弦图

任意长度大于3的环都有一个弦的图称为弦图。很多在一般图上的NP-Hard问题在弦图中都有优秀的**线性**时间复杂度解法。

基本概念

弦：连接环中不相邻两点的边。

子图：点集和边集均为原图点集和边集的子集的图。

团：完全子图，即满足任意两点都恰有一条边相连的子图。

导出子图：点集为原图点集子集，边集为所有满足**两端点均在选定点集中**的图。

最大独立集：最大的点集使得点集中任意两点都没有边直接相连。

最小团覆盖数：用最少的团覆盖所有的点。

单纯点：设$N(v)$表示与点v相邻的点集。一个点称为单纯点当$\{v\}+N(v)$的诱导子图为一个团。任何一个弦图都有至少一个单纯点，不是完全图的弦图至少有两个不相邻的单纯点。

完美图：一个图G称为完美图若它的每一个诱导子图都满足$\omega(G)=\chi(G)$

伴完美图：一个图G称为伴完美图若它的每一个诱导子图都满足$\alpha(G)=\kappa(G)$

定理

①：团数小于等于色数。$w(G) \le \chi(G)$

对最大团的导出子图进行染色，至少需要$w(G)$种。

②：最大独立集数小于等于最小团覆盖数。$\alpha(G) \le \kappa(G)$

每个团至多选择一个点。

③：弦图的任意导出子图一定都是弦图。

④: 弦图的任意导出子图一定不可能是一个点数大于3的环。（与③同理）

⑤最大独立集数 = 最小团覆盖数

⑥完美图属于伴完美图

⑦弦图属于完美图

弦图判断

朴素算法

依次判断$\{v_{i+1},...,v_n\}$中所有与v_i相邻的点是否构成了一个团。

时间复杂度$O(\sum(deg(v))^2) = O(nm)$

优化后的算法

设$\{v_{i+1},...,v_n\}$中所有与v_i相邻的点依次为$v_{j1},...,v_{jk}$

只需判断v_{j1}是否与$v_{j2},...,v_{jk}$相邻即可。

时间复杂度$O(n+m)$

```
//bzoj1242 zju1015 Fishing Net弦图判定
#include<bits/stdc++.h>
#define pb push_back
using namespace std;
const int N=1010,M=4e6+10;

int n,m,g[N][N],q[N],f[N],rk[N];
int head[N],to[M],nxt[M],tot;
vector<int>nt[N];bool vs[N];

inline void lk(int u,int v)
{to[++tot]=v;nxt[tot]=head[u];head[u]=tot;}

void MSC()
{
    int i,j,x,y,bst=0;
    for(i=1;i<=n;++i) lk(0,i);
    for(j=n;j>0;--j){
        for(;;){
            for(int &i=head[bst];i;i=nxt[i])
                if(!vs[to[i]]) break;
            if(head[bst]){
```

```
                x=to[head[bst]];q[j]=x;rk[x]=j;vs[x]=true;
                for(i=nt[x].size()-1;i>=0;--i) if(!vs[nt[x][i]]){
                    y=nt[x][i];f[y]++;lk(f[y],y);
                    bst=max(bst,f[y]);
                }
                break;
            }else bst--;
        }
    }
}

int stk[N],top;

bool ck()
{
    int i,j,x,y,mng,mnv;
    for(j=n;j>0;--j){
        x=q[j];top=0;mng=n+1;
        for(i=nt[x].size()-1;i>=0;--i) if(rk[nt[x][i]]>j){
            y=nt[x][i];stk[++top]=y;
            if(rk[y]<mng) mng=rk[y],mnv=y;
        }
        if(mng==n+1) continue;
        for(i=1;i<=top;++i) if(stk[i]!=mnv)
         if(!g[mnv][stk[i]]) return false;
    }
    return true;
}

int main(){
    int i,j,x,y;
    scanf("%d%d",&n,&m);
    for(i=1;i<=m;++i){
        scanf("%d%d",&x,&y);
        g[x][y]=g[y][x]=1;
        nt[x].pb(y);nt[y].pb(x);
    }
    MSC();
    if(ck()) printf("Perfect");
    else printf("Imperfect");
    return 0;
}
```

结论：弦图判定的问题可以在线性时间内解决！

完美消除序列

一个完美消除序列是一个排列$a_1,a_2,...,a_n$，满足对于任意的i,都有$a_i,a_{i+1},...,a_n$形成的导出子图与a_i相邻的点两两之间有边。一个图是弦图的充要条件是它有一个完美序列。

朴素算法求完美消除序列

①每次找一个单纯点，加入到完美消除序列中。

②将v以及相关的边从图中删掉。

③重复以上过程直到所有点都被删除或者不存在单纯点v(图不是弦图)。

MCS算法求完美消除序列

最大势算法是一种可以再$O(n+m)$的时间内求出无向图的完美消除序列的方法。具体步骤为：

①逆序给结点编号，即按从n到1的顺序给点标号。

②设$label_x$表示第x个点与多少个已经标号的点相邻，每次选择$label$值最大的未标号结点进行标号。

③用链表维护对于每个i,满足$label_x=i$的x。

④由于每条边对$\sum_{i=1}^{n}label_i$的贡献最多是2，时间复杂度为$O(n+m)$

除此之外，还有**LexBFS算法**（字典序广度优先搜索）。

弦图的色数

给定一个n个点m条边的弦图，求它的色数。

思路：先求出完美消除序列，然后贪心倒着染色。与当前点相邻的染过色的点显然颜色互不相同，因此可以证明是对的。枚举颜色数$O(\sqrt{m})$，所以总复杂度为$O(n\sqrt{m})$。

模板 P3196 [HNOI2008]神奇的国度

按完美消除序列从后往前依次给每个点染色，给每个点染上可以染的最小颜色。

```
//#pragma GCC optimize("Ofast", "inline", "-ffast-math")
//#pragma GCC target("avx,sse2,sse3,sse4,mmx")
#include<bits/stdc++.h>
#define inf 0x3f3f3f3f
#define int long long
using namespace std;
const int N=2e5+7;
const int mod=1e9+7;

//int read(){    int x=0,f=1;char ch=getchar();while(ch<'0'||ch>'9'){if(ch=='-')
f=f*-1;ch=getchar();}while(ch>='0'&&ch<='9'){x=x*10+ch-'0';ch=getchar();}return
x*f;}

int n,m,a[N],c[N],l[N],r[N],col[N],vis[N];
```

```
bool b[N];
vector<int>G[N];

void get_array(){
    for(int i=0;i<=2*n;i++){
        l[i]=r[i]=-1;
    }
    for(int i=1,t=0;i<=n;i++){
        l[n+i]=t,r[t]=n+i,t=n+i;
    }
    int mx=0;
    for(int k=1;k<=n;k++){
        while(r[mx]==-1) mx--;
        int u=r[mx]-n,t=r[u+n];
        l[t]=mx,r[mx]=t;
        a[k]=u,b[u]=1;
        for(int i=0,v,lx,rx;i<G[u].size();i++){
            v=G[u][i];
            if(b[v]) continue;
            lx=l[n+v],rx=r[n+v];
            r[lx]=rx,l[rx]=lx,c[v]++;
            l[n+v]=r[n+v]=-1;
            if(~r[c[v]]) l[r[c[v]]]=n+v,r[n+v]=r[c[v]];
            r[c[v]]=n+v,l[n+v]=c[v];
        }
        mx++;
    }
}

void Solve(){
    cin>>n>>m;
    for(int i=1,u,v;i<=m;i++){
        cin>>u>>v;
        G[u].push_back(v);
        G[v].push_back(u);
    }
    get_array();
    int ans=0;
    for(int k=1,i;k<=n;k++){
        i=a[k];
        for(int j=0;j<G[i].size();j++) vis[col[G[i][j]]]=i;
        for(int j=1;j<=n;j++){
            ans=max(ans,j);
            if(vis[j]!=i){
                col[i]=j;
                break;
            }
        }
    }
    cout<<ans<<"\n";
}

signed main(){
    int T=1;
    //cin>>T;
    while(T--){
        Solve();
    }
```

```
    return 0;
}
```

区间图

给定一些区间，定义一个相交图为每个顶点表示一个区间，两个点有边当且仅当两个区间的交集非空。

一个图为区间图当它是若干个区间的相交图。

性质

①区间图一定是弦图

②给定n个区间的区间图为G，G的一个完美消除序列为：将所有的区间按照右端点从小到大排序。

应用

注：下面两个例子写起来都是很简单的贪心，不过相信大多数人开始时都没有相同是怎么证明的。

区间图的最大独立集

给定n个区间，要求选择最多的区间使得区间不互相重叠。

思路：将所有的区间按照右端点从小到大排序，依次处理每个区间，如果与已选区间没有重叠则选择该区间。复杂度$O(nlogn)$

区间图的最小染色

Tetris:有n个积木，高度均为1，第i个积木的宽度范围是$[L_i, R_i]$，选择一个积木的下落顺序使得最后积木总高度尽可能小。

思路：将所有的区间按照右端点从大到小排序，依次处理每个区间，选择一个可以染的最小的颜色染色。线段树或堆维护，时间复杂度$O(nlogn)$

参考资料

OI-Wiki

弦图和区间图-陈丹琦

https://blog.csdn.net/corsica6/article/details/88979383