AlanTu

随笔 - 952, 文章 - 0, 评论 - 45, 阅读 - 200万

导航

博客园
首页
新随笔
联系
订阅 XML
管理

< 2021年12月 >

日	一	二	三	四	五	六
28	29	30	1	2	3	4
5	6	7	8	9	10	11
12	13	14	15	16	17	18
19	20	21	22	23	24	25
26	27	28	29	30	31	1
2	3	4	5	6	7	8

搜索

找找看
谷歌搜索

常用链接

我的随笔
我的评论
我的参与
最新评论
我的标签

随笔分类

C++(128)
Linux定时测量(14)
Linux进程管理(29)
Linux内存管理(29)
Linux内核(102)
linux设备驱动(7)
Linux同步(19)
Linux文件系统(27)
Linux中断异常(16)
TCP/IP协议族(5)
编译原理(12)
电影观后感(2)
个人总结(146)
进程间通信(26)
面试(56)
设计模式(12)
数学基础(7)
算法与数据结构(129)
网络编程(86)

prim算法

一个连通图的生成树是一个极小的连通子图，它包含图中全部的顶点（n个顶点），但只有n-1条边。

最小生成树：构造连通网的最小代价（最小权值）生成树。

prim算法在严蔚敏树上有解释，但是都是数学语言，很深奥。

最小生成树MST性质：假设N=（V,{E}）是一个连通网，U是顶点集V的一个非空子集。若（u,v）是一条具有最小权值（代价）的边，

其中u∈U，v∈V-U，则必存在一颗包含边（u,v）的最小生成树。

prim算法过程为：

假设N=（V,{E}）是连通图，TE是N上最小生成树中边的集合。算法从U={u_0}（u_0∈V），TE={}开始，

重复执行下述操作：

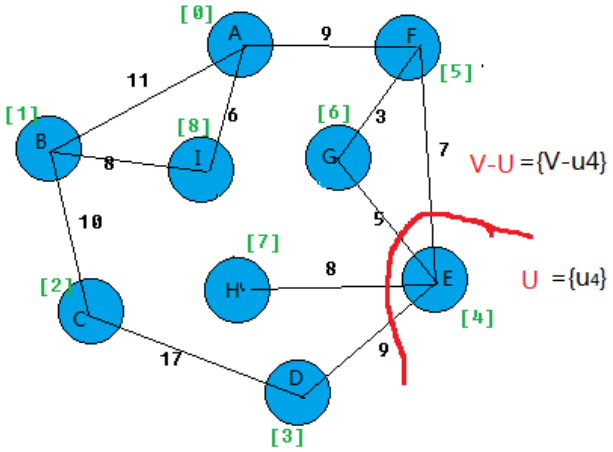
在所有u∈U，v∈V-U的边（u，v）∈E中找一条代价最小的边（u_0，v_0）并入集合TE，同时v_0 并入U，直至U=V为止。

此时TE中必有n-1条边，则T=（V，{TE}）为N的最小生成树。

我以图为例，看看算法过程。

1. 初始

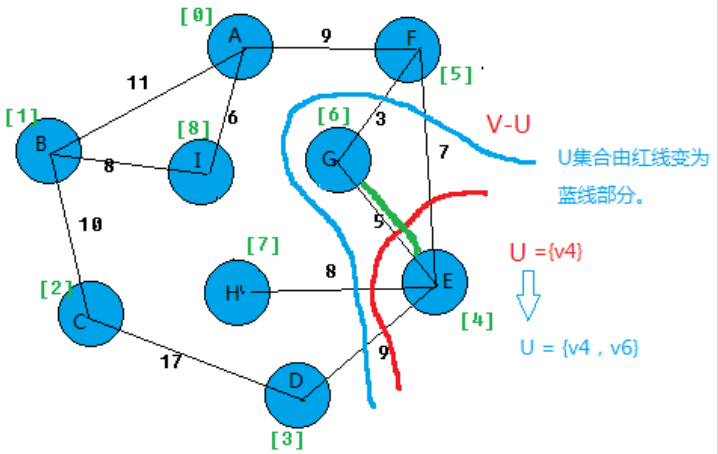
V={图中所有的顶点}



我们选择从E（[4]）顶点开始。

对于整个图，当我们选择E开始之后，整个图可以分成两个部分。

2.



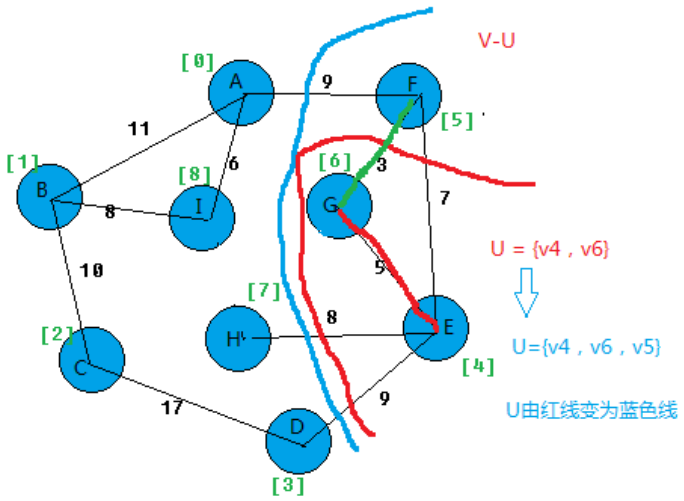
1.U（v4即点E）与V-U一共有四条边，其中最小的边是（v4，v6）。

2.TE={(v4,v6)}。

3.v6并入到U。

评论排行榜

3

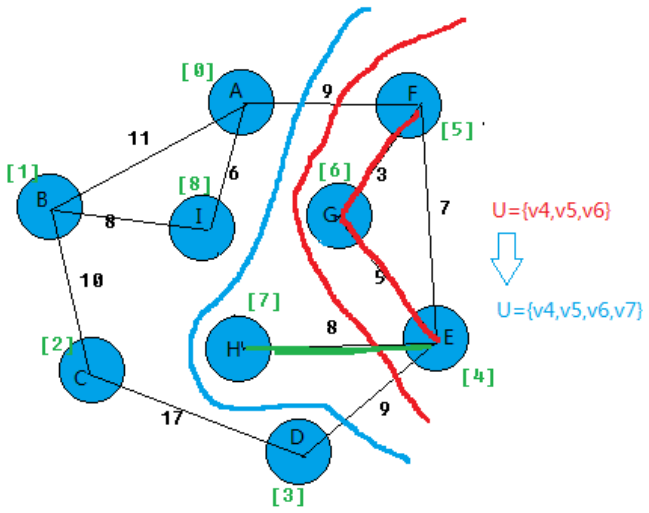


1.U={v4，v6} 与V-U的边最小为（v6，v5） = 3

2.（v6，v5）纳入TE中。

3.v5纳入U中。

4



1。U和V-U的边有（v5，v0）、（v4，v7）、（v4，v3），

2。其中最小为（v4，v7），纳入到TE中

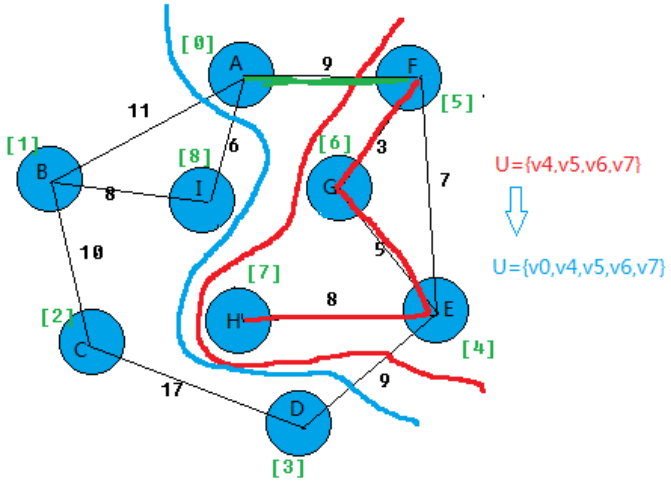
3。v7纳入到U中

推荐排行榜

最新评论

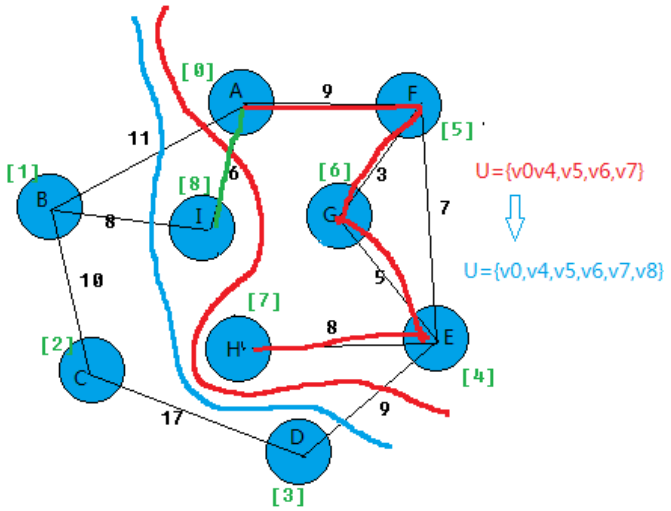
1. Re:Linux深入理解Socket异常
有个疑问, "crash的一端即没发送FIN也没发送RST"这种情况是什么时候会出现呢? 服务器断电, 宕机还是什么其他原因
--Durian_yang
2. Re:c++拷贝构造函数详解
如果定义了拷贝构造函数, 那也必须重载赋值操作符.这句话怎么理解? 上面的例子没有重载赋值运算符呀
--出手无招
3. Re:linux select函数详解
select 必须搭配轮询吗?
--明明1109
4. Re:Linux深入理解Socket异常
写的真好
--喜欢兰花山丘
5. Re:Ext4文件系统架构分析(一)
@miaoqingcoding 换个浏览器， Firefox可以看到图，chrome看不到...
--Tudou_Blog
6. Re:Ext4文件系统架构分析(一)
大佬你这怎么没图啊
--miaoqingcoding
7. Re:Linux下逻辑地址、线性地址、物理地址详细总结
第一小节讲的很棒
--小北觅
8. Re:AOE网与关键路径简介
文章中有一处错误。应该是你截图的书的勘误。 “可以发现，在计算ltv时，其实是把拓扑序列倒过来进行而已，因此可以得到计算顶点vk最晚发生时间即求ltv[k] 的公式是：” 这句话下面的图中，公式里的第...
--lulipro

5



1。U与V-U边（v5，v0）、（v4，v3），由于两个相等，按v0优先，
2。（v0，v5）纳入到TE
3。v0纳入到U

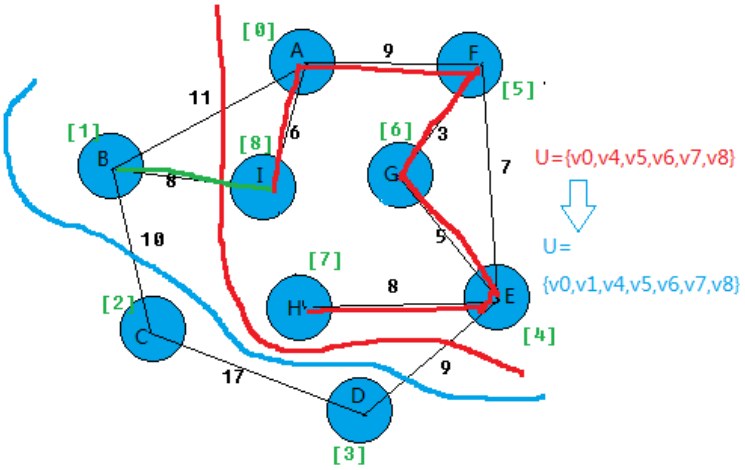
6



1. U和V-U边（v0，v1）、（v0，v8）、（v4，v3），其中
2.（v0，v8）最小，纳入到TE中
3. v8纳入到U中

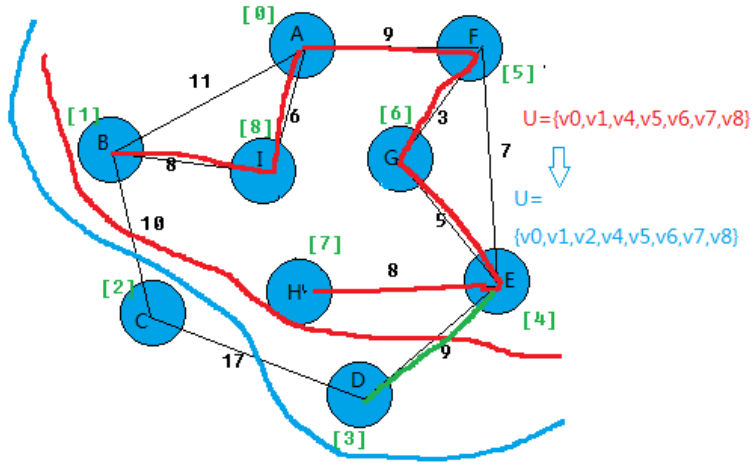
9. Re:深入理解Linux内存分配
两脸懵逼
--不知也
10. Re:prim算法
图清晰
--pubby
11. Re:c++拷贝构造函数详解
讲的正好
--杀马特彪少
12. Re:c++拷贝构造函数详解
最后一句：“如果定义了拷贝构造函数，那也必须重载赋值操作符。”是指赋值操作符也要自己重新定义？
--noone3001
13. Re:二维码的生成细节和原理
大佬，数据和数据纠错码填充是从右下角开始。
--amisarex
14. Re:利用recv和readn函数实现readline函数
学习了，但是有一个问题：recv_peek 之后，如果找到了 \r\n，就需要清空缓存区。但是清空又得重复 Copy 缓存区，有没有只清空指定长度的缓存区，不重新 Copy 的方式？
--陈鑫伟
15. Re:【转】对 Rust 语言的分析
看到lifetime，我就感觉这门语言有点复杂
--sabiameno
16. Re:c++拷贝构造函数详解
@jarven_0728编译器优化的问题，你用2019版的就应该没问题...
--流霜抚墨
17. Re:c++拷贝构造函数详解
这个界面是怎么回事.....
--NoneType
18. Re:c++拷贝构造函数详解
顶顶顶!
--小小程序梦
19. Re:Linux内存初始化(一)
“假设我们分配4个page分别保存Level 0到level 3的translation table，那么可以建立的最大的地址映射范围是512（level 3中有512个entry） X 4k = 2...
--飘零剑客
20. Re:c++拷贝构造函数详解
菜鸡在线感激
--无趣趣无
21. Re:二维码的生成细节和原理
您好，您的pdf连接好像打不开了，能发一份给我吗？百度网盘链接也可以，谢谢。

7



1.U与V-U边（v0，v1）、（v8，v1）、（v4，v3），其中
2.（v8，v1）最小，纳入TE中
3. v1纳入U中。

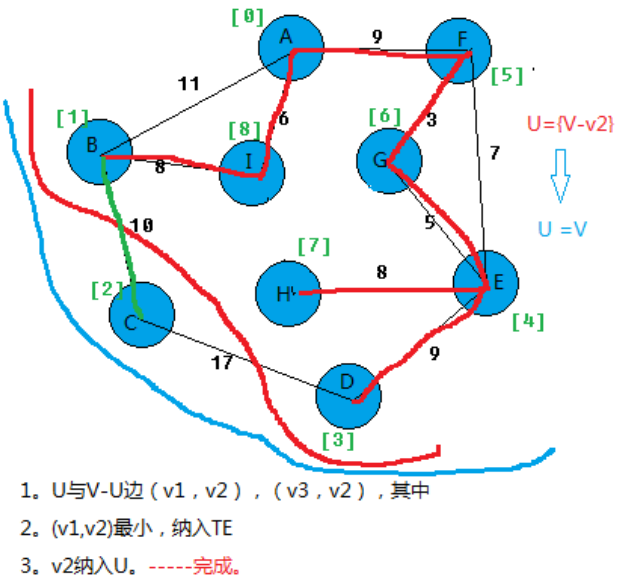
8



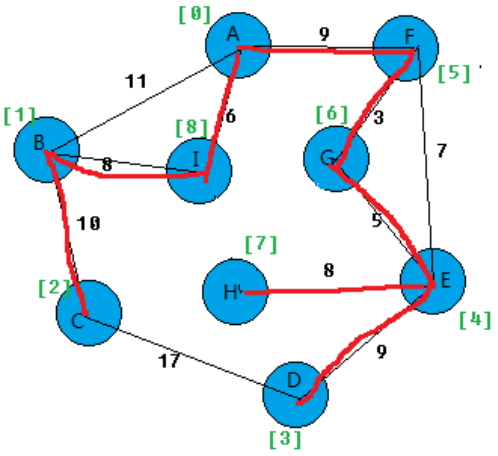
1。U与V-U边（v1，v2）、（v3，v4），其中
2。（v3，v4）最小，纳入TE
3。v3纳入U中

--ruxia_TJY
22. Re:【转】对 Rust 语言的分析
（），在Rust就是元组，
--m12
23. Re:P问题、NP问题和NPC问题
请问，一个图中是否不存在Hamilton回路这样的问题是不是就属于NP-Hard问题了？
--破碎中的麦粒
24. Re:prim算法
很棒!
--菜鸟qiz
25. Re:Linux文件系统详解
博主内容很好，希望排版能改善
--疱丁
26. Re:C语言二维数组作为函数的参数
mark!
--shiyideliutang
27. Re:深入理解Linux内存分配
一脸懵逼
--折花刀
28. Re:Linux文件系统详解
博主您好，您的博文您自己看了阅读版面效果吗。 能稍微改进一下，对阅读友好点吗。
--Tudou_Blog
29. Re:c++拷贝构造函数详解
顶
--你的雷哥
30. Re:linux下判断文件和目录是否存在
使用opendir函数时需要包含头文件<dirent.h>
--红城客栈蓝精灵
31. Re:windows下LIB和DLL的区别与使用
谢谢 理解了
--EasyWorld
32. Re:深入理解Linux内存分配
MARK
--小星星的大猩猩
33. Re:malloc的内存分配原理
这个图片都挂了
--WOTGL
34. Re:贪心算法 - 0/1背包问题
@ 笑里藏道对，这是动态规划算法，不是贪心算法...
--FyuNaru
35. Re:c++拷贝构造函数详解
大赞博主
--ssif
36. Re:Linux内存初始化(一)

--xiashaohua2010
37. Re:贪心算法 - 0/1背包问题



1。U与V-U边（v1，v2），（v3，v2），其中
2。(v1,v2)最小，纳入TE
3。v2纳入U。-----完成。



上面基本就把prim算法思想给表达出来。

代码部分：

这里我使用的是邻接矩阵来表示图，其中边的值就是权值。

```
#include<stdio.h>
#include<stdlib.h>
typedef int bool;
typedef char VertexType;
typedef int  EdgeType;
#define false 0
```

这个不是贪心，是记忆化搜索（或者备忘录，别名很多）吧，跟动态规划类似，你的证明过程其实也没有证明贪心选择性质

--笑里藏道

38. Re:理解矩阵乘法

优秀

--Happy_lei

39. Re:AOV网与拓扑排序

写得非常清晰易懂！赞一个~

--坚持的彼岸是星辰大海

40. Re:c++拷贝构造函数详解

函数的返回值是类的对象，这个时候，为什么我运行的结果没有调用赋值构造函数?

--jarven_0728

```
#define true 1
#define MAXVEX 100
#define IFY 65535

VertexType g_init_vexs[MAXVEX] = {'A','B','C','D','E','F','G','H','I'

EdgeType g_init_edges[MAXVEX][MAXVEX] = {
    {0,11,IFY,IFY,IFY,9,IFY,IFY,6},    //'A'
    {11,0,10,IFY,IFY,IFY,IFY,IFY,8},  //'B'
    {IFY,10,0,17,IFY,IFY,IFY,IFY,IFY},//'C'
    {IFY,IFY,17,0,9,IFY,IFY,IFY,IFY},//'D'
    {IFY,IFY,IFY,9,0,7,5,8,IFY},    //'E'
    {9,IFY,IFY,IFY,7,0,3,IFY,IFY},    //'F'
    {IFY,IFY,IFY,IFY,5,3,0,IFY,IFY},    //'G'
    {IFY,IFY,IFY,IFY,8,IFY,IFY,0,IFY},    //'H'
    {6,8,IFY,IFY,IFY,IFY,IFY,IFY,0},    //'I'
};

typedef struct {
    VertexType vexs[MAXVEX];
    EdgeType Mat[MAXVEX][MAXVEX];
    int numVexs,numEdges;
}MGraph;

//打印矩阵,使用二维指针
void prt_maxtix(EdgeType (*p)[MAXVEX],int vexs)
{
    int i,j;
    for(i=0;i<vexs;i++)
    {
        printf("\t");
        for(j=0;j<vexs;j++)
        {
             if( *(*p + j) == IFY)
            {
                printf("  $ ");
            }
            else
            {
                printf(" %2d ", *(*p + j));
            }
        }
         p++;
         printf("\n");
    }
}

//check the number of vextex
int getVexNum(VertexType *vexs)
{
    VertexType *pos = vexs;
    int cnt=0;
    while(*pos <= 'Z' && *pos >= 'A')
    {
        cnt++;
        pos++;
    }
    return cnt;
```

```
}
//检查矩阵是否对称
bool checkMat(EdgeType *p,VertexType numvex)
{
    int i,j;
    for (i=0;i<numvex;i++)
    {
        for(j=i+1;j<numvex;j++)
        {
            //printf("[%d][%d] = %d\t",i,j,*(p + MAXVEX*i + j));
            //printf("[%d][%d] = %d\n",j,i,*(p + MAXVEX*j + i));
            if (*(p + MAXVEX*i + j) != *(p + MAXVEX*j +i) )
            {
                printf("ERROR:Mat[%d][%d] or Mat[%d][%d] not equal!\n
                return false;
            }
        }
    }
    return true;
}
//用已知的一维数组和二维数组分别初始化顶点和边
void init_Grp(MGraph *g,VertexType *v,EdgeType *p)
{
    int i,j;
    // init vex num
    (*g).numVexs = getVexNum(v);

    //init vexter
    for (i=0;i<(*g).numVexs;i++)
    {
        (*g).vexs[i]=*v;
        v++;
    }

    //init Mat
    for (i=0;i<(*g).numVexs;i++)
    {
        for (j=0;j<(*g).numVexs;j++)
        {
            (*g).Mat[i][j] = *(p + MAXVEX*i + j);
        }
    }
    if(checkMat(&((*g).Mat[0][0]),(*g).numVexs) == false)
    {
        printf("init error!\n");
        exit(0);
    }
}

/*void prim(MGraph G,int num)
{
    int sum=0;
    int min,i,j,k;
    int adjvex[MAXVEX];
    int lowcost[MAXVEX];

    lowcost[num] = 0;
    adjvex[num] = 0;
```

```
    for (i = 0; i < G.numVexs;i++ )
    {
        if (num == i)
        {
            continue;
        }
        lowcost[i]=G.Mat[num][i];     //存放起始顶点到各个顶点的权值。
        adjvex[i] = num;
    }

    for (i=0;i<G.numVexs;i++)
    {
        //1.找权最短路径
        //2.把权最短路径的顶点纳入已找到的顶点集合中，重新查看新集合中最短路径
        if(num == i)
        {
            continue;
        }
        min = IFY;
        j=0;k=0;
        while (j<G.numVexs)
        {
            if (lowcost[j] != 0 && lowcost[j] < min)
            {
                min = lowcost[j];
                k = j;

            }
            j++;
        }
        printf(" (%d,%d) --> ",adjvex[k],k);
        sum += G.Mat[adjvex[k]][k];
        lowcost[k]=0;
        for (j=0;j<G.numVexs;j++)
        {

            if (lowcost[j] != 0 && G.Mat[k][j] < lowcost[j])
            {
                lowcost[j] = G.Mat[k][j];
                adjvex[j]=k;
            }
        }

    }
    printf("total:sum=%d\n",sum);
}*/

void Prim(MGraph G,int num)
{
    int sum,i,j,min,k;
    int adjvex[MAXVEX];
    int lowcost[MAXVEX];
    sum=0;
    adjvex[num]=0;
    lowcost[num]=0;
    for(i=0;i<G.numVexs;i++)
    {
```

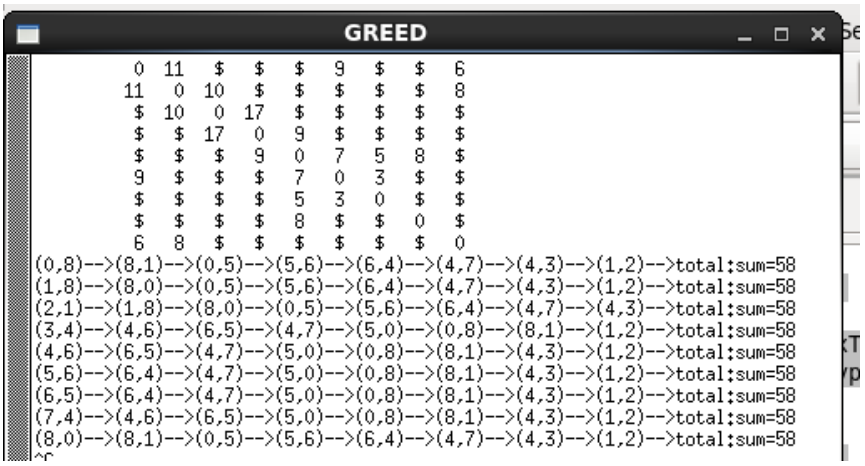
```
            if(i==num)
                continue;
            adjvex[i]=num;
            lowcost[i]=G.Mat[num][i];
        }
        for(i=0;i<G.numVexs;i++)
        {
            if(i==num)
                continue;
            min=IFY;
            k=0;
            //求出代价最小的点
            for(j=0;j<G.numVexs;j++)
            {
                if(lowcost[j]!=0&&lowcost[j]<min)
                {
                    min=lowcost[j];
                    k=j;
                }
            }
            printf("(%d,%d)-->",adjvex[k],k);
            sum+=G.Mat[adjvex[k]][k];
            lowcost[k]=0;
            for(j=0;j<G.numVexs;j++)
            {
                if(lowcost[j]!=0&&G.Mat[k][j]<lowcost[j])
                {
                    lowcost[j]=G.Mat[k][j];
                    adjvex[j]=k;
                }
            }
        }
        printf("total:sum=%d\n",sum);
    }

    int main(int argc, char* argv[])
    {
        MGraph grp;
        //init
        init_Grp(&grp,g_init_vexs,&g_init_edges[0][0]);
        //print Matix
        prt_maxtix(grp.Mat,grp.numVexs);

        //prim(grp,4);
        int i;
        for (i=0;i<grp.numVexs;i++)
        {
            Prim(grp,i);
        }
        //prim(grp,3);


        getchar();
        return 0;
    }
```

运行结果如下：



Jungle Roads

Problem Description

The Head Elder of the tropical island of Lagrishan has a problem. A burst of foreign aid money was spent on extra roads between villages some years ago. But the jungle overtakes roads relentlessly, so the large road network is too expensive to maintain. The Council of Elders must choose to stop maintaining some roads. The map above on the left shows all the roads in use now and the cost in aacms per month to maintain them. Of course there needs to be some way to get between all the villages on maintained roads, even if the route is not as short as before. The Chief Elder would like to tell the Council of Elders what would be the smallest amount they could spend in aacms per month to maintain roads that would connect all the villages. The villages are labeled A through I in the maps above. The map on the right shows the roads that could be maintained most cheaply, for 216 aacms per month. Your task is to write a program that will solve such problems.

The input consists of one to 100 data sets, followed by a final line containing only 0. Each data set starts with a line containing only a number n, which is the number of villages, 1 < n < 27, and the villages are labeled with the first n letters of the alphabet, capitalized. Each data set is completed with n-1 lines that start with village labels in alphabetical order. There is no line for the last village. Each line for a village starts with the village label followed by a number, k, of roads from this village to villages with labels later in the alphabet. If k is greater than 0, the line continues with data for each of the k roads. The data for each road is the village label for the other end of the road followed by the monthly maintenance cost in aacms for the road. Maintenance costs will be positive integers less than 100. All data fields in the row are separated by single blanks. The road network will always allow travel between

all the villages. The network will never have more than 75 roads. No village will have more than 15 roads going to other villages (before or after in the alphabet). In the sample input below, the first data set goes with the map above.

The output is one integer per line for each data set: the minimum cost in aacms per month to maintain a road system that connect all the villages. Caution: A brute force solution that examines every possible set of roads will not finish within the one minute time limit.

Sample Input

```
9 A 2 B 12 I 25 B 3 C 10 H 40 I 8 C 2 D 18 G 55 D 1 E 44 E 2 F
60 G 38 F 0 G 1 H 35 H 1 I 35 3 A 2 B 10 C 40 B 1 C 20 0
```

Sample Output

```
216 30
```

```
//HDOJ1301
#include<iostream>
#include<cstring>
using namespace std;
#define MAX 99999
#define LEN 30
int dist[LEN];//某点的权值  起始点到目标点的权值
int map[LEN][LEN];//某点到某点两点之间的权值
bool isvisitd[LEN];//表示某点是否访问过

//初始化map数组  设置为无穷大
void  init(int n){
    for(int  i = 0;i<=n;i++) {
         for(int  j = 0;j<=n;j++)
            map[i][j] = MAX;
     }
}

//prim最小生成树的算法
int prim(int n){
    int i,j,min,pos,sum;
    sum = 0;   //最小生成树的权值

    //初始化，表示没有一点走过
    memset(isvisitd,false,sizeof(isvisitd));

    //初始化给dist数组赋值
    for(i = 1;i<=n;i++){
        dist[i] = map[1][i];
    }


    isvisitd[1] = true;//标记1已被访问  从1开始

    //找到权值最小点并记下位置
```

```
        //找到权值最小点并记下位置
        for(i = 1;i<n;i++){
                min = MAX;
                for(j = 1;j<=n;j++){
                if(!isvisitd[j]&&dist[j]<min){
                    min = dist[j];
                    pos = j;//记录下该位置
                }
            }

            sum+=min;
            isvisitd[pos] = true;

            //更新权值
            for(j = 1;j<=n;j++)     {
                if(!isvisitd[j]&&dist[j]>map[pos][j]){
                    dist[j] = map[pos][j];
                }
            }

        }
        return sum;
    }
    int main(){
        int i,j,n,m,len;
        char start,end;
        while(scanf("%d",&n)!=EOF){
            if(n==0){
                break;
            }

            init(n);//初始化

            for(i=0;i<n-1;i++){
                cin>>start>>m;
                for( j = 0;j<m;j++){
                    cin>>end>>len;
                    map[i+1][end-'A'+1] = len;
                    map[end-'A'+1][i+1] = len;
                }
            }
            cout<<prim(n)<<endl;
        }
        return 0;

    }
```

运行结果如下：

```
[root@localhost GREED]# vim acm.cpp
[root@localhost GREED]# g++ -o acm acm.cpp -std=c++11
[root@localhost GREED]# ./acm
9
A 2 B 12 I25
^C
[root@localhost GREED]# ./acm
9
A 2 B 12 I 25
B 3 C 10 H 40 I 8
C 2 D 18 G 55
D 1 E 44
E 2 F 60 G 38
F 0
G 1 H 35
H 1 I 35
216
```

最小生成树 **Prim**算法的实现及应用

关于**prim**算法

先把有的点放于一个集合（或者数组）里，这个集合里存放的是所有走过的点。初始值为0或者false表示还没有点

声明一个一维数组用于记录各点的权值[可理解为起始点到目标点的距离]，

声明一个二维数组用于记录某点到某一点的权值，如果这两点不可达到，则设置为无穷大

具体执行过程：

先从某一点开始，把这一个开始的点放于声明的一个数组或者集合里，表明这一点已经被访问过。然后再从余下的n-1个点里去找那个权值最小的点并记录该点的位置然后再加上这一点的权值，同时将该点放于集合里表明该点已初访问。再更新权值

再看下图，从下图分析：

1、

先选取一个点作起始点，然后选择它邻近的权值最小的点（如果有多个与其相连的相同最小权值的点，随便选取一个）。如1作为起点。

```
isvisited[1]=1;    //表明把1加进来说明是已经访问过

pos=1; //记录该位置

//用dist[]数组不断刷新最小权值，dist[i](0<i<=点数)的值为：i点到邻近点（未被标记）的最小距离。

dist[1]=0;  //起始点i到邻近点的最小距离为0

dist[2]=map[pos][2]=4;

dist[3]=map[pos][3]=2;

dist[4]==map[pos][4]=3;

dist[5]=map[pos][5]=MaxInt;  //无法直达

dist[6]=map[pos][6]=MaxInt;
```

2、

再在伸延的点找与它邻近的两者权值最小的点。

//dist[]以3作当前位置进行更新

isvisited[3]=1;

pos=3;

dist[1]=0; //已标记，不更新

dist[2]=map[pos][2]=4; //比5小，不更新

dist[3]=2; //已标记，不更新

dist[4]=map[pos][4]=3; //比1大，更新

dist[5]=map[pos][5]=MaxInt;

dist[6]=map[pos][6]=MaxInt;

3、最后的结果：

当所有点都连同后，结果最生成树如上图所示。

所有权值相加就是最小生成树，其值为2+1+2+4+3=12。

prim算法的实现：

```
//prim算法

int prim(int n){

        int i,j,min,pos;

        int sum=0;

        memset(isvisited,false,sizeof(isvisited));


        //初始化

        for(i=1;i<=n;i++){

                dist[i]=map[1][i];

        }


        //从1开始

        isvisited[1]=true;

        dist[1]=MAX;


        //找到权值最小点并记录下位置
```

```
32.
33.         for(i=1;i<n;i++){
34.
35.                 min=MAX;
36.
37.                 //pos=-1;
38.
39.                 for(j=1;j<=n;j++){
40.
41.                         if(!isvisited[j] && dist[j]<min){
42.
43.                                 min=dist[j];
44.
45.                                 pos=j;
46.
47.                         }
48.
49.                 }
50.
51.                 sum+=dist[pos];//加上权值
52.
53.                 isvisited[pos]=true;
54.
55.
56.
57.                 //更新权值
58.
59.                 for(j=1;j<=n;j++){
60.
61.                         if(!isvisited[j] && dist[j]>map[pos][j])
    {
62.
63.                                 dist[j]=map[pos][j];
64.
65.                         }
66.
67.                 }
68.
69.         }
70.
71.         return sum;
72.
73. }
74.
75.
```

算法的应用：

应用上面的这个模板基本上能解决一些常见的最小生成树的算法，例如像杭电上的ACM题目：

题目链接地址：

HDOJ1863：http://acm.hdu.edu.cn/showproblem.php?pid=1863

HDOJ1875：http://acm.hdu.edu.cn/showproblem.php?pid=1875

HDOJ1879：http://acm.hdu.edu.cn/showproblem.php?pid=1879

HDOJ1233：http://acm.hdu.edu.cn/showproblem.php?pid=1233

HDOJ1162：http://acm.hdu.edu.cn/showproblem.php?pid=1162

HDOJ1301：http://acm.hdu.edu.cn/showproblem.php?pid=1301