

# 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：	专业：
上机实践名称：编程实现图搜索算法	学号：20151910042	上机实践日期：2018-10-16
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;  
Cygwin GCC, G++编译器;

### 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**      **SEARCHING**, Breadth-first traversal algorithm.

**Input**            图  $G = (V, E)$ ，并假定图  $G$  是无向图；  
图  $G$  中的某个起点  $v_1$

**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  $\varepsilon - \text{CLOSURE}$

**Begin**

**Step 1**          **for each** vertex  $u \in G.V - \{v_1\}$   
                   $u.\text{color} = \text{White}$   
                   $u.d = \infty$   
                   $u.\pi = \text{NIL}$

**Step 2**           $v_1.\text{color} = \text{Gray}$

$$v_1.d = 0$$

$$v_1.\pi = \text{NIL}$$

**Step 3**       $Q = \phi$

**Step 4**      ENQUEUE( $Q, v_1$ )

**Step 5**      **while**  $Q \neq \phi$   
                   $u = \text{DEQUEUE}(Q)$   
                  **for each**  $v \in G.\text{ADJ}[u]$   
                        $v.\text{color} = \text{Gray}$   
                        $v.d = u.d + 1$   
                        $v.\pi = u$   
                       ENQUEUE( $Q, v$ )  
           $v_1.\text{color} = \text{Black}$

**End**

## 五、 程序代码

```

1  //
2  //  bfs.c
3  //  Algorithms - Graph breadth-first search
4  //
5  //  Created by YourtionGuo on 08/05/2017.
6  //  Copyright © 2017 Yourtion. All rights reserved.
7  //
8
9  #include <stdlib.h>
10
11 #include "bfs.h"
12 #include "graph.h"
13 #include "list.h"
14 #include "queue.h"
15
16 #pragma mark - Public
17
18 int bfs(Graph *graph, BfsVertex *start, List *hops) {
19     Queue      queue;
20     AdjList     *adjlist, *clr_adjlist;
21     BfsVertex   *clr_vertex, *adj_vertex;
22     ListElmt    *element, *member;
23
24     /// 初始化图的所有顶点
25

```

```

26     for (element = list_head(&graph_adjlists(graph)); element != NULL; element = list_next(element)) {
27
28         clr_vertex = ((AdjList *)list_data(element))->vertex;
29
30         if (graph->match(clr_vertex, start)) {
31
32             /// 初始化起点
33             clr_vertex->color = gray;
34             clr_vertex->hops = 0;
35
36         } else {
37
38             /// 初始化其他顶点
39             clr_vertex->color = white;
40             clr_vertex->hops = -1;
41
42         }
43     }
44
45     /// 根据起点邻接表初始化队列
46
47     queue_init(&queue, NULL);
48
49     if (graph_adjlist(graph, start, &clr_adjlist) != 0) {
50
51         queue_destroy(&queue);
52         return -1;
53     }
54
55     if (queue_enqueue(&queue, clr_adjlist) != 0) {
56
57         queue_destroy(&queue);
58         return -1;
59     }
60
61     /// 执行广度优先搜索
62
63     while (queue_size(&queue) > 0) {
64
65         adjlist = queue_peek(&queue);
66
67         /// 在当前邻接表遍历每个顶点
68
69         for (member = list_head(&adjlist->adjacent); member != NULL; member = list_next(member))
70     {
71         adj_vertex = list_data(member);
72

```

```

73         /// 确定下个邻接顶点的颜色
74
75         if (graph_adjlist(graph, adj_vertex, &clr_adjlist) != 0) {
76
77             queue_destroy(&queue);
78             return -1;
79         }
80
81         clr_vertex = clr_adjlist->vertex;
82
83         /// 将白顶点着色为灰色并将邻接表入队
84
85         if (clr_vertex->color == white) {
86
87             clr_vertex->color = gray;
88             clr_vertex->hops = ((BfsVertex *)adjlist->vertex)->hops + 1;
89
90             if (queue_enqueue(&queue, clr_adjlist) != 0) {
91
92                 queue_destroy(&queue);
93                 return -1;
94             }
95         }
96     }
97
98     /// 将当前邻接表出队并将其着色为黑色
99
100    if (queue_dequeue(&queue, (void **)&adjlist) == 0) {
101
102        ((BfsVertex *)adjlist->vertex)->color = black;
103
104    } else {
105
106        queue_destroy(&queue);
107        return -1;
108    }
109 }
110
111 queue_destroy(&queue);
112
113 /// 回传每个顶点的跳数到表中
114
115 list_init(hops, NULL);
116
117 for (element = list_head(&graph_adjlists(graph)); element != NULL; element = list_next(element)) {
118
119     /// 去掉跳数为 -1 的（不可达）
120

```

```
121     clr_vertex = ((AdjList *)list_data(element))->vertex;
122
123     if (clr_vertex->hops != -1) {
124
125         if (list_ins_next(hops, list_tail(hops), clr_vertex) != 0) {
126
127             list_destroy(hops);
128             return -1;
129         }
130     }
131 }
132
133 return 0;
134 }
```

## 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>