图

一个图对应一个偶对，G = (V, E)。
V（vertex）是顶点的非空集合；E（edge）是无需集V&V的一个子集，记为E(G)，元素为弧（边Arc）：

G = (V, E)
V = {v | v ∈ data object}
E = {<v, w> | V, W ∈ V ^ p(v, w)}
P(v, w)表示从顶点v到顶点w有一条直通通路

图分为有向图Digraph和无向图Undigraph（E = {(v, w) | ... }）。

设图中顶点数为n，边的数目为e，则
对于无向图：e ∈ [0, n(n-1)/2]；
对于有向图：e ∈ [0, n(n-1)]。

408要求中都是简单图，不含平行边和自环。

图的种类

1. 完全无向图
   具有 n(n-1)/2 条边的无向图，即不同顶点间均有一条无向边。

2. 完全有向图
   具有n(n-1)条边的有向图，即不同顶点间均有一条弧。

3. 稀疏图和稠密图
   有很少的边或弧的图（e<nlogn）称为稀疏图，反之为稠密图。

4. 子图
   设有图 G=(V, E) 和 G'=(V', E')，V'∈V且E'∈E，则G'是G的子图。

5. 包含子图
   子图包含原图的全部顶点和部分边，称其为包含子图（V'=V, E'∈E）。

图的基本概念

设图G=(v, w)，若为有向图则为G=<v, w>

1. 如果两点互为邻接点，则边(v, w)依附于v和w。对于有向图，若两点相互有弧，则弧<v, w>与v和w相关联。

2. 度：某一顶点为起点的有向边的数量为顶点的出度OD；反之以其为终点的则为入度ID。该顶点的出度和入度之和称为度TD=OD+ID。

3. 路径长度：路径上边的数量。

4. 简单路径：一条路径中没有重复同一个顶点。

5. 回路（环）：一条路径中第一个顶点和最后一个顶点相同。

6. 简单回路（简单环）： 回路中除了头尾外没有重复顶点。
7. 连通图：图中 任意vi,vj∈V 都是联通的。反之则是非连通图
8. 连通分量：极大的连通子图。
9. 强连通图：任意两个点相互为起点、终点出发，都有有向路径。反之则为非强连通图，其极大的强连通子图称为G的强连通分量。
10. 生成树：一个连通无向图的一个极小连通子图，它有原图的全部顶点以及刚好构成一棵树的边数量。对于生成树：
    1. n个顶点的生成树有且仅有n-1条边；
    2. 如果一个图有n个顶点和n-1条边，则是非连通图；
    3. 多于n-1条边，则一定有环；
    4. 有n-1条边的不一定是生成树。
11. 网：边带有权值的图。

补充

- 对于求n条边至少要多少个顶点为连通的问题，设有x个顶点：
    - 先设置两个极端的情况：
        - 0条边
        - 构成完全图（ n(n-1)/2 ）
    - 那么此时可以考虑减少一个顶点，即可得到 (x-1)(x-2)/2 ≥ n。
    - 对于n个顶点求至少要多少条连通边亦是如此。

存储结构

邻接矩阵，邻接链表

邻接矩阵（数组）表示法

用一维数组`vexs[n]`存放顶点，用二维数组`A[n][n]`存放顶点之间的关系。

邻接矩阵是对称方阵，并且是唯一的；对于顶点vi，其度数是第i行的非0元素的个数；无向图的边数是上三角（或下三角）矩阵中的非零元素个数。

特性：

1. 给定图，邻接矩阵唯一；
2. 所占空间开销`O(N^2)`，N为顶点个数，与边的个数无关；
3. 适用于稠密图，不适用于稀疏图；
4. 计算度需要用遍历或列来实现。

无向图的邻接矩阵

1. 无权无向图：
   对于`A[i][j]`：若`(vi, vj) ∈ E`，二者邻接；否则不邻接。

2. 带权无向图：
   对于`A[i][j]`：若`Wij (vi, vj) ∈ E`，二者邻接，权值为Wij；否则∞，不邻接。

有向图的邻接矩阵

1. 无权有向图：
   对于`A[i][j]`：若`<vi, vj> ∈ E`，从vi到vj有弧；否则没有弧。

2. 带权有向图：
   对于`A[i][j]`：若`Wij (vi, vj) ∈ E`，二者邻接，权值为Wij；否则∞，不邻接。

连接矩阵的定义和基本操作

```
#define INFINITY MAX_VAL     //最大值∞
#define MAX_VEX 30           //最大顶点数

typedef enum {DG, AG, WDG, WAG} GraphKind;  //{有向图，无向图，带权有向图，带权无向图}

typedef struct ArcType{
    int vex1, vex2;          //弧或边所依附的两个顶点
    ArcValType arcVal;       //弧或边的权值
    ArcInfoType arcInfo;     //弧或边的其它信息
}arcType;    //弧或边的定义

typedef sturct{
    GraphKind kind;              //图的类型
    int vexnum, arcnum;          //图当前顶点数和弧数
    int vexs[MAX_VEX];           //顶点向量
    int adj[MAX_VEX][MAX_VEX];   //连接矩阵
}MGraph;    //结构定义

/*创建图*/
AdjGraph *createGraph(MGraph *G){
    printf("请输入图种类标志：");
    scanf("%d", &G->kind);
    G->vexnum = 0;   //初始化顶点数量
    return(G);
}

/*顶点定位
（确定一个顶点在vexs数组中的位置（下标），等同于在顺序存储的线性表中查找一个数据元素）*/
int locateVex(MGraph *G, int *vp){
    int k;
    for(k=0; k < G->vexnum; k++){
        if(G->vexs[k] == *vp) return(k);
    }
    return(-1);     //图中无此顶点
}

/*增加顶点
```

```
（类似在线性表的末尾增加一个元素）*/
int addVertex(MGraph *G, int *vp){
    int k, j;
    k = G->vexnum;
    G->vexs[G->vesnum++] = *vp;
    if(G->kind == DG || G->kind == AG){     //是否为带权的有向图或无向图
        for(j=0; j < G->vexnum; j++){
            G->adj[j][k].arcVal = G->adj[k][j].arcVal = 0;
        }
    } else {
        for(j=0; j < G->vexnum; j++){
            G->adj[j][k].arcVal = G->adj[k][j].arcVal = INFINITY;
        }
    }
    return(k);
}

/*增加一条边或弧*/
int addArc(MGraph *G, arcType *arc){
    int k, j;
    k = locateVex(G, &arc->vex1);
    j = locateVex(G, &arc->vex1);
    if(G->kind == DG || G->kind == WDG){    //是否为有向图或带权有向图
        G->adj[k][j].arcVal = arc->arcVal;
        G->adj[k][j].arcInfo = arc->arcInfo;
    } else {    //对于无向图或带权无向图，需要对称赋值
        G->adj[k][j].arcVal = arc->arcVal;
        G->adj[j][k].arcVal = arc->arcVal;
        G->adj[k][j].arcInfo = arc->arcInfo;
        G->adj[j][k].arcInfo = arc->arcInfo;
    }
    return(1);
}
```

邻接表

分别设各个顶点为头结点，将与顶点相邻接的边构成一个单链表。于是分别需要表结点和顶点结点

表结点：adjvex | info | nextarc 【邻接点域 | 数据域 | 下一邻接表结点】
顶点结点（表头结点）：data | firstarc 【数据域 | 链域】

特点：

1. 表头中每个分量就是一个单链表的头结点，分量个数是图中顶点的数目；
2. 稀疏图的情况下更省空间；
3. 无向图中，Vi的度就是第i个链表的节点数；
4. 有向图中：正邻接表出度直观；逆邻接表入度直观。

邻接表的定义和基本操作

```
```

```
#define MAX_VEX 30      //最大顶点数

typedef enum{DG, AG, WDG, WAG}GraphKind;

/*弧或边的结构定义*/
typedef struct ArcType{
    int vex1, vex2;
    infoType info;      //与弧相关的信息，例如权值
}arcType;

/*图的结构定义*/
typedef sturct{
    GraphKind kind;
    int vexnum;
    vexNode adjList[MAX_VEX];
}ALGraph;

/*表结点定义*/
typedef struct LinkNode{
    int adjvex;     //邻接点在头结点数组中的位置（下标）
    infoType info;  //弧或边的相关信息，例如权值
    struct LinkNode *nextarc;    //指向下一个表结点
}linkNode;

/*创建图*/
ALGraph *createGraph(ALGraph *G){
    printf("请输入图类型的标志：");
    scanf("%d", &G->kind);
    G->vexnum = 0;
    return(G);
}

/*图顶点的定位*/
int locateVex(ALGraph *G, int *vp){
    int k;
    for(k=0; k < G->vexnum; k++){
        if(G->adjList[k].data == *vp) return(k);
    }
    return(-1);     //图中没有这个顶点
}

/*增加顶点
直接在顶点结点插入*/
int addVertex(ALGraph *G, vexType *vp){
    int k, j;
    G->adjList[G->vexnum].data = *vp;
    G->adjList[G->vexnum].degree = 0;
    G->adjList[G->vexnum].firstarc = NULL;
    k = ++G->vexnum;
    return(k);
}

/*增加边（弧）*/
```

```
int addArc(ALGraph *G, arcType *arc){
    int k, j;
    linkNode *p, *q;
    k = locateVex(G, &arc->vex1);
    j = locateVex(G, &arc->vex2);
    p = (linkNode *)malloc(sizeof(linkNode));
    p->adjvex = arc->vex1;
    p->info = arc->info;
    p->nextarc = NULL;       //边的起始表结点赋值
    q = (linkNode *)malloc(sizeof(linkNode));
    q->adjvex = arc->vex2;
    q->info = arc->info;
    q->nextarc = NULL;       //边的末尾表结点赋值
    if(G->kind == AG || G->kind == WAG){
        /*无向图，用头插法插入到两个单链表*/
        q->nextarc = G->adjList[k].firstarc;
        G->adjList[k].firstarc = q;
        p->nextarc = G->adjList[j].firstarc;
        G->adjList[j].firstarc = p;
    } else {
        /*有向图，用头插法*/
        q->nextarc = G->adjList[k].firstarc;
        G->adjList[k].firstarc = q;     //简历正邻接链表
    }
    return(1);
}
```

补充内容

- 邻接矩阵表示是唯一的，邻接表表示不唯一

遍历【2009-2023都没考代码】

从某一点出发各访问一次其余点。在遍历过程中借助一个辅助向量`Visited[1...n]`记录被访问过的点。

图的遍历算法分为广度优先搜索算法和深度优先搜索算法。
采用邻接表时，由于邻接表不唯一，遍历结果不唯一，但给定邻接表后遍历结果唯一；邻接矩阵的遍历结果唯一。

不同的遍历顺序会生成不同的树；非连通图则会生成森林。

深度优先遍历DFS

从一点出发沿着一条路径深入，直到无法深入后再转回前一个可继续深入其它路径的点，直到遍历完全部顶点。
深度优先遍历利用栈来存储。图中有e条边时时间复杂度为`O(e)`，总时间复杂度为`O(n+e)`；有n个顶点式，`O(n)`。
通常用于拓扑排序、连通分量等。

广度优先遍历BFS

从一点触发访问所有相邻点，再依次访问这些点的相邻点，直到遍历完。
广度优先遍历用队列来存储。图中有e条边时时间复杂度为`O(e)`，总时间复杂度为`O(n+e)`；有n个顶点式，`O(n)`。
通常用于求最短路径。