图论的两个实用算法的编程

贺定球 石 磊

摘 要 本文通过C++语言编程，利用VC++开发平台，实现了图论中的两个有用算法：在一有向有环图中，①判断出从一个顶点到另外一个顶点是否存在路径，②找出从一个顶点到另外一个顶点的所有路径。

关键词 VC++，MFC，图论，算法

笔者在开发一个“城市公交车查询系统”的项目时遇到两个问题：①判断出从一个地点是否可以通过坐公交车到达另外一个地点，②找出从一个地点到另外一个地点的所有路径。这两个问题转化为图论的观点就是：在一有向有环图中，①判断出从一个顶点到另外一个顶点是否存在路径，②找出从一个顶点到另外一个顶点的所有路径。本文就这两个算法的实现与读者一起探讨。

一、图的存储方式设计

图的存储方式有多种，比如邻接矩阵（又称数组表示法）、邻接表、十字链表等。这里我们选择用邻接矩阵来实现图的存储。一个有向有环图（如图1所示）可以用一个邻接矩阵（如下表所示）来表示各节点的联系，表中的“1”表示所对应的行顶点到所对应的列顶点存在直接路径，比如第一行与第二列的1表示顶点1到顶点2存在直接的路径。

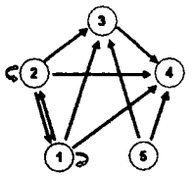

图1 实例图

邻接矩阵

0	1	1	1	0
1	0	1	1	0
0	0	0	1	0
0	0	0	0	0
0	0	1	1	0

这样的邻接矩阵一般用数组来存储，但是用数组来存储有些缺点，就是不能动态定义数组的大小，也不能动态改变它的大小。这里，笔者设计一矩阵类（Matrix）来解决这个问题，为了符合表达习惯，矩阵的下标从1开始。Matrix类的定义如下：

```
class Matrix
{          public:
   void ReSetRowCol(int r , int c);//用来重新改变矩阵的大小
   Matrix();
   int& operator()(int i, int j) const;
   Matrix(int r, int c ); //构造一r行,c列的矩阵
   virtual ~Matrix();
   int rows, cols; // 矩阵的行与列
private:
          int *element;
};
```

Matrix类的实现如下(只给出几个关键函数,具体的代码请看本刊的网络版)：

```
Matrix::Matrix(int r, int c)
{
   rows = r; cols = c;
   element = new int [r * c];
    for ( int i = 0 ; i < r*c ; i++)
       element[i] = 0;
}
int& Matrix::operator ()(int i, int j) const
{
//一个重载操作符,相当数组的[][].
   return element[(i - 1) * cols + j - 1];
}
void Matrix::ReSetRowCol(int r, int c)
{
   if (element != NULL)
       delete []element;//释放空间
   rows = r; cols = c;
   element = new int [r * c];//重新定义矩阵的大小
    for ( int i = 0 ; i < r*c ; i++)
       element[i] = 0;
}
```

利用Matrix可以非常方便地实现邻接矩阵的定义、存储、改变矩阵的大小。

比如要定义一个5行5列的矩阵，只要用 Matrix m(5,5)即可，要使这个矩阵的变成r行c列只要用m.ReSetRowCol(r,c)即可。

二、判断从一个顶点到另外一个顶点是否存在路径的算法

关于有向有环图的路径问题有两个经典算法，其一是求从一个源点到其余各顶点的最短路径的迪杰斯特拉算法，其二是求每一对顶点之间的最短路径的弗洛伊德算法。这两个算法都可以实现判断从一个顶点到另外一个顶点是否存在路径，因为只要从一个顶点到另外一个顶点存在最短路径，那么至少存在一条路径从一个顶点到另外一个顶点。但是由于这两个算法的时间复杂度、空间复杂度都很高，因为它们都得保存路径，根据路径来做进一步判断。为了减少时间复杂度、空间复杂度，笔者找到了另外一种算法。这个算法就是基于 transitive closure（传递闭包）的算法，它是 Warshall 发现的，也叫 Warshall 算法。这个算法的编程思想很简单也很巧妙，它可以求出图中所有顶点间是否可以相互到达的问题，其 C++ 语言描述如下：

```
transclosure(const Matrix&  adjmat,  Matrix&  path)
{
  int i,j,k;
 for(i = 1; i <= adjmat.rows; i++)  //拷贝保存,
以免 adjmat 受到破坏
  for(j = 1; j <= adjmat.rows; j++)
      path(i,j) = adjmat(i,j);
 for(i = 1;i <= adjmat.rows; i++)
  for(j = 1;j <=  adjmat.rows; j++)   //进行计算
处理
   if(path(i,j) == 1)
    for(k = 1; k <=  adjmat.rows;k++) // 这一
步就是所谓的" 传递闭包"
      if(path(j,k) == 1)
        path(i,k) = 1;
  for(i = 1;i <= adjmat.rows; i++) //输出结果
      {
        for(j = 1;j <=  adjmat.rows; j++)
            printf("%d ",path(i,j));
          printf("\n");
 }
}
```

对于图 1 所示的图，用 Warshal 算法实现的的程序如下：

```
Matrix m_adjmat(5,5),m_path(5,5); //定义矩阵
m_adjmat(1,2) =m_adjmat(1,3) =m_adjmat(1,4) = 1;
 m_adjmat(2,1) =m_adjmat(2,3) =m_adjmat(2,4) =1;
m_adjmat(3,4) = 1;//表示顶点 3 与顶点 4 直接相连
m_adjmat(5,3) = m_adjmat(5,4) = 1;
transclosure(m_adjmat,m_path);     //调用 Warshal 算法。
```

其计算结果如图 2 所示：

从结果中很容易看出来从顶点 1 出发可以达到顶点 2，3，4。对角线上的 1 表示该点存在一个环，从该点出发可以再找回该点，比如顶点 1 有路径（1~2~1）的环，同样顶点 2 也有（2~1~2）的环。

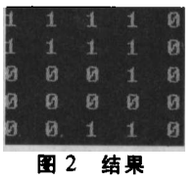

图 2 结果

三、找出从一个顶点到另外一个顶点的所有路径的算法

这个问题也是图论中一个很经典的问题，由于图的有向有环性，实现起来比较复杂与繁琐，这也是迪杰斯特拉算法与弗洛伊德算法在找最短路径时没有先找出所有路径，再在所有路径中比较出一条最短的路径的原因。笔者在一位数学教授的启发下，想到了下面的这种算法。

假如要从如图 1 所示的有向有环图中找出顶点 1 到顶点 4 的所有路径，可以通过展开成如图 3 所示的层次图的来计算，展开原则是：

从要查找的路径的源顶点出发，找出源点能够直接达到的顶点的集合作为第一层，分别找出第一层中各顶点所能够直接达到的顶点集合作为第二层（第二层中不能与其相关的前几层中的数据有重复，比如第一层中的顶点 2 的后续层不能够再出顶点 1，因为将出现（1~2~1~2~1）这样的死循环），以此类推下去，直到找到目的顶点或者再也找不到满足条件的层为止。这是层的展开原则，同时也是算法的流程。

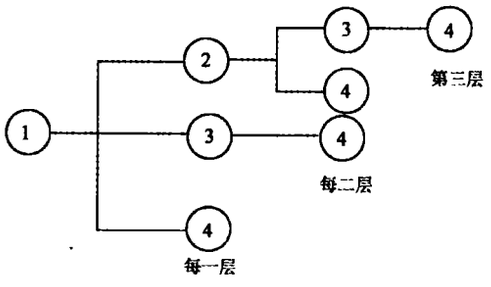

图 3 展开层次图

程序实现如下：

定义 4 个类：CpathData（用来保存路径）、ClayData（用来保存层）、CfindPath（搜索路径的主要类）、Matrix（矩阵类）。由于篇幅的限制，不作过多的细节描述，具体的代码请看本刊的网络版。在这里只给出使用方法。

```
CFindPath m_findpath(5);   // 构造一个类，并且定义了
它含有 5 个顶点
    m_findpath.from = 1;   //  起点为 1
    m_findpath.to = 4;     //  终点是 4
//以下是初使化矩阵
```

（下转第 87 页）

杂，详情可参考相关资料。

（6）通过软件工具检查内存泄露

通过 Optimizeit Profiler, Jprobe Profiler, JinSight, Rational 公司的 Purify 等工具，可以找出那些无用但被引用的对象。

五、结束语

本文对 Java 中 GC 工作的原理和产生内存泄露的原因进行了研究分析，并根据在 Java 应用程序开发中的实践经验，摸索出了几条避免或改进内存泄露问题的方法，希望为开发更优的 Java 应用提供一些参考。

参考文献

1. Bruce Eckel . Thinking in Java[M]. 北京机械工业出版社，2002.

2. Bloch J. Java 高级编程指南[M]. 北京机械工业出版社，2002.

3. Jim Patrick. Handling memory leaks in Java programs [EB/OL].

http://www-106.ibm.com/developerworks/library/j-leaks/index.html, 2003.

4. SUN Microsystems, Inc. Tuning Garbage collection with 1.3.1 Java TM Virtual

Machine[EB/OL]. http://www.java.sun.com/docs/hotspot/gc/index.html, 2002.

（收稿日期：2005 年 2 月 10 日）

（上接第 55 页）

从算法实现上也可以看出，SML 映射规则是一种有偏的映射规则，某些范围的灰度级会被有偏地映射到接近开始计算的灰度级；而 GML 映射规则是统计无偏的，从根本上就避免了上述问题的出现。通过分析可以看出 GML 映射规则总会比 SML 映射规则更能体现规定直方图的意图，而且通常产生的误差只有 SML 映射规则的十几分之一。

六、结论

本文从理论上讲述了直方图变换处理中常用的直方图均衡化、采取单映射和组映射规则的直方图规定化变换方法，通过程序算法实现了上述图像增强过程并给出了通过三种算法实验得出的处理图像。实验表明本文介绍的方法对于暗、弱信号的原始图像的目标识别和图像增强等有着良好的处理效果，尤其是通过组映射规则的直方图规定化变换方法结合设计良好的规定直方图可以得到更佳的图像处理效果。本文给出的程序代码在 Windows 2000 Professional 下由 Microsoft Visual C++ 6.0 编译通过。

参考文献

1. 章毓晋．图像处理和分析．清华大学出版社，2001.

2. Kenneth R. Catleman, Digital Image Processing, Prentice Hall. 1996.

3. Gonzalez R C, Wintz P. Digital Image Processing. Addion-Wesley. 1987.

（收稿日期：2005 年 3 月 6 日）

（上接第 73 页）

```
m_findpath.m_adj(1, 2) = m_findpath.m_adj(1, 3) = m_findpath.m_adj(1,4) =1;
m_findpath.m_adj(2, 1) = m_findpath.m_adj(2, 3) = m_findpath.m_adj(2,4) =1;
m_findpath.m_adj(3,4) = 1 ;
m_findpath.m_adj(5, 4) = m_findpath.m_adj(5, 3) = 1;
m_findpath.FindPath();  //调用函数，查找路径，并输出结果.
```

```
从顶点1到顶点4的所有路径如下：
1       4
1       2       4
1       3       4
1       2       3       4
```

图 4 路径结果

其计算结果如图 4 所示：

这表明从顶点 1 到顶点 4 有 4 条路径。分别是（1～4）、（1～2～4）、（1～3～4）、（1～2～3～4）。

四、结语

本文实现的这两个算法非常具有实用价值，它们不但速度快、占用存储空间少，而且使用方便。本文的第二个问题也许有更好的算法，希望大家一起探讨。

参考文献

Sartaj Sahni. 数据结构算法与应用．机械工业出版社，1999

（收稿日期：2005 年 1 月 20 日）