```cpp
#include<stdio.h>
#include<stdlib.h>
#include "S_Q.h"
#include<iostream>
using namespace std;

//**********************自定义符号常量**************************

#define OVERFLOW -2          //内存溢出错误常量
#define OK 1                 //表示操作正确的常量
#define ERROR 0              //表示操作错误的常量
#define TRUE 1               //表示逻辑真的常量
#define FALSE 0              //表示逻辑假的常量

//**********************自定义数据类型**************************

typedef int Status;         //指定状态码的类型是int

//-------------图的数组（邻接矩阵）存储表示

//一个输入的权值很难达到的最大值 ∞
#define INFINITY 65535

//最大顶点个数
#define MAX_VERTEX_NUM 20

//使用枚举类型定义图的四种基本类型
typedef enum {

    //（有向图=0，有向网=1，无向图=2，无向网=3）
    DG, DN, UDG, UDN
} GraphKind;

//指定顶点关系类型为int
typedef int VRType;

//指定顶点类型为int
typedef int VertexType;

typedef struct ArcCell{

    //VRType是顶点关系类型。
    //对无权图，用1或0，表示相邻与否，对带权图，则为权值类型
    VRType adj;

    //我修改了书上的定义删除了info指针。
    //原因是：info指针完全没必要设置，因为无论是图还是网，
    //        adj这个数据域都已经足够存储所有有用信息了。
    //        作者设置info指针的本意是指向存储网权值的内存空间，
    //        但是由于adj本身就能存储网中的权值，也不会引起歧义，
    //        而且设置了info之后还会涉及很多内存的分配、释放和指针操作，
    //        为了使一个函数适用于四种图，需要根据图的不同类型对这个指针
    //        执行不同的操作。去掉info可以简化操作。

}ArcCell, AdjMatrix[MAX_VERTEX_NUM][MAX_VERTEX_NUM];

typedef struct{

    //顶点向量:存储了图中所有的顶点
    VertexType vexs[MAX_VERTEX_NUM];

    //邻接矩阵：存储了顶点之间的邻接关系以及权值
    AdjMatrix arcs;

    //图当前的顶点数和弧数
    int vexnum, arcnum;

    //图的种类标志
    GraphKind kind;
}MGraph;


//-------------------------------------------------------------------------------------------------
/*
    函数：LocateVex
    参数：MGraph G 图G（邻接矩阵存储结构）
    返回值：若G中存在顶点v，则返回该顶点在图中位置；否则返回-1
    作用：顶点定位函数，在图G中查找顶点v的位置
*/
int LocateVex(MGraph G, int v){

    //遍历邻接矩阵查找顶点v
    for(int i = 0; i < G.vexnum; ++i) {
```

```
84          //若找到结点返回i
85          if(G.vexs[i] == v) {
86              return i;
87          }//if
88      }//for
89
90      //否则返回-1，表示没有找到
91      return -1;
92  }//LocateVex
93
94  /*
95      函数：CreateUDN
96      参数：MGraph &G 图的引用
97      返回值：状态码，操作成功返回OK
98      作用：采用数组（邻接矩阵）表示法，构造无向网G
99  */
100 Status CreateUDN(MGraph &G){
101     cout <<"Please input: vexnum(no more than 20) arcnum(no more than vexnum*vexnum)" <<
    endl;
102     cin >> G.vexnum >> G.arcnum;
103     cout << "构造顶点向量" << endl;
104     for(int i=0; i<G.vexnum; i++)      //构造顶点向量
105         cin >> G.vexs[i];
106
107     for(int i=0; i<G.vexnum; i++)    //初始化邻接矩阵
108         for(int j=0; j<G.vexnum; j++)
109             G.arcs[i][j].adj = INFINITY;
110
111     VertexType v1,v2; //顶点
112     int i,j,value;
113     cout << "以‘起点 终点 权值’的方式依次输入每一条边(例如：1 2
    16\t起点：1，终点：2，权值：16)： " << endl;
114     int k=0;
115     for(k=0; k<G.arcnum; ++k){  //构造邻接矩阵
116         cin >>v1>>v2>>value;
117         //定位v1,v2在G中的位置
118         i=LocateVex(G,v1);
119         j=LocateVex(G,v2);
120         G.arcs[i][j].adj = value;    //弧<v1,v2>的权值
121         G.arcs[j][i] = G.arcs[i][j];
122     }
123     return OK;
124 }// CreateUDN
125
126 /*
127     函数：CreateUDG
128     参数：MGraph &G 图的引用
129     返回值：状态码，操作成功返回OK
130     作用：采用数组（邻接矩阵）表示法，构造无向图G
131 */
132 Status CreateUDG(MGraph &G){
133     cout <<"Please input: vexnum(no more than 20) arcnum(no more than vexnum*vexnum)" <<
    endl;
134     cin >> G.vexnum >> G.arcnum;
135     cout << "构造顶点向量" << endl;
136     for(int i=0; i<G.vexnum; i++)      //构造顶点向量
137         cin >> G.vexs[i];
138
139     for(int i=0; i<G.vexnum; i++)    //初始化邻接矩阵
140         for(int j=0; j<G.vexnum; j++)
141             G.arcs[i][j].adj = 0;
142
143     VertexType v1,v2; //顶点
144     int i,j;
145     cout << "以‘起点 终点’的方式依次输入每一条边(例如：1 2\t起点：1，终点：2)： " << endl;
146     int k=0;
147     for(k=0; k<G.arcnum; ++k){  //构造邻接矩阵
148         cin >>v1>>v2;
149         //定位v1,v2在G中的位置
150         i=LocateVex(G,v1);
151         j=LocateVex(G,v2);
152         G.arcs[i][j].adj = 1;    //弧<v1,v2>的权值
153         G.arcs[j][i] = G.arcs[i][j];
154     }
155     return OK;
156 }// CreateUDG
157
158 /*
159     函数：CreateDN
160     参数：MGraph &G 图的引用
161     返回值：状态码，操作成功返回OK
162     作用：采用数组（邻接矩阵）表示法，构造有向网G
163 */
164 Status CreateDN(MGraph &G){
```

```
165   cout <<"Please input: vexnum(no more than 20) arcnum(no more than vexnum*vexnum)" << endl;
166       cin >> G.vexnum >> G.arcnum;
167       cout << "构造顶点向量" << endl;
168       for(int i=0; i<G.vexnum; i++)      //构造顶点向量
169           cin >> G.vexs[i];
170
171       for(int i=0; i<G.vexnum; i++)    //初始化邻接矩阵
172           for(int j=0; j<G.vexnum; j++)
173               G.arcs[i][j].adj = INFINITY;
174
175       VertexType v1,v2; //顶点
176       int i,j,value;
177       cout << "以'起点 终点 权值'的方式依次输入每一条边(例如: 1 2
      16\t起点: 1, 终点: 2, 权值: 16): " << endl;
178       int k=0;
179       for(k=0; k<G.arcnum; ++k){  //构造邻接矩阵
180           cin >>v1>>v2>>value;
181           //定位v1,v2在G中的位置
182           i=LocateVex(G,v1);
183           j=LocateVex(G,v2);
184           G.arcs[i][j].adj = value;    //弧<v1,v2>的权值
185       }
186       return OK;
187   }// CreateDN
188
189   /*
190       函数: CreateDG
191       参数: MGraph &G 图的引用
192       返回值: 状态码, 操作成功返回OK
193       作用: 采用数组（邻接矩阵）表示法, 构造有向图G
194   */
195   Status CreateDG(MGraph &G){
196       cout <<"Please input: vexnum(no more than 20) arcnum(no more than vexnum*vexnum)" <<
      endl;
197       cin >> G.vexnum >> G.arcnum;
198       cout << "构造顶点向量" << endl;
199       for(int i=0; i<G.vexnum; i++)      //构造顶点向量
200           cin >> G.vexs[i];
201
202       for(int i=0; i<G.vexnum; i++)    //初始化邻接矩阵
203           for(int j=0; j<G.vexnum; j++)
204               G.arcs[i][j].adj = 0;
205
206       VertexType v1,v2; //顶点
207       int i,j;
208       cout << "以'起点 终点'的方式依次输入每一条边(例如: 1 2\t起点: 1, 终点: 2): " << endl;
209       int k=0;
210       for(k=0; k<G.arcnum; ++k){  //构造邻接矩阵
211           cin >>v1>>v2;
212           //定位v1,v2在G中的位置
213           i=LocateVex(G,v1);
214           j=LocateVex(G,v2);
215           G.arcs[i][j].adj = 1;    //弧<v1,v2>的权值
216       }
217       return OK;
218   }// CreateDG
219
220   /*
221       函数: CreateGraph
222       参数: MGraph &G 图的引用
223       返回值: 状态码, 操作成功返回OK
224       作用: 采用数组（邻接矩阵）表示法, 构造图G
225   */
226   Status CreateGraph(MGraph &G){
227
228       //输入构造图的类型
229
      printf("请输入您想构造的图的类型:有向图输入0,有向网输入1,无向图输入2,无向网输入3):");
230       scanf("%d", &G.kind);
231
232       //根据图的不同类型调用图的构造函数
233       switch(G.kind){
234           case  DG:  return CreateDG(G);  //构造有向图G
235           case  DN:  return CreateDN(G);  //构造有向网G
236           case UDG:  return CreateUDG(G); //构造无向图G
237           case UDN:  return CreateUDN(G); //构造无向网G
238           default :  return ERROR;
239       }//switch
240   }//CreateGraph
241
242   /*
243       函数: DestroyGraph
244       参数: MGraph &G 图的引用
245       返回值: 状态码, 操作成功返回OK
```

```
246         作用：销毁图G
247     */
248     Status DestroyGraph(MGraph &G) {
249
250         //若是网
251         if(G.kind % 2) {
252
253             //重置邻接矩阵所有顶点之间不可达
254             for(int i = 0; i < G.vexnum; i++) {
255                 for(int j = 0; j < G.vexnum; j++) {
256                     G.arcs[i][j].adj = INFINITY;
257                 }//for
258             }//for
259         }//if
260         else { //若是图
261
262             //重置邻接矩阵所有顶点之间不可达
263             for(int i = 0; i < G.vexnum; i++) {
264                 for(int j = 0; j < G.vexnum; j++) {
265                     G.arcs[i][j].adj = 0;
266                 }//for
267             }//for
268         }//else
269
270         //重置顶点数为0
271         G.vexnum = 0;
272
273         //重置边数为0
274         G.arcnum = 0;
275     }//DestroyGraph
276
277     /*
278         函数：PrintAdjMatrix
279         参数：MGraph G 图G
280         返回值：状态码，操作成功返回OK
281         作用：打印某个图的邻接矩阵
282     */
283     Status PrintAdjMatrix(MGraph G){
284
285         //输出左上角多余的空白
286         printf("      ");
287
288         //输出邻接矩阵的上坐标（全部顶点）
289         for(int i = 0; i < G.vexnum; i++) {
290
291             printf(" %3d ", G.vexs[i]);
292         }//for
293
294         printf("\n");
295
296         //输出左上角多余的空白
297         printf("     +");
298
299         //输出一条横线
300         for(int i = 0; i < G.vexnum; i++) {
301             printf("-----");
302         }//for
303
304         printf("\n");
305
306         //输出邻接矩阵的左坐标（全部顶点）和内容
307         for(int i = 0; i < G.vexnum; i++) {
308
309             //输出邻接矩阵左边的坐标
310             printf(" %3d |", G.vexs[i]);
311
312             for(int j = 0; j < G.vexnum; j++) {
313                 if(G.arcs[i][j].adj == INFINITY) {
314                     printf("  ∞ ");
315                 }//if
316                 else {
317                     printf(" %3d ", G.arcs[i][j].adj);
318                 }//else
319             }//for
320             printf("\n     |\n");
321         }//for
322     }//PrintAdjMatrix
323
324     /*
325         函数：GetVex
326         参数：MGraph G 图G
327               int v v是G中某个顶点的序号
328         返回值：返回v的值
329         作用：得到序号为v的顶点值
```

```
*/
VertexType GetVex(MGraph G, int v) {

    //检查参数v是否合法：v是否越界
    if(v >= G.vexnum || v < 0) {
        exit(ERROR);
    }//if

    //返回序号为v的顶点值
    return G.vexs[v];
}//GetVex

/*
    函数：PutVex
    参数：MGraph &G 图的引用
         VertexType v v是G中某个顶点
         VertexType value 将序号为v的顶点的值修改为value
    返回值：状态码，操作成功返回OK，否则返回ERROR
    作用：修改序号为v的顶点值
*/
Status PutVex(MGraph &G, VertexType v, VertexType value) {

    //k为顶点v在图G中的序号
    int k = LocateVex(G, v);

    //检查图中是否存在顶点v
    if(k < 0) {
        return ERROR;
    }//if

    //将顶点v的值置为value
    G.vexs[k] = value;

    //操作成功
    return OK;
}//PutVex


/*
    函数：FirstAdjVex
    参数：MGraph G 图G
         VertexType v 顶点v
    返回值：若找到邻接点，返回邻接点的顶点位置，否则返回-1
    作用：求顶点v在图G中的第一个邻接点
*/
int FirstAdjVex(MGraph G, VertexType v){

    //i表示不可达，在图中，0表示不可达，在网中INFINITY表示不可达
    int j = 0;

    //v是顶点，不是序号！需要定位
    //k就是顶点v在顶点数组中的序号
    int k = LocateVex(G, v);

    //若是网，则INFINITY表示不可达
    if(G.kind == DN || G.kind == UDN) {
       j = INFINITY;
    }//if

    //在顶点v对应的第k行查找第一个邻接点的序号i
    for(int i = 0; i < G.vexnum; ++i) {

        //若找到顶点v的第一个邻接点则返回i
        //G.arcs[k][i].adj != j 表示顶点G.arcs[k][i]可达
        if(G.arcs[k][i].adj != j) {
            return i;
        }//if
    }//for

    //若未找到返回-1
    return -1;
}//FirstAdjVex

/*
    函数：NextAdjVex
    参数：MGraph G 图G
         VertexType v 顶点v
    返回值：若找到邻接点，返回邻接点的顶点位置，否则返回-1
    作用：求顶点v在图G中相对于邻接点w的下一个邻接点
*/
int NextAdjVex(MGraph G, VertexType v, VertexType w){

    //i表示不可达，在图中，0表示不可达，在网中INFINITY表示不可达
    int j = 0;
```

```
414
415         //k1是顶点v在顶点数组中的位序
416         int k1 = LocateVex(G, v);
417
418         //k2是顶点w在顶点数组中的位序
419         int k2 = LocateVex(G, w);
420
421         //若是网，则使用INFINITY表示不可达
422         if(G.kind == DN || G.kind == UDN) {
423             j = INFINITY;
424         }//if
425
426         //在图G中查找顶点v相对于顶点w的下一个邻接点的位置
427         for(int i= k2 + 1; i < G.vexnum; ++i) {
428
429             //若找到则返回i
430             if(G.arcs[k1][i].adj != j) {
431                 return i;
432             }//if
433         }//for
434
435         //若未找到返回-1
436         return -1;
437     }//NextAdjVex
438
439     /*
440         函数：InsertVex
441         参数：MGraph &G 图的引用
442               VertexType v 顶点v
443         返回值：状态码，操作成功返回OK
444         作用：在图G中增添新顶点v
445     */
446     Status InsertVex(MGraph &G, VertexType v) {
447
448         //i表示不可达，在图中，0表示不可达，在网中INFINITY表示不可达
449         int j = 0;
450
451         //若是网，则使用INFINITY表示不可达
452         if(G.kind % 2) {
453             j = INFINITY;
454         }//if
455
456         //构造新顶点向量
457         G.vexs[G.vexnum] = v;
458
459         //初始化邻接矩阵
460         for(int i = 0; i <= G.vexnum; i++) {
461
462             //初始化矩阵的每个存储单元为不可达
463             G.arcs[G.vexnum][i].adj = G.arcs[i][G.vexnum].adj = j;
464         }//for
465
466         //图G的顶点数加1
467         G.vexnum++;
468
469         //操作成功
470         return OK;
471     }//InsertVex
472
473     /*
474         函数：DeleteVex
475         参数：MGraph &G 图的引用
476               VertexType v 顶点v
477         返回值：状态码，操作成功返回OK
478         作用：删除G中顶点v及其相关的弧
479     */
480     Status DeleteVex(MGraph &G, VertexType v) {
481
482         //m表示不可达，在图中，0表示不可达，在网中INFINITY表示不可达
483         VRType m = 0;
484
485         //若是网，则使用INFINITY表示不可达
486         if(G.kind % 2) {
487             m = INFINITY;
488         }//if
489
490         //k为待删除顶点v的序号
491         int k = LocateVex(G, v);
492
493         //检查v是否是图G的顶点
494         if(k < 0) { //v不是图G的顶点
495
496             //操作失败
497             return ERROR;
```

```
    }//if

    //删除边的信息
    for(int j = 0; j < G.vexnum; j++) {

        //有入弧
        if(G.arcs[j][k].adj != m) {

            //删除弧
            G.arcs[j][k].adj = m;

            //弧数-1
            G.arcnum--;
        }//if

        //有出弧
        if(G.arcs[k][j].adj != m) {

            //删除弧
            G.arcs[k][j].adj = m;

            //弧数-1
            G.arcnum--;
        }//if
    }//for

    //序号k后面的顶点向量依次前移
    for(int j = k + 1; j < G.vexnum; j++) {
        G.vexs[j-1] = G.vexs[j];
    }//for

    //移动待删除顶点之右的矩阵元素
    for(int i = 0; i < G.vexnum; i++) {
        for(int j = k + 1; j < G.vexnum; j++) {
            G.arcs[i][j - 1] = G.arcs[i][j];
        }//for
    }//for

    //移动待删除顶点之下的矩阵元素
    for(int i = 0; i < G.vexnum; i++) {
        for(int j = k + 1; j < G.vexnum; j++) {
            G.arcs[j - 1][i] = G.arcs[j][i];
        }//for
    }//for

    //图的顶点数-1
    G.vexnum--;

    //操作成功
    return OK;
}//DeleteVex

/*
    函数: InsertArc
    参数: MGraph &G 图的引用
          VertexType v 顶点v（弧尾）
          VertexType w 顶点w（弧头）
    返回值: 状态码，操作成功返回OK
    作用: 在G中增添弧<v,w>，若G是无向的，则还增添对称弧<w,v>
*/
Status InsertArc(MGraph &G, VertexType v, VertexType w) {

    //弧尾顶点v在图中的序号v1
    int v1 = LocateVex(G, v);

    //弧头顶点w在图中的序号w1
    int w1 = LocateVex(G, w);

    //检查顶点v和顶点w在图中是否存在
    if(v1 < 0 || w1 < 0) {  //v或w有一个不是图G中的顶点

        //操作失败
        return ERROR;
    }//if

    //弧或边数加1
    G.arcnum++;

    //如果是图G是网，还需要输入权值
    //if(G.kind % 2) <=> if(G.kind % 2 != 0)
    if(G.kind % 2) {
        printf("请输入此弧或边的权值: ");
        scanf("%d", &G.arcs[v1][w1].adj);
    }//if
```

```
582         else { // 图
583             G.arcs[v1][w1].adj = 1;
584         }//else
585 
586         //如果是无向图或无向网还需要置对称的边
587         if(G.kind > 1) {
588             G.arcs[w1][v1].adj = G.arcs[v1][w1].adj;
589         }//if
590 
591         //操作成功
592         return OK;
593     }//InsertArc
594 
595     /*
596         函数: DeleteArc
597         参数: MGraph &G 图的引用
598               VertexType v 顶点v(弧尾)
599               VertexType w 顶点w(弧头)
600         返回值: 状态码, 操作成功返回OK, 操作失败返回ERROR
601         作用: 在G中删除弧<v,w>, 若G是无向的, 则还删除对称弧<w,v>
602     */
603     Status DeleteArc(MGraph &G, VertexType v, VertexType w) {
604 
605         //i表示不可达, 在图中, 0表示不可达, 在网中INFINITY表示不可达
606         int j = 0;
607 
608         //若是网, 则使用INFINITY表示不可达
609         if(G.kind % 2) {
610             j = INFINITY;
611         }//if
612 
613         //弧头顶点v在图中的序号v1
614         int v1 = LocateVex(G, v);
615 
616         //弧尾顶点w在图中的序号w1
617         int w1 = LocateVex(G, w);
618 
619         //检查顶点v和顶点w在图中是否存在
620         if(v1 < 0 || w1 < 0) { //v或w有一个不是图G中的顶点
621             return ERROR;
622         }//if
623 
624         //将顶点v与顶点w之间设为不可达
625         G.arcs[v1][w1].adj = j;
626 
627         //如果是无向图或网, 还需要删除对称弧<w,v>
628         if(G.kind >= 2) {
629             G.arcs[w1][v1].adj = j;
630         }//if
631 
632         //弧数-1
633         G.arcnum--;
634 
635         //操作成功
636         return OK;
637     }//DeleteArc
638 
639 
640     //-----------------------图的遍历-----------------------------
641 
642     //访问标志数组
643     int visited[MAX_VERTEX_NUM];
644 
645     //函数变量
646     Status (*VisitFunc)(int v);
647 
648     /*  函数: Print
649         参数: int v 被访问的顶点v
650         返回值: 状态码, 操作成功返回OK, 操作失败返回ERROR
651         作用: 元素访问函数, 作为函数变量被调用
652     */
653     Status Print(int v){
654 
655         //设置元素访问方式为控制台打印
656         printf(" %3d ", v);
657 
658         //操作成功
659         return OK;
660     }//Print
661 
662     //----------------------DFS遍历-----------------------------
663     /*函数: DFS
664         参数: MGraph G 图G
665               int v 从序号为v的顶点出发
```

```
666         返回值：无
667         作用：从第v个顶点出发递归地深度优先遍历图G
668     */
669     void DFS(MGraph G, int v){
670
671         //先将访问标志数组该元素的访问标志改为True，
672         //含义是序号为v的顶点已被访问
673         visited[v] = TRUE;
674
675         //访问第v个顶点
676         VisitFunc(G.vexs[v]);
677
678         //依次访问v的各个邻接顶点（向v的邻接点延伸）
679         for(int w = FirstAdjVex(G, G.vexs[v]); w >= 0;
680                 w = NextAdjVex(G, G.vexs[v], G.vexs[w])){
681
682             //对v的尚未被访问的邻接点w递归调用DFS
683             if(!visited[w]) {
684                 DFS(G, w);
685             }//if
686         }//for
687     }//DFS
688
689     /*函数：DFSTraverse
690         参数：MGraph G 图G
691               Status (*Visit)(int v) 函数指针，指向元素访问函数
692         返回值：无
693         作用：对图G作深度优先遍历，调用Visit()函数访问结点
694     */
695     void DFSTraverse(MGraph G, Status (*Visit)(int v)){
696
697         //使用全局变量VisitFunc，使DFS不必设函数指针参数
698         VisitFunc = Visit;
699
700         //预置标志数组visited所有值为FALSE
701         for(int v = 0; v < G.vexnum; ++v) {
702             visited[v] = FALSE;
703         }//for
704
705         //深度优先遍历主循环
706         //写成循环是为了保证在图分为多个连通分量时能对
707         //每个连通分量进行遍历
708         for(int v = 0; v < G.vexnum; ++v) {
709
710             //若该顶点未被访问，则调用DFS()访问该节点
711             if(!visited[v]) {
712                 DFS(G, v);
713             }//if
714         }//for
715     }//DFSTraverse
716
717     //----------------广度优先遍历（需要使用队列）BFS------------
718     //说明：队列的相关代码包含在"SqQueue.cpp"中，关于队列的详细方法请阅读该文件
719
720     /*函数：BFSTraverse
721         参数：MGraph G 图G
722               Status (*Visit)(int v) 函数指针，指向元素访问函数
723         返回值：无
724         作用：按广度优先非递归遍历图G，使用辅助队列Q和访问标志数组visited
725     */
726     void BFSTraverse(MGraph G, Status (*Visit)(int v)){
727
728         int u;
729
730         //广度优先遍历使用到遍历
731         SqQueue Q;
732
733         //预置标志数组visited所有值为FALSE
734         for(int v = 0; v < G.vexnum; ++v) {
735             visited[v] = FALSE;
736         }//for
737
738         //初始化辅助队列Q，得到一个空队列
739         InitQueue_Sq(Q);
740
741         //广度优先遍历主循环
742         for(int v = 0; v < G.vexnum; ++v) {
743
744             //v尚未访问
745             if(!visited[v]){
746
747                 //设置v已经被访问
748                 visited[v] = TRUE;
749
```

BFS算法不完善

```
750             //访问第v顶点
751             Visit(G.vexs[v]);
752 
753             //v入队列
754             EnQueue_Sq(Q, v);
755 
756             //队列不空
757             while(!QueueEmpty_Sq(Q)){
758 
759                 //队头元素出队并置为u
760                 DeQueue_Sq(Q, u);
761 
762                 //依次访问第u顶点的邻接顶点
763                 for(int w = FirstAdjVex(G, G.vexs[u]); w >= 0;
764                         w = NextAdjVex(G, G.vexs[u], G.vexs[w])){
765 
766                     //w为u尚未访问的邻接顶点
767                     if(!visited[w]){
768 
769                         //设置第w顶点已被访问
770                         visited[w] = TRUE;
771 
772                         //访问第w顶点
773                         Visit(G.vexs[w]);
774                     }//if
775                 }//for
776             }//while
777         }//if
778     }//for
779 
780     //销毁循环队列
781     DestroyQueue_Sq(Q);
782 }//BFSTraverse
783 
784 
785 /*
786 //----------------------主函数----------------------
787 int main(){
788     MGraph G;
789     CreateGraph(G);
790 
791     PrintAdjMatrix(G);
792 
793     cout << "DFS结果: " << endl;
794     DFSTraverse(G,Print);
795     cout << endl;
796 
797     cout << "BFS结果: " << endl;
798     BFSTraverse(G,Print);
799     cout << endl;
800 
801     return 0;
802 }
803 */
804 
```

添一句：EnQueue_Sq(Q, w)