refresh_new_relation)
label3 = frame.add_label(' ')

input_start = frame.add_input('Set start', input_start_handler, 50)
label4 = frame.add_label(' ')

input_goal = frame.add_input('Set goal', input_goal_handler, 50)
label5 = frame.add_label(' ')

button4 = frame.add_button('Find BFS', button_breadth_first_search)
button5 = frame.add_button('Find DFS', button_depth_first_search)


# Program starts here
frame.start()
```

六、 参考文献

[1] **林锐**. 高质量 C++/C 编程指南 [M]. 1.0 ed., 2001.

[2] 算法精解：C 语言描述：https://github.com/yourtion/LearningMasteringAlgorithms-C

[3] https://github.com/Daytron/graph-bfs-dfs-gui