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ε-CLOSURE $=$ SEARCH$(G,\ v_1)$

Begin

Step 1 // 初始化染色
for each vertex $u \in (V-\{v\})$:

$u.\mathbf{color} = \mathbf{White}$
$u.\mathbf{d} = \infty$
$u.\boldsymbol{\pi} = \text{NIL}$

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

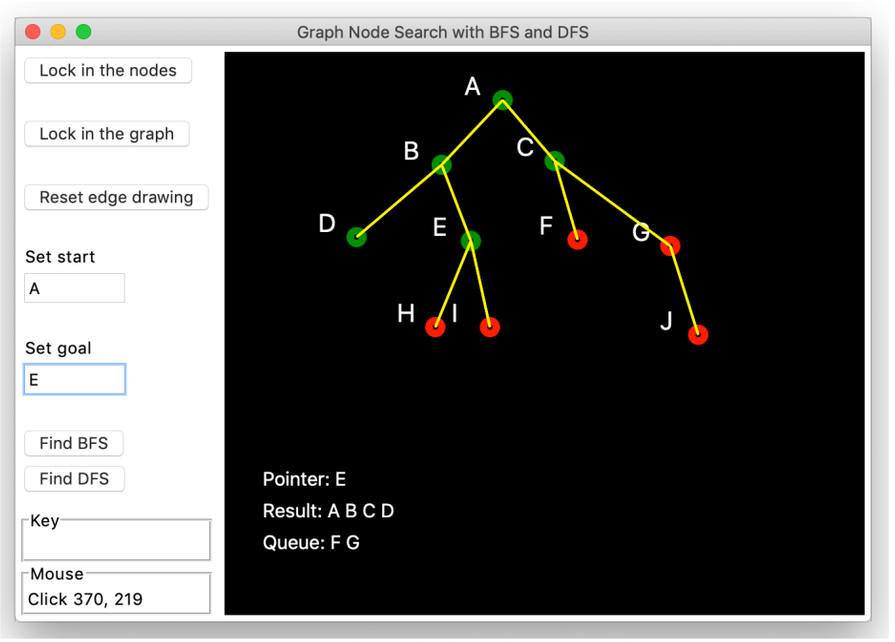
Step 3 **while** $Q \neq \emptyset$:
$u = \text{DEQUEUE}(Q)$
for each vertex $x \in \mathbf{\Phi}_G(u)$:
if $x.\mathbf{color} == \mathbf{White}$:
ENQUEUE$(Q,\ x)$
$x.\mathbf{color} = \mathbf{Gray}$
$x.\mathbf{d} = u.\mathbf{d} + 1$
$x.\boldsymbol{\pi} = u$
$u.\mathbf{color} = \mathbf{Black}$

End

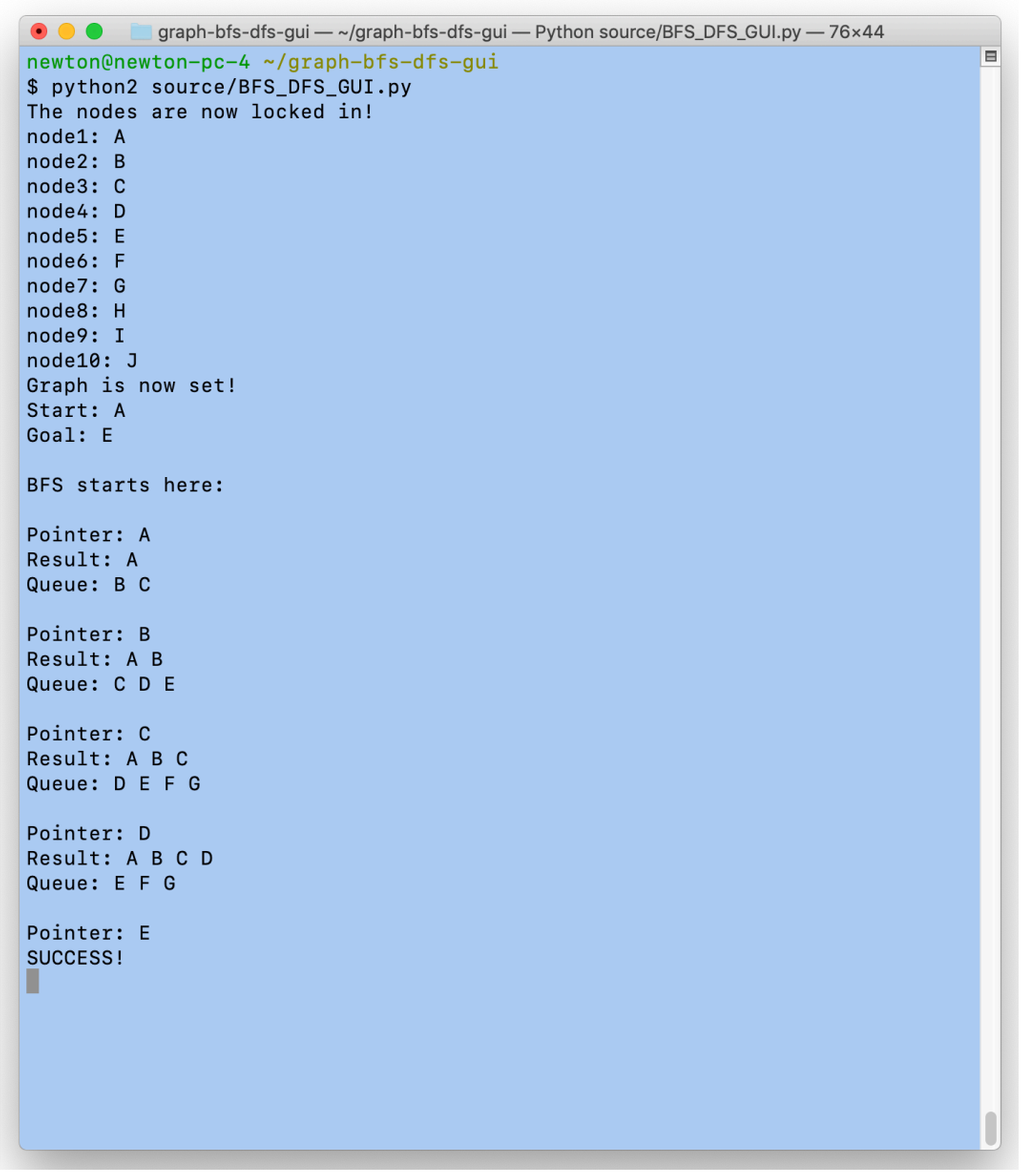
根据这个算法，可以很快写出图的极大连通分支搜索算法SUB-GRAPH(G)、图的连通性判断算法IS$-$CONNECTED(G)等。

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。



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。



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
```

```
16  NODE_LABEL_COLOR = "White"
17  NODE_COLOR = "Red"
18  NODE_MARK_COLOR = "Green"
19
20  # Global variables
21  start = 0
22  goal = 0
23  placeNodes = True
24  setNodesRelation = False
25  draw_relations = False
26  draw_mark_relations =  False
27  setGoal = False
28  setStart = False
29  displayResult = False
30  lock_nodes = False
31  nodes = []
32
33  pos1 = [0,0]
34  pos2 = [0,0]
35  pos_lock = False
36  indx = 0
37  letter_label_default = '@'
38  letter_pos = 1
39  current_node_letters_low = []
40  current_node_letters_up = []
41
42
43  class Point:
44      def __init__(self,pos,node_colour,node_mark_colour):
45          self.pos = pos
46          self.children = []
47          self.radius = 5
48          self.colour = node_colour
49          self.mark_colour = node_mark_colour
50          self.index = 0
51          self.is_mark = False
52          self.label = '@'
53
54      def draw(self,canvas):
55          if self.is_mark == False:
56              canvas.draw_circle(self.pos, self.radius, 6, self.colour)
57          else:
58              canvas.draw_circle(self.pos, self.radius, 6, self.mark_colour)
59
60
61
62  def mouseclick(pos):
63      global pos1, pos2, pos_lock, indx, draw_relations, draw_mark_relations, nodes, indx_mark_color
64      global letter_label_default, letter_pos
65
66      # Creates new instance of point(node) if the last position of
67      # the mouseclick is not on  top of a previous node
68      allow_place_node = True
69
70      if placeNodes == True:
71          if nodes: # Checks if the nodes are not empty
72              for p, location in enumerate(nodes):
73                  if ((pos[0] >= (nodes[p].pos[0]-NODE_SPACE_ALLOWANCE) and pos[0] <= (nodes[p].pos[0]+NODE_SPACE_ALLOWANCE)) and
```

```
74                      (pos[1] >= (nodes[p].pos[1]-NODE_SPACE_ALLOWANCE) and pos[1] <= (nodes[p].pos[1]+NODE_SPACE_ALLOWANCE))):
75                      print "Warning: Cannot create node on top of another node!"
76                      allow_place_node = False
77                      break
78              # Creates new instance of Point class if no nodes detected in
79              # the vicinity of mouseclick position
80              if allow_place_node == True:
81                  nodes.append(Point(pos,NODE_COLOR,NODE_MARK_COLOR))
82                  nodes[-1].label = chr(ord(letter_label_default) + letter_pos)
83                  letter_pos += 1
84          # Else creates a node for first time
85          else:
86              nodes.append(Point(pos,NODE_COLOR,NODE_MARK_COLOR))
87              nodes[-1].label = chr(ord(letter_label_default) + letter_pos)
88              letter_pos += 1
89
90      # Sets up the edges or links
91      if setNodesRelation == True:
92
93          # If mouseclick pos is on top of a current node mark that node
94          for i, position in enumerate(nodes):
95              if ((pos[0] >= (nodes[i].pos[0]-NODE_SPACE_ALLOWANCE) and pos[0] <= (nodes[i].pos[0]+NODE_SPACE_ALLOWANCE)) and
96                  (pos[1] >= (nodes[i].pos[1]-NODE_SPACE_ALLOWANCE) and pos[1] <= (nodes[i].pos[1]+NODE_SPACE_ALLOWANCE))):
97                  if pos_lock == False:
98                      pos1[0] = pos[0]
99                      pos1[1] = pos[1]
100
101                     indx = i
102                     indx_mark_color = i
103                     pos_lock = True
104                     draw_mark_relations = True
105                     break
106
107                 else:
108                     # If it is the second node that is not the same of
109                     # the first marked node, then creates a new relation/edge
110                     if i != indx:
111                         pos2[0] = pos[0]
112                         pos2[1] = pos[1]
113                         nodes[indx].children.append(i)
114                         nodes[i].children.append(indx)
115
116                         pos_lock = False
117                         draw_relations = True
118                         draw_mark_relations = False
119                         break
120                     else:
121                         print "Warning: Recursion or self loop is not allowed."
122
123
124 def button_refresh_new_relation():
125     global pos_lock, pos1, pos2, nodes, draw_relations, draw_mark_relations
126
127     if lock_nodes == False and setNodesRelation == True:
128         pos_lock = False
129         draw_mark_relations = False
130         draw_relations = False
131         pos1[0] = 0
```

```
132         pos1[1] = 0
133         pos2[0] = 0
134         pos2[1] = 0
135 
136         # This empties the list of children attribute of Point class
137         for i, child in enumerate(nodes):
138             del nodes[i].children[:]
139     else:
140         print "Warning: This action is not allowed."
141 
142 
143 def button_lock_nodes():
144     global placeNodes, setNodesRelation, current_node_letters_up, nodes, current_node_letters_low
145 
146     # Can only lock nodes if the set-up is right
147     # Prevents locking nodes later in the program
148     if placeNodes == True and setNodesRelation == False and setStart == False and setGoal == False:
149         placeNodes = False
150         setNodesRelation = True
151 
152         # Fills two new lists of all node labels(letters)
153         # for later use in input start and goal
154         if nodes:
155             for n, obj in enumerate(nodes):
156                 current_node_letters_up.append(nodes[n].label)
157 
158             for let in current_node_letters_up:
159                 current_node_letters_low.append(let.lower())
160 
161         print "The nodes are now locked in!"
162     else:
163         print "Warning: This action is not allowed."
164 
165 def button_lock_graph():
166     global placeNodes, setNodesRelation, nodes, lock_nodes
167 
168     if setNodesRelation is True:
169         placeNodes = False
170         setNodesRelation = False
171         lock_nodes = True
172 
173         # Sets the index of nodes list and apply it to each index attribute of Point class
174         # for index/element reference only, for later use in BFS and DFS functions
175         for d, dot in enumerate(nodes):
176             nodes[d].index = d
177             print "node"+str(d+1)+":", nodes[d].label
178 
179             # This is important
180             # This sorts the elements of children attribute list in ascending order
181             nodes[d].children.sort()
182 
183         print "Graph is now set!"
184     else:
185         print "Warning: This action is not allowed."
186 
187 
188 def input_start_handler(start_string):
189     global start, nodes, setStart
```

```
190
191     setStart = False
192     if start_string.isdigit():
193         # Allows number as input for starting node
194         # 1 for A, 2 for B and so on and so forth
195         temp_start = int(start_string) - 1
196         for element, num in enumerate(nodes):
197             if temp_start == element:
198
199                 # Minus one because node label starts at 1 not zero(index)
200                 start = temp_start
201                 print "Start:", chr(start+65)
202                 setStart = True
203                 break
204         if setStart == False:
205             print "Warning: This number is outside of the nodes!"
206     else:
207         # Allows letter as input for starting node
208         if start_string in current_node_letters_up:
209             start = ord(start_string) - 65
210             setStart = True
211             print "Start:", chr(start+65)
212         else:
213             if start_string in current_node_letters_low:
214                 start = ord(start_string) - 97
215                 setStart = True
216                 print "Start:", chr(start+65)
217             else:
218                 print "Warning: Unknown input. Enter a number or the node letter."
219
220
221 def input_goal_handler(goal_string):
222     global goal, nodes, setGoal
223
224     setGoal = False
225     if goal_string.isdigit():
226
227         # Allows number as input for goal node
228         # 1 for A, 2 for B and so on and so forth
229         temp_goal = int(goal_string) - 1
230         for element, num in enumerate(nodes):
231             if temp_goal == element:
232                 #minus one because node label starts at 1 not zero(index)
233                 goal = temp_goal
234                 print "Goal:", chr(goal+65)
235                 setGoal = True
236                 break
237         if setGoal == False:
238             print "Warning: This number is outside of the nodes!"
239     else:
240         # Allows letter as input for goal node
241         if goal_string in current_node_letters_up:
242             goal = ord(goal_string) - 65
243             setGoal = True
244             print "Goal:", chr(goal+65)
245         else:
246             if goal_string in current_node_letters_low:
247                 goal = ord(goal_string) - 97
```

```
248                 setGoal = True
249                 print "Goal:", chr(goal+65)
250             else:
251                 print "Warning: Unknown input. Enter a number or the node letter."
252 
253 
254 def button_breadth_first_search():
255     global nodes, displayResult, result_string, queue_string, pointer_string
256     displayResult = False
257     pointer_string = ""
258 
259     # Resets all nodes markings (color)
260     for d, marking_obj in enumerate(nodes):
261         nodes[d].is_mark = False
262 
263     in_queue_result = False
264 
265     if placeNodes == False and setNodesRelation == False and setStart == True and setGoal == True:
266         print " "
267         print "BFS starts here:"
268 
269         # Checks queue if defined,
270         # if it is, then go to else and empty the list; otherwise create a new list
271         try:
272             queue
273         except:
274             queue = []
275         else:
276             del queue[:]
277 
278         queue.append(nodes[start])
279         queue[0].is_mark = True
280 
281         try:
282             result
283         except:
284             result = []
285         else:
286             del result[:]
287 
288         while queue:
289             pointer = queue[0]
290             queue.pop(0)
291 
292             pointer.is_mark = True
293             print " "
294             print "Pointer:", pointer.label
295 
296             if pointer.index == goal:
297                 pointer_string =  "Pointer: " + pointer.label
298                 result_string = "Result: "
299                 queue_string = "Queue: "
300 
301                 for obj in result:
302                     result_string += str(obj.label)
303                     result_string += " "
304                 for objt in queue:
305                     queue_string += str(objt.label)
```

```
306                     queue_string += " "
307
308                 displayResult = True
309                 print "SUCCESS!"
310                 break
311             else:
312                 result.append(pointer)
313
314                 for neighbor in pointer.children:
315                     in_queue_result = False
316                     for i in queue:
317                         #print "neighbor:", neighbor+1, "queue:", i.index+1
318                         if neighbor == i.index:
319                             in_queue_result = True
320
321                     for j in result:
322                         #print "neighbor:", neighbor+1, "result:", j.index+1
323                         if neighbor == j.index:
324                             in_queue_result = True
325
326                     if in_queue_result == False:
327                         for objct in nodes:
328                             if objct.index == neighbor:
329                                 queue.append(nodes[objct.index])
330             result_string = "Result: "
331             queue_string = "Queue: "
332             for obj in result:
333                 result_string += str(obj.label)
334                 result_string += " "
335             print result_string
336
337             for objt in queue:
338                 queue_string += str(objt.label)
339                 queue_string += " "
340             print queue_string
341
342
343 def button_depth_first_search():
344     global nodes, displayResult, result_string, queue_string, pointer_string
345     displayResult = False
346     pointer_string = ""
347
348 # Resets all nodes markings (color)
349     for d, marking_obj in enumerate(nodes):
350         nodes[d].is_mark = False
351
352     in_queue_result = False
353
354     if placeNodes == False and setNodesRelation == False and setStart == True and setGoal == True:
355         print " "
356         print "DFS starts here:"
357
358         # Checks queue if defined,
359         # if it is, then go to else and empty the list; otherwise create new list
360         try:
361             queue
362         except:
363             queue = []
```

```
364         else:
365             del queue[:]
366 
367         #print queue
368         queue.append(nodes[start])
369         queue[0].is_mark = True
370         #print "queue:", queue
371         try:
372             result
373         except:
374             result = []
375         else:
376             del result[:]
377 
378 
379         try:
380             temp_list
381         except:
382             temp_list = []
383         else:
384             del temp_list[:]
385 
386 
387         while queue:
388             pointer = queue[0]
389             queue.pop(0)
390             #print "pointer is", pointer
391             pointer.is_mark = True
392             print " "
393             print "Pointer:", pointer.label
394 
395 
396             if pointer.index == goal:
397                 pointer_string =  "Pointer: " + pointer.label
398                 result_string = "Result: "
399                 queue_string = "Queue: "
400 
401                 for obj in result:
402                     result_string += str(obj.label)
403                     result_string += " "
404                 for objt in queue:
405                     queue_string += str(objt.label)
406                     queue_string += " "
407 
408                 displayResult = True
409                 print "SUCCESS!"
410                 break
411             else:
412                 result.append(pointer)
413                 del temp_list[:]
414 
415                 for neighbor in pointer.children:
416                     in_queue_result = False
417 
418                     for i in queue:
419                         #print "neighbor:", neighbor+1, "queue:", i.index+1
420                         if neighbor == i.index:
421                             in_queue_result = True
```

```
422
423                     for j in result:
424                         #print "neighbor:", neighbor+1, "result:", j.index+1
425                         if neighbor == j.index:
426                             in_queue_result = True
427
428                     if in_queue_result == False:
429                         for obj in nodes:
430                             if obj.index == neighbor:
431                                 temp_list.append(nodes[obj.index])
432
433                 if temp_list:
434                     queue[0:0] = temp_list
435
436             result_string = "Result: "
437             queue_string = "Queue: "
438             for obj in result:
439                 result_string += str(obj.label)
440                 result_string += " "
441             print result_string
442
443             for objt in queue:
444                 queue_string += str(objt.label)
445                 queue_string += " "
446             print queue_string
447
448
449 def draw_handler(canvas):
450     global result_string, queue_string, pointer_string
451     global placeNodes, setNodesRelation, setStart, setGoal, pos1
452
453     # Draws nodes
454     if draw_mark_relations == True and setNodesRelation == True:
455         canvas.draw_circle(nodes[indx_mark_color].pos, 15, 3, "Yellow", "Black")
456
457     if nodes:
458         for i, vertex in enumerate(nodes):
459             nodes[i].draw(canvas)
460             canvas.draw_text(nodes[i].label, (nodes[i].pos[0]-30, nodes[i].pos[1]), 20, NODE_LABEL_COLOR)
461
462     # Draws edges
463     if draw_relations == True:
464         for n, point in enumerate(nodes):
465             if nodes[n].children:
466                 for child in nodes[n].children:
467                     canvas.draw_line(nodes[n].pos, nodes[child].pos, EDGE_SIZE, EDGE_COLOR)
468
469     # Display results
470     if displayResult == True:
471         canvas.draw_text(pointer_string, (30, 345), 15, NODE_LABEL_COLOR)
472         canvas.draw_text(result_string, (30, 370), 15, NODE_LABEL_COLOR)
473         canvas.draw_text(queue_string, (30, 395), 15, NODE_LABEL_COLOR)
474
475
476 # Creates the frame window
477 frame = simplegui.create_frame("Graph Node Search with BFS and DFS",WIDTH,HEIGHT)
478
479 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