图论 1

最短路

非负权边单源最短路

Dijkstra 算法板子

时间复杂度 $O(m\log m)$

```
template <typename T>
struct dijkstra{
    T dis[MAXN];
    bool vis[MAXN];
    priority_queue<pair<T,int>,vector<pair<T,int> >,greater<pair<T,int> >
>q;
    void solve(int src,int n){
        mem(vis,0);
        _rep(i,1,n)
        dis[i]=Inf;
        dis[src]=0;
        q.push(make_pair(dis[src],src));
        while(!q.empty()){
            pair<T,int> temp=q.top();q.pop();
            int u=temp.second;
            if(vis[u])
            continue;
            vis[u]=true;
            for(int i=head[u];i;i=edge[i].next){
                int v=edge[i].to;
                if(dis[v]>edge[i].w+dis[u]){
                    dis[v]=edge[i].w+dis[u];
                    q.push(make_pair(dis[v],v));
                }
            }
        }
    }
};
```

例题

洛谷p1462

题意

给定 n 个城市□m 条边以及起点、终点。

要求选择一条路径，满足路径边权和不超过给定值，且路径上的最大点权最小。

题解

二分点权上界，跑 dijkstra 时跳过点权大于该上界的点，计算起点到终点的边权和最短路，如果不超过给定值则更新答案。

时间复杂度 $O(n\log m\log v)$

```
#include <iostream>
#include <cstdio>
#include <cstdlib>
#include <algorithm>
#include <string>
#include <sstream>
#include <cstring>
#include <cctype>
#include <cmath>
#include <vector>
#include <set>
#include <map>
#include <stack>
#include <queue>
#include <ctime>
#include <cassert>
#define _for(i,a,b) for(int i=(a);i<(b);++i)
#define _rep(i,a,b) for(int i=(a);i<=(b);++i)
#define mem(a,b) memset(a,b,sizeof(a))
using namespace std;
typedef long long LL;
inline int read_int(){
    int t=0;bool sign=false;char c=getchar();
    while(!isdigit(c)){sign|=c=='-';c=getchar();}
    while(isdigit(c)){t=(t<<1)+(t<<3)+(c&15);c=getchar();}
    return sign?-t:t;
}
inline LL read_LL(){
    LL t=0;bool sign=false;char c=getchar();
    while(!isdigit(c)){sign|=c=='-';c=getchar();}
    while(isdigit(c)){t=(t<<1)+(t<<3)+(c&15);c=getchar();}
    return sign?-t:t;
}
inline char get_char(){
    char c=getchar();
    while(c==' '||c=='\n'||c=='\r')c=getchar();
    return c;
```

```
}
inline void write(LL x){
    register char c[21],len=0;
    if(!x)return putchar('0'),void();
    if(x<0)x=-x,putchar('-');
    while(x)c[++len]=x%10,x/=10;
    while(len)putchar(c[len--]+48);
}
inline void space(LL x){write(x),putchar(' ');}
inline void enter(LL x){write(x),putchar('\n');}
const int MAXN=1e4+5,MAXM=5e4+5,Inf=2e9;
struct Edge{
    int to,w,next;
}edge[MAXM<<1];
int head[MAXN],edge_cnt,val[MAXN],limit;
void Insert(int u,int v,int w){
    edge[++edge_cnt].next=head[u];
    edge[edge_cnt].to=v;edge[edge_cnt].w=w;
    head[u]=edge_cnt;
}
template <typename T>
struct dijkstra{
    T dis[MAXN];
    bool vis[MAXN];
    priority_queue<pair<T,int>,vector<pair<T,int> >,greater<pair<T,int> >
>q;
    void solve(int src,int n){
        mem(vis,0);
        _rep(i,1,n)
        dis[i]=Inf;
        dis[src]=0;
        q.push(make_pair(dis[src],src));
        while(!q.empty()){
            pair<T,int> temp=q.top();q.pop();
            int u=temp.second;
            if(vis[u])
            continue;
            vis[u]=true;
            for(int i=head[u];i;i=edge[i].next){
                int v=edge[i].to;
                if(val[v]>limit)
                continue;
                if(dis[v]>edge[i].w+dis[u]){
                    dis[v]=edge[i].w+dis[u];
                    q.push(make_pair(dis[v],v));
                }
            }
        }
    }
};
dijkstra<LL> dj;
```

```
int main()
{
    int n=read_int(),m=read_int(),b=read_int(),u,v,w,lef=Inf,rig=0;
    _rep(i,1,n){
        val[i]=read_int();
        lef=min(val[i],lef);
        rig=max(val[i],rig);
    }
    while(m--){
        u=read_int(),v=read_int(),w=read_int();
        Insert(u,v,w);
        Insert(v,u,w);
    }
    int mid,ans=-1;
    while(lef<=rig){
        mid=lef+rig>>1;
        limit=mid;
        dj.solve(1,n);
        if(dj.dis[n]<=b){
            ans=mid;
            rig=mid-1;
        }
        else
        lef=mid+1;
    }
    if(ans>=0)
    enter(ans);
    else
    puts("AFK");
    return 0;
}
```

带负权边单源最短路

SPFA 算法板子

平均时间复杂度 $O(Km)$，最坏时间复杂度 $O(nm)$。

```
template <typename T>
struct SPFA{
    T dis[MAXN];
    int len[MAXN];
    bool inque[MAXN];
    bool solve(int src,int n){
        queue<int>q;
        mem(inque,0);mem(len,0);
        _rep(i,1,n)
        dis[i]=Inf;
```

```
        dis[src]=0;len[src]=1;
        q.push(src);
        inque[src]=true;
        while(!q.empty()){
            int u=q.front();q.pop();
            inque[u]=false;
            for(int i=head[u];i;i=edge[i].next){
                int v=edge[i].to;
                if(dis[v]>dis[u]+edge[i].w){
                    dis[v]=dis[u]+edge[i].w;
                    len[v]=len[u]+1;
                    if(len[v]>n)
                    return false;
                    if(!inque[v]){
                        q.push(v);
                        inque[v]=true;
                    }
                }
            }
        }
        return true;
    }
};
```

例题

洛谷p5960

题意

解方程 $$ \left \{ \begin{array}{l} x_{a1}-x_{b1}\le y_1\\ x_{a2}-x_{b2}\le y_2\\ \cdots\\ x_{am}-x_{bm}\le y_m \end{array} \right. $$

题解

将 $x_j-x_i\le y$ 移项，得 $x_j\le x_i+y$发现该式与单源最短路的三角不等式 $\text{dist}_j\le \text{dist}_i+\text{w}_{i\to j}$ 相似。

考虑添加超级源点 $x_0$$x_0$ 向所有其他点连一条权为 0 (事实上边权数值无特殊要求，边权相当于为所有解加上一个初始值)的单向边。

然后跑最短路算法即可，解得 $x_i=\text{dist}_i+k$ 为方程的一组可行解。

```
#include <iostream>
#include <cstdio>
#include <cstdlib>
#include <algorithm>
```

```
#include <string>
#include <sstream>
#include <cstring>
#include <cctype>
#include <cmath>
#include <vector>
#include <set>
#include <map>
#include <stack>
#include <queue>
#include <ctime>
#include <cassert>
#define _for(i,a,b) for(int i=(a);i<(b);++i)
#define _rep(i,a,b) for(int i=(a);i<=(b);++i)
#define mem(a,b) memset(a,b,sizeof(a))
using namespace std;
typedef long long LL;
inline int read_int(){
    int t=0;bool sign=false;char c=getchar();
    while(!isdigit(c)){sign|=c=='-';c=getchar();}
    while(isdigit(c)){t=(t<<1)+(t<<3)+(c&15);c=getchar();}
    return sign?-t:t;
}
inline LL read_LL(){
    LL t=0;bool sign=false;char c=getchar();
    while(!isdigit(c)){sign|=c=='-';c=getchar();}
    while(isdigit(c)){t=(t<<1)+(t<<3)+(c&15);c=getchar();}
    return sign?-t:t;
}
inline char get_char(){
    char c=getchar();
    while(c==' '||c=='\n'||c=='\r')c=getchar();
    return c;
}
inline void write(LL x){
    register char c[21],len=0;
    if(!x)return putchar('0'),void();
    if(x<0)x=-x,putchar('-');
    while(x)c[++len]=x%10,x/=10;
    while(len)putchar(c[len--]+48);
}
inline void space(LL x){write(x),putchar(' ');}
inline void enter(LL x){write(x),putchar('\n');}
const int MAXN=1e4+5,MAXM=5e5+5,Inf=1e9;
struct Edge{
    int to,w,next;
}edge[MAXM<<1];
int head[MAXN],edge_cnt;
void Insert(int u,int v,int w){
    edge[++edge_cnt].next=head[u];
```

```
    edge[edge_cnt].to=v;edge[edge_cnt].w=w;
    head[u]=edge_cnt;
}
template <typename T>
struct SPFA{
    T dis[MAXN];
    int len[MAXN];
    bool inque[MAXN];
    bool solve(int src,int n){
        queue<int>q;
        mem(inque,0);mem(len,0);
        _rep(i,1,n)
        dis[i]=Inf;
        dis[src]=0;len[src]=1;
        q.push(src);
        inque[src]=true;
        while(!q.empty()){
            int u=q.front();q.pop();
            inque[u]=false;
            for(int i=head[u];i;i=edge[i].next){
                int v=edge[i].to;
                if(dis[v]>dis[u]+edge[i].w){
                    dis[v]=dis[u]+edge[i].w;
                    len[v]=len[u]+1;
                    if(len[v]>n)
                    return false;
                    if(!inque[v]){
                        q.push(v);
                        inque[v]=true;
                    }
                }
            }
        }
        return true;
    }
};
SPFA<int> spfa;
int main()
{
    int n=read_int(),m=read_int(),u,v,w;
    _rep(i,1,n)Insert(n+1,i,0);
    while(m--){
        u=read_int(),v=read_int(),w=read_int();
        Insert(v,u,w);
    }
    if(spfa.solve(n+1,n+1)){
        _rep(i,1,n){
            if(i>1)putchar(' ');
            write(spfa.dis[i]);
        }
    }
```

```
    else
    puts("NO");
    return 0;
}
```

带负权边全源最短路

Floyd 算法板子

时间复杂度 $O(n^3)$无法判断负环。

```
int n,dis[MAXN][MAXN];
void Floyd(){
    _for(i,0,n)
    _for(j,0,n)
    dis[i][j]=Inf;
    _for(i,0,n)
    dis[i][i]=0;
    _for(k,0,n)
    _for(i,0,n)
    _for(j,0,n)
    dis[i][j]=min(dis[i][j],dis[i][k]+dis[k][j]);
}
```

例题

洛谷p1119

题意

给定 n 个城市m 条边，每个城市在第 t_i 天起才加入点集(保证 t_i 升序)。

q 个询问，每次询问第 t 天的 $dis(i,j)$保证询问的 t 升序。

题解

考虑 Floyd 算法本质其实是 dp

$dis[k][i][j]$ 表示只使用前 k 个点作为中转点时 $i$$j$ 间的最短路，可以用滚动数组省去一维。

所以本题只需要按时间更新即可。

```
#include <iostream>
#include <cstdio>
```

```cpp
#include <cstdlib>
#include <algorithm>
#include <string>
#include <sstream>
#include <cstring>
#include <cctype>
#include <cmath>
#include <vector>
#include <set>
#include <map>
#include <stack>
#include <queue>
#include <ctime>
#include <cassert>
#define _for(i,a,b) for(int i=(a);i<(b);++i)
#define _rep(i,a,b) for(int i=(a);i<=(b);++i)
#define mem(a,b) memset(a,b,sizeof(a))
using namespace std;
typedef long long LL;
inline int read_int(){
    int t=0;bool sign=false;char c=getchar();
    while(!isdigit(c)){sign|=c=='-';c=getchar();}
    while(isdigit(c)){t=(t<<1)+(t<<3)+(c&15);c=getchar();}
    return sign?-t:t;
}
inline LL read_LL(){
    LL t=0;bool sign=false;char c=getchar();
    while(!isdigit(c)){sign|=c=='-';c=getchar();}
    while(isdigit(c)){t=(t<<1)+(t<<3)+(c&15);c=getchar();}
    return sign?-t:t;
}
inline char get_char(){
    char c=getchar();
    while(c==' '||c=='\n'||c=='\r')c=getchar();
    return c;
}
inline void write(LL x){
    register char c[21],len=0;
    if(!x)return putchar('0'),void();
    if(x<0)x=-x,putchar('-');
    while(x)c[++len]=x%10,x/=10;
    while(len)putchar(c[len--]+48);
}
inline void space(LL x){write(x),putchar(' ');}
inline void enter(LL x){write(x),putchar('\n');}
const int MAXN=200+5,Inf=1e9;
int dis[MAXN][MAXN],t[MAXN];
int main()
{
    int n=read_int(),m=read_int(),u,v,w;
    _for(i,0,n)
```

```
    t[i]=read_int();
    _for(i,0,n)
    _for(j,0,n)
    dis[i][j]=Inf;
    while(m--){
        u=read_int(),v=read_int(),w=read_int();
        dis[u][v]=dis[v][u]=w;
    }
    _for(i,0,n)
    dis[i][i]=0;
    int q=read_int(),pos=0,temp;
    while(q--){
        u=read_int(),v=read_int(),temp=read_int();
        while(t[pos]<=temp&&pos<n){
            _for(i,0,n)
            _for(j,0,n)
            dis[i][j]=min(dis[i][j],dis[i][pos]+dis[pos][j]);
            pos++;
        }
        if(t[u]>temp||t[v]>temp||dis[u][v]==Inf)
        enter(-1);
        else
        enter(dis[u][v]);
    }
    return 0;
}
```

Johnson 算法板子

洛谷p5905

加入虚拟节点，虚拟节点向每个点连一条权值为 0 的单向边。

跑一遍 SPFA 得到每个点到虚拟节点的距离，记该距离为每个点的势能 h_i

将原图中的所有边权修改 $w+h_u-h_v$ 则新图的每条路径 $s\to t$ 的长度为

\begin{equation}(w_{s,p1}+h_s-h_{p1})+(w_{p1,p2}+h_{p1}-h_{p2})+\cdots+(w_{pk,t}+h_{pk}-h_t)=w_{s,p1}+w_{p1,p2}+\cdots+w_{pk,t}+h_s-h_t\end{equation}

所以新图的最短路与原图等效。

另外，根据 SPFA 结果，有 $h_v\le h_u+w_{u,v}$ 即 $w_{u,v}+h_u-h_v\ge 0$

由于新图所有边权非负，所以可以跑 n 轮 Dijkstra 求出全源最短路。

时间复杂度 $O(nm\log m)$ 带负环判断。

```
#include <iostream>
```

```cpp
#include <cstdio>
#include <cstdlib>
#include <algorithm>
#include <string>
#include <sstream>
#include <cstring>
#include <cctype>
#include <cmath>
#include <vector>
#include <set>
#include <map>
#include <stack>
#include <queue>
#include <ctime>
#include <cassert>
#define _for(i,a,b) for(int i=(a);i<(b);++i)
#define _rep(i,a,b) for(int i=(a);i<=(b);++i)
#define mem(a,b) memset(a,b,sizeof(a))
using namespace std;
typedef long long LL;
inline int read_int(){
    int t=0;bool sign=false;char c=getchar();
    while(!isdigit(c)){sign|=c=='-';c=getchar();}
    while(isdigit(c)){t=(t<<1)+(t<<3)+(c&15);c=getchar();}
    return sign?-t:t;
}
inline LL read_LL(){
    LL t=0;bool sign=false;char c=getchar();
    while(!isdigit(c)){sign|=c=='-';c=getchar();}
    while(isdigit(c)){t=(t<<1)+(t<<3)+(c&15);c=getchar();}
    return sign?-t:t;
}
inline char get_char(){
    char c=getchar();
    while(c==' '||c=='\n'||c=='\r')c=getchar();
    return c;
}
inline void write(LL x){
    register char c[21],len=0;
    if(!x)return putchar('0'),void();
    if(x<0)x=-x,putchar('-');
    while(x)c[++len]=x%10,x/=10;
    while(len)putchar(c[len--]+48);
}
inline void space(LL x){write(x),putchar(' ');}
inline void enter(LL x){write(x),putchar('\n');}
const int MAXN=3e3+5,MAXM=9e3+5,Inf=1e9;
struct Edge{
    int to,w,next;
}edge[MAXM];
int head[MAXN],edge_cnt;
```

```
void Insert(int u,int v,int w){
    edge[++edge_cnt].next=head[u];
    edge[edge_cnt].to=v;edge[edge_cnt].w=w;
    head[u]=edge_cnt;
}
template <typename T>
struct SPFA{
    T dis[MAXN];
    int len[MAXN];
    bool inque[MAXN];
    bool solve(int src,int n){
        queue<int>q;
        mem(inque,0);mem(len,0);
        _rep(i,1,n)
        dis[i]=Inf;
        dis[src]=0;len[src]=1;
        q.push(src);
        inque[src]=true;
        while(!q.empty()){
            int u=q.front();q.pop();
            inque[u]=false;
            for(int i=head[u];i;i=edge[i].next){
                int v=edge[i].to;
                if(dis[v]>dis[u]+edge[i].w){
                    dis[v]=dis[u]+edge[i].w;
                    len[v]=len[u]+1;
                    if(len[v]>n)
                    return false;
                    if(!inque[v]){
                        q.push(v);
                        inque[v]=true;
                    }
                }
            }
        }
        return true;
    }
};
template <typename T>
struct dijkstra{
    T dis[MAXN];
    bool vis[MAXN];
    priority_queue<pair<T,int>,vector<pair<T,int> >,greater<pair<T,int> >
>q;
    void solve(int src,int n){
        mem(vis,0);
        _rep(i,1,n)
        dis[i]=Inf;
        dis[src]=0;
        q.push(make_pair(dis[src],src));
```

```
        while(!q.empty()){
            pair<T,int> temp=q.top();q.pop();
            int u=temp.second;
            if(vis[u])
            continue;
            vis[u]=true;
            for(int i=head[u];i;i=edge[i].next){
                int v=edge[i].to;
                if(dis[v]>edge[i].w+dis[u]){
                    dis[v]=edge[i].w+dis[u];
                    q.push(make_pair(dis[v],v));
                }
            }
        }
    }
};
SPFA<int> spfa;
dijkstra<int> dj;
int main()
{
    int n=read_int(),m=read_int(),u,v,w;
    _for(i,0,m){
        u=read_int(),v=read_int(),w=read_int();
        Insert(u,v,w);
    }
    _rep(i,1,n)
    Insert(n+1,i,0);
    if(!spfa.solve(n+1,n+1)){
        puts("-1");
        return 0;
    }
    _rep(u,1,n){
        for(int i=head[u];i;i=edge[i].next){
            int v=edge[i].to;
            edge[i].w+=spfa.dis[u]-spfa.dis[v];
        }
    }
    _rep(i,1,n){
        LL ans=0;
        dj.solve(i,n);
        _rep(j,1,n) if(dj.dis[j]!=Inf)
        dj.dis[j]-=spfa.dis[i]-spfa.dis[j];
        _rep(j,1,n)
        ans+=1LL*j*dj.dis[j];
        enter(ans);
    }
    return 0;
}
```

连通分量

无向图的割顶与桥

定义

若删除节点 u 将导致无向图的连通分量增加，则称 u 为无向图的割顶。

若删除边 e 将导致无向图的连通分量增加，则称 e 为无向图的桥。

算法思想

考虑 dfs 过程中建树。如果某条边指向的节点是第一次访问，则该边为树边，否则为反向边。易知，不同子树间不存在树边与反向边。

记 $\text{low}(u)$ 为 u 及其后代不经过 u 与 fa_u 的树边能连回的最早祖先的 pre 值。

顶点 u 为割顶当且仅当 u 为根节点且 u 在树中有两个子节点或 u 为非根节点且存在 u 的一个子节点 v 满足 $\text{low}(v)\ge \text{dfs_id}(u)$

另外若此时还有 $\text{low}(v)\gt \text{dfs_id}(u)$则 (u,v) 为桥。

只需要 dfs 过程维护一下 low 数组即可，需要从树边的贡献和反向边的贡献两方面考虑，时间复杂度 $O(n+m)$

```
int low[MAXN],dfs_id[MAXN],dfs_t;
bool iscut[MAXN];
void dfs(int u,int fa){
    low[u]=dfs_id[u]=++dfs_t;
    int child=0;
    for(int i=head[u];i;i=edge[i].next){
        int v=edge[i].to;
        if(v==fa)continue;
        if(!dfs_id[v]){//树边贡献
            dfs(v,u);
            low[u]=min(low[u],low[v]);
            if(low[v]>=dfs_id[u]&&u!=fa)
            iscut[u]=true;
            child++;
        }
        else
        low[u]=min(low[u],dfs_id[v]);//反向边贡献
    }
    if(u==fa&&child>=2)//根节点特判
    iscut[u]=true;
}
```

无向图的双连通分量

定义

给定一个连通图，以下条件等价：

- 任意两点间至少存在两条点不重复的路径
- 任意两条边都至少可以找到一个包含它们的简单环。
- 图内部无割顶

若满足上述条件，则称该图是点-双连通的。对一个无向图，称点-双连通的极大子图为点-双连通分量。

类似的，给定一个连通图，以下条件等价：

- 任意两点间至少存在两条边不重复的路径
- 每条边都至少可以找到一个包含它的简单环。
- 图内部无桥

若满足上述条件，则称该图是边-双连通的。对一个无向图，称点-双连通的极大子图为边-双连通分量。

算法思想

对点-双连通分量，有如下性质：

- 每条边恰好属于一个点-双连通分量
- 任意两个双连通分量间最多有一个公共点，且该点为割顶
- 任意割顶至少属于两个不同的双连通分量

求点-双连通分量有两种方法，一种先一次 dfs 求出割顶，再一次 dfs 染色，染色过程中不经过割顶。

第二种方法为每求出一个割顶，立刻处理该连通分量，下面给出第二种方法的板子，时间复杂度 $O(n+m)$

```
int low[MAXN],dfs_id[MAXN],dfs_t,bcc_id[MAXN],bcc_cnt;
vector<int> bcc[MAXN];
stack<pair<int,int> >Stack;
bool iscut[MAXN];
void dfs(int u,int fa){
    low[u]=dfs_id[u]=++dfs_t;
    int child=0;
    for(int i=head[u];i;i=edge[i].next){
        int v=edge[i].to;
        if(v==fa)continue;
        if(!dfs_id[v]){
            Stack.push(make_pair(u,v));
            dfs(v,u);
            low[u]=min(low[u],low[v]);
            if(low[v]>=dfs_id[u]){
                iscut[u]=true;
                pair<int,int> temp;
```

```
                bcc[++bcc_cnt].clear();
                while(true){
                    temp=Stack.top();Stack.pop();
                    if(bcc_id[temp.first]!=bcc_cnt){
                        bcc_id[temp.first]=bcc_cnt;
                        bcc[bcc_cnt].push_back(temp.first);
                    }
                    if(bcc_id[temp.second]!=bcc_cnt){
                        bcc_id[temp.second]=bcc_cnt;
                        bcc[bcc_cnt].push_back(temp.second);
                    }
                    if(temp.first==u&&temp.second==v)
                    break;
                }
            }
            child++;
        }
        else if(dfs_id[v]<dfs_id[u]){
            Stack.push(make_pair(u,v));
            low[u]=min(low[u],dfs_id[v]);
        }
    }
    if(u==fa&&child<2)
    iscut[u]=false;
}
void find_bcc(int n){
    mem(dfs_id,0);
    mem(iscut,0);
    mem(bcc_id,0);
    dfs_t=bcc_cnt=0;
    _rep(i,1,n){
        if(!dfs_id[i])
        dfs(i,i);
    }
}
```

对边-双连通分量，有如下性质：

- 除了桥不属于任意一个边-双连通分量，其他每条边恰好属于一个点-双连通分量
- 任意两个双连通分量间无公共点和公共边

求边-双连通分量有两种方法，与求点-双连通分量类似。

例题

洛谷p3225

题意

给定 n 个点□m 条边。要求选择若干点作为逃生点，使得删去任意一个点后任意其他点均能达到某个逃生点。

输出最少需要选择的点数和选择最少的点数的方案数。

题解

考虑先求出所有点-双连通分量，记点-双连通分量的度为该点-双连通分量中的割点数。

易知若点-双连通分量的度等于 0，该点-双连通分量中必须设立两个逃生点，逃生点位置任意。

若点-双连通分量的度等于 1，该点-双连通分量中必须设立一个逃生点，逃生点不能是割点。

若点-双连通分量的度大于 1，则删去任意一个点后该点-双连通分量中仍然可以到达其他度为 1 的点-双连通分量，不需要设置逃生点。

若某个点为孤立点，按之前给出的求点-双连通分量的算法它无法被计入任意一个连通分量，需要额外设立一个逃生点。

```
#include <iostream>
#include <cstdio>
#include <cstdlib>
#include <algorithm>
#include <string>
#include <sstream>
#include <cstring>
#include <cctype>
#include <cmath>
#include <vector>
#include <set>
#include <map>
#include <stack>
#include <queue>
#include <ctime>
#include <cassert>
#define _for(i,a,b) for(int i=(a);i<(b);++i)
#define _rep(i,a,b) for(int i=(a);i<=(b);++i)
#define mem(a,b) memset(a,b,sizeof(a))
using namespace std;
typedef long long LL;
inline int read_int(){
    int t=0;bool sign=false;char c=getchar();
    while(!isdigit(c)){sign|=c=='-';c=getchar();}
    while(isdigit(c)){t=(t<<1)+(t<<3)+(c&15);c=getchar();}
    return sign?-t:t;
}
inline LL read_LL(){
    LL t=0;bool sign=false;char c=getchar();
```

```
    while(!isdigit(c)){sign|=c=='-';c=getchar();}
    while(isdigit(c)){t=(t<<1)+(t<<3)+(c&15);c=getchar();}
    return sign?-t:t;
}
inline char get_char(){
    char c=getchar();
    while(c==' '||c=='\n'||c=='\r')c=getchar();
    return c;
}
inline void write(LL x){
    register char c[21],len=0;
    if(!x)return putchar('0'),void();
    if(x<0)x=-x,putchar('-');
    while(x)c[++len]=x%10,x/=10;
    while(len)putchar(c[len--]+48);
}
inline void space(LL x){write(x),putchar(' ');}
inline void enter(LL x){write(x),putchar('\n');}
const int MAXN=505,MAXM=505;
struct Edge{
    int to,next;
}edge[MAXM<<1];
int head[MAXN],edge_cnt;
void Insert(int u,int v){
    edge[++edge_cnt].next=head[u];
    edge[edge_cnt].to=v;
    head[u]=edge_cnt;
}
int low[MAXN],dfs_id[MAXN],dfs_t,bcc_id[MAXN],bcc_cnt;
vector<int> bcc[MAXN];
stack<pair<int,int> >Stack;
bool iscut[MAXN];
void dfs(int u,int fa){
    low[u]=dfs_id[u]=++dfs_t;
    int child=0;
    for(int i=head[u];i;i=edge[i].next){
        int v=edge[i].to;
        if(v==fa)continue;
        if(!dfs_id[v]){
            Stack.push(make_pair(u,v));
            dfs(v,u);
            low[u]=min(low[u],low[v]);
            if(low[v]>=dfs_id[u]){
                iscut[u]=true;
                pair<int,int> temp;
                bcc[++bcc_cnt].clear();
                while(true){
                    temp=Stack.top();Stack.pop();
                    if(bcc_id[temp.first]!=bcc_cnt){
                        bcc_id[temp.first]=bcc_cnt;
```

```
                        bcc[bcc_cnt].push_back(temp.first);
                    }
                    if(bcc_id[temp.second]!=bcc_cnt){
                        bcc_id[temp.second]=bcc_cnt;
                        bcc[bcc_cnt].push_back(temp.second);
                    }
                    if(temp.first==u&&temp.second==v)
                    break;
                }
            }
            child++;
        }
        else if(dfs_id[v]<dfs_id[u]){
            Stack.push(make_pair(u,v));
            low[u]=min(low[u],dfs_id[v]);
        }
    }
    if(u==fa&&child<2)
    iscut[u]=false;
}
void find_bcc(int n){
    mem(dfs_id,0);
    mem(iscut,0);
    mem(bcc_id,0);
    dfs_t=bcc_cnt=0;
    _rep(i,1,n){
        if(!dfs_id[i])
        dfs(i,i);
    }
}
int main()
{
    int kase=0,n,m,u,v;
    while(m=read_int()){
        n=0,edge_cnt=0;
        mem(head,0);
        while(m--){
            u=read_int(),v=read_int();
            n=max(n,u),n=max(n,v);
            Insert(u,v);
            Insert(v,u);
        }
        find_bcc(n);
        int ans1=0;
        unsigned long long ans2=1;
        _rep(i,1,bcc_cnt){
            int cnt=0;
            _for(j,0,bcc[i].size()){
                if(iscut[bcc[i][j]])
                cnt++;
            }
```

```
            if(cnt==1)
            ans1++,ans2*=bcc[i].size()-1;
            else if(cnt==0)
            ans1+=2,ans2*=bcc[i].size()*(bcc[i].size()-1)/2;
        }
        _rep(i,1,n){
            if(!bcc_id[i])
            ans1++;
        }
        printf("Case %d: %d %llu\n",++kase,ans1,ans2);
    }
    return 0;
}
```

有向图的强连通分量