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： |  |

# 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；

Cygwin GCC, G++编译器；

# 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**  **Step 4**  **Step 5**  **End** | **SEARCHING,** Breadth-first traversal algorithm.  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  **for** **each** vertex        **while**    **for** **each** |

# 程序代码

|  |  |
| --- | --- |
| 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 | //  // bfs.c  // Algorithms - Graph breadth-first search  //  // Created by YourtionGuo on 08/05/2017.  // Copyright © 2017 Yourtion. All rights reserved.  //  #include <stdlib.h>  #include "bfs.h"  #include "graph.h"  #include "list.h"  #include "queue.h"  #pragma mark - Public  int bfs**(**Graph **\***graph**,** BfsVertex **\***start**,** List **\***hops**)** **{**  Queue queue**;**  AdjList **\***adjlist**,** **\***clr\_adjlist**;**  BfsVertex **\***clr\_vertex**,** **\***adj\_vertex**;**  ListElmt **\***element**,** **\***member**;**  /// 初始化图的所有顶点  **for** **(**element **=** list\_head**(&**graph\_adjlists**(**graph**));** element **!=** **NULL;** element **=** list\_next**(**element**))** **{**  clr\_vertex **=** **((**AdjList **\*)**list\_data**(**element**))->**vertex**;**  **if** **(**graph**->**match**(**clr\_vertex**,** start**))** **{**  /// 初始化起点  clr\_vertex**->**color **=** gray**;**  clr\_vertex**->**hops **=** 0**;**  **}** **else** **{**  /// 初始化其他顶点  clr\_vertex**->**color **=** white**;**  clr\_vertex**->**hops **=** **-**1**;**  **}**  **}**  /// 根据起点邻接表初始化队列  queue\_init**(&**queue**,** **NULL);**  **if** **(**graph\_adjlist**(**graph**,** start**,** **&**clr\_adjlist**)** **!=** 0**)** **{**  queue\_destroy**(&**queue**);**  **return** **-**1**;**  **}**  **if** **(**queue\_enqueue**(&**queue**,** clr\_adjlist**)** **!=** 0**)** **{**  queue\_destroy**(&**queue**);**  **return** **-**1**;**  **}**  /// 执行广度优先搜索  **while** **(**queue\_size**(&**queue**)** **>** 0**)** **{**  adjlist **=** queue\_peek**(&**queue**);**  /// 在当前邻接表遍历每个顶点  **for** **(**member **=** list\_head**(&**adjlist**->**adjacent**);** member **!=** **NULL;** member **=** list\_next**(**member**))** **{**  adj\_vertex **=** list\_data**(**member**);**  /// 确定下个邻接顶点的颜色  **if** **(**graph\_adjlist**(**graph**,** adj\_vertex**,** **&**clr\_adjlist**)** **!=** 0**)** **{**  queue\_destroy**(&**queue**);**  **return** **-**1**;**  **}**  clr\_vertex **=** clr\_adjlist**->**vertex**;**  /// 将白顶点着色为灰色并将邻接表入队  **if** **(**clr\_vertex**->**color **==** white**)** **{**  clr\_vertex**->**color **=** gray**;**  clr\_vertex**->**hops **=** **((**BfsVertex **\*)**adjlist**->**vertex**)->**hops **+** 1**;**  **if** **(**queue\_enqueue**(&**queue**,** clr\_adjlist**)** **!=** 0**)** **{**  queue\_destroy**(&**queue**);**  **return** **-**1**;**  **}**  **}**  **}**  /// 将当前邻接表出队并将其着色为黑色  **if** **(**queue\_dequeue**(&**queue**,** **(**void **\*\*)&**adjlist**)** **==** 0**)** **{**  **((**BfsVertex **\*)**adjlist**->**vertex**)->**color **=** black**;**  **}** **else** **{**  queue\_destroy**(&**queue**);**  **return** **-**1**;**  **}**  **}**  queue\_destroy**(&**queue**);**  /// 回传每个顶点的跳数到表中  list\_init**(**hops**,** **NULL);**  **for** **(**element **=** list\_head**(&**graph\_adjlists**(**graph**));** element **!=** **NULL;** element **=** list\_next**(**element**))** **{**  /// 去掉跳数为 -1 的（不可达）  clr\_vertex **=** **((**AdjList **\*)**list\_data**(**element**))->**vertex**;**    **if** **(**clr\_vertex**->**hops **!=** **-**1**)** **{**    **if** **(**list\_ins\_next**(**hops**,** list\_tail**(**hops**),** clr\_vertex**)** **!=** 0**)** **{**    list\_destroy**(**hops**);**  **return** **-**1**;**  **}**  **}**  **}**    **return** 0**;**  **}** |

# 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