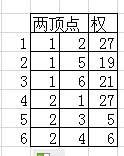
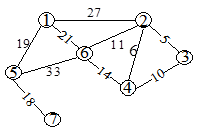
图论算法与STL

1. 最小生成树

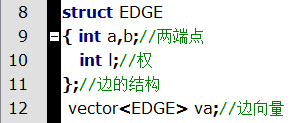
*使用vector 存储边结点，并且按边的长度排序。*

1．图的表示方法：边数组。

如下图，一般情况下可以用一个二维数组表示一个有权的无向图，表示有向图当然也可以的。



使用vector存储边时，设计包含两个顶点和权值结构体。



例一：【codevs.cn 1078】



假设要在n个居民点铺设煤气管道，最多用n(n-1)条管道，最少需（n-1）条，另外还需考虑每一条管道的费用。如何从这n(n-1)条管道中优先出（n-1）条路线，构成煤气管道网络，既能边能n个居民点，又能使花费最少。

算法：边数组存储各边。

1)按边的权从小到大排序

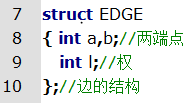
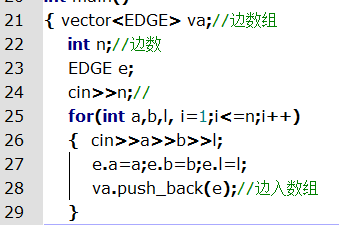
2)考虑权最小的边，若不构成环则选用（使用并查集），

否则丢弃

3)返回2）直到先了n-1条边

4)输出选用边权值之和。完成

图的存储

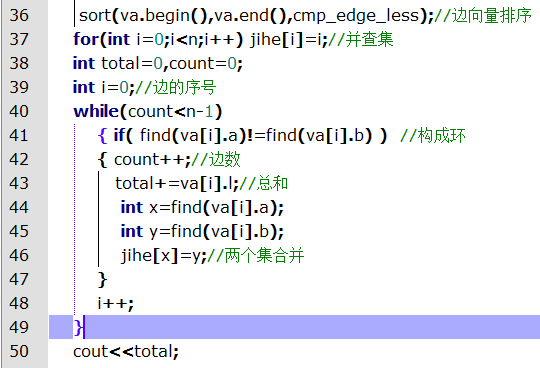
边排序：

因为被排序的对象是结构体，我们必须告诉SORT函数，如何比较大小。cmp\_edge\_less()就是这样的函数。



也就是当第一个参数的值较小时，返回真。实现递增排序。

以下程序片段，完成选边和统计工作，使用并查集。



代码：

#include<iostream>

#include<vector>

#include<string>

#include<algorithm>

#include<iterator>

#define max\_n 101

**using** **namespace** std**;**

struct EDGE

**{** int a**,**b**;**//两端点

int l**;**//权

**};**//边的结构

vector**<**EDGE**>** va**;**//边向量

bool cmp\_edge\_less**(**const EDGE**&** a**,**const EDGE**&** b**)**

**{** **return** a**.**l**<**b**.**l **;** **}**

int jihe**[**max\_n**];**

int find**(**int i**)**

**{** **if(**jihe**[**i**]==**i**)** **return** i**;**

**{** int k**=**find**(**jihe**[**i**]);**

jihe**[**i**]=**k**;**

**return** k**;**

**}**

**}**

int main**()**

**{**

int n**;**//边数

EDGE e**;**

cin**>>**n**;**//

**for(**int i**=**0**;**i**<**n**;**i**++)**

**for(**int j**=**0**;**j**<**n**;**j**++)**

**{** int l**;**

cin**>>**l**;**

**if(**i**!=**j**){**

e**.**a**=**i**;**e**.**b**=**j**;**e**.**l**=**l**;**

va**.**push\_back**(**e**);**//边入向量

**}**

**}**

sort**(**va**.**begin**(),**va**.**end**(),**cmp\_edge\_less**);**//边向量排序

**for(**int i**=**0**;**i**<**n**;**i**++)** jihe**[**i**]=**i**;**//并查集

int total**=**0**,**count**=**0**;**

int i**=**0**;**//边的序号

**while(**count**<**n**-**1**)**

**{** **if(** find**(**va**[**i**].**a**)!=**find**(**va**[**i**].**b**)** **)** //构成环

**{** count**++;**//边数

total**+=**va**[**i**].**l**;**//总和

int x**=**find**(**va**[**i**].**a**);**

int y**=**find**(**va**[**i**].**b**);**

jihe**[**x**]=**y**;**//两个集合并

**}**

i**++;**

**}**

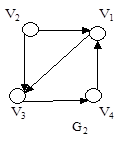
cout**<<**total**;**

**}**

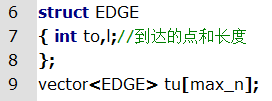
1. 最短路的DJ算法

*使用vector数组来存储图的邻接表*

如下图，一般情况下可以用二维数组来存储邻接表。



使用二维的vector<EDGE>来存储邻接表，如下图



向量tu[i]存储编号为i的点所能到达的点to和距离dis。

另外，算法在要找距离值最小的点的编号，因此设计了一个结构体，包括点的编号和它的距离，这个结构体将被放入一个优先队列中。如下图

**例：[codevs 1079]**现在是晚餐时间,而母牛们在外面分散的牧场中。 农民约翰按响了电铃,所以她们开始向谷仓走去。 你的工作是要指出哪只母牛会最先到达谷仓(在给出的测试数据中,总会有且只有一只最快的母牛)。 在挤奶的时候(晚餐前),每只母牛都在她自己的牧场上,一些牧场上可能没有母牛。 每个牧场由一条条道路和一个或多个牧场连接(可能包括自己)。 有时，两个牧场(可能是字母相同的)之间会有超过一条道路相连。 至少有一个牧场和谷仓之间有道路连接。 因此,所有的母牛最后都能到达谷仓,并且母牛总是走最短的路径。 当然,母牛能向着任意一方向前进,并且她们以相同的速度前进。 牧场被标记为'a'..'z'和'A'..'Y',在用大写字母表示的牧场中有一只母牛,小写字母中则没有。 谷仓的标记是'Z',注意没有母牛在谷仓中。

注意'm'和'M'不是同一个牧场否则错误上面的意思是说：输入数据中可能会同时存在M,m（郁闷ing)，比如M a a m m z

**输入：**第 1 行: 整数 P(1<= P<=10000),表示连接牧场(谷仓)的道路的数目。

第 2 ..P+1行: 用空格分开的两个字母和一个整数:

被道路连接牧场的标记和道路的长度(1<=长度<=1000)。

**输出：**单独的一行包含二个项目:最先到达谷仓的母牛所在的牧场的标记,和这只母牛走过的路径的长度。

算法：

1）各点距离值设极大值，各点设为未完成点

2）起点距离值设为0

3）从未完成点找距离值最小的点start（使用优先队列）

4）对于与start相接的所有未完成点p，

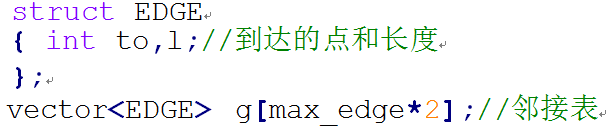
若: p的距离值 > start的距离值+ start则修改p的距离值

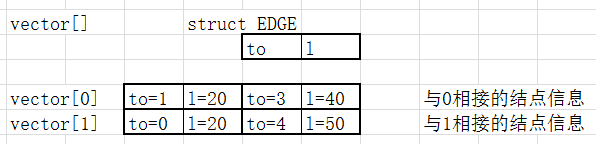
则更新p的距离值为上述值（该点入优先队列）

5）将start设为完成点

6）返回3），直到所有点都完成。

数据存储：



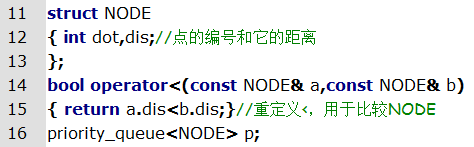


邻接表g[]：如g[1]是一个vector，存储与编号1的点相接的点。

例如：g[1][0]就是一个EDGE类型的结点，g[1][0].to与1相边的结点编号，g[1][0].l就是边长。

记录各点最小值的数组 dis[i]

为了方便找到距离最小值的编号，将点的编号与最小距离值做在一个结构体里，使用优先队列。



使用优先队列找到最小距离值的编号。当修改了一个点的距离值之后，将该点的NODE(包含点编号和距离)入优先队列。从优先队列的顶可以取到最小距离值的NODE。

这个算法过程可以描述成：

1）设置起点的距离值0，起点的NODE入优先队列

2）从优先队列的TOP取NODE，

修改与之相连的未完成点的距离值

被修改距离值的点NODE入优先队列

3）返回2），直到队列空。

4）完成

#include<iostream>

#include<vector>

#include<algorithm>

#include<queue>

#include<limits.h>

#define max\_dot 60

#define max\_edge 10001

**using** **namespace** std**;**

struct EDGE

**{** int to**,**l**;**//到达的点和长度

**};**

vector**<**EDGE**>** g**[**max\_edge**\***2**];**//邻接表

struct NODE

**{** int dot**,**dis**;**//点的编号和它的距离

**};**

bool **operator<(**const NODE**&** a**,**const NODE**&** b**)**

**{** **return** a**.**dis**<**b**.**dis**;}**//重定义<，用于比较NODE

priority\_queue**<**NODE**>** p**;**

NODE dots**[**max\_dot**];**

int dot\_dis**[**max\_dot**];**//记录每点的dis值

bool finish**[**max\_dot**];**//记录每点是否完成

int edges**;**//边数

void datainput**()**

**{** cin**>>**edges**;**

**for(**int i**=**1**;**i**<=**edges**;**i**++)**

**{** char x**,**y**;**int l**;**

cin**>>**x**>>**y**>>**l**;**

int a**=**int**(**x**),**b**=**int**(**y**);**

a**=**a**>**96**?** a**-**71**:**a**-**65**;**

b**=**b**>**96**?** b**-**71**:**b**-**65**;**

EDGE tmp**;**

tmp**.**to**=**b**;**tmp**.**l**=**l**;**

g**[**a**].**push\_back**(**tmp**);**//加入边

tmp**.**to**=**a**;**tmp**.**l**=**l**;**

g**[**b**].**push\_back**(**tmp**);**//无向图

**}**

**}**

void dj**()**

**{** NODE T**;**int start**,**dis**;**

**while(**p**.**size**()>**0**)**

**{**

T**=**p**.**top**();**//取优先队列的顶元素

p**.**pop**();**//顶元素出队

start**=**T**.**dot**;**dis**=**T**.**dis**;**

**if(**finish**[**start**])** **break;**//已完成点跳过

**for(**int i**=**0**;**i**<**g**[**start**].**size**();**i**++)**

**{**

int to**=**g**[**start**][**i**].**to**;**

**if(**finish**[**to**])** **continue;**//已完成点跳过

int l**=**g**[**start**][**i**].**l**;**

**if(**dot\_dis**[**start**]+**l**<**dot\_dis**[**to**])**

**{** dot\_dis**[**to**]=**dot\_dis**[**start**]+**l**;**

NODE T2**;**

T2**.**dot**=**to**;**T2**.**dis**=**dot\_dis**[**to**];**

p**.**push**(**T2**);**

**}**

**}**

finish**[**start**]=**1**;**

**}**

**}**

int main**()**

**{** datainput**();**

**for(**int i**=**0**;**i**<**max\_dot**;**i**++)** dot\_dis**[**i**]=**INT\_MAX**;**

dot\_dis**[**25**]=**0**;**//初始化各点距离值

**for(**int i**=**0**;**i**<**max\_dot**;**i**++)** finish**[**i**]=**0**;**

NODE tmp**;**

tmp**.**dis**=**0**;**tmp**.**dot**=**25**;**//起点入队列

p**.**push**(**tmp**);**

dj**();**

int min**=**dot\_dis**[**0**];**

int ans**=**0**;**

**for(**int i**=**1**;**i**<**25**;**i**++)**

**if(** min**>**dot\_dis**[**i**]** **){**ans**=**i**;**min**=**dot\_dis**[**i**];}**

ans**+=**65**;**

cout**<<**char**(**ans**)<<**" "**<<**min**;**

**return** 0**;**

**}**