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： |  |

# 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。

# 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。

# 实验平台

Windows 10 Pro 1809；

MacOS Mojave 10.14.2；

Python 2；

# 算法设计

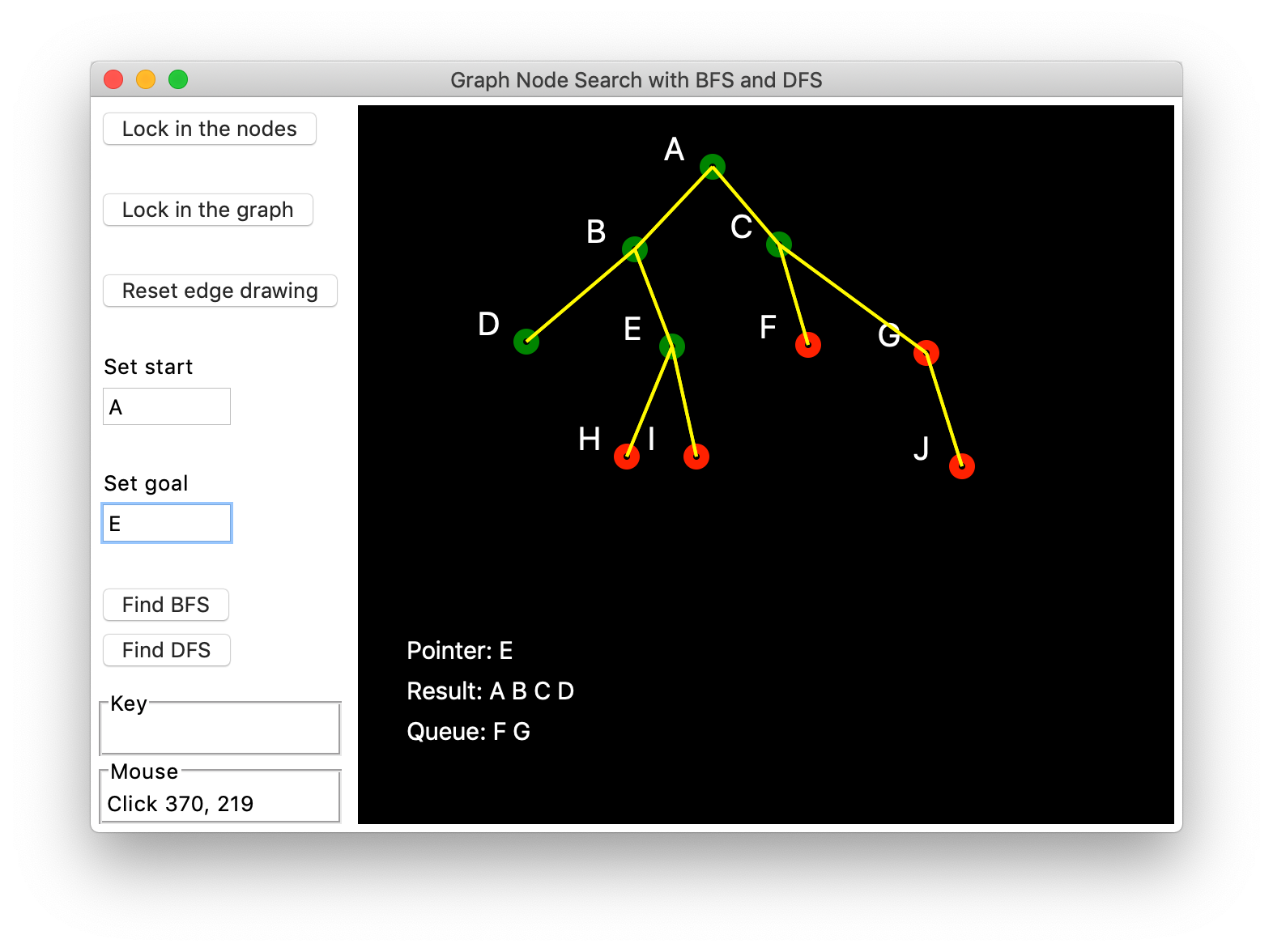
本次理论课上所讲的Searching算法在图论中一般被称为广度优先的图遍历算法（Breath First Searching, BFS）。在一定规则下循环地使用这个算法可以对一个图进行遍历，并得到所有的连通子图（连通分支）。这个算法十分重要，它是Dijkstra算法以及更一般的Prim算法的基础与原型。下面对Searching算法（广度优先图遍历算法）进行形式化描述。

|  |  |
| --- | --- |
| **Algorithm**  **Input**  **Output**  **Begin**  **Step 1**  **Step 2**  **Step 3**  **End** | 图的反圈遍历算法，记此算法为  图，并假定图是无向图；  图中的某个起点  自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-  // 初始化染色  **for** **each** vertex :        // 初始化给定的起点  **while** :    **for** **each** vertex :  **if** : |

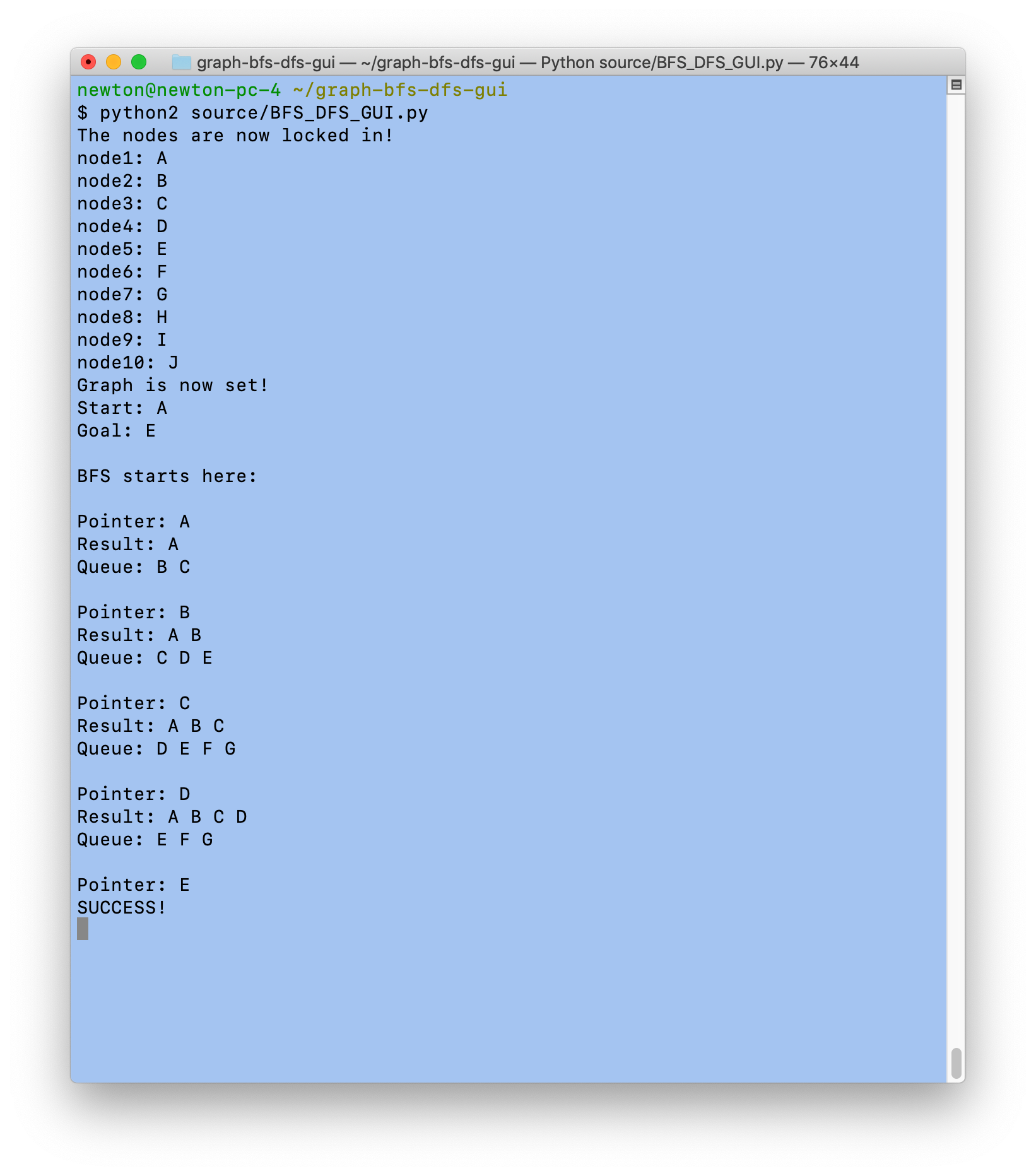
根据这个算法，可以很快写出图的极大连通分支搜索算法-、图的连通性判断算法等。

# 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。



## 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。

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90  91  92  93  94  95  96  97  98  99  100  101  102  103  104  105  106  107  108  109  110  111  112  113  114  115  116  117  118  119  120  121  122  123  124  125  126  127  128  129  130  131  132  133  134  135  136  137  138  139  140  141  142  143  144  145  146  147  148  149  150  151  152  153  154  155  156  157  158  159  160  161  162  163  164  165  166  167  168  169  170  171  172  173  174  175  176  177  178  179  180  181  182  183  184  185  186  187  188  189  190  191  192  193  194  195  196  197  198  199  200  201  202  203  204  205  206  207  208  209  210  211  212  213  214  215  216  217  218  219  220  221  222  223  224  225  226  227  228  229  230  231  232  233  234  235  236  237  238  239  240  241  242  243  244  245  246  247  248  249  250  251  252  253  254  255  256  257  258  259  260  261  262  263  264  265  266  267  268  269  270  271  272  273  274  275  276  277  278  279  280  281  282  283  284  285  286  287  288  289  290  291  292  293  294  295  296  297  298  299  300  301  302  303  304  305  306  307  308  309  310  311  312  313  314  315  316  317  318  319  320  321  322  323  324  325  326  327  328  329  330  331  332  333  334  335  336  337  338  339  340  341  342  343  344  345  346  347  348  349  350  351  352  353  354  355  356  357  358  359  360  361  362  363  364  365  366  367  368  369  370  371  372  373  374  375  376  377  378  379  380  381  382  383  384  385  386  387  388  389  390  391  392  393  394  395  396  397  398  399  400  401  402  403  404  405  406  407  408  409  410  411  412  413  414  415  416  417  418  419  420  421  422  423  424  425  426  427  428  429  430  431  432  433  434  435  436  437  438  439  440  441  442  443  444  445  446  447  448  449  450  451  452  453  454  455  456  457  458  459  460  461  462  463  464  465  466  467  468  469  470  471  472  473  474  475  476  477  478  479  480  481  482  483  484  485  486  487  488  489  490  491  492  493  494  495  496  497  498  499  500  501  502  503 | """  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**    **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**()** |

# 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