Notes for ACMICPC World Finals 2014

ACMICPC World Finals 2014 **参考资料**

Chinese Edition **中文版**

*Shanghai Jiao Tong University* : **Secret;Weapon**



***Coach* 教 练**

Yong YU 俞 勇

***Team member* 队 员**

Xiaoxu GUO 郭晓旭

Qi LIU 刘 奇

Haobin NI 倪昊斌

目录

[KM 2](#_Toc419028903)

[费用流 2](#_Toc419028904)

[无向图最小割 3](#_Toc419028905)

[一般图最大匹配\_片段 3](#_Toc419028906)

[有向图最小生成树 4](#_Toc419028907)

[Hopcroft 5](#_Toc419028908)

[Manacher 5](#_Toc419028909)

[ExtKMP 5](#_Toc419028910)

[SA 6](#_Toc419028911)

[最大团搜索算法 7](#_Toc419028912)

[极大团的计数 7](#_Toc419028913)

[字符串的最小表示 8](#_Toc419028914)

[Exact Cover 8](#_Toc419028915)

[后缀自动机 9](#_Toc419028916)

[弦图的完美消除序列 10](#_Toc419028917)

[综合 10](#_Toc419028918)

## KM

**const** **int** maxn=**200**;**const** **int** oo=**0x7fffffff**;

**int** w[maxn][maxn],x[maxn],y[maxn],px[maxn],py[maxn],sy[maxn],slack[maxn];

int par[maxn];**int** n;**int** pa[**200**][**2**],pb[**200**][**2**],n0,m0,na,nb;**char** s[**200**][**200**];

**void** adjust(**int** v){ sy[v]=py[v]; **if** (px[sy[v]]!=-**2**) adjust(px[sy[v]]);}

**bool** find(**int** v){**for** (int i=**0**;i<n;i++)

**if** (py[i]==-**1**){

**if** (slack[i]>x[v]+y[i]-w[v][i]){

slack[i]=x[v]+y[i]-w[v][i]; par[i]=v;}

**if** (x[v]+y[i]==w[v][i]){

py[i]=v; **if** (sy[i]==-**1**){adjust(i); **return** **1**;}

**if** (px[sy[i]]!=-**1**) **continue**; px[sy[i]]=i;

**if** (find(sy[i])) **return** **1**;

}}**return** **0**;}

**int** km(){**int** i,j,m;

**for** (i=**0**;i<n;i++) sy[i]=-**1**,y[i]=**0**;

**for** (i=**0**;i<n;i++) {x[i]=**0**; **for** (j=**0**;j<n;j++) x[i]=max(x[i],w[i][j]);}

**bool** flag;

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

**for** (j=**0**;j<n;j++) px[j]=py[j]=-**1**,slack[j]=oo;

px[i]=-**2**; **if** (find(i)) **continue**; flag=**false**;

**for** (;!flag;){

m=oo; **for** (j=**0**;j<n;j++) **if** (py[j]==-**1**) m=min(m,slack[j]);

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

**if** (px[j]!=-**1**) x[j]-=m;

**if** (py[j]!=-**1**) y[j]+=m;

**else** slack[j]-=m;}

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

**if** (py[j]==-**1**&&!slack[j]){

py[j]=par[j];

**if** (sy[j]==-**1**){ adjust(j); flag=**true**; **break**;}

px[sy[j]]=j; **if** (find(sy[j])){flag=**true**;**break**;}

}}}}

**int** ans=**0**; **for** (i=**0**;i<n;i++) ans+=w[sy[i]][i];**return** ans;}

## 费用流

int modlable**()** **{**

int delta **=** inf**;**

**for(**int i **=** 1**;** i **<=** T**;** i**++)** **{**

**if** **(!**visit**[**i**]** **&&** slack**[**i**]** **<** delta**)**

delta **=** slack**[**i**];**

slack**[**i**]** **=** inf**;**

**}**

**if** **(**delta **==** inf**)**

**return** 1**;**

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

**if** **(**visit**[**i**])**

dis**[**i**]** **+=** delta**;**

**return** 0**;**

**}**

int dfs**(**int x**,** int flow**)** **{**

**if** **(**x **==** T**)** **{**

totFlow **+=** flow**;**

totCost **+=** flow **\*** **(**dis**[**S**]** **-** dis**[**T**]);**

**return** flow**;**

**}**

visit**[**x**]** **=** 1**;**

int left **=** flow**;**

**for(**int i **=** e**.**last**[**x**];** **~**i**;** i **=** e**.**succ**[**i**])**

**if** **(**e**.**cap**[**i**]** **>** 0 **&&** **!**visit**[**e**.**other**[**i**]])** **{**

int y **=** e**.**other**[**i**];**

**if** **(**dis**[**y**]** **+** e**.**cost**[**i**]** **==** dis**[**x**])** **{**

int delta **=** dfs**(**y**,** min**(**left**,** e**.**cap**[**i**]));**

e**.**cap**[**i**]** **-=** delta**;**

e**.**cap**[**i **^** 1**]** **+=** delta**;**

left **-=** delta**;**

**if** **(!**left**)** **{**

visit**[**x**]** **=** 0**;**

**return** flow**;**

**}**

**}** **else** **{**

slack**[**y**]** **=** min**(**slack**[**y**],** dis**[**y**]** **+** e**.**cost**[**i**]** **-** dis**[**x**]);**

**}**

**}**

**return** flow **-** left**;**

**}**

pair**<**int**,** int**>** minCost**()** **{**

totFlow **=** 0**,** totCost **=** 0**;**

fill**(**dis **+** 1**,** dis **+** T **+** 1**,** 0**);**

**do** **{**

**do** **{**

fill**(**visit **+** 1**,** visit **+** T **+** 1**,** 0**);**

**}** **while(**dfs**(**S**,** inf**));**

**}** **while(!**modlable**());**

**return** make\_pair**(**totFlow**,** totCost**);**

**}**

## 无向图最小割

#define typec int // type of res (or long long)

**const** typec inf = **0x3f3f3f3f**; // max of res

**const** typec maxw = **1000**; // maximum edge weight, g[i][j]=g[j][i]

typec g[V][V], w[V]; **int** a[V], v[V], na[V];

typec mincut(**int** n){

**int** i, j, pv, zj; typec best = maxw \* n \* n;

**for** (i = **0**; i < n; i++) v[i] = i; // vertex: 0 ~ n-1

**while** (n > **1**) {

**for** (a[v[**0**]] = **1**, i = **1**; i < n; i++) {

a[v[i]] = **0**; na[i - **1**] = i; w[i] = g[v[**0**]][v[i]];}

**for** (pv = v[**0**], i = **1**; i < n; i++ ) {

**for** (zj = -**1**, j = **1**; j < n; j++ )

**if** (!a[v[j]] && (zj < **0** || w[j] > w[zj])) zj = j;

a[v[zj]] = **1**;

**if** (i == n - **1**) {

**if** (best > w[zj]) best = w[zj];

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

g[v[i]][pv] = g[pv][v[i]]+=g[v[zj]][v[i]];

v[zj] = v[--n]; **break**;

} pv = v[zj];

**for** (j = **1**; j < n; j++) **if**(!a[v[j]]) w[j] += g[v[zj]][v[j]];

}} **return** best;}

## 一般图最大匹配\_片段

**const** **int** maxn=**310**;

vector<**int**> link[maxn];

**int** n; **int** match[maxn]; **int** Queue[maxn], head, tail; **int** pred[maxn], base[maxn];

**bool** InQueue[maxn], InBlossom[maxn]; **bool** use[maxn]; //===这个点是否有用

**int** start, finish; **int** newbase;

**void** push(**int** u) { Queue[tail++] = u; InQueue[u] = **true**; }

**int** pop() { **return** Queue[head++];}

**int** FindCommonAncestor(**int** u, **int** v) {

**bool** InPath[maxn]; **for** (**int** i = **0**; i < n; i++) InPath[i] = **0**;

**while**(**true**) {

u = base[u]; InPath[u] = **true**;

**if**(u == start) **break**; u = pred[match[u]];}

**while**(**true**) {v = base[v]; **if**(InPath[v]) **break**; v = pred[match[v]]; }

**return** v;}

**void** ResetTrace(**int** u) {

**int** v;

**while**(base[u] != newbase) {

v = match[u]; InBlossom[base[u]] = InBlossom[base[v]] = **true**;

u = pred[v]; **if**(base[u] != newbase) pred[u] = v;}}

**void** BlossomContract(**int** u, **int** v) {

newbase = FindCommonAncestor(u, v);

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

ResetTrace(u); ResetTrace(v);

**if**(base[u] != newbase) pred[u]=v;**if**(base[v] != newbase) pred[v]=u;

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

**if**(InBlossom[base[i]]) {base[i]=newbase; **if**(!InQueue[i]) push(i);}}

**bool** FindAugmentingPath(**int** u) {

**bool** found = **false**;

**for**(**int** i = **0**; i < n; ++i) pred[i] = -**1**, base[i] = i;

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

start = u; finish = -**1**; head = tail = **0**; push(start);

**while**(head < tail) {

**int** u = pop();

**for**(**int** i = link[u].size() - **1**; i >= **0**; i--) {

**int** v = link[u][i];

**if**(use[u] && use[v] && base[u] != base[v] && match[u] != v)

**if**(v == start || (match[v] >= **0** && pred[match[v]] >= **0**))

BlossomContract(u, v);

**else** **if**(pred[v] == -**1**) {pred[v] = u;

**if**(match[v] >= **0**) push(match[v]);

**else** {finish = v; **return** **true**;}

}}} **return** found;}

**void** AugmentPath() {

**int** u, v, w; u = finish;

**while**(u >= **0**) { v = pred[u];w = match[v];match[v] = u; match[u] = v;u = w;}}

**void** FindMaxMatching() {

**for**(**int** i = **0**; i < n; ++i) match[i] = -**1**;

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

**if**(match[i] == -**1** && use[i])**if**(FindAugmentingPath(i)) AugmentPath();}

**int** main() {

**foru**(i,**0**,n) link[i].clear(); memset(use,**1**,**sizeof**(use));

//========编号从0~n-1 ， link[i] push\_back所有i号点连向的点。 双向边

FindMaxMatching(); k=**0**;**rep**(i,n) **if** (match[i]>=**0**) k++;

printf(**"%d\n"**,k/**2**); **return** **0**;

}

## 有向图最小生成树

**const** **int** maxn=**1100**; **int** n,m , g[maxn][maxn] , used[maxn] , pass[maxn] ;

**int** eg[maxn] , more , queue[maxn];

**void** combine (**int** id , **int** &sum ) {

**int** tot = **0** , from , i , j , k ;

**for** ( ; id!=**0** && !pass[ id ] ; id=eg[id] ) {

queue[tot++]=id ; pass[id]=**1**; }

**for** ( from=**0**; from<tot && queue[from]!=id ; from++);

**if** ( from==tot ) **return** ;

more = **1** ;

**for** ( i=from ; i<tot ; i++) {

sum+=g[eg[queue[i]]][queue[i]] ;

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

used[queue[i]]=**1**;

**for** ( j = **1** ; j <= n ; j++) **if** ( !used[j] )

**if** ( g[queue[i]][j]<g[id][j] ) g[id][j]=g[queue[i]][j] ;}}

**for** ( i=**1**; i<=n ; i++) **if** ( !used[i] && i!=id ) {

**for** ( j=from ; j<tot ; j++){ k=queue[j];

**if** ( g[i][id]>g[i][k]-g[eg[k]][k] ) g[i][id]=g[i][k]-g[eg[k]][k]; }}}

**int** mdst( **int** root ) { // return the total length of MDST

**int** i , j , k , sum = **0** ;

memset ( used , **0** , **sizeof** ( used ) ) ;

**for** ( more =**1**; more ; ) {

more = **0** ; memset (eg,**0**,**sizeof**(eg)) ;

**for** ( i=**1** ; i <= n ; i ++) **if** ( !used[i] && i!=root ) {

**for** ( j=**1** , k=**0** ; j <= n ; j ++) **if** ( !used[j] && i!=j )

**if** ( k==**0** || g[j][i] < g[k][i] ) k=j ;

eg[i] = k ;

} memset(pass,**0**,**sizeof**(pass));

**for** ( i=**1**;i<=n;i++) **if** (!used[i] && !pass[i] && i!= root )combine(i,sum);

}

**for** ( i =**1**; i<=n ; i ++) **if** ( !used[i] && i!= root ) sum+=g[eg[i]][i];

**return** sum ; }

**int** main(){

**int** i,j,k,test,cases; cases=**0**; scanf(**"%d%d"**,&n,&m);

**foru**(i,**1**,n) **foru**(j,**1**,n) g[i][j]=**1000001**;

**foru**(i,**1**,m) {scanf(**"%d%d"**,&j,&k);j++;k++;scanf(**"%d"**,&g[j][k]);}

k=mdst(**1**); **if** (k>**1000000**) printf(**"Possums!\n"**); //===no

**else** printf(**"%d\n"**,k); **return** **0**;}

## Hopcroft

#define maxn 50005 #define maxm 150005

**inline** **int** Maxmatch(){

memset(mk,-**1**,**sizeof**(mk));memset(cx,-**1**,**sizeof**(cx));

memset(cy,-**1**,**sizeof**(cy));

**for** (**int** p=**1**,fl=**1**,h,tail;fl;++p){

fl=**0**; h=tail=**0**;

**for** (**int** i=**0**;i<n;++i) **if** (cx[i]==-**1**)

q[++tail]=i,pre[i]=-**1**,src[i]=i;

**for** (h=**1**;h<=tail;++h){

**int** u=q[h]; **if** (cx[src[u]]!=-**1**) **continue**;

**for** (**int** pp=head[u],v=vtx[pp];pp;pp=next[pp],v=vtx[pp])

**if** (mk[v]!=p) { mk[v]=p; q[++tail]=cy[v];

**if** (cy[v]>=**0**) {

pre[cy[v]]=u; src[cy[v]]=src[u];**continue**;

} **int** d,e,t;

**for** (--tail,fl=**1**,d=u,e=v;d!=-**1**;t=cx[d],cx[d]=e,cy[e]=d,e=t,d=pre[d]);

**break**; } } }

**int** res=**0**; **for** (**int** i=**0**;i<n;++i) res+=(cx[i]!=-**1**);**return** res;}

## Manacher

**void** manacher**(char** text**[],** **int** n**,** **int** palindrome**[])** **{**

palindrome**[**0**]** **=** 1**;**

**for** **(int** i **=** 1**,** j **=** 0**,** i **<** **(**n **<<** 1**)** **-** 1**;** **++** i**)** **{**

**int** p **=** i **>>** 1**;**

**int** q **=** i **-** p**;**

**int** r **=** **(**j **+** 1 **>>** 1**)** **+** palindrome**[**j**]** **-** 1**;**

palindrome**[**i**]** **=** r **<** q**?** 0**:** min**(**r **-** q **+** 1**,** palindrome**[(**j **<<** 1**)** **-** i**]);**

**while** **(**0 **<=** p **-** palindrome**[**i**]** **&&** q **+** palindrome**[**i**]** **<** n

**&&** text**[**p **-** palindrome**[**i**]]** **==** text**[**q **+** palindrome**[**i**]])** **{**

palindrome**[**i**]** **++;**

**}**

**if** **(**q **+** palindrome**[**i**]** **-** 1 **>** r**)** **{**

j **=** i**;**

**}**

**}**

**}**

## ExtKMP

void ExtendedKMP(char \*a, char \*b, int M, int N, int \*Next, int \*ret) {// a -> 模式串 b -> 匹配串  
    int i, j, k;  
    for (j = 0; 1 + j < M && a[j] == a[1 + j]; j++);  
    Next[1] = j;  
    k = 1;  
    for (i = 2; i < M; i++) {  
        int Len = k + Next[k], L = Next[i - k];  
        if (L < Len - i) {  
            Next[i] = L;  
        } else {  
            for (j = max(0, Len - i); i + j < M && a[j] == a[i + j]; j++);  
            Next[i] = j;  
            k = i;  
        }  
    }  
    for (j = 0; j < N && j < M && a[j] == b[j]; j++);  
    ret[0] = j;  
    k = 0;  
    for (i = 1; i < N; i++) {  
        int Len = k + ret[k], L = Next[i - k];  
        if (L < Len - i) {  
            ret[i] = L;  
        } else {  
            for (j = max(0, Len - i); j < M && i + j < N && a[j] == b[i + j]; j++);  
            ret[i] = j;  
            k = i;  
        }  
    }  
}

## SA

int n,a[20010],sa[20010],rank[20010],height[20010];

void build()

{

a[n+1]=-1;

void sort(int \*);

int count(int \*,int \*);

int b[20010],c[20010];

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

c[i]=a[i],b[i]=-1,sa[i]=i;

sort(c),count(c,b);

for (int k=1;;k<<=1)

{

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

c[i]=rank[i],b[i]=i+k<=n?rank[i+k]:0;

sort(b),sort(c);

if (count(c,b)>=n)

break;

}

k=0;

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

{

k=k?k-1:0;

if (rank[i]==1)

{

height[rank[i]]=0;

continue;

}

int p=sa[rank[i]-1],q=sa[rank[i]];

while (a[p+k]==a[q+k])

k++;

height[rank[i]]=k;

}

}

void sort(int \*a)

{

int f[20010],x[20010],t=0;

memset(f,0,sizeof(f));

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

f[a[i]]++,t=max(t,a[i]);

for (int i=1;i<=t;i++)

f[i]+=f[i-1];

for (int i=n;i>=1;i--)

x[f[a[sa[i]]]--]=sa[i];

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

sa[i]=x[i];

}

int count(int \*a,int \*b)

{

rank[sa[1]]=1;

int t=1;

for (int i=2;i<=n;i++)

{

if (a[sa[i]]!=a[sa[i-1]] || b[sa[i]]!=b[sa[i-1]])

t++;

rank[sa[i]]=t;

}

return(t);

}

## 最大团搜索算法

Int g[][]为图的邻接矩阵。 MC(V)表示点集V的最大团

令Si={vi, vi+**1**, ..., vn}, mc[i]表示MC(Si). 倒着算mc[i]，那么显然MC(V)=mc[**1**]

此外有mc[i]=mc[i+**1**] **or** mc[i]=mc[i+**1**]+**1**

**void** init(){

**int** i, j;**for** (i=**1**; i<=n; ++i) **for** (j=**1**; j<=n; ++j) scanf(**"%d"**, &g[i][j]);

}

**void** dfs(**int** size){

**int** i, j, k;

**if** (len[size]==**0**) { **if** (size>ans) { ans=size; found=**true**;} **return**;}

**for** (k=**0**; k<len[size] && !found; ++k) {

**if** (size+len[size]-k<=ans) **break**;

i=list[size][k]; **if** (size+mc[i]<=ans) **break**;

**for** (j=k+**1**, len[size+**1**]=**0**; j<len[size]; ++j)

**if** (g[i][list[size][j]]) list[size+**1**][len[size+**1**]++]=list[size][j];

dfs(size+**1**);}}

**void** work(){

**int** i, j; mc[n]=ans=**1**;

**for** (i=n-**1**; i; --i) { found=**false**; len[**1**]=**0**;

**for** (j=i+**1**; j<=n; ++j) **if** (g[i][j]) list[**1**][len[**1**]++]=j;

dfs(**1**); mc[i]=ans;}}

## 极大团的计数

Bool g[][] 为图的邻接矩阵，图点的标号由1至n。

**void** dfs(**int** size){

**int** i, j, k, t, cnt, best = **0**; **bool** bb;

**if** (ne[size]==ce[size]){**if** (ce[size]==**0**) ++ans;**return**;}

**for** (t=**0**, i=**1**; i<=ne[size]; ++i) {

**for** (cnt=**0**, j=ne[size]+**1**; j<=ce[size]; ++j)

**if** (!g[list[size][i]][list[size][j]]) ++cnt;

**if** (t==**0** || cnt<best) t=i, best=cnt;

}

**if** (t && best<=**0**) **return**;

**for** (k=ne[size]+**1**; k<=ce[size]; ++k) {

**if** (t>**0**){

**for** (i=k; i<=ce[size]; ++i)

**if** (!g[list[size][t]][list[size][i]]) **break**;

swap(list[size][k], list[size][i]);

}

i=list[size][k]; ne[size+**1**]=ce[size+**1**]=**0**;

**for** (j=**1**; j<k; ++j)**if** (g[i][list[size][j]])

list[size+**1**][++ne[size+**1**]]=list[size][j];

**for** (ce[size+**1**]=ne[size+**1**], j=k+**1**; j<=ce[size]; ++j)

**if** (g[i][list[size][j]]) list[size+**1**][++ce[size+**1**]]=list[size][j];

dfs(size+**1**); ++ne[size]; --best;

**for** (j=k+**1**, cnt=**0**; j<=ce[size]; ++j) **if** (!g[i][list[size][j]]) ++cnt;

**if** (t==**0** || cnt<best) t=k, best=cnt;

**if** (t && best<=**0**) **break**;

}}

**void** work(){

**int** i; ne[**0**]=**0**; ce[**0**]=**0**; **for** (i=**1**; i<=n; ++i) list[**0**][++ce[**0**]]=i;

ans=**0**; dfs(**0**);}

## 字符串的最小表示

A[1..n]; A[n+1..n+n]=A[1..n]; i:=**1**; j:=**2**; k:=**0**; t:=**0**;

while (j<=n) { k=**0**; **while** (a[i+k]=a[j+k]) k++;

**if** (a[i+k]>a[j+k]) i=i+k+**1;** **else** j=j+k+**1**;

**if** (i==j) j++; **if** (i>j) swap(i,j);

} printf(“%d\n”,i);

## Exact Cover

void del(int x)

{

a[a[x].l].r=a[x].r;

a[a[x].r].l=a[x].l;

for (int i=a[x].d;i!=x;i=a[i].d)

for (int j=a[i].r;j!=i;j=a[j].r)

{

sum[a[j].y]--;

a[a[j].u].d=a[j].d;

a[a[j].d].u=a[j].u;

}

}

void renew(int x)

{

a[a[x].l].r=x;

a[a[x].r].l=x;

for (int i=a[x].u;i!=x;i=a[i].u)

for (int j=a[i].l;j!=i;j=a[j].l)

{

sum[a[j].y]++;

a[a[j].u].d=j;

a[a[j].d].u=j;

}

}

bool search()

{

if (a[0].r==0)

return(true);

int k,min=20000000;

for (int i=a[0].r;i!=0;i=a[i].r)

if (sum[i]<min)

min=sum[k=i];

del(k);

for (int i=a[k].d;i!=k;i=a[i].d)

{

for (int j=a[i].r;j!=i;j=a[j].r)

del(a[j].y);

if (search())

return(true);

for (int j=a[i].l;j!=i;j=a[j].l)

renew(a[j].y);

}

renew(k);

return(false);

}

void del(int x)

{

for (int i=a[x].d;i!=x;i=a[i].d)

{

sum[a[i].y]--;

a[a[i].l].r=a[i].r;

a[a[i].r].l=a[i].l;

}

}

void renew(int x)

{

for (int i=a[x].u;i!=x;i=a[i].u)

{

sum[a[i].y]++;

a[a[i].l].r=i;

a[a[i].r].l=i;

}

}

bool vis[60];

int best()

{

memset(vis,0,sizeof(vis));

int ans=0;

for (int i=a[0].r;i!=0;i=a[i].r)

{

if (vis[i])

continue;

ans++;

for (int j=a[i].d;j!=i;j=a[j].d)

for (int k=a[j].r;k!=j;k=a[k].r)

vis[a[k].y]=true;

}

return(ans);

}

bool DLX(int dep)

{

if (a[0].r==0)

return(true);

if (dep+best()>m)

return(false);

int k,mi=1<<30;

for (int i=a[0].r;i!=0;i=a[i].r)

if (sum[i]<mi)

mi=sum[k=i];

for (int i=a[k].d;i!=k;i=a[i].d)

{

del(i);

for (int j=a[i].r;j!=i;j=a[j].r)

del(j);

if (DLX(dep+1))

return(true);

for (int j=a[i].l;j!=i;j=a[j].l)

renew(j);

renew(i);

}

return(false);

}

## 后缀自动机

**struct** State **{**

**static** vector **<**State**\*>** states**;**

**int** id**,** length**;**

State **\***parent**;**

State**\*** go**[**C**];**

State**(int** length**)** **:** id**((int)**states**.**size**()),** length**(**length**),** parent**(NULL)** **{**

memset**(**go**,** **NULL,** **sizeof(**go**));**

states**.**push\_back**(this);**

**}**

State**\*** extend**(**State**\*** start**,** **int** token**)** **{**

State **\***p **=** **this;**

State **\***np **=** **new** State**(**length **+** 1**);**

**while** **(**p **&&** **!**p**->**go**[**token**])** **{**

p**->**go**[**token**]** **=** np**;**

p **=** p**->**parent**;**

**}**

**if** **(!**p**)** **{**

np**->**parent **=** start**;**

**}** **else** **{**

State **\***q **=** p**->**go**[**token**];**

**if** **(**p**->**length **+** 1 **==** q**->**length**)** **{**

np**->**parent **=** q**;**

**}** **else** **{**

State **\***nq **=** **new** State**(**p**->**length **+** 1**);**

memcpy**(**nq**->**go**,** q**->**go**,** **sizeof(**q**->**go**));**

nq**->**parent **=** q**->**parent**;**

np**->**parent **=** q**->**parent **=** nq**;**

**while** **(**p **&&** p**->**go**[**token**]** **==** q**)** **{**

p**->**go**[**token**]** **=** nq**;**

p **=** p**->**parent**;**

**}**

**}**

**}**

**return** np**;**

**}**

**};**

## 弦图的完美消除序列

从n到1的顺序依次给点标号（标号为i的点出现在完美消除序列的第i个）

设lable[i]表示第i个点与多少已标号的点相邻，每次选择label[i]最大的未标号点进行标号。

任取一个已标号的与当前新标号的点相邻的点，如果与其他的已标号的且与当前点相邻的点之间没有边，则无解。

1.团数 ≤ 色数

2.最大独立集数 ≤ 最小团覆盖数

3.任何一个弦图都至少有一个单纯点，不是完全图的弦图至少有两个不相邻的单纯点。

4.设第i个点在弦图的完美消除序列第p(i)个。令N(v) = {w | w与v相邻且p(w) > p(v)}弦图的极大团一定是v∪N(v)的形式。

5.弦图最多有n个极大团。

6.设next(v) 表示N(v)中最前的点。令w\*表示所有满足A∈B的w中最后的一个点。判断v∪N(v)是否为极大团,只需判断是否存在一个w，满足Next(w) = v且|N(v)| + 1 ≤ |N(w)|即可。

7.最小染色：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的颜色。//团数=色数

8.最大独立集：完美消除序列从前往后能选就选。

9.最小团覆盖：设最大独立集为{p1 , p2 , …, pt}，则{p1∪N(p1), …, pt∪N(pt)}为最小团覆盖。 //最大独立集数 = 最小团覆盖数!!!

## 综合

有上下界网络流，可行流增广的流量不是实际流量。若要求实际流量应该强算一遍源点出去的流量。

求最小下届网络流： 方法一：加t-s的无穷大流，求可行流，然后把边反向后（减去下届网络流），在残留网络中从汇到源做最大流。

方法二：在求可行流的时候，不加从汇到源的无穷大边，得到最大流X， 加上从汇到源无穷大边后，再求最大流得到Y。那么Y即是答案最小下届网络流。