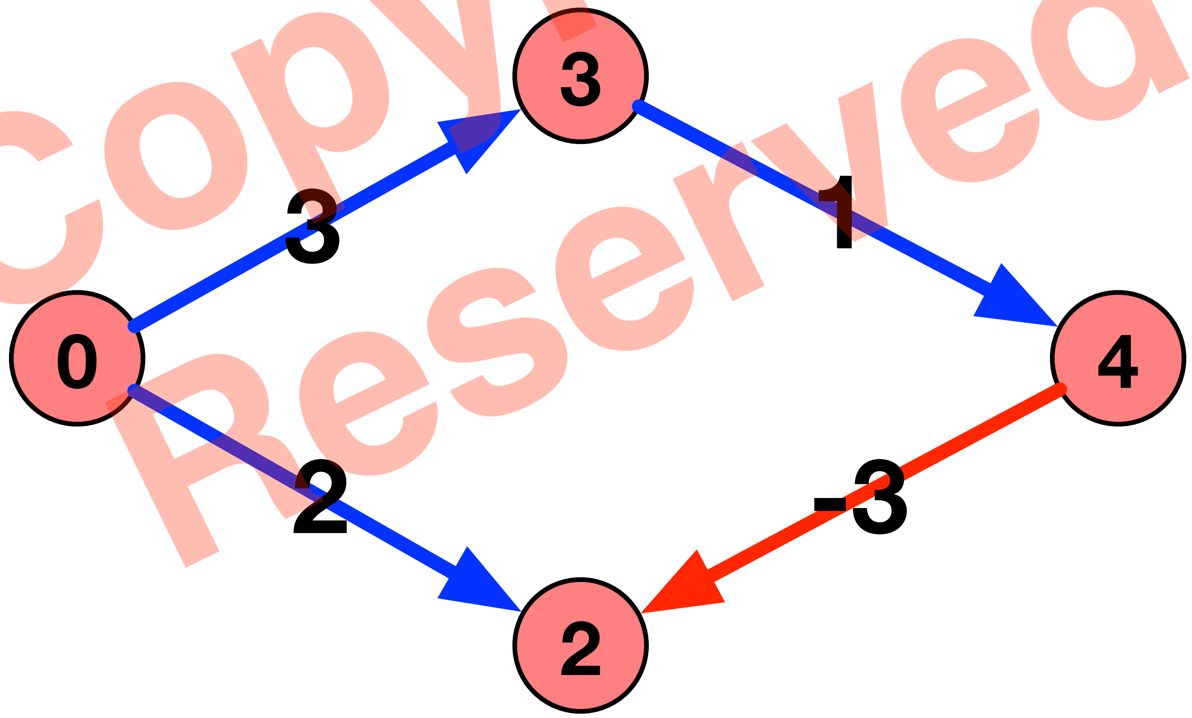
作业回顾

1.dijkstra算法反例

点内是d的数值

红色的边未被松弛过，导致下面顶点最短路不对



2.请柬分发：建模

→ **有向图，求1出发和到达1的最短路**

到达1的只要将所有的边反向即可，所以就是**两次单源最短路**

→ **使用dijkstra或spfa算法均可（最短路实现我就不贴了）**

dijkstra需要使用优先队列优化（见后）

```
int main() {
    scanf("%d%d",&n,&m);
    for (int i=0;i<m;i++) { // 建图
        scanf("%d%d%d",&es[i].s,&es[i].t,&es[i].w);
        g[es[i].s].push_back(i);
    }
    LL ans=spfa();  // 单源最短路

    for (int i=1;i<=n;i++)  g[i].clear();
    for (int i=0;i<m;i++) { // 反向边并重新建图
        swap(es[i].s,es[i].t);
        g[es[i].s].push_back(i);
    }
    ans+=spfa();  // 单源最短路
    printf("%lld\n",ans);
    return 0;
}
```

→ **复杂度：O(m)**

多源最短路径：Floyd算法

- → 多源最短路是指求**任意两个节点之间**的最短路径
- → DP

  状态：f(i,j,k)从i到j，且**中间节点id不超过k**的最短路径长度

  转移方程：

$$f(i,j,k) = min\{f(i,j,k-1), f(i,k,k-1) + f(k,j,k-1)\}$$

  边界：f(i,j,0)=w(i,j)

```
int n,m;
int map[N][N];
int main(){
    memset(map,0x3f,sizeof(map));
    cin >> n >> m;
    for(int i=1;i<=m;i++){
        int x,y,z;
        cin >> x >> y >> z;
        map[x][y]=min(map[x][y],z);
    }
    for(int k=1;k<=n;k++)
        for(int i=1;i<=n;i++)
            for(int j=1;j<=n;j++)
                map[i][j]=min(map[i][k]+map[k][j],map[i][j]);
```

→ 复杂度：O(n^3)（空间可以使用滚动数组）

4.最短路计数：建模

→ **在松弛操作过程中同时维护最短路的距离和计数（最省事）**

```
设一个集合X，表示已经计算出最短路径的顶点（初始为空）
设一个数组d，用来记录当前某个节点的最短路（初始d[0]=0，其他为∞）
设一个数组f，用来记录当前某个节点的最短路数量（初始f[0]=1，其他为0）
for (int i=0;i<n;i++) {
  选择不在X中且d最小的顶点s，s加入X
  对所有s出发的边（s,t）{
    if (t不在X中 && d[s]+w<d[t]) { // w为边长
      d[t]=d[s]+w  // 松弛操作
      f[t]=f[s]  // 重新计数
    } else if (t不在X中 && d[s]+w=d[t]) {
      f[t]+=f[s]  // 累加计数
    }
  }
}
输出f
```

4.最短路计数：建图

```
const int N=1000010,M=2000010,MOD=100003;

struct edge {   // struct链表
    int to,val,next;    // next是后向指针
} G[M];
int head[N],tot=0;    // head与G构成邻接表

int f[N];    // 最短路计数
struct node{    // 顶点类型
    int i,di;
};
bool operator <(node a,node b){
    return a.di>b.di;
}

void add_oriented_edge(int start,int end,int weight){ // 邻接表增加有向边
    G[tot].to=end;
    G[tot].val=weight;
    G[tot].next=head[start];
    head[start]=tot++;
}
void add_edge(int node1,int node2,int weight){ // 邻接表增加无向边
    add_oriented_edge(node1,node2,weight);
    add_oriented_edge(node2,node1,weight);
}
```

→ 使用了struct数组方式实现的邻接表

4.最短路计数：解答

```
int s=0;
double dmin=inf+1;
for (int i=1;i<=N;i++){
    if (d[i]<dmin && !visited[i]) {
        s=i; dmin=d[i];
    }
}
```

```
void dijkstra(int o){    // dijkstra+优先队列
    int dis[N];
    memset(dis,0x7f,sizeof(dis));    // 填充INF
    memset(f,0,sizeof(f));  // 填充0

    priority_queue<node>q;
    q.push((node){st,0});

    dis[o]=0; f[o]=1;
    for(;!q.empty();){
        node now=q.top(); q.pop();     // 这里用堆优化选择dis最小的
        if(dis[now.i]!=now.di) continue;     // 这句实现了visited功能（想一想为什么）
        for(int i=head[now.i];i!=-1;i=G[i].next){    // 遍历出边
            int v=G[i].to;
            if(now.di+G[i].val==dis[v]){ //如果通过tmp点访问v点刚好是最短路，累加路径条数
                f[v]=(f[now.i]+f[v])%MOD;
            }
            if(now.di+G[i].val<dis[v]){ //如果更新了tmp点到v点的最短路，之前的不算，重新计数
                dis[v]=now.di+G[i].val;
                f[v]=f[now.i];
                q.push((node){v,dis[v]});
            }
        }
    }
}
```

→ **使用了优先队列获取dis最小的节点）**

注意：同一顶点可能多次进入优先队列

→ **复杂度：O(MlogN)**

→ **算出最短路径后，为什么不用矩阵快速幂统计数量?**

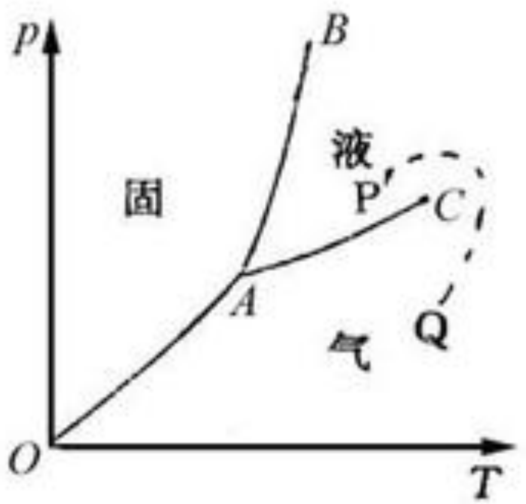
$O(N^3\log N)$，矩阵乘法本身太慢了

→ **如果权值不是都为1，本算法仍然适用**

CS102 算法进阶

生成树

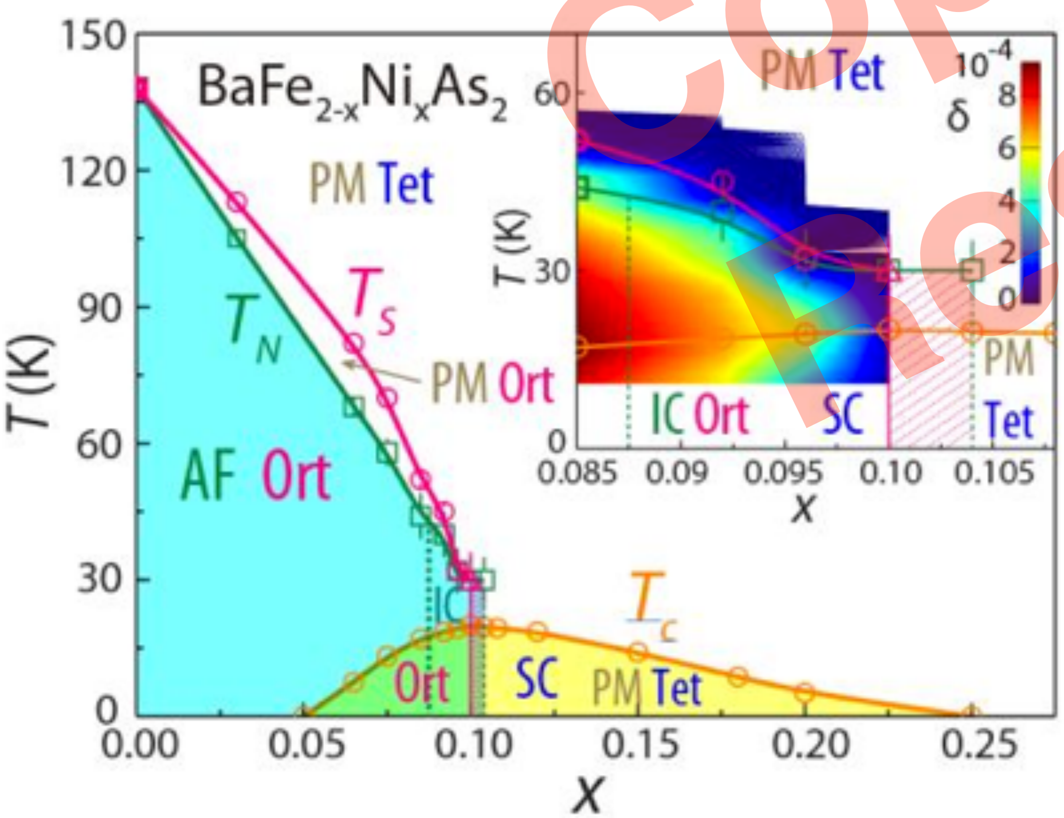
先讲一点科学：相变



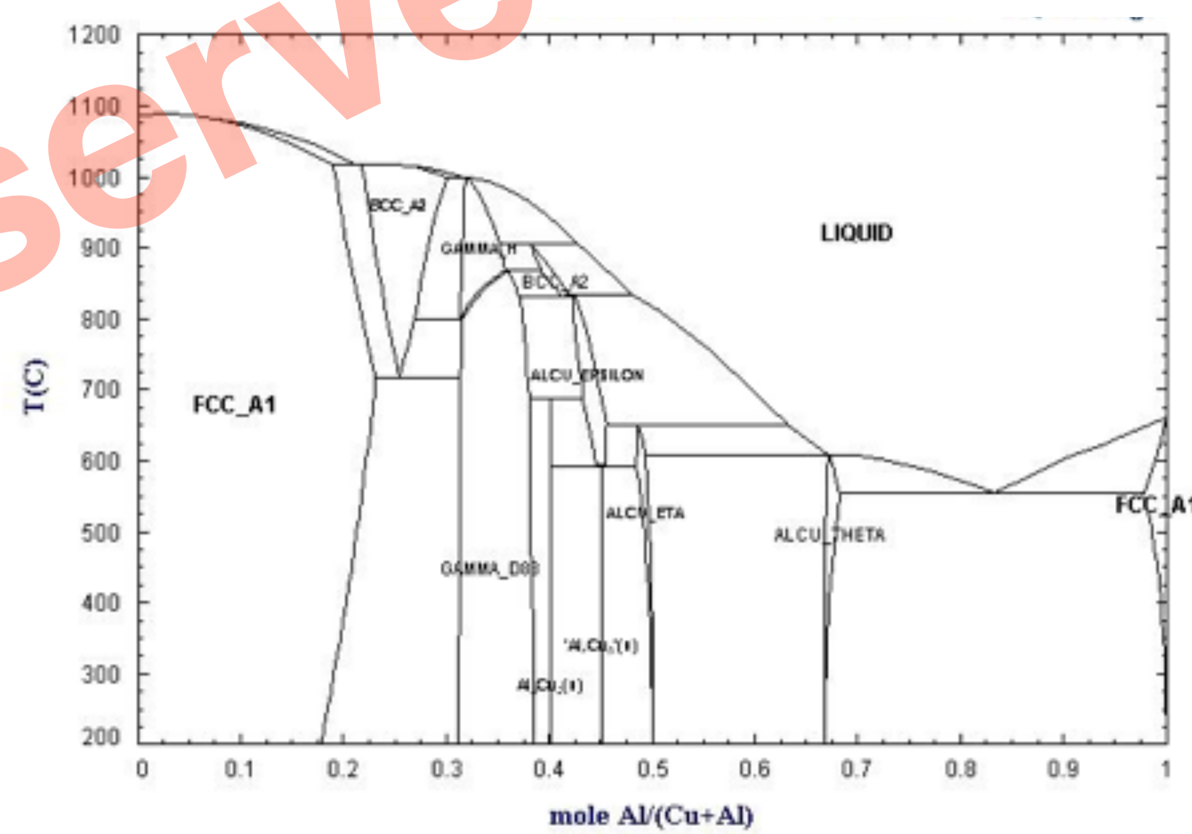
→ 相(Phase)指物质物理/化学性质相同的区域
如水有气、液、固三相

→ 此处“区域”不是指空间上，而是某些物理属性张成的抽象空间
有点类似我们说的“状态空间”

→ 理论物理对相变附近的行为特别感兴趣



高温超导体相图



铜铝合金相图

最短网络

lester被选为他们镇的镇长，他其中一个竞选承诺就是在镇上建立起互联网，并连接到所有的农场。当然，他需要你的帮助。lester已经给他的农场安排了一条高速的网络线路，他想把这条线路共享给其他农场。为了用最小的消费，他想铺设最短的光纤去连接所有的农场

你将得到一份各农场之间连接费用的列表，你必须找出能连接所有农场并所用光纤最短的方案。每两个农场间的距离不会超过100000

输入输出格式：

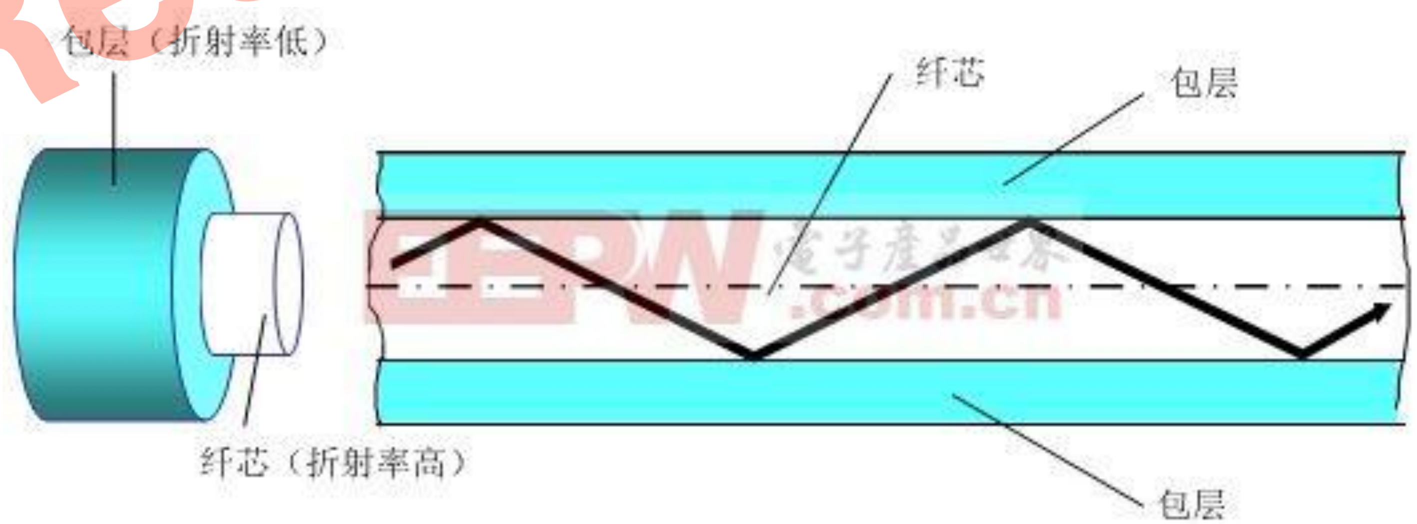
输入第一行：农场的个数N（3<=N<=100）

后N行包含了一个N*N的矩阵,表示每个农场之间的距离

输出一个数，连接到每个农场的光纤的最小长度

样例输入：

4
0 4 9 21
4 0 8 17
9 8 0 16
21 17 16 0



样例输出：

28

最短网络：建模

→ **建图没什么悬念，输入就是相邻矩阵**

→ **最优化问题**

状态空间：无向图上**边的子集**全体

限制条件：连接所有顶点（连通性）

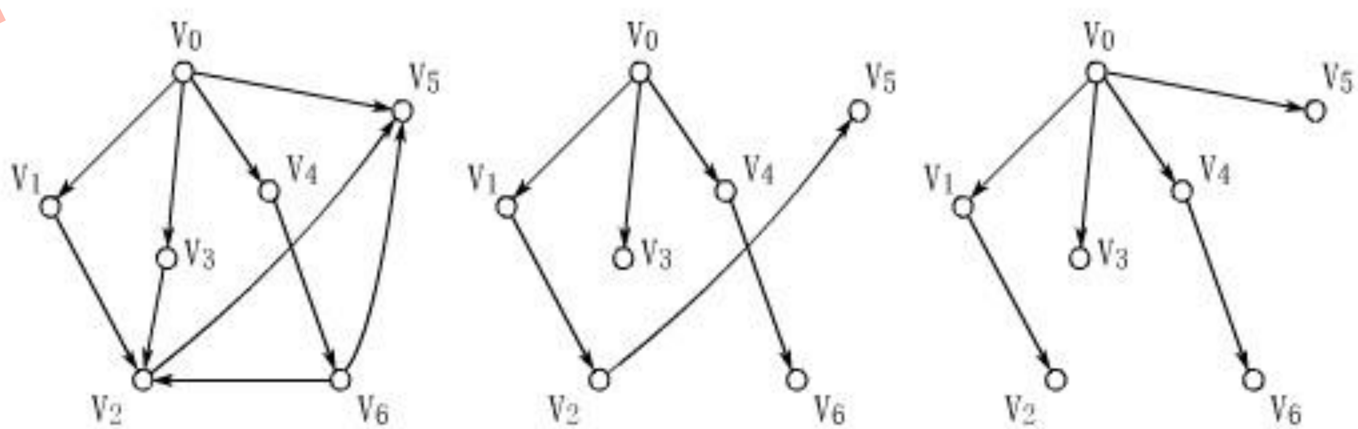
最优化条件：总权值最小

→ **一些初步推理**

1.一定没有回路，否则随便去掉回路上的一条边，仍符合条件

2.连通性（当然前提是图本身连通）

→ **没有回路+连通性=树**



最小生成树

→ **连通的无向图G=(V,E)中一个边的子集T，满足**

1.T覆盖到所有顶点

2.T是连通的

3.T中没有回路

→ **则T称为图G的一个生成树（Spanning Tree）**

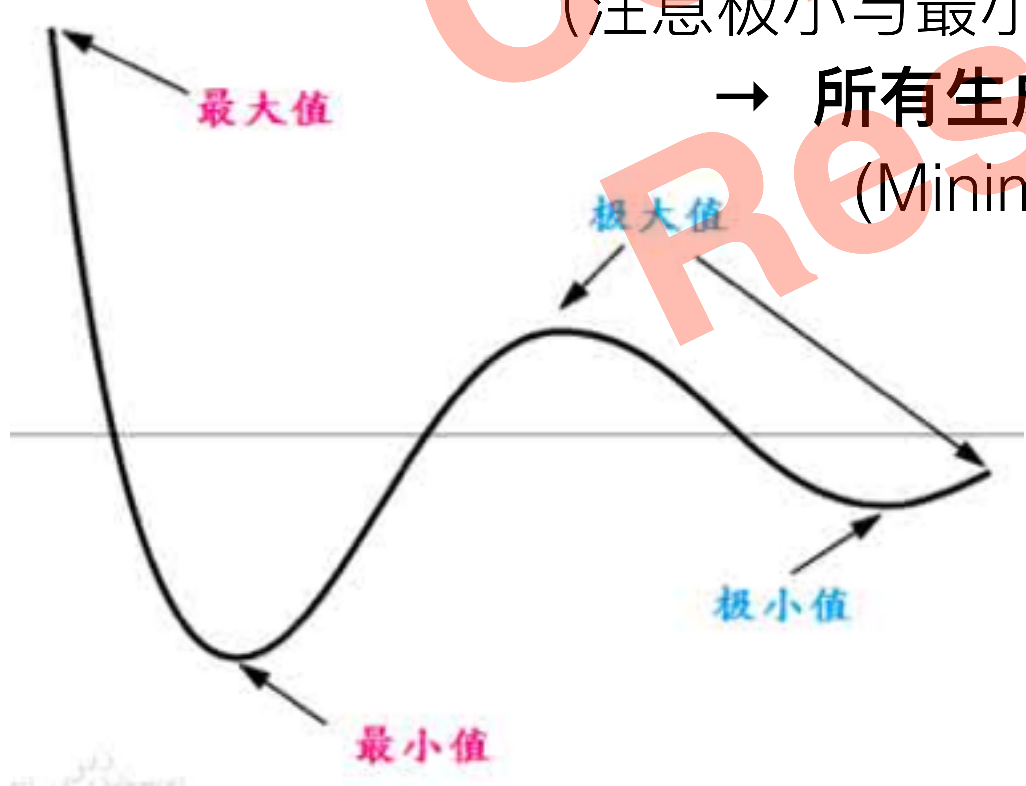
→ **生成树可以理解为使G保持连通的极小的边的集合**

极小指**去掉任何一条边都会破坏规定的特性**

（注意极小与最小不是同一个概念）

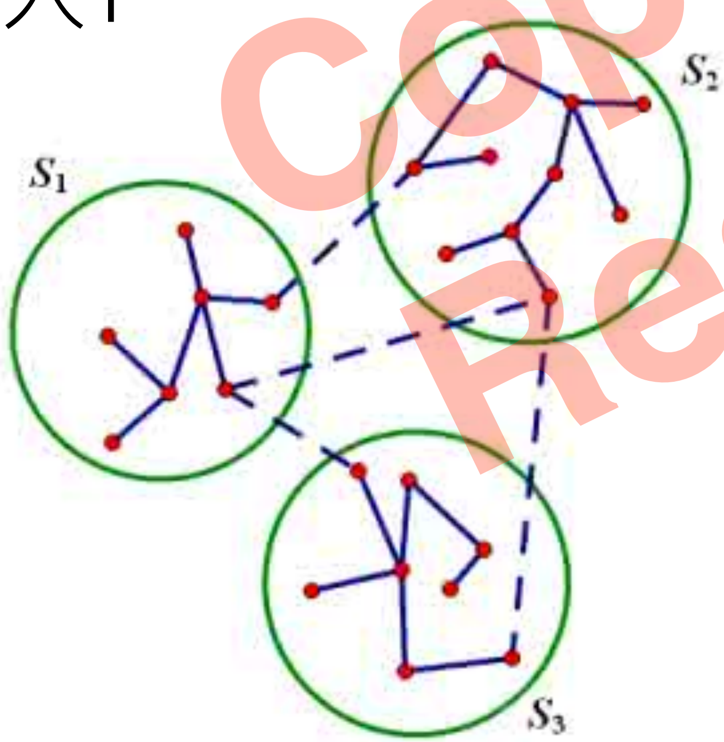
→ **所有生成树中边权总和最小的称为G的最小生成树**

(Minimum Spanning Tree, MST)



最小生成树：Kruskal算法

初始T为空
for (int i=1;i<=n-1;i++) {
  选择满足如下条件的权值最小的边e=(u,v)
    u,v分属于**T的两个不同连通分支**
  e加入T
}



→ **从连通性入手**

开始没有任何边

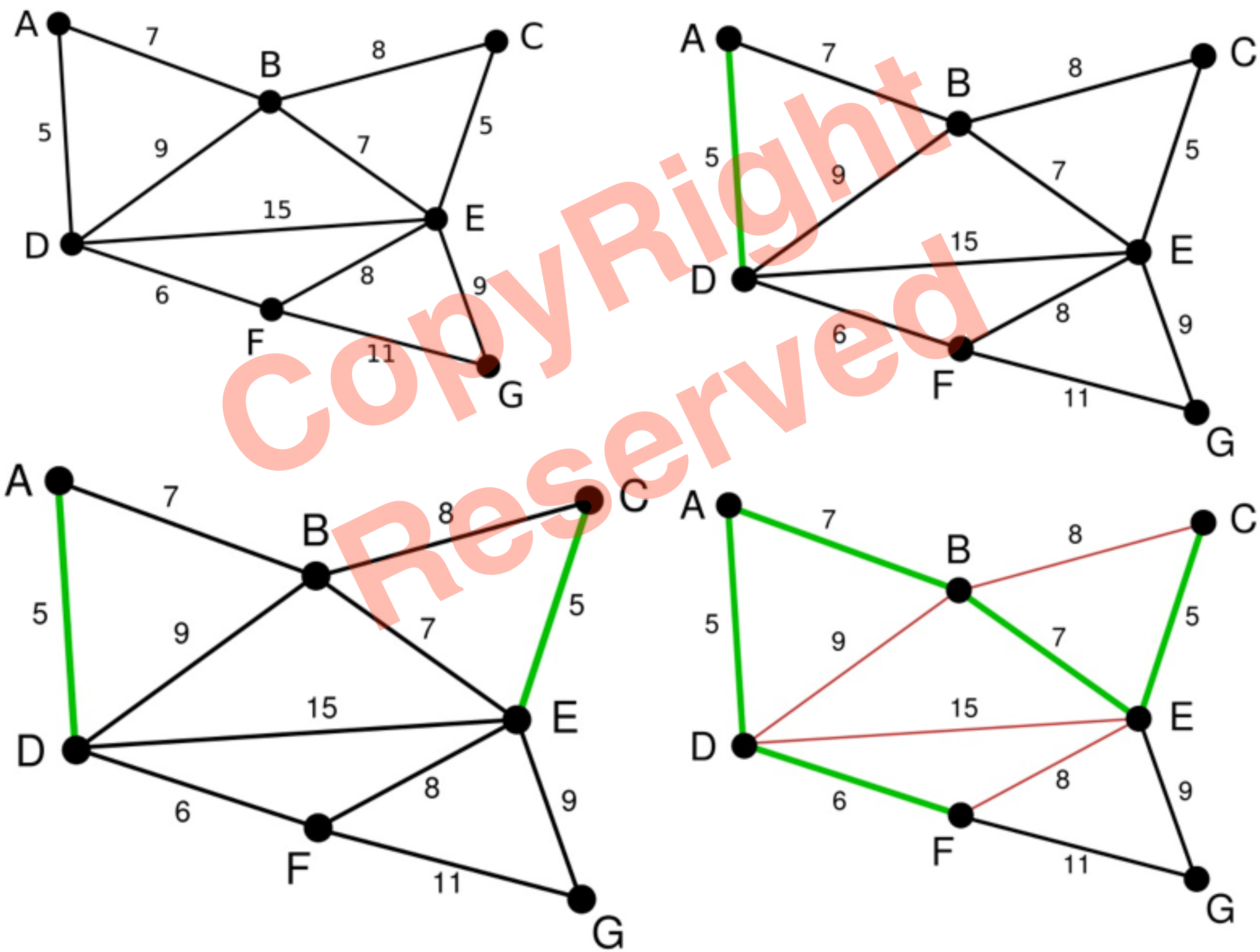
依次找n-1条边，把n个顶点连接起来

整个过程就是**连通分支减少的过程**

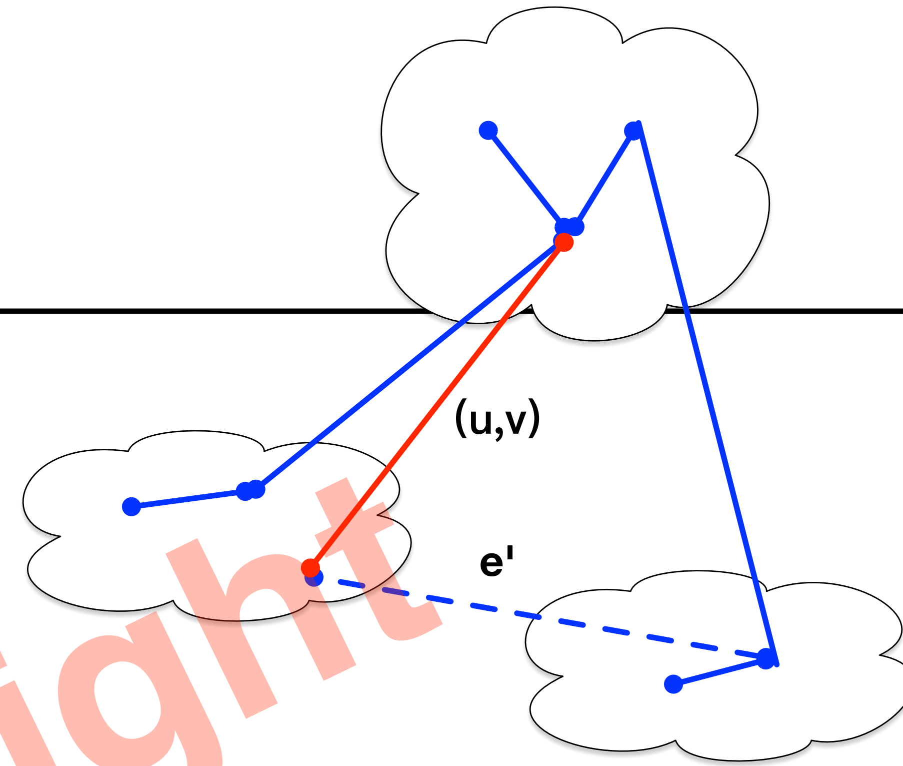
→ **贪心的猜想**

每次都找最短的边（恭喜你猜对了）

Kruskal算法：演示



Kruskal算法：证明



对顶点数**归纳法**

1.n=1时，trivial

2.n>1时

先证明**选出的边e=(u,v)一定在某个MST中**

假设e∉MST，将e强制添加到MST，则会形成一个环R

∃边e'∈R && e'跨当前T的不同连通分支

根据e的选择方式，w(e')≥w(e)

从MST中删除e'，增加形成一个新的生成树T'

w(T')=w(MST)+w(e)-w(e')≤w(MST)，则T'也是最小生成树

将u,v**收缩成一个顶点**，形成一个新图G'，顶点数为n-1

∀G的生成树T，T\e一定是G'的生成树

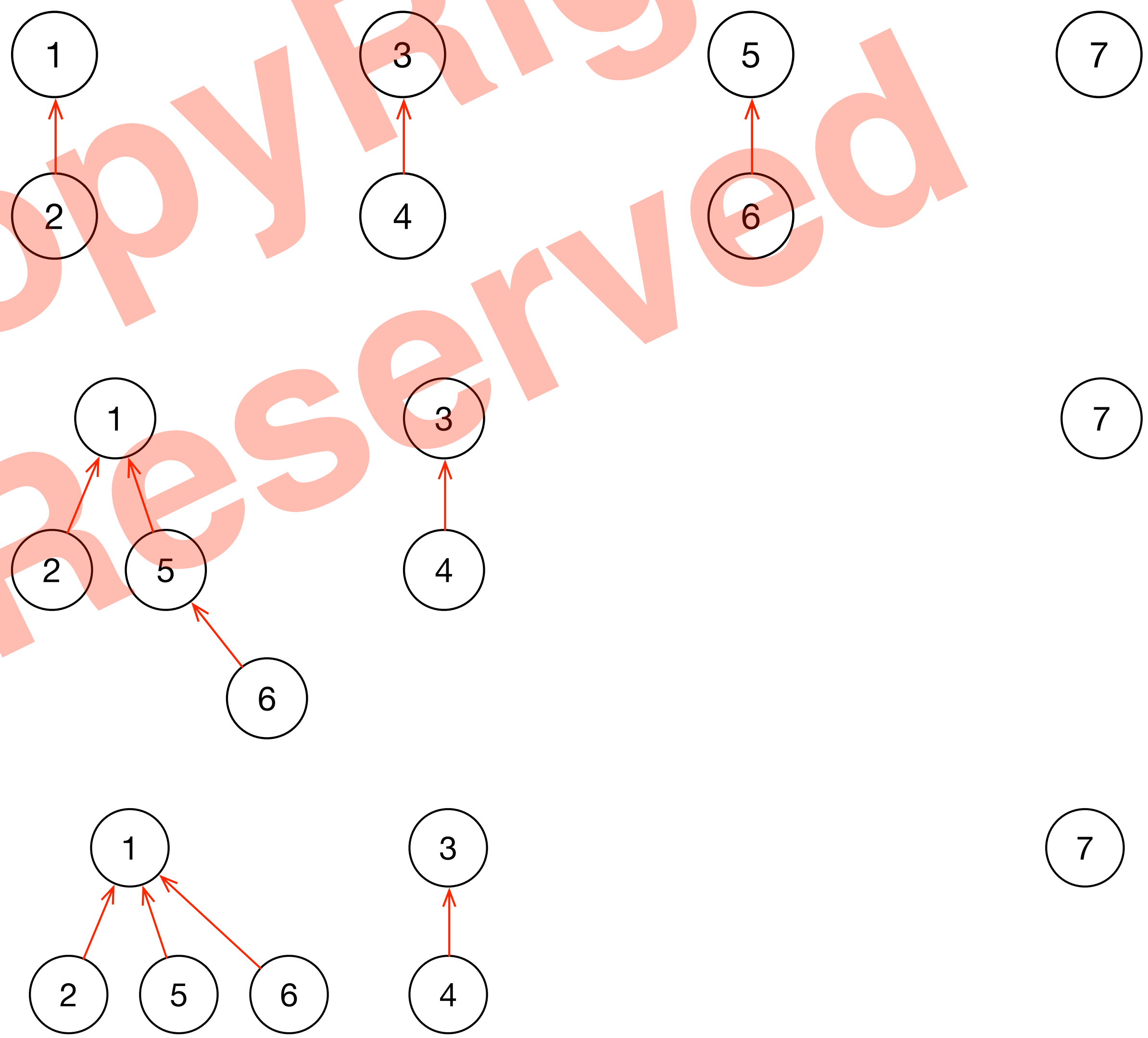
根据归纳法，证毕

合并连通分支：并查集

→ **需要动态合并连通分支以及判断两个点是否在同一连通分支**

CS100时提过一种数据结构专门处理这个问题：**并查集**

忘记的去看CS100第四周讲义

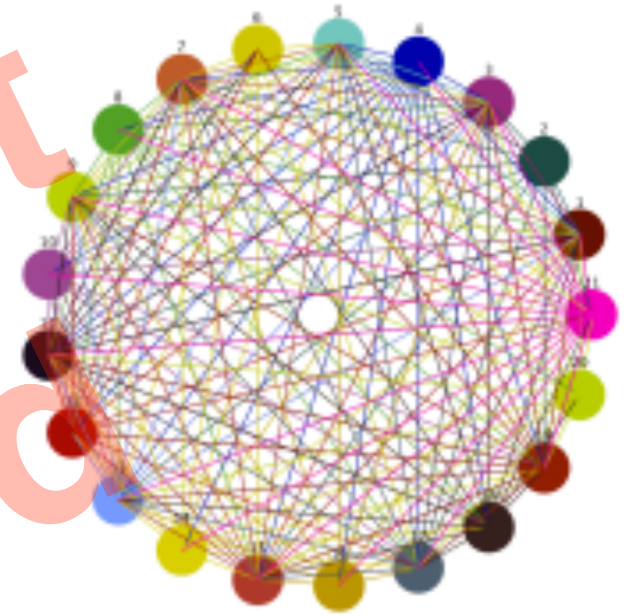


最短网络：解答-并查集

```
#define N 105

using namespace std;

int n,p[N],tot=0,m=0,x;    // p为顶点连通性的（并查集）
struct edge {    // 边类型
    int s,t,w;
}e[N*N];

int cmp(const edge &e1,const edge &e2) {
    return e1.w<e2.w;
}

int findLeader(int x) {
    if (p[x]!=x) p[x]=findLeader(p[x]); // 路径压缩
    return p[x];
}
```

最短网络：解答-Kruskal算法

```
int main() {
    cin>>n;
    for(int i=1;i<=n;i++){
        for(int j=1;j<=n;j++){
            cin>>x;
            if(x!=0) {
                e[++m].s=i, e[m].t=j; e[m].w=x;
            }
        }
    }
    for(int i=1;i<=n;i++)  p[i]=i;   // 初始时所有顶点都是连通分支

    sort(e+1,e+m+1,cmp);  // 排序所有边
    for(int i=1,k=0;i<=m && k<n-1;i++) { // 从短到长遍历所有边
        int la=findLeader(e[i].s);
        int lb=findLeader(e[i].t);
        if(la!=lb) {     // 属于不同连通分支，添加到ST
            p[la]=lb; // 合并连通分支
            tot+=e[i].w;  // 累加边长
            k++;
        }

    }
    cout<<tot<<endl;
    return 0;
}
```



→ **复杂度：O(mlogm)**

以上为Kruskal算法一般的复杂度，m为边数

本题中图是稠密的，所以m=O(n2)

最短网络II

供应商告诉lester，光纤的费用不是跟总长度有关，而**只跟其中最长的一根有关**，因为单根越长的光纤工艺要求越高（好吧我编不下去了）。于是lester需要修改方案，使方案中最长一根光纤的长度最短

输入输出格式：

同前，输出改为最长的那根长度最小值

样例输入：

4
0 4 9 21
4 0 8 17
9 8 0 16
21 17 16 0

样例输出：

16

最短网络II：建模

→ **看了样例数据，似乎跟前一题的解一样的**

也就是**从MST中找条最长的那条边**输出就行了（老师你真敢想...）

→ **不巧，还就蒙对了...**

终极蒙题大法

不是什么人我都告诉哦：

不会做就选C

最短网络II：证明

反证法：

设T为无向图G的MST，(u,v)为T的最长边

假设$\exists$生成树T'及其最长边(u',v')，s.t. $w(u',v')<w(u,v)$

$T\backslash(u,v)$包含2个连通分支A、B

$\therefore \exists(x,y)\in T'$ s.t. $x\in A$ && $y\in B$

则$T''\equiv T\backslash(u,v)+(x,y)$也是生成树，且$w(x,y)\le w(u',v'$

$w(T'')=w(T)-w(u,v)+w(x,y)\le w(T)-w(u,v)+w(u',v')<w(T)$

矛盾，证毕

最小生成树：Prim算法

维护一个顶点集合V'，初始包含任意一个顶点

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

  选择满足如下条件的权值最小的边e=(u,v)

    $u \in V'$ && $v \notin V'$

  e加入T，v加入V'

}

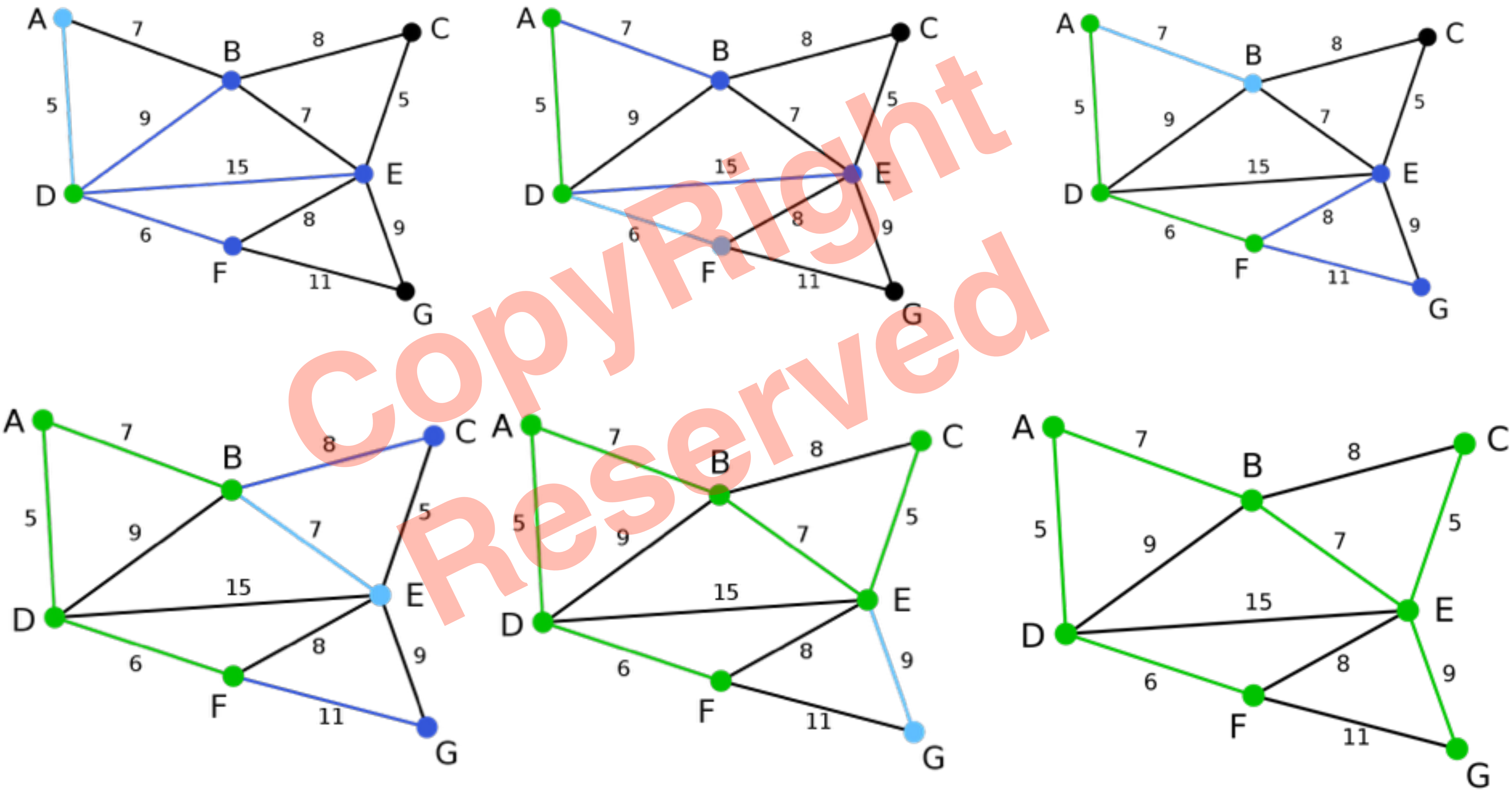
→ **本质上还是贪心的思想**

思想是**从一个点开始不断扩展**生成树

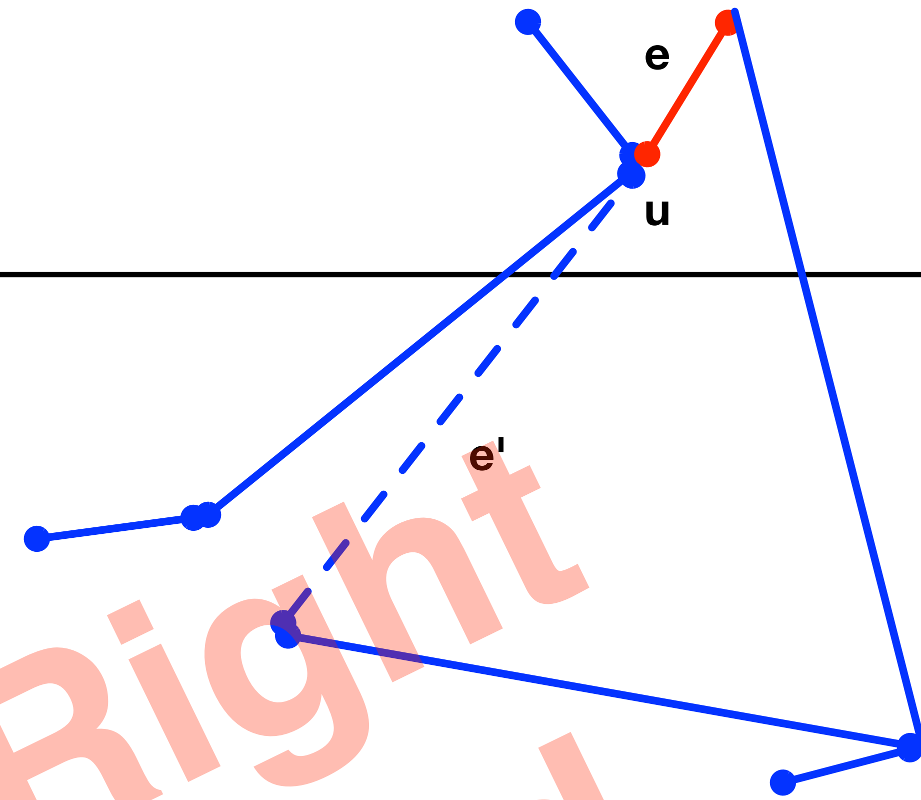
→ **为啥不用Kruskal算法?**

同一个算法连讲两遍多没劲~

Prim算法：演示



Prim算法：证明



还是归纳法（谁想来试一下？）

1.当n=1时，trivial

2.当n>1时

先证明**对任意顶点u，u出发的边里最短的一条e一定在MST中**

假设e∉MST，将e强制添加到MST，则会形成一个环R（又来了）

R上一定存在另一条边e'以u为端点

根据e的选择方式，w(e')≥w(e)

从MST中删除e'，增加形成一个新的生成树T'

w(T')=w(MST)+w(e)-w(e')≤w(MST)，则T'也是最小生成树

将u,v看做一个顶点，形成一个新图G'（顶点数为n-1）

∀G的生成树T，T'=T\e一定是G'的生成树

根据归纳法，证毕

Prim算法：实现

维护一个顶点集合V’，初始包含任意一个顶点

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

　选择满足如下条件的权值最小的边e=(u,v)

　　$u \in V'$ && $v \notin V'$

　e加入T，v加入V'

}

→ **需要动态寻找最小的边**

动态最值，用优先队列

→ **为什么Kruskal算法不需要用到优先队列?**

Kruskal算法中，待选边的集合**只会删不会增**

Prim算法中，待选边的集合**既会删又会增**

→ **思考题：并查集能不能删?**

最短网路II：解答

```
#define N 105
int n,visited[N*N]={0,1},tot=0,x;
struct edge {    // 边类型
    int s; int t; int w;
    bool operator < (edge e) const {
        return w>e.w;
    }
};
vector<edge> es[N];   // 邻接表的vector方式实现

int main() {
    cin>>n;
    for(int i=0;i<n;i++){    // 建图
        for(int j=0;j<n;j++){
            cin>>x;
            if(x!=0) {
                es[i].push_back((edge){i,j,x});
            }
        }
    }
```

→ **题目给的相邻矩阵，为啥用邻接表?**

涉及**遍历某个顶点出发的所有边**，邻接表最合适

教你邻接表的另一种实现方式（vector数组）

万一部分数据是稀疏的呢?

最短网路II：解答

```
priority_queue<edge> heap;  // 边优先队列
for (edge e:es[1]){
    heap.push(e);
}

for(int k=0;k<n-1 && !heap.empty();) {
    edge e=heap.top();
    heap.pop();
    if(!visited[e.t]) {    // 另一头不在ST中，添
          tot+=e.w;
        tot=max(tot,e.w);  // 更新边长
        visited[e.t]=1;
        k++;
        for (edge e2:es[e.t]){
            heap.push(e2);
        }
    }
}
cout<<tot<<endl;
```

- → **这里变相实现了堆的删除（非根节点）**
  这TM也可以?
  这种技术称为**标记-延迟（Tag-Lazy）**
  以后学到线段树，还会用到这种方法
- → **复杂度：O(n+mlogm)**

作业

1.Kruskal/Prim算法对于有否负权边有要求吗?

如果觉得有要求，举一个反例

如果觉得没有，简述下理由

（提示：证明过程有否隐含要求边权非负？）

2.如果原图可能不连通，这时候也有极大连通边集

可以称为**最小生成森林**

如果要Kruskal和Prim算法兼容不连通的情况，要改哪些地方?

3.买礼物

又到了一年一度的lester大神的生日了，lester想要买B样东西，这些东西价格都是A元。但是商店老板说最近有促销活动，也就是如果你买了第I样东西，再买第J样，那么就可以只花K[I,J]元，并且K[I,J]总是等于K[J,I]。如果K[I,J]=0，那么表示这两样东西之间不会导致优惠。同一件商品的优惠不能叠加（只能选一种优惠方案）。现在lester想知道，他最少要花多少钱

输入输出格式：

输入第一行两个整数，A（<=1000）,B（<=500）

接下来B行，每行B个数，第I行第J个为K[I,J]（<=1000，保证K[I,J]=K[J,I]并且K[I,I]=0）

输出一个整数，为最小要花的钱数

样例输入：

3 3
0 2 4
2 0 2
4 2 0

样例输出：

7

4.忽悠奶牛

lester有N个牧场，编号依次为1到N。每个牧场里住着一头奶牛。连接这些牧场的有P条道路，每条道路都是双向的。第j条道路连接的是牧场Sj和Ej，通行需要Lj的时间。lester打算在保持各牧场连通的情况下去掉尽量多的道路。在道路被强拆后，奶牛会非常伤心（O__O "…），所以lester计划拆除道路之后就去忽悠她们。lester可以选择从任意一个牧场出发开始他维稳工作。当他走访完所有的奶牛之后，还要回到他的出发地。每次路过牧场i的时候，他必须花Ci的时间和奶牛交谈，即使之前已经做过工作了，也要留下来再谈一次。注意约翰在出发和回去的时候，都要和出发地的奶牛谈一次话。请你计算一下，lester要拆除哪些道路，才能让忽悠奶牛的总时间最少

输入输出格式：

输入第一行两个整数n,p，（n<=10000,p<=100000）

第2行为n个整数Ci。接下来p行每行3个数Si,Ei,Li（Ci，Li<=10000）

输出一个整数，忽悠行动需要的最少时间

样例输入：

5 7
10 10 20 6 30
1 2 5
2 3 5
2 4 12
3 4 17
2 5 15
3 5 6
4 5 12

样例输出：

176

扩展阅读：哥当年翻译的图论

算法图论（Graph Theoretic Algorithms）

第一部分：区间图、弦图、相似图、完美图

本书英文版共 29 讲，第一部分共 12 讲，系译者学习的同时为备以后查阅复习之便作的翻译，内容基本忠实于原著，且大部分是直译，同时加入了译者的少许想法，并补上了少部分略去的证明过程（限于译者的水平只能证明一些力所能及的证明）。

第 1 讲是概论导读；第 2 到 10 讲比较详细地介绍了区间图、弦图及其相关的性质、算法（包括证明）以及延伸内容，其中第 7 到 9 讲涉及相似图的一些内容，部分定理尚未证明；第 11、12 讲介绍完美图、相交图及其部分特例，侧重于介绍性，概念结论较多但证明较少，可以作为知识性的阅读。

《尚为解决的问题》是译者翻译学习过程中遇到的一些不懂的问题，原文中未给出详细证明，而译者目前还无法自己证明的，这些在译文中将用粗括号扩出。各路高手如有知道方法的情与译者联系，不胜感激！

复旦附中 葛潇

http://www.doc88.com/p-9932763538951.html

这些内容会讲吗?

目前尚无规划，可能要CS105+吧