阅读：50,172　　作者：解学武

普里姆算法(Prim算法)求最小生成树

通过前面的学习，对于含有 n 个顶点的连通图来说可能包含有多种生成树，例如图 1 所示：

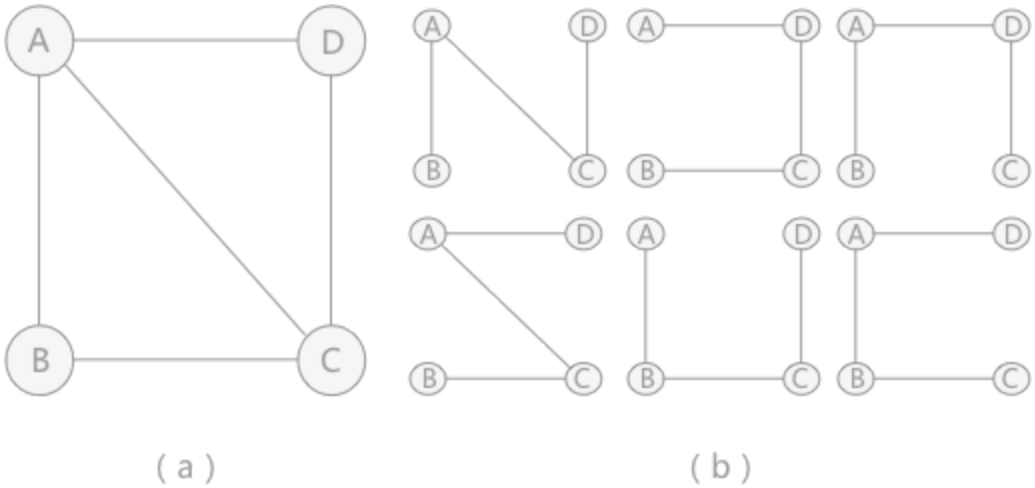


图 1 连通图的生成树

图 1 中的连通图和它相对应的生成树，可以用于解决实际生活中的问题：假设A、B、C 和 D 为 4 座城市，为了方便生产生活，要为这 4 座城市建立通信。对于 4 个城市来讲，本着节约经费的原则，只需要建立 3 个通信线路即可，就如图 1（b）中的任意一种方式。

在具体选择采用（b）中哪一种方式时，需要综合考虑城市之间间隔的距离，建设通信线路的难度等各种因素，将这些因素综合起来用一个数值表示，当作这条线路的权值。

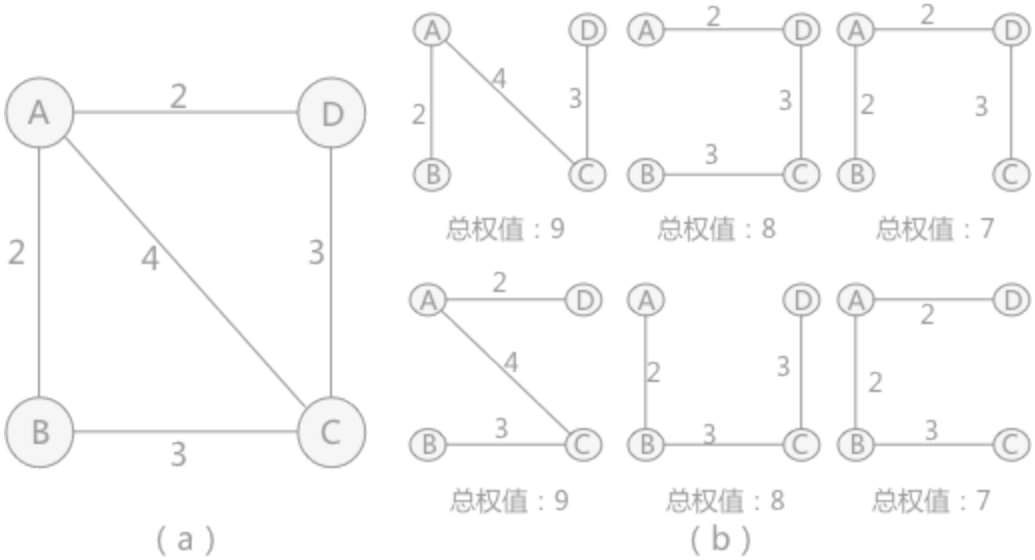


图 2 无向网

假设通过综合分析，城市之间的权值如图 2（a）所示，对于（b）的方案中，选择权值总和为 7 的两种方案最节

约经费。

这就是本节要讨论的最小生成树的问题，简单得理解就是给定一个带有权值的连通图（连通网），如何从众多的生成树中筛选出权值总和最小的生成树，即为该图的最小生成树。

给定一个连通网，求最小生成树的方法有：普里姆（Prim）算法和克鲁斯卡尔（Kruskal）算法。

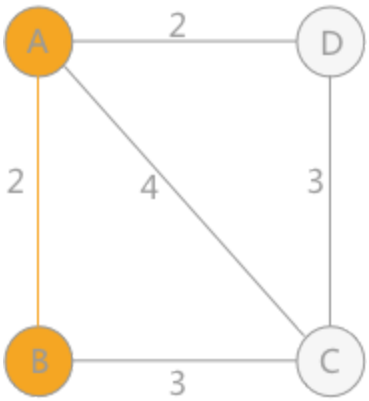
普里姆算法

普里姆算法在找最小生成树时，将顶点分为两类，一类是在查找的过程中已经包含在树中的（假设为 A 类），剩下的是另一类（假设为 B 类）。

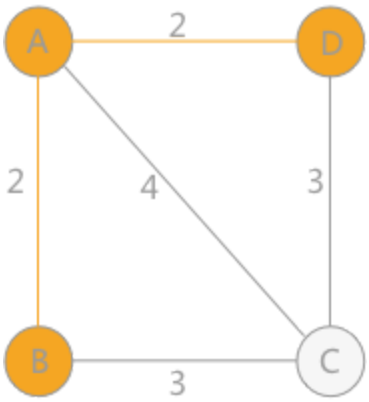
对于给定的连通网，起始状态全部顶点都归为 B 类。在找最小生成树时，选定任意一个顶点作为起始点，并将之从 B 类移至 A 类；然后找出 B 类中到 A 类中的顶点之间权值最小的顶点，将之从 B 类移至 A 类，如此重复，直到 B 类中没有顶点为止。所走过的顶点和边就是该连通图的最小生成树。

例如，通过普里姆算法查找图 2（a）的最小生成树的步骤为：

假如从顶点A出发，顶点 B、C、D 到顶点 A 的权值分别为 2、4、2，所以，对于顶点 A 来说，顶点 B 和顶点 D 到 A 的权值最小，假设先找到的顶点 B：

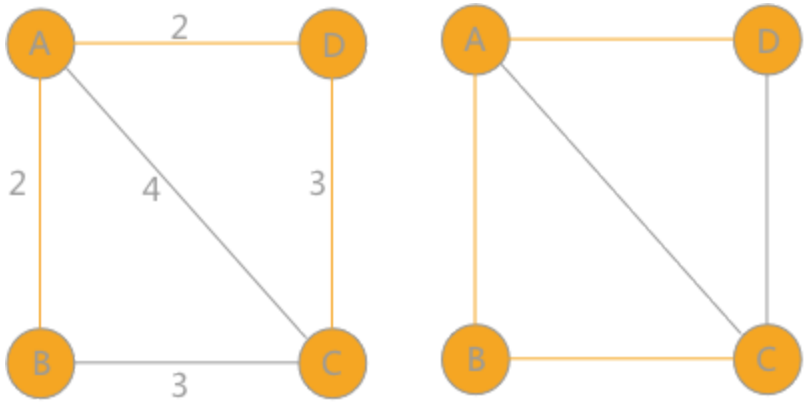


继续分析顶点 C 和 D，顶点 C 到 B 的权值为 3，到 A 的权值为 4；顶点 D 到 A 的权值为 2，到 B 的权值为无穷大（如果之间没有直接通路，设定权值为无穷大）。所以顶点 D 到 A 的权值最小：



最后，只剩下顶点 C，到 A 的权值为 4，到 B 的权值和到 D 的权值一样大，为 3。所以该连通图有两个最小生成

树：



具体实现代码为：

```
#include <stdio.h>
#include <stdlib.h>
#define VertexType int
#define VRType int
#define MAX_VERtEX_NUM 20
#define InfoType char
#define INFINITY 65535
typedef struct {
    VRType adj;                                 //对于无权图，用 1 或 0 表示是否相邻；对于带权图，直
    InfoType * info;                            //弧额外含有的信息指针
}ArcCell,AdjMatrix[MAX_VERtEX_NUM][MAX_VERtEX_NUM];

typedef struct {
    VertexType vexs[MAX_VERtEX_NUM];            //存储图中顶点数据
    AdjMatrix arcs;                             //二维数组，记录顶点之间的关系
    int vexnum,arcnum;                          //记录图的顶点数和弧（边）数
}MGraph;

//根据顶点本身数据，判断出顶点在二维数组中的位置
int LocateVex(MGraph G,VertexType v){
    int i=0;
    //遍历一维数组，找到变量v
    for (; i<G.vexnum; i++) {
        if (G.vexs[i]==v) {
            return i;
        }
    }
    return -1;
}
//构造无向网
void CreateUDN(MGraph* G){
    scanf("%d,%d",&(G->vexnum),&(G->arcnum));
    for (int i=0; i<G->vexnum; i++) {

```

```
34.         scanf("%d",&(G->vexs[i]));
35.     }
36.     for (int i=0; i<G->vexnum; i++) {
37.         for (int j=0; j<G->vexnum; j++) {
38.             G->arcs[i][j].adj=INFINITY;
39.             G->arcs[i][j].info=NULL;
40.         }
41.     }
42.     for (int i=0; i<G->arcnum; i++) {
43.         int v1,v2,w;
44.         scanf("%d,%d,%d",&v1,&v2,&w);
45.         int m=LocateVex(*G, v1);
46.         int n=LocateVex(*G, v2);
47.         if (m==-1 ||n==-1) {
48.             printf("no this vertex\n");
49.             return;
50.         }
51.         G->arcs[n][m].adj=w;
52.         G->arcs[m][n].adj=w;
53.     }
54. }
55.
56. //辅助数组，用于每次筛选出权值最小的边的邻接点
57. typedef struct {
58.     VertexType adjvex;//记录权值最小的边的起始点
59.     VRType lowcost;//记录该边的权值
60. }closedge[MAX_VERtEX_NUM];
61. closedge theclose;//创建一个全局数组，因为每个函数中都会使用到
62. //在辅助数组中找出权值最小的边的数组下标，就可以间接找到此边的终点顶点。
63. int minimun(MGraph G,closedge close){
64.     int min=INFINITY;
65.     int min_i=-1;
66.     for (int i=0; i<G.vexnum; i++) {
67.         //权值为0，说明顶点已经归入最小生成树中；然后每次和min变量进行比较，最后找出最小的。
68.         if (close[i].lowcost>0 && close[i].lowcost < min) {
69.             min=close[i].lowcost;
70.             min_i=i;
71.         }
72.     }
73.     //返回最小权值所在的数组下标
74.     return min_i;
75. }
76. //普里姆算法函数，G为无向网，u为在网中选择的任意顶点作为起始点
77. void miniSpanTreePrim(MGraph G,VertexType u){
78.     //找到该起始点在顶点数组中的位置下标
79.     int k=LocateVex(G, u);
80.     //首先将与该起始点相关的所有边的信息：边的起始点和权值，存入辅助数组中相应的位置，例如（1，2）边，ad
```

```
81.       for (int i=0; i<G.vexnum; i++) {
82.           if (i !=k) {
83.               theclose[i].adjvex=k;
84.               theclose[i].lowcost=G.arcs[k][i].adj;
85.           }
86.       }
87.       //由于起始点已经归为最小生成树，所以辅助数组对应位置的权值为0，这样，遍历时就不会被选中
88.       theclose[k].lowcost=0;
89.       //选择下一个点，并更新辅助数组中的信息
90.       for (int i=1; i<G.vexnum; i++) {
91.           //找出权值最小的边所在数组下标
92.           k=minimun(G, theclose);
93.           //输出选择的路径
94.           printf("v%d v%d\n",G.vexs[theclose[k].adjvex],G.vexs[k]);
95.           //归入最小生成树的顶点的辅助数组中的权值设为0
96.           theclose[k].lowcost=0;
97.           //信息辅助数组中存储的信息，由于此时树中新加入了一个顶点，需要判断，由此顶点出发，到达其它各顶点
98.           for (int j=0; j<G.vexnum; j++) {
99.               if (G.arcs[k][j].adj<theclose[j].lowcost) {
100.                  theclose[j].adjvex=k;
101.                  theclose[j].lowcost=G.arcs[k][j].adj;
102.              }
103.          }
104.      }
105.      printf("\n");
106.  }
107.
108.  int main(){
109.      MGraph G;
110.      CreateUDN(&G);
111.      miniSpanTreePrim(G, 1);
112.  }
```

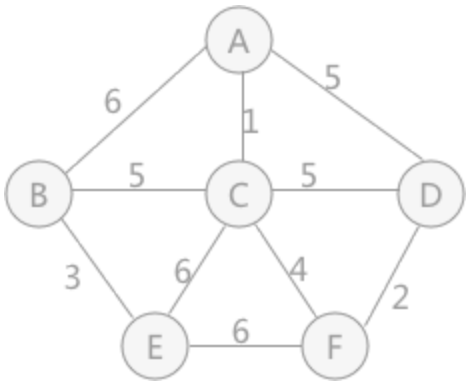


图 3 无向网

使用普里姆算法找图 3 所示无向网的最小生成树的测试数据为：

```
6,10
1
```

2
3
4
5
6
1,2,6
1,3,1
1,4,5
2,3,5
2,5,3
3,4,5
3,5,6
3,6,4
4,6,2
5,6,6
运行结果为：
v1 v3
v3 v6
v6 v4
v3 v2
v2 v5

普里姆算法的运行效率只与连通网中包含的顶点数相关，而和网所含的边数无关。所以普里姆算法适合于解决边稠密的网，该算法运行的时间复杂度为： $O(n^2)$ 。

> 如果连通网中所含边的绸密度不高，则建议使用克鲁斯卡尔算法求最小生成树（下节详细介绍）。