第12次实验

## 1、最小生成树问题

一个有N个点的图，边一定是大于等于N-1条的。图的最小生成树，就是在这些边中选择N-1条出来，连接所有的N个点。这N-1条边的边权之和是所有方案中最小的。参见图1。

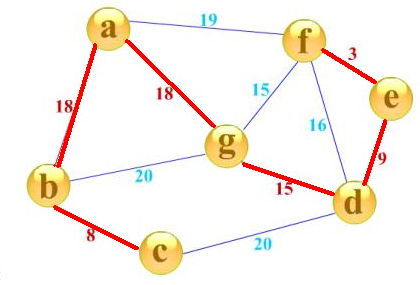


图1 一个最小生成树的例子

### 最小生成树——普里姆算法

普里姆算法是从某个顶点出发，逐步“生长”出其他所有顶点。考查图1所示的网络。不妨取V0为初始点，可以将顶点分成两个集合：S={V0}和T={除去V0的其他点}。

* 第1步，连接S和T共有3条边，在生成树中这两个集合必然连通，即3条黑色的边中必有一条在生成树中。为了使生成树边上权值之和最小，根据贪心的原则，应取三条黑色的边中权最小的(V0,V2)，如图1。

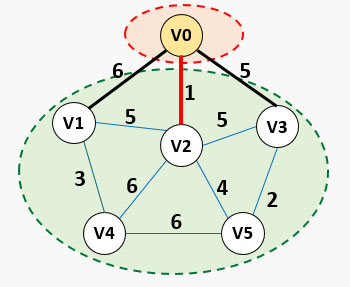
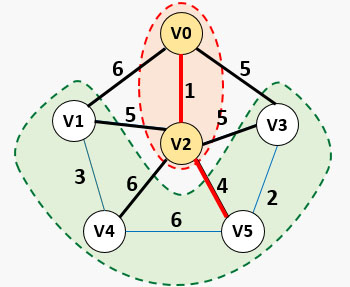
 

图1 prim算法第1步 图2 prim算法第2步

* 第2步，设S={V0,V2}、T={除去V0、V2的其他点}。连接S和T共有6条边，在生成树中这两个集合必然连通，即6条黑色的边中必有一条在生成树中。为了使生成树边上权值之和最小，根据贪心的原则，应取6条黑色的边中的(V2,V5)，如图2。

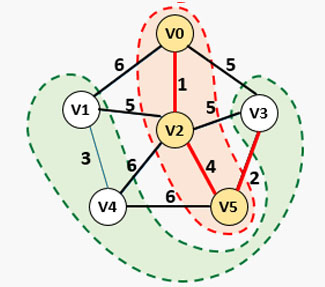
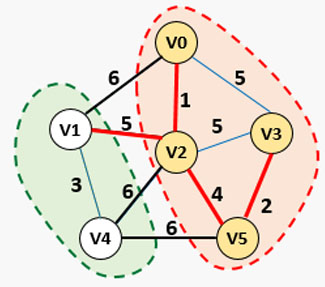
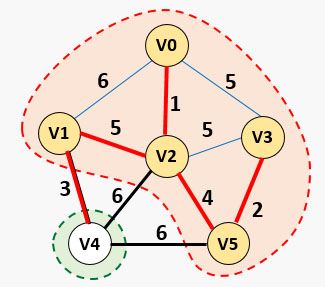
  

图3 prim算法第3步 图4 prim算法第4步 图5 prim算法第5步

* 第3步，设S={V0,V2,V5}、T={V1,V4,V3}。连接S和T共有7条边，继续“生长”出其他顶点，取黑色的边中的(V3,V5)。
* 第4步，设S={V0,V2,V3,V5}、T={V1,V4}。为了使生成树边上权值之和最小，应取黑色的边中的(V1,V2)。
* 第5步，设S={V0,V1,V2,V3,V5}、T={V4}。为了使生成树边上权值之和最小，应取黑色的边中的(V1,V4)。

【Prim算法】设U为生成树顶点的集合，初始状态仅包含起点u0

1. 从含权连通图的某一顶点 u0 出发，选择与它关联的具有最小权值的边（u0,v），将其顶点 v 加入到生成树的顶点集合U中，同时记录边（u0,v）为生成树的一部分
2. 从一个顶点在U中（记作u），而另一个顶点不在U中（记作v）的各条边中选择权值最小的边（u,v），把顶点 v 加入集合U，同时记录边（u,v）为生成树的一部分
3. 将上面步骤反复作下去，直到图中的所有顶点都加入到生成树顶点集合U中

### 最小生成树——克鲁斯卡尔算法

克鲁斯卡尔算法从另一途径求最小生成树，该算法着眼于边考虑解法。

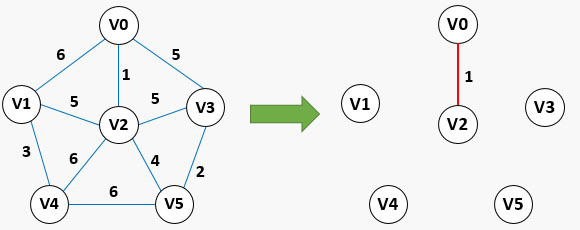
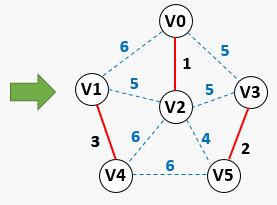
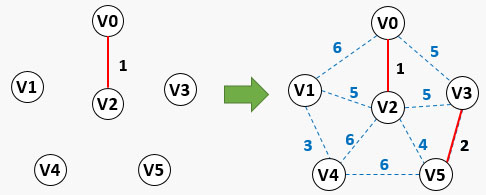


图6 第1步

将原图拆解为互不相连的N个连通分量(单结点树)，考察所有**连接不同分量的边。**

* 为了使生成树边上权值之和最小，根据贪心的原则，这里取所有连接不同分量的边中权值最小的边 (V0,V2)作为生成树的一部分。
* 继续**选择连接两个不同分量的权值最小的边**



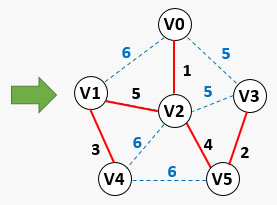
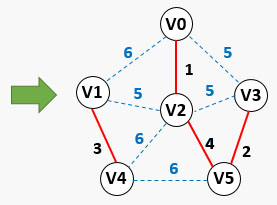


图7 第2~5步

【克鲁斯卡尔算法】

1. 假设含权连通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
2. 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，否则舍去此边而选择下一条代价最小的边
3. 以此类推，直至T中所有顶点都在同一连通分量上为止

## 2、实验：克鲁斯卡尔算法编程

用邻接矩阵表示图。比如下面左侧的图，可用右侧矩阵表示。这里M为很大的整数

表示不直接连通。

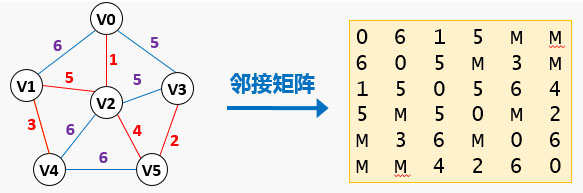


图8 邻接矩阵

在矩阵中以上排序操作不方便编程，所以定义下面结构体存储边的信息。

typedef struct

{

int begin; // 边的起点

int end; // 边的终点

int weight; // 边的权重

bool selected; // 是否作为最小生成树的边，初值flase

}Edge;

// 定义存放边的数组

Edge edges[MAXEDGE];

* 将邻接矩阵中边的信息读入edges[]，见图9

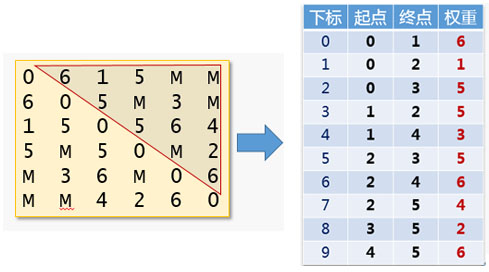
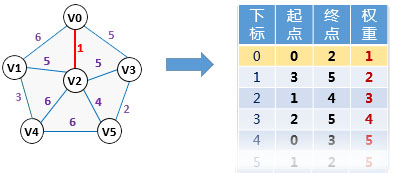
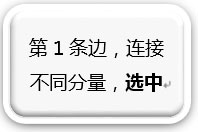
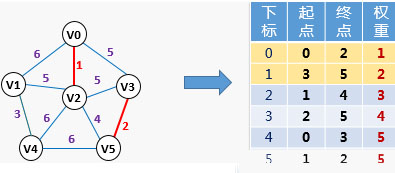
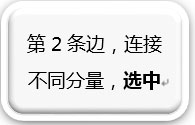
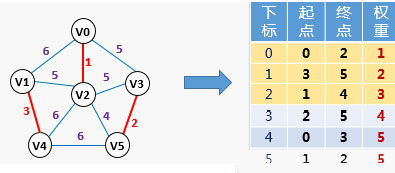
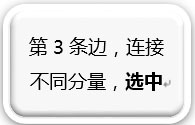
 

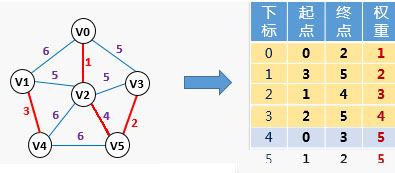
图9 读邻接矩阵到edges[]，再排序

* 将数组中的边**按权重从小到大排序**
* （**核心步骤**）依次读取数组中的边，并做如下操作：
* 若该边连接不同分量则边加入生成树（将selected值设为true）
* 否则，跳过此边，处理下一条边

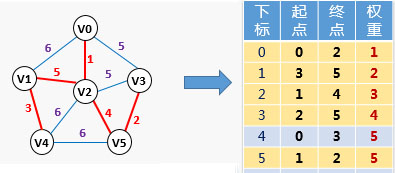
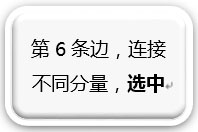
 

**第5条边（V0，V3）**，**连接相同分量**，**放弃！！**

**已经选中N-1条边，结束！**

* **如何判断一条边连接两个不同分量？**

可以在每个分量中设一个根。对一条边而言，如果其起点和终点的根不同，就是连接不同分量。

* 定义一个数组parent[]，其中parent[i]存储 i 号点的父结点或根结点 。初始状态parent[]值全为-1，表示 i 号点的根就是自己

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 0 | 1 | 2 | 3 | 4 | 5 |
| -1 | -1 | -1 | -1 | -1 | -1 |

* 生成树每加入一条边后，执行 **parent[终点]=起点**
  + 加入第1条边(V0,V2)，执行parent[2]=0
  + 加入第2条边(V3,V5)，执行parent[5]=3
  + 加入第3条边(V1,V4)，执行parent[4]=1

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 0 | 1 | 2 | 3 | 4 | 5 |
| -1 | -1 | 0 | -1 | 1 | 3 |

* + 加入第4条边(V2,V5)，查V2的根：parent[2]为0 ，parent[0]为-1，所以V2根为0；查V5的根：parent[5]为3， parent[3]为-1，故V5的根为3；综上，边(V2,V5)连接不同分量，可加入，执行parent[3]=0 （即parent[终点的根]=起点的根）

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 0 | 1 | 2 | 3 | 4 | 5 |
| 1 | -1 | 0 | 0 | 1 | 3 |

* + 可以加入边(V0,V3)吗？因parent[0]为-1，V0根为0；查V3的根：parent[3]为0 ，继续找，parent[0]为-1，V3的根为0；所以V0和V3的根相同，**舍弃**
  + 加入第5条边(V1,V2)，执行parent[0]=1（0为2的根）

// 找根的函数

int Find(int \*parent, int f)

{

while (parent[f] >= 0) { f = parent[f]; }

return f;

}

* **算法概要**

int parent[MAXVEX]; //用于寻找根节点的数组，初始化为 -1

Edge edges[MAXEDGE]; //定义存储边的数组

// 初始化 edges 数组

for (i = 0; i < 顶点数-1; i++)

for (j = i + 1; j <顶点数-1; j++)

读入邻接矩阵边的信息到 edges[]

Sort(edges，边数); // Sort函数为边数组Edge排序

for (i = 0; i < 边数; i++)

{

n = Find(parent, edges[i].begin); //找边的起点的根

m = Find(parent, edges[i].end); //找边的终点的根

// 两个顶点不在一棵子树内

if(n != m) {

parent[m] = n;

设置边i的selected为true;

}

}

输出所有选中的边；