图

图的定义和术语

1、 **顶点**：**在图中的数据元素通常称作顶点**（Vertex），**V** 是顶点的有穷非空集合。
2、 弧、弧尾、弧头
   a) **VR 是两个顶点之间的关系的集合**
      i. **弧**：若<v,w>∈VR，**则<v,w>表示从 v 到 w 的一条弧**。
      ii. **弧尾**：称 v 为弧尾（或初始点）
      iii. **弧头**：称 w 为弧头（或终端点）
3、 有向图、无向图
   a) **有向图**：称 2 中所叙述的图为有向图。
   b) **无向图**：若以**无序对**代表上述的有序对，则称这样的图为**无向图**。
4、 **完全图：有 n(n-1)/2 条边的无向图称为完全图**。
5、 **完全有向图：有 n(n-1)条弧的图称为完全有向图。**
6、 稀疏图、稠密图
   a) 稀疏图：有很少条边或弧。
   b) 稠密图：有很多条边或弧。
7、 子图
8、 回路（环）、简单路径
   a)
9、 连通、连通图、连通分量
10、 强连通图、强连通分量
   a) 强连通图：在有向图 G 中，如果对于每一对 vi，vj∈V，vi≠vj，从 vi 到 vj 和从 vj 到 vi 之间都存在路径，则 G 为强连通图。
   b) 强连通分量：有向图中的极大强连通子图称作有向图的强连通分量。

图的存储结构

1、 数组表示法（略）
2、 邻接表（略）
3、 十字链表
   a) **十字链表**：是有向图的另一种链式存储结构。可以看成是将有向图的邻接表和逆连接表结合起来得到的一种链表。在十字链表中，对应于有向图中每一条弧有一个结点，对应于每个顶点也有一个结点。这些结点的结构如下所示：

弧结点

Tailvex	Headvex	Hlink	Tlink	info

顶点结点

Data	Firstin	firstout

其中：在弧结点中有五个域：
①tailvex 指向弧尾顶点在图中所在位置。
②haedvex 指向弧头顶点在图中所在位置。
③hlink 指向弧头相同的下一条弧
④tlink 指向弧尾相同的下一条弧
⑤info 指向该弧的相关信息

在顶点结点中有三个域
①data 存储和顶点相关信息。
②firstin 指向以该顶点为弧头的第一个弧结点。
③firstout 指向以该顶点为弧尾的第一个弧结点。

（注：若将有向图的邻接矩阵看成是稀疏矩阵的话，则十字链表也可以看成是邻接矩阵的链表存储结构）

```
typedef struct ArcBox
{
    int tailvex,headvex;//该弧的尾和头顶点的位置
    struct ArcBox *hlink,*tlink;//分别指向弧头相同的下一条弧和弧尾相同的下一条弧
    int info;//信息
}ArcBox;
typedef struct VexNode
{
    int data;
    ArcBox *firstin,*firstout;
}VexNode;
```

图的遍历

1、深度优先搜索

类似于树的先序遍历，是树的先序遍历推广。

假设初始状态是图中所有顶点未曾被访问，则深度优先搜索可以从图中某个顶点 v 出发，访问此顶点，然后以此从 v 的未被访问的临界点出发深度优先遍历图。直至图中所有和 v 有路径相通的顶点都被访问到；若此时图中尚又顶点未被访问，则另选图中一个未曾被访问的顶点作为起始点，重复上述过程，直至图中所有顶点都被访问过。

（注：遇到没访问的就走，如果到了一个顶点，这个顶点的邻接点都被访问过了，则退回到上一个点）

2、广度优先搜索

类似于树的层序遍历。

假设从图中某顶点 v 出发，在访问了 v 之后依次访问 v 的各个未曾访问过的邻接点，然后分别从这些邻接点出发以此访问它们的邻接点，并使“先被访问的顶点的邻接点”先访问，直至图中所有已被访问的顶点作为起始点。若此时图中尚有顶点未访问，则另选图中一个未曾被访问过的顶点作起始点，重复上述过程。

（一层一层的访问）

图的连通性问题

1、 无向图的连通分量和生成树
   a) 在对无向图进行遍历时，对于连通图，仅需从图中任一顶点出发，进行深度优先搜索或者广度优先搜索即可。但对于非连通图，则需从多个顶点出发进行搜索，而每一次从一个新的起始点出发进行搜索的过程中得到的定点访问序列恰为其各个连通分量中的顶点集。
   b) 生成树：广度优先搜索得到的为广度优先生成树；深度优先搜索得到的为深度优先生成树。
2、 有向图的强连通分量
   深度优先搜索是求有向图的强连通分量的一个新的有效方法。假设以十字链表作有向图的存储结构，则步骤如下：
   a)
3、 最小生成树
   a) MST 性质：假设 N=(V,{E})是一个连通网，U 是顶点集 V 的非空子集。若（u，v）是一条具有最小权值的边，其中 u∈U，v∈V-U，则必定存在一棵包含（u，v）的最小生成树。
   b) PRIM 算法
      假设 N=（V,{E}）是连通网，TE 是 N 上最小生成树中边的集合。算法从 U={u0}，TE={}开始，重复执行下述过程：在所有 u∈U，v∈V-U 的边（u，v）∈E 中找一条代价最小的边（u0，v0）并入集合 TE，同时 v0 并入 U，直至 U=V 为止。此时 TE 中必有 n-1 条边，则 T=（V，{TE}）为 N 的最小生成树。
      （注：时间复杂度：O（n^2））
   c) 克鲁斯卡尔算法
      克鲁斯卡尔算法恰恰相反，它的时间复杂度为 O（eloge），eloge 为网中的边数。假设连通网 N=（V，{E}），则令最小生成树的初始状态为只有 n 个顶点而无边的非连通图 T=（V，{}），图中每个顶点自成一个连通分量。在 E 中选择代价最小的边，若该边依附的顶点落在 T 中不同的连通分量上，则将此边加入到 T 中，否则发舍去此边而选择下一条代价最小的边。以此类推，直至 T 中所有顶点都在同一连通分量上为止。
4、 关节点和重连通分量
   a) 关节点：假若在删去**顶点 v 以及和 v 相关联的各边**之后，将图的**一个连通分量分割成两个或两个以上的连通分量**，则称顶点 v 为该图的一个**关节点**。
   b) **重连通图**：一个**没有关节点**的连通图称为**重连通图**。
   c) **连通度：**在重连通图上，任意一对顶点之间至少存在两条路径。**若在连通图上至少删去 k 个顶点才能破坏图的连通性，则称此图的连通度为 k**。
   d) **求关节点的方法**：
      i. **主要思路：利用深度优先搜索**。
      ii. 由深度优先生成树可得出两类关节点的特性
         1. 若生成树的根由两棵或两棵以上的子树，那么根节点必为关节点。
         2. 若生成树中某个非叶子顶点 v，其某棵子树的根和子树中的其他结点均没有指向 v 的祖先的回边，则 v 为关节点。即 v 的下面（某棵子树）没有往

v 上指的（通俗说法）。

有向无环图及其应用

1、**有向无环图：一个无环的有向图称作有向无环图**。

最短路径

1、从某个源点到其余各个顶点的最短路径（Dijkstra 算法）

a) 引入向量：distance[N]表示从 v0 到其他点的最短距离
b) 选择 vj 使得 distance[j]=Min{D[i]，vi∈V-S}，vj 就是当前求得的一条从 v 出发的最短路径的终点。将 vj 加入 S 中。
c) 修改从 v0 出发到集合 V-S 上任意一点 vk 可达到的最短路径长度。如果原来的 distance[k]>从该点到 k 点的距离+v 到 j 的距离，则修改 distance[k]
d) 重复 b，c 共 n-1 次。最后的 distance[N]就是所求。

2、每一对顶点之间的最短路径

附 1：自己写的 PRIM 简易算法（c 语言）

```
#include <stdio.h>
#include <stdlib.h>
#define MAXSIZE 6

int V[MAXSIZE][MAXSIZE]={0};
int exist[MAXSIZE]={0};
int num=0;

void init_sysm();
void PRIM();
void findmin();

int main()
{
	init_sysm();
	PRIM();
	return 0;
}

void init_sysm()
{
```

```
    for(int i=0;i<MAXSIZE;i++)
    {
        for(int j=0;j<MAXSIZE;j++)
        {
            V[i][j]=1000;
        }
    }
    V[0][1]=6;
    V[0][2]=1;
    V[0][3]=5;
    V[1][0]=6;
    V[2][0]=1;
    V[3][0]=5;
    V[1][2]=5;
    V[2][1]=5;
    V[1][4]=3;
    V[4][1]=3;
    V[2][4]=6;
    V[2][5]=4;
    V[2][3]=5;
    V[4][2]=6;
    V[5][2]=4;
    V[3][2]=5;
    V[3][5]=2;
    V[5][3]=2;
    V[4][5]=6;
    V[5][4]=6;
}

void PRIM()
{
    //从 V1 开始搜索
    exist[0]=1;//V！被标记
    num++;
    for(int i=0;i<MAXSIZE-1;i++)
    {
        findmin();//从 U 中寻找代价最小的路
    }
}

void findmin()
{
    int minval=1000;
    int in=0,out=0;
```

```
    for(int i=0;i<MAXSIZE;i++)
    {
        if(exist[i]==1)//存在
        {
            for(int j=0;j<MAXSIZE;j++)
            {
                if(minval>V[i][j]&&i!=j&&exist[j]!=1)
                {
                    minval=V[i][j];
                    in=i;
                    out=j;
                }
            }
        }
    }
    V[in][out]=1000;
    V[out][in]=1000;
    exist[out]=1;//存在
    exist[in]=1;
    printf("%d->%d\n",in+1,out+1);
}
```

附 2：自己写的 Dijkstra 算法（c 语言）

```
#include <stdio.h>
#include <stdlib.h>
#define MAX 1000 //定义最大长度
#define N 6
int arcs[N][N]={0};
int exist[N]={0};
int path[N][N]={0};

void init_vex();
void print_vex();
void dijkstra(int distance[],int num);
void findmin(int distance[],int *index,int *min);

int main()
{

    int distance[N]={0};
    init_vex();
    print_vex();
```

```
	printf("\n\n");
	dijkstra(distance,0);
	for(int i=0;i<N;i++)
	{
		printf("%d\n",distance[i]);
	}
	return 0;
}
void init_vex()
{
	//初始化
	for(int i=0;i<N;i++)
	{
		for(int j=0;j<N;j++)
		{
			arcs[i][j]=MAX;
		}
	}
	arcs[0][2]=10;
	arcs[1][2]=5;
	arcs[0][4]=30;
	arcs[0][5]=100;
	arcs[2][3]=50;
	arcs[4][3]=20;
	arcs[3][5]=10;
	arcs[4][5]=60;
}

void print_vex()
{
	for(int i=0;i<N;i++)
	{
		for(int j=0;j<N;j++)
		{
			printf("\t%d",arcs[i][j]);
		}
		printf("\n");
	}
}

void findmin(int distance[],int *index,int *min)
{
	int tmin=distance[0];
	int tindex=0;
```

```
    for(int i=1;i<N;i++)
    {
        if(distance[i]<tmin&&exist[i]==0)
        {
            tmin=distance[i];
            tindex=i;
        }
    }
    *min=tmin;
    *index=tindex;
}
void dijkstra(int distance[],int num)
{
    //第一个参数：临时向量存储最短距离
    //第二个参数：从哪个起点出发
    int index=0;//找到的最小点的坐标
    int min=0;
    exist[num]=1;//在 S 中存在
    int temp[N];
    for(int i=0;i<N;i++)
    {
        distance[i]=arcs[num][i];
        path[i][num]=0;
    }
    for(int i=1;i<N;i++)
    {
        findmin(distance,&index,&min);
        exist[index]=1;
        for(int j=0;j<N;j++)
        {
            temp[j]=arcs[index][j];
        }
        //更新 distance 数据
        for(int j=0;j<N;j++)
        {
            if(distance[j]>temp[j]+min)
            {
                distance[j]=temp[j]+min;
            }
        }
    }

}
```