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)

set nocompatible sta hls wrap ruler cindent nu nobackup noswapfile autoindent ts=4 noet sts=4 sw=4

syntax on

autocmd FileType c,cpp nmap <F8> <ESC>:w <CR><ESC>:!g++ % -o %< <CR>

autocmd FileType c,cpp nmap <F9> :!time ./%< <./%<.in <CR>

autocmd FileType c,cpp nmap <F10> :!time ./%< <CR>

nmap <F2> :vs %<.in <CR>

## 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

typedef **int** ValueType;

**const** ValueTyep MOD = 0x3f3f3f3f3f3f3f3fLL;

ValueType flow, cost, value, dist[Maxn];

**int** visit[Maxn], src, des;

deque<**int**> Q;

void adde(**int** u, **int** v, ValueType c, ValueType w) {}

ValueType Aug(**int** u, ValueType m) {

if(u == des) {

cost += value \* m; flow += m;

return m; }

visit[u] = true; **int** j, v;

ValueType l = m, c, w, del;

for(j = last[u]; j != -1; j = e[j].next) {

v = e[j].v; c = e[j].c; w = e[j].w;

if(c && !w && !visit[v]) {

del = Aug(v, l < c ? l : c);

e[j].c -= del; e[j ^ 1].c += del; l -= del;

if(!l) return m; } }

return m - l; }

bool Modlabel(**int** src, **int** des, **int** n) {

**int** i, j, u, v; ValueType c, w, del;

memset(dist, 0x3f, sizeof(dist[0])\*(n + 3));

while(!Q.empty()) Q.pop\_back();

dist[src] = 0; Q.push\_back(src);

while(!Q.empty()) {

u = Q.front(); Q.pop\_front();

for(j = last[u]; j != -1; j = e[j].next) {

v = e[j].v; c = e[j].c; w = e[j].w;

if(c && (del = dist[u] + w) < dist[v]) {

dist[v] = del;

if(Q.empty() || del <= dist[Q.front()]) Q.push\_front(v);

else Q.push\_back(v); } } }

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

for(j = last[i]; j != -1; j = e[j].next)

e[j].w -= dist[e[j].v] - dist[i];

}

value += dist[des];

return dist[des] < MOD; }

void zkw(**int** src, **int** des, **int** n) {

value = cost = flow = 0;

while(Modlabel(src, des, n)){

do {

memset(visit, 0, sizeof(visit[0]) \* (n + 3));

}while(Aug(src, MOD)); } }

## 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++;

pr**int**f(**"%d\n"**,k/**2**); **return** **0**;

}

## 有向图最小生成树

/\* O(VE),根不固定，添加一个根节点与所有点连无穷大的边！

\* 如果求出比2\*MOD大, 则不连通; 根和虚拟根相连的结点

\* 根据pre的信息能构造出这棵树！

\* 注意结点必须从0~n-1\*/

typedef **int** mytype; mytype inv[Maxn];

**int** visit[Maxn], pre[Maxn], belong[Maxn], ROOT;

mytype dirtree(**int** n, **int** m, **int** root) {

mytype sum = 0; **int** i, j, k, u, v;

while (1) {

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

inv[i] = MOD; pre[i] = -1; belong[i] = -1; visit[i] = -1; }

inv[root] = 0;

for (i = 0; i < m; i++) {//除原点外，找每个点的最小入边

u = e[i].u; v = e[i].v;

if (u != v) {

if (e[i].w < inv[v]) {

inv[v] = e[i].w; pre[v] = u;

if(u == root) ROOT = i; //记录根所在的边,输出根时利用ROOT-m计算是原图哪个结点

} } }

for (i = 0; i < n; i++) if (inv[i] == MOD) return -1;

**int** num = 0;

for (i = 0; i < n; i++) { //找圈，收缩圈

if (visit[i] == -1) {

j = i;

for(j = i; j != -1 && visit[j] == -1 && j != root; j = pre[j]) visit[j] = i;

if (j != -1 && visit[j] == i) {

for (k = pre[j]; k != j; k = pre[k]) belong[k] = num;

belong[j] = num ++ ; } }

sum += inv[i]; }

if (num == 0) return sum;

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

if (belong[i] == -1) belong[i] = num ++ ;

for (i = 0; i < m; i++) { //重新构图

e[i].w = e[i].w - inv[e[i].v]; e[i].v = belong[e[i].v];

e[i].u = belong[e[i].u]; }

n = num; root = belong[root]; } }

## Hopcroft

int n, m, match = 0; queue<int> Q;

int mx[Maxn], my[Maxn], dx[Maxn], dy[Maxn], dis, visit[Maxn];

int ux[Maxn], uy[Maxn], px[Maxn], py[Maxn], pv[Maxn];

void adde(int u, int v) {}

bool searchPath() {

int i, j; dis = MOD; for(i = 0; i < n; i++) dx[i] = -1;

for(j = n; j < n + m; j++) dy[j] = -1; while(!Q.empty()) Q.pop();

for(int i = 0; i < n; i++) if(-1 == mx[i]) Q.push(i);

int u, v;

while(!Q.empty()) {

u = Q.front(); Q.pop(); if(dx[u] > dis) break;

for(int j = last[u]; j != -1; j = e[j].next) {

v = e[j].v; if(-1 != dy[v]) continue; dy[v] = dx[u] + 1;

if(-1 == my[v]) dis = dy[v];

else {dx[my[v]] = dy[v] + 1; Q.push(my[v]);} } }

return dis != MOD; }

bool dfs(int u){ int v;

for(int j = last[u]; j != -1; j = e[j].next) {

v = e[j].v; if(visit[v] || dx[u] + 1 != dy[v]) continue;

if(dy[v] == dis && my[v] != -1) continue; visit[v] = true;

if(-1 == my[v] || dfs(my[v])){my[v] = u; mx[u] = v;return true;} } return false; }

int solve(){int i, j; match = 0;

for(i = 0; i < n; i++) mx[i] = -1;

for(j = n; j < n + m; j++) my[j] = -1;

while(searchPath()){

for(j = n; j < n + m; j++) visit[j] = false;

for(int i = 0; i < n; i++)if(-1 == mx[i] && dfs(i)) match++;}

return match; }

## 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);

} pr**int**f(“%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。