数据结构中图的遍历算法

余云，邬奇梅

（安徽新华学院 信息工程学院，安徽 合肥 230088）

摘要：本文根据图的存储结构、搜索路径及编制算法的方法不同，详尽地给出八种不同的图的遍历算法思想。即：从图的邻接矩阵和邻接表两种不同存储结构，再考虑到递归和非递归两种遍历思想的不同，分别把深度优先搜索遍历和广度优先搜索遍历分为四种不同的遍历方法。其目的是：通过本文的讨论，使初学者及高职学生能充分掌握数据结构中图的不同遍历算法并能正确的给出图的各种遍历程序。

关键词：图；存储结构；算法

中图分类号：TP301　　**文献标识码**：A　　**文章编号**：1009-3044(2008)17-21516-03

Traversing Gragh Algorithm in Data Structure

YU Yun, WU Qi-mei

(Information Engineering Institute,Anhui Xinhua University,Hefei 230088,China)

Abstract:This article according to the difference of method in storage structure of graph、search path and design algorithm is to introduce eight different kinds of traversing graph algorithm in detail. That is to say: from two different storage structure of adjacency matrix and adjacency list and two kinds of difference traversing graph algorithm include recursion and not recursion, depth precedence search and breadth precedence search are divided into four kinds of different traversing graph algorithm. This paper is to make beginners and college students master different traversing grasp algorithm well and write out all kinds of graph traversing programs exactly.

Key words:Graph;storage structure;algorithm

遍历算法在数据结构中是最普通的运算方法也是所有其它算法的基础。由于图的遍历比线性表、树的结构的遍历算法要复杂，因此着重对图的遍历算法进行讨论，具有更普遍的意义。图的遍历就是从图中指定的某顶点作为遍历的起始出发点，按照一定搜索遍历路径，对图中所有顶点仅作一次访问的过程。

因为图的复杂性，在图中的任何顶点都可能和其余顶点相邻接，故在访问了某个顶点之后，可能沿着某条路径又回到了该顶点。为了避免重复访问图中的同一个顶点，在搜索访问过程中必须记住每个顶点是否被访问过，为此可设置一个向量 visited[vexnum],它的初始值为{0}，一旦访问了第 i 顶点，便将 visited[i]置为 1。

根据搜索路径方向的不同，遍历图的方法可分深度优先搜索遍历和广度优先搜索遍历，又根据编制算法的方法不同，可分为递归遍历算法和非递归遍历算法，再考虑到图的邻接矩阵和邻接表两种不同存储结构，综合起来图的搜索遍历共可分为八种。

下面以图 1 为例，分别介绍图的各种优先遍历方法。

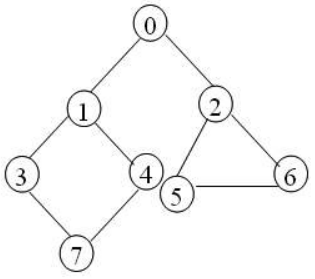

图 1　图的优先遍历

Matrix=

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

图 2　G 的邻接矩阵 Matrix 存储结构

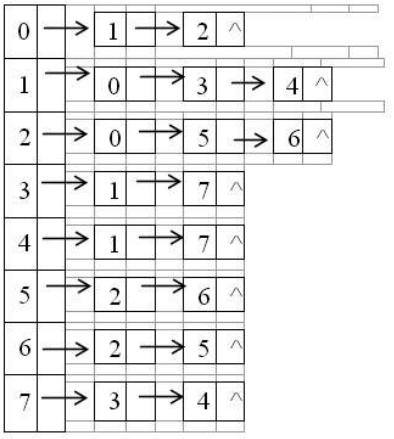

图 3　G 的邻接表存储结构

收稿日期：2008-03-27

作者简介：余云（1979-），女，安徽六安人，助教，学士，研究方向：软件工程；邬奇梅（1936-），男，江西萍乡人，教授，研究方向：软件工程。

1 深度优先搜索遍历

假定给定图 G 的初态是所有顶点均未访问过，在 G 中任选一顶点 i 作为遍历的起始点，则深度优先搜索遍历基本思想是：

首先访问起始顶点 i,并将其访问标记置为已访问过，即 visited[i]=1；然后依次从顶点 i 的未被访问的邻接点中选顶点 j，若 j 未被访问过，则访问它，并将顶点 j 的访问标记置为已访问过，即 visited[j]=1，再从 j 开始重复此过程。若 j 已访问，再看与顶点 i 有边相连的其他顶点，若与 i 有边相连的顶点都被访问过，则退回到前一个访问顶点并重复前过程，直到图中所有顶点都被访问完为止。

以图 1 中无向图 G 为例可以得到不同深度优先搜索遍历顶点序列。

例如：0→1→3→7→4→2→5→6

0→1→4→7→3→2→6→5

2 广度优先搜索遍历

假定给定图 G 的初态是所有顶点未被访问，在 G 中任选一顶点 i 作为遍历的起始点，则广度优先搜索遍历的基本思想是：

首先访问起始点 i,并将其访问标记置为已访问过，既 visited[i]=1,接着依次访问顶点 i 的未被访问过的邻接点 w1、w2、……、wt，然后再依次访问与 w1、w2、……、wt 相邻接的未被访问过的顶点，依次类推，直到图中所有顶点都被访问完为止。

同样以图 1 中无向图 G 为例可得出不同的广度优先搜索遍历序列：

例如：0→1→2→3→4→5→6→7

0→2→1→6→5→4→3→7

3 图的存储结构数据类型的定义

为了讨论方便，下面用 C 语言来描述

1)图的邻接矩阵数据类型描述：

```
#define  vexnum  maxver
#define  edgenum  maxedge
typedef  struct { int vertex[vexnum]; int matvix[vexnum] [vexnum];}adjmatrix;
```

2)图的邻接表数据类型描述：

```
typedef  struct  acrnode{int  adjvex; struct  acrnode  *nextedge;  }edgenode;
typedef  struct  vnode{  int  data; edgenode  *firstedge; }vexnode;
typedef  struct{ vexnode vx[vexnum]; }adjlist;
```

4 深度优先搜索遍历算法

4.1 用邻接矩阵实现图的深度优先搜索

1)深度优先搜索遍历递归算法[2]

```
void  dfsm( int i)
{int j; G=&g; printf(" %3d ",G- >vextex[i]); vixited[i]=1;
for(j=0;j<vexnum;j++) if((G- >matrix[i][j]= =1&&(!visited[j])) dfsm(j);}
void  mdfs( )
{int  k;for(k=0;k<vexnum;k++)if(visited[k]= =0) dfsm(k);}
```

用上述算法和图(b)的存储结构，可以描述从顶点 0 出发的深度优先搜索遍历顶点序列为：

0→1→3→7→4→2→5→6

2)深度优先搜索遍历非递归算法[1]

```
void  dfsm1( )
{int visited[vexnum];  int stack[vexnum];  int top=0, k, j;
for(k=0;k<vexnum;k++) visited[k]=0;
for(k=0;k<vexnum;k++) { if(visited[k]= =0) { top++; stack[top]=k;
while(top!=0)
{ i=stack[top];  top- - ; if(visited[i]!=1) {visited[i]=1;  printf(" %3d ",i); }
for(j=0;j<vexnum;j++)
if(g.matrix[i][j]= =0&&visited[j]= =0){ top++;stack[top]=j;} } }}
```

用此算法得出的深度优先搜索遍历序列为：0→2→6→5→1→4→7→3

4.2 用邻接表实现图的深度优先搜索

1)深度优先搜索遍历递归算法[2][3]

```
void dfsL(int i){ edgenode *p; printf(" %3d ",g.vx[i].data); visited[i]=1; p=g.vx[i].firstedge;       while(p!=NULL){ if(visited[p- >adjvex]= =0)
dfsL(p- >adjvex); p=p- >nextedge; } }
void  Ldfs( ){ int  k; for(k=0;k<vexnum;k++) if(visited[x]= =0) dfsL(k); }
```

用上述算法和图(c)的存储结构，可以描述从顶点 0 出发搜索遍历的顶点序列为：

0→1→3→7→4→2→5→6

2)深度优先搜索遍历非递归算法[1]

```
void  dfsL1( )
{ edgenode  *p; int visited[vexnum]={0}; int stack[vexnum]; int top=0, i,j;
for(i=0;i<vexnum;i++) { if(visited[i]= =0){ top++; stack[top]=i;
while(top!=0) {j=stack[top]; top- - ; if(visited[j]!=1) {visited[j]=1;printf(" %3d ",j);}
for(p=g.vx[j].firstedge;p!=NULL;p=p- >nextedge)
if(visited[p- >adjvex]= =0){ top++;stack[top]=p- >adjvex; } } }}}
```

5 广度优先搜索遍历算法

5.1 用邻接矩阵实现图的广度优先搜索

1)广度优先搜索遍历递归算法[2][3]

```
void bfsm( )
{ int i;int q[vexnum+1]; int front=0; int rear=0;
for(i=0;i<vexnum;i++) { if(visited[i]= =0){rear++;q[rear]=i; } mbfs( );}}
void mbfs( )
{int j,i; while(rear!=front) {front++;i=q[front]; if(visited[i]= =0) { visited[i]=1; printf(" %3d ",i);}
for(j=0;j<vexnum;j++) if(G- >matrix[i][j]= =1&&(!visited[j])) { rear++;q[rear]=j; } } }
```

2)**广度优先搜索遍历非递归算法和深度优先搜索遍历非递归算法相似，不同之处在于前者用栈存储数据，后者用队列储数据。**

5.2 **用邻接表实现图的广度优先搜索**

1)**广度优先搜索遍历递归算法**

```
void bfsL( )
{ int i; for(i=0;i<vexnum;i++)visited[i]=0; for(i=0;i<vexnum;i++)
{ if(visited[i]= =0) {rear++; q[rear]=i;}Lbfs( ); } }
void Lbfs( )
{ int j;edgenode *p;while(front!=rear) { front++; j=q[front];
if(visited[j]= =0) {visited[j]=1; printf(" %3d ",j);}
for(p=g.vx[j].firstedge;p!=NULL;p=p- >nextedge)
{ if(visited[p- >adjvex]= =0) { rear++; q[rear]=p- >adjvex; } } Lbfs( ); } }
```

2)**广度优先搜索遍历非递归算法**[1][3]

```
void bfsL1( )
{ edgenode *p; int q[vexnum];int front=- 1; int rear=- 1; int visited[vexnum]={0};
for(i=0;i<vexnum;i++){ if(visited[i]= =0) { rear++; q[rear]=i;
while(front!=rear) {front++;j=q[front]; if(visited[j]!=1) {visited[j]=1;printf(" %3d ",j);}
for(p=g.vx[j].firstedge;p!=NULL;p=p- >nextedge)
if(visited[p- >adjvex]= =0) { rear++;[rear]=p- >adjvex;} } } }}
```

参考文献：

[1] 杨枨.数据结构[M].高等教育出版社,2002.
[2] 严蔚敏,吴伟民.数据结构[M].清华大学出版社,1997.
[3] 李根强.数据结构[M].中国水科水电出版社,2001.

(上接第 1506 页)

选切片垂直于 Y 轴的索引结构；如果视点位于 3 区和 4 区，则选切片垂直于 X 轴的索引结构；视点位于 5，6，7，8 区，则可任选一组索引结构。对于三维的情况可以类推。选择不同的索引结构的目的，是为了成像时便于由近及远地处理面片，以便更好地利用前面面片对后面面片的遮挡性，避免对不可见面片的处理。

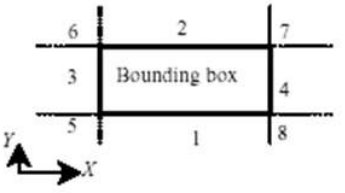

图 4 根据视点相对于场景包围盒的方位选择合适的索引结构

在选择了索引结构后，我们就可以根据成像面上的可见像素来选择可见面。不失一般性，设所选择的索引结构的切片是垂直于 Z 轴的，而视点的 Z 坐标是小于这些切片的 Z 坐标的。由此，对这些切片进行由近及远的逐个处理，并在处理每一个切片时，根据成像面的可见像素在此切片的实网格中挑选出可见面片。当可见像素被真正离它最近的面片填充后，它就变成了填充像素，不再参与选择可见面的操作。

5 动态物体的处理

新方法能够很好地处理带有刚性运动物体的场景。首先，预处理时我们为每一个动态物体按其局部坐标系建立层次索引结构；在绘制时，可以先对动态场景和背景的层次索引结构进行合并；然后按照相同的方法绘制所有的面片。但是，合并索引结构的操作需要很大的开销。

为了方便地处理动态物体，并与静态场景进行协调的绘制，我们采用以下的方式：确定视点和成像面以后，首先分别绘制所有的动态物体；在绘制这些物体时，所用的坐标系是每个物体的层次索引结构所在的局部坐标系，不同的是，需要将成像面上像素的深度坐标从物体所在的局部坐标系转换到场景对应的全局坐标系下，这样就可以根据转换后的深度坐标进行比较来选择动态物体中的可见面进行绘制；绘制完动态物体后，再对场景进行绘制。这时，一些成像面上像素的深度坐标已经改变，而且选择了一些临时的可见面。但是，我们把所有这些像素仍然当作可见像素，以进行对静态场景中可见面的选择。这样，由于像素上深度坐标的存在，动态物体上的可见面能得到保留，而不可见面会被自动地覆盖掉。

6 总结

本文介绍了常见的遮挡算法，针对各种遮挡算法的不足提出的可见面选取方法对面片进行基于法向的分类，并对每类面片进行基于空间位置的有序的层次化索引构造。这样，各条视线可以方便地将法向背离视点的不可见面片进行成群的剔除，视线在寻找可见面时可以利用高效的检索技术加快前行的步长，以提高检测速度，因为索引构造能够很好地、有序地反映面片的空间位置。此外，成像时，基于可见像素来选择可见面，能够很好地利用已绘制的可见面片隐含地剔除大量被遮挡的面片，节省了大量时间。由于新方法的预处理结果是可以相对于物体的局部空间来计算的，物体运动对预处理结果没什么改变，因此新方法能够方便地处理动态场景。总之，新方法对可见面的选取具有很高的精度，并且速度很快，能方便地处理动态场景。

参考文献：

[1] 魏峰,王文成,吴恩华.快速高精度的可见面选择[J].软件学报,2006(10).
[2] 王益.基于视域剪裁的三维场景快速消隐算法[J].系统仿真学报,2001(11).
[3] 马自萍.基于线性八叉树的一种消隐算法[J].宁夏工程技术,2006(9).
[4] 任重,华炜,鲍虎军,等.全局遮挡图[J].计算机学报,2005(6).
[5] 志庚,傅国良,王毅刚.一种基于视角剖分的可见性算法[J].系统仿真学报,2004(8).