目录

[KM 1](#_Toc419229111)

[Hopcroft 1](#_Toc419229112)

[稳定婚姻匹配 1](#_Toc419229113)

[一般图最大匹配 2](#_Toc419229114)

[LCA 3](#_Toc419229115)

[点双连通(重建图) 3](#_Toc419229116)

[dinic 4](#_Toc419229117)

[费用流 4](#_Toc419229118)

[无向图最小割 4](#_Toc419229119)

[有向图最小生成树 5](#_Toc419229120)

[最大团搜索算法 5](#_Toc419229121)

[极大团的计数 6](#_Toc419229122)

[弦图的完美消除序列 6](#_Toc419229123)

[Manacher 7](#_Toc419229124)

[ExtKMP 7](#_Toc419229125)

[Aho-Corasick Automaton(部分代码) 7](#_Toc419229126)

[SA 8](#_Toc419229127)

[字符串的最小表示 8](#_Toc419229128)

[DLX 8](#_Toc419229129)

[Tips-Lquartz 9](#_Toc419229130)

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

typedef **double** ValueType;//结点0~n-1

const **int** maxn=200; const ValueType MOD=**(**1e20**)**;

ValueType x**[**maxn**]**, y**[**maxn**]**, w**[**maxn**][**maxn**]**, slack**[**maxn**]**;

**int** sy**[**maxn**]**, px**[**maxn**]**, py**[**maxn**]**, par**[**maxn**]**;

**int** pa**[**200**][**2**]**,pb**[**200**][**2**]**,n0,m0,na,nb;

char s**[**200**][**200**]**, n;

**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; **}**

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

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

**for** **(**i=0;i<n;i++**)** **{** x**[**i**]**=-MOD;

**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**]**=MOD;

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

**for** **(**;!flag;**){** m=MOD;

**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**;**}**

**}** **}** **}** **}** ValueType ans=0;

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

## 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, u, v; 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(**i = 0; i < n; i++**)** **if(**-1 == mx**[**i**])** Q.push**(**i**)**;

**while(**!Q.empty**())** **{**

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

**for(**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; **}**

## 稳定婚姻匹配

//延迟认可算法**(**Gale-Shapley算法**)**

**int** mx**[**Maxn**]**, my**[**Maxm**]**, cur**[**Maxn**]**, n, m; //x匹配的y, y匹配的x

//yorder表示在x眼中y的顺序,0~m-1为喜爱度递减的y的编号

//xorder表示在y眼中x的顺序,0~n-1为编号0~n-1的x的重要度,越重要, 值越小

**int** yorder**[**Maxn**][**Maxm**]**, xorder**[**Maxm**][**Maxn**]**; queue<**int**> que;

**void** GaleShapley**()** **{** **int** i, j, v;

**for(**i = 0; i <= n; i++**)** mx**[**i**]** = -1,cur**[**i**]** = 0;//初始化

**for(**j = 0; j <= m; j++**)** my**[**j**]** = -1; **while(**!que.empty**())** que.pop**()**;

**for(**i = 0; i < n; i++**)** que.push**(**i**)**;//将x加入队列

**while(**!que.empty**())** **{**//x还有没找到朋友的

i = que.front**()**; que.pop**()**; **if(**cur**[**i**]** >= m**)** **continue**;

v = yorder**[**i**][**cur**[**i**]**++**]**;

**if(**my**[**v**]** == -1**)** **{**mx**[**i**]** = v; my**[**v**]** = i; **}** //y没有匹配

**else** **if(**xorder**[**v**][**i**]** < xorder**[**v**][**my**[**v**]])** **{**//i比之前的好

mx**[**my**[**v**]]** = -1; que.push**(**my**[**v**])**; my**[**v**]** = i; mx**[**i**]** = v; **}**

**else** que.push**(**i**)**;//i比以前的差, i找下一个y

**}**/\***while**\*/ **}**/\*func\*/

## 一般图最大匹配

**int** n, head, tail, Start, Finish, Q**[**Maxn**]**, mark**[**Maxn**]**;

**int** InBlossom**[**Maxn**]**, inqueue**[**Maxn**]**;**int** match**[**Maxn**]**;//表示哪个点匹配了哪个点 **int** father**[**Maxn**]**;//这个是增广路径的father

**int** base**[**Maxn**]**;//该点属于哪朵花 **bool** mp**[**Maxn**][**Maxn**]**; //邻接关系

**void** BlossomContract**(int** x,**int** y**){**memset**(**mark, **false**, sizeof**(**mark**))**;

memset**(**InBlossom, **false**, sizeof**(**InBlossom**))**;

#define pre father**[**match**[**i**]]**

**int** lca, i; **for(** i = x; i; i = pre**)** **{**i = base**[**i**]**; mark**[**i**]** = **true**; **}**

**for(**i = y; i; i = pre**)** **{**i = base**[**i**]**; //寻找lca

**if(**mark**[**i**])** **{**lca = i; **break**; **}** **}**

**for(**i = x; base**[**i**]** != lca; i = pre**)** **{**

**if(**base**[**pre**]** != lca**)** father**[**pre**]** = match**[**i**]**;

//对于BFS树中的父边是匹配边的点, father向后跳

InBlossom**[**base**[**i**]]** = **true**; InBlossom**[**base**[**match**[**i**]]]** = **true**; **}**

**for(**i = y; base**[**i**]** != lca; i = pre**)** **{**

**if(**base**[**pre**]** != lca**)** father**[**pre**]** = match**[**i**]**; // 同理

InBlossom**[**base**[**i**]]** = **true**; InBlossom**[**base**[**match**[**i**]]]** = **true**; **}**

#undef pre

**if(**base**[**x**]** != lca**)** father**[**x**]** = y; **if(**base**[**y**]** != lca**)** father**[**y**]** = x;

**for(**i = 1; i <= n; i++**)** **if(**InBlossom**[**base**[**i**]])** **{** base**[**i**]** = lca;

**if(**!inqueue**[**i**])** **{**Q**[**tail++**]** = i; inqueue**[**i**]** = **true**; **}** **}** **}**

**void** Change**()** **{** **int** x, y, z = Finish;

**while(**z**){**y= father**[**z**]**;x= match**[**y**]**;match**[**y**]** = z;match**[**z**]** = y; z = x;**}** **}**

**void** FindAugmentPath**()** **{** **int** i; memset**(**father, 0, sizeof**(**father**))**;

memset**(**inqueue, **false**, sizeof**(**inqueue**))**;

**for(**i = 1; i <= n; i++**)** base**[**i**]** = i;

head = tail = 0; Q**[**tail++**]** = Start; inqueue**[**Start**]** = 1;

**while(**head < tail**)** **{** **int** x = Q**[**head++**]**;

**for(int** y = 1; y <= n; y++**)**

**if(**mp**[**x**][**y**]** && base**[**x**]** != base**[**y**]** && match**[**x**]** != y**)** **{**

**if(**Start == y || match**[**y**]** && father**[**match**[**y**]])**

BlossomContract**(**x, y**)**;

**else** **if(**!father**[**y**])** **{** father**[**y**]** = x;

**if(**match**[**y**]){**Q**[**tail++**]**=match**[**y**]**;inqueue**[**match**[**y**]]**= **true**;**}**

**else** **{**Finish = y; Change**()**; return; **}** **}** **}** **}** **}**

**void** Edmonds**()** **{** memset**(**match, 0, sizeof**(**match**))**;

**for(**Start=1;Start<= n;Start++**)if(**match**[**Start**]**== 0**)** FindAugmentPath**()**;**}**

**void** output**()** **{** memset**(**mark, **false**, sizeof**(**mark**))**;

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

/\* printf**(**"%d\n", cnt**)**; //输出匹配关系

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

**if(**!mark**[**i**]** && match**[**i**])** **{** mark**[**i**]** = **true**;

mark**[**match**[**i**]]** = **true**; printf**(**"%d %d\n", i, match**[**i**])**;**}}** //\*/

**if(**cnt < n**)**printf**(**"NO\n"**)**;**else** printf**(**"YES\n"**)**; **}**

## LCA

//倍增算法加边之前使用initLCA**()**初始化数组

//调用solveLCA**()**初始化LCA, 调用getLCA**(**x,y**)**返回x和y的LCA

#define STEP 17

**void** initLCA**()** **{}**//初始化tot, last, depth, fa**[][]**

**void** dfsLCA**(int** u**)** **{}**//确定结点深度depth**[]**和fa**[**0**][**u**]**

**int** getLCA **(int** x, **int** y**)** **{** **int** i, d**if** = abs**(**depth**[**x**]** - depth**[**y**])**;

**if** **(**depth**[**x**]** < depth**[**y**])** swap**(**x, y**)**;

**for** **(**i = STEP - 1; i >= 0; i--**)** **{**

**if** **((**1 << i**)** & d**if)** **{** d**if** -= **(**1 << i**)**; x = fa**[**i**][**x**]**; **}}**

**for** **(**i = STEP - 1; i >= 0; i--**)** **{**

**if** **(**fa**[**i**][**x**]** != fa**[**i**][**y**])** **{**x = fa**[**i**][**x**]**; y = fa**[**i**][**y**]**; **}}**

**if** **(**x == y**)** return x; **else** return fa**[**0**][**x**]**; **}**

**void** solveLCA**(){int** i, j, root = 1;**for(**i = 0; i <= n; i++**)** depth**[**i**]** = -1;

fa**[**0**][**root**]** = root; depth**[**root**]** = 0; dfsLCA**(**root**)**;

**for** **(**i = 1; i < STEP; i++**)** **for(**j = 0; j <= n; j++**)**

fa**[**i**][**j**]** = fa**[**i-1**][**fa**[**i-1**][**j**]]**; **}**

//Tarjan离线LCA, 利用struct Graph, e存放树边, q存放query

struct GRAPH **{}**e, q;

**void** tarjanLCA**(int** u**)** **{** **int** i, j, v, f; fa**[**u**]** = u; visit**[**u**]** = 1;

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

**if(**!visit**[**v**])** **{** tarjanLCA**(**v**)**; fa**[**v**]** = u; **}** **}**

**for(**j = q.last**[**u**]**; j != -1; j = q.adj**[**j**]**.next**)** **{** v = q.adj**[**j**]**.v;

**if(**visit**[**v**])** **{**lca**[**q.adj**[**j**]**.n**]** = f = getfa**(**v**)**;

ans**[**q.adj**[**j**]**.n**]** = dist**[**v**]** + dist**[**u**]** - 2 \* dist**[**f**]**; **}** **}** **}**

//LCA转RMQ

graph: 1-2, 1- 7, 2-3, 2-4, 4-5, 5-6, 7-8

step 1: dfs遍历树, 依次记录每次到达的点, 以及每个点的深度得到序列:

结点访问顺序是: 1 2 3 2 4 5 4 6 4 2 1 7 8 7 1 //共2n-1个值

结点对应深度是: 0 1 2 1 2 3 2 3 2 1 0 1 2 1 0

step 2: 利用ST计算任意区间最小深度的点的ID

step 3: 对于每次查询, 查询u第一次出现位置到v第一次出现位置区间的最小值

## 点双连通(重建图)

**int** dfn**[**Maxn**]**, low**[**Maxn**]**, iscut**[**Maxn**]**, belong**[**Maxm**]**, color**[**Maxm**]**;

**int** lcnt**[**Maxm**]**, nindex, ncnt, visit**[**Maxm**]**, n, m, q;

stack<**int**> sta; set<PII> S;

**void** gao**(int** u**)** **{}**//遍历所有与u连通的点,标记visit**[**v**]**从-1变为1

**void** newAdde**(int** u, **int** v**)** **{if(**u > v**)** swap**(**u, v**)**; PII ss = MP**(**u, v**)**;

**if(**S.find**(**ss**)** == S.ED**)** **{**S.insert**(**ss**)**;e1.adde**(**u, v**)**;e1.adde**(**v, u**)**;**}}**

**void** Tarjan**(int** u,**int** from**){int** v, child = 0; dfn**[**u**]** = low**[**u**]** = ++nindex;

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

**if(**j == from**)** **continue**; v = e.adj**[**j**]**.v;

**if(**dfn**[**v**]** < dfn**[**u**])** **{** sta.push**(**j**)**;

**if(**!dfn**[**v**]){**child++;Tarjan**(**v,j^ 1**)**;low**[**u**]**= min**(**low**[**u**]**,low**[**v**])**;

**if(**low**[**v**]** >= dfn**[**u**])** **{** ncnt++;

**while(**sta.top**()**!=j**){**belong**[**sta.top**()**/2**]**=ncnt;sta.pop**()**;**}**

belong**[**j / 2**]** = ncnt; sta.pop**()**; iscut**[**u**]** = 1; **}**

**}** **else** low**[**u**]** = min**(**low**[**u**]**, dfn**[**v**])**; **}** **}**/\***for**\*/

**if(**from < 0 && child == 1**)**iscut**[**u**]** = -1; **}**//child

**void** buildGraph**()** **{int** i, j, u, v, x, y; e1.init**(**ncnt + 10**)**; S.clear**()**;

**for(**j = 0; j < e.tot; j += 2**)** **{**u = e.adj**[**j**]**.u; v = e.adj**[**j**]**.v;

**if(**iscut**[**u**]** == -1 && iscut**[**v**]** == -1**)** **continue**;

**else** **{if(**iscut**[**u**]**!=-1**){**x=iscut**[**u**]**;y = belong**[**j/2**]**;newAdde**(**x,y**)**;**}**

**if(**iscut**[**v**]**!=-1**){**x=iscut**[**v**]**;y=belong**[**j/2**]**;newAdde**(**x,y**)**;**}}}**

**for(**i = 0; i <= ncnt; i++**)** visit**[**i**]** = -1;

**for(**i = 1; i <= ncnt; i++**)** **{**

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

gao**(**i**)**; e1.adde**(**0, i**)**; e1.adde**(**i, 0**)**;**}}**/\***for**\*/ **}**/\*func\*/

**void** solve**()** **{int** i, j; memset**(**dfn, 0, sizeof**(**dfn**))**; ncnt = nindex = 0;

/\*memset low->0, iscur->-1, color->0, belong->-1\*/

**for(**i = 1; i <= n; i++**)** **{**

**if(**!dfn**[**i**])** **{while(**!sta.empty**())** sta.pop**()**;Tarjan**(**i, -1**)**; **}** **}**

**for(**i = 1; i <= n; i++**)** **{**

**if(**iscut**[**i**]** == 1**)** **{**color**[**++ncnt**]** = 1;iscut**[**i**]** = ncnt; **}**

**}**/\***for**\*/ buildGraph**()**; **}**

## dinic

bool bfs**(int** s, **int** t, **int** n**)** **{}**//注意对边.c!=0的判断

**int** dinic**(int** s, **int** t, **int** n**)** **{int** i, j, u, v; **int** maxflow = 0;

**while(**bfs**(**s, t, n**))** **{for(**i = 0; i < n; i++**)** cur**[**i**]** = last**[**i**]**;

u = s; top = 0;

**while(**cur**[**s**]** != -1**)** **{**

**if(**u == t**)** **{int** tp = MOD;//tp最小值,修改流量,修改top

u = e**[**sta**[**top**]]**.u; **}**

**else** **if(**cur**[**u**]** != -1 && e**[**cur**[**u**]]**.c > 0 && dist**[**u**]** + 1 == dist**[**e**[**cur**[**u**]]**.v**])** **{** sta**[**top++**]** = cur**[**u**]**; u = e**[**cur**[**u**]]**.v; **}**

**else** **{while(**u != s && cur**[**u**]** == -1**)**u = e**[**sta**[**--top**]]**.u;

cur**[**u**]** = e**[**cur**[**u**]]**.next; **}**

**}**/\***while(**cur**)**\*/ **}**/\***while** bfs\*/ return maxflow; **}**

## 费用流

**typedef** **int** ValueType; deque<**int**>Q; **const** ValueTyep MOD=0x3f3f3f3f3f3fLL;

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

**int** visit**[**Maxn**]**, src, des;//注意全局变量src,des必须初始化

**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

**typedef** **int** ValueType;//K连通块计数, 注意节点下标0~n-1

ValueType edge**[**Maxn**][**Maxn**]**, g**[**Maxn**][**Maxn**]**, minCut, maxi;

**int** n, m, k, S, T, top, sta**[**Maxn**]**, comb**[**Maxn**]**, node**[**Maxn**]**;

vector<**int**> parta, partb, belong**[**Maxn**]**;

ValueType Search **(int** n**)** **{int** i, j, u, vis**[**Maxn**]**;

ValueType wet**[**Maxn**]**,minCut= 0,maxi;**int** temp= -1,top= 0;S= -1,T= -1;

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

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

**for** **(**j = 0; j < n; j++**)** **{** u = node**[**j**]**;

**if(**!comb**[**u**]**&& !vis**[**u**]** && wet**[**u**]**> maxi**){**temp= u;maxi= wet**[**u**]**;**}}**

sta**[**top++**]** = temp;vis**[**temp**]** = **true**;**if(**i == n - 1**)** minCut = maxi;

**for** **(**j = 0; j < n; j++**)** **{** u = node**[**j**]**;

**if** **(**!comb**[**u**]** && !vis**[**u**])** wet**[**u**]** += edge**[**temp**][**u**]**; **}** **}**

S = sta**[**top - 2**]**; T = sta**[**top - 1**]**;

**for** **(**i = 0; i < top; i++**)** node**[**i**]** = sta**[**i**]**; return minCut; **}**

ValueType StoerWagner **(**vector<**int**> & li**)** **{**

**int** i, j, k, n = li.SZ, u, v, used**[**Maxn**]**;

ValueType cur, ans = MOD; memset**(**comb, 0, sizeof**(**comb**))**;

**for** **(**i = 0; i < n; i++**){**node**[**i**]** = i; belong**[**i**]**.clear**()**;belong**[**i**]**.PB**(**i**)**;**}**

**for** **(**i = 1; i < n; i++**)** **{**k = n - i + 1; cur = Search**(**k**)**;

**if** **(**cur < ans**)** **{** ans = cur; **for(**j = 0; j < n; j++**)** used**[**j**]** = 0;

**for(**j = 0; j < belong**[**T**]**.SZ; j++**)** used**[**belong**[**T**][**j**]]** = 1; **}**

**for(**j = 0; j < belong**[**T**]**.SZ; j++**)** belong**[**S**]**.PB**(**belong**[**T**][**j**])**;

**if** **(**ans == 0**)** **break**; comb**[**T**]** = **true**;

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

**if** **(**!comb**[**j**])** **{**edge**[**S**][**j**]** += edge**[**T**][**j**]**;

edge**[**j**][**S**]** += edge**[**j**][**T**]**; **}** **}** **}**

parta.clear**()**; partb.clear**()**;

**for(**j = 0; j < n; j++**)** **{if(**used**[**j**])** parta.PB**(**li**[**j**])**;

**else** partb.PB**(**li**[**j**])**; **}** return ans; **}**

**int** dfs**(**vector<**int**> &li**)** **{int** n = li.SZ, i, j;

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

edge**[**i**][**j**]** = g**[**li**[**i**]][**li**[**j**]]**;

**ValueType** cur = StoerWagner**(**li**)**; **if(**cur >= k**)** return 1;

vector<**int**> a**(**parta**)**, b**(**partb**)**; return dfs**(**a**)** + dfs**(**b**)**; **}**

## 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** ValueType; ValueType inv**[**Maxn**]**;

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

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

ValueType 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; //除原点外,找每个点的最小入边

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

**}**/\***if**\*/ **}**/\***if**\*/ **}**/\***for**\*/

**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**]**; **}** **}**

## 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**;//第size个点选择点i

**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)**;**}**

## 弦图的完美消除序列

最大势算法:简单的弦图判定,先求完美消除序列L,再利用L判断是否弦图

**int** adj**[**Maxn**][**Maxn**]**, n, m, L**[**Maxn**]**, cnt**[**Maxn**]**, visit**[**Maxn**]**, mpL**[**Maxn**]**;

priority\_queue<PII> que;

//利用MSC最大势算法求完美消除序列L,无合法序列返回**false**

**int** getList**()** **{** **int** i, j, k, u, v, w;

**for(**i = 1; i <= n; i++**)** cnt**[**i**]** = 0, visit**[**i**]** = 0;

**while(**!que.empty**())** que.pop**()**; que.push**(**MP**(**0, 1**))**; k = n;

**while(**!que.empty**())** **{**u = que.top**()**.BB; w = que.top**()**.AA; que.pop**()**;

**if(**w != cnt**[**u**])** **continue**; visit**[**u**]** = 1; mpL**[**u**]** = k; L**[**k--**]** = u;

**for(**v = 1; v <= n; v++**)** **if(**!visit**[**v**]** && adj**[**u**][**v**])** **{**

cnt**[**v**]**++; que.push**(**MP**(**cnt**[**v**]**, v**))**; **}** **}**

**if(**k < 1**)** return **true**; **else** return **false**; **}**

//利用完美消除序列判断是否弦图

**int** check**()** **{int** i, j, k, u, v, w;

**for(**i = n - 1; i >= 1; i--**)** **{**u = L**[**i**]**; k = -1;

**for(**j= i + 1; j <= n; j++**){**v = L**[**j**]**;**if(**adj**[**u**][**v**]){**k = v; **break**;**}** **}**

**if(**k != -1**)** **for(** j++; j <= n; j++**)** **{**

v = L**[**j**]**; **if(**adj**[**u**][**v**]** && !adj**[**k**][**v**])** return **false**; **}**

**}** return **true**; **}**

1. 极大团：此团不是其他团的子集 2. 最大团：点数最多的团 -> 团数

3. 最小染色：用最少的颜色给点染色使相邻点颜色不同 -> 色数

4. 最大独立集：原图点集的子集，任意两点在原图中没有边相连

6. 最小团覆盖：用最少个数的团覆盖所有的点

推论 -> 团数<=色数，最大独立集数<=最小团覆盖数

6.弦图：图中任意长度大于3的环都至少有1个弦

推论 -> 弦图的每一个诱导子图一定是弦图,弦图的任一个诱导子图不同构于Cn(n>3)

7.单纯点：记N(v)为点v相邻点的集合，若N(v)+{v}是一个团，则v为单纯点

引理 -> 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

8.弦图最多有n个极大团.

9.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.

10.完美消除序列：点的序列v1,v2,..,vn，满足vi在{vi,vi+1,..,vn}中是单纯点

定理 -> 一个无向图是弦图，当且仅当它有一个完美消除序列

构造算法 -> 令cnt[i]为第i个点与多少个已标记的点相邻，初值全为零,每次选择一个cnt[i]最大的结点并打上标记,标记顺序的逆序则为完美消除序列

判定算法 -> 对于每个vi，其出边为vi1,vi2,..,vik,然后判断vi1与vi2,vi3,..,vik是否都相邻,若存在不相邻的情况，则说明不是完美消除序列

11.弦图各类算法：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

最大独立集：完美消除序列从前往后能选就选.

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!!!

12.区间图：坐标轴上的一些区间看作点，任意两个交集非空的区间之间有边

定理: 区间图一定是弦图 \*/

13.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.

## Manacher

//s为原串, str为插入$和#的串, 读入s后, 调用init**(**s, str, len**)**,

//最后调用Manacher**(**str,p,len**)**, 求解遍历p数组求最大值, 注意输出ans-1

最长回文子串对应原串T中的位置:l = **(**i - p**[**i**])**/2; r = **(**i + p**[**i**])**/2 - 2;

**int** len, p**[**Maxn**]**; char s**[**Maxn**]**, str**[**Maxn**]**;

**void** init**(**char s**[]**, char str**[]**, **int**& len**)** **{**

**int** i, j, k; str**[**0**]** = '$'; str**[**1**]** = '#';

**for** **(**i = 0; i < len; i++**)** **{** str**[**i \* 2 + 2**]** = s**[**i**]**;

str**[**i \* 2 + 3**]** = '#'; **}**

len = len \* 2 + 2; s**[**len**]** = 0; **}**

**void** Manacher **(**char str**[]**, **int** p**[]**, **int** len**)** **{**

**int** i, mx = 0, id; **for** **(**i = len; i < Maxn; i++**)** str**[**i**]** = 0;

**for** **(**i = 1; i < len; i++**)** **{**

**if** **(** mx > i **)** p**[**i**]** = min **(** p**[**2 \* id - i**]**, p**[**id**]** + id - i **)**;

**else** p**[**i**]** = 1;

**for** **(**; str**[**i + p**[**i**]]** == str**[**i - p**[**i**]]**; p**[**i**]**++**)**;

**if** **(** p**[**i**]** + i > mx **)** **{**mx = p**[**i**]** + i; id = i;**}** **}** **}**

## ExtKMP

char S**[**Maxn**]**, T**[**Maxn**]**; **int** next**[**Maxn**]**, B**[**Maxn**]**;

**void** preExKmp**(**char T**[]**, **int** LT, **int** next**[])** **{**

**int** i, ind = 0, k = 1; next**[**0**]** = LT;

**while(**ind + 1 < LT && T**[**ind + 1**]** == T**[**ind**])** ind++; next**[**1**]** = ind;

**for(**i = 2; i < LT; i++**)** **{**

**if(**i <= k + next**[**k**]** - 1 && next**[**i - k**]** + i < k + next**[**k**])**

next**[**i**]** = next**[**i - k**]**;

**else** **{** ind = max**(**0, k + next**[**k**]** - i**)**;

**while(**ind + i < LT && T**[**ind + i**]** == T**[**ind**])** ind++;

next**[**i**]** = ind; k = i; **}** **}** **}**

**void** exKmp**(**char S**[]**, **int** LS, char T**[]**, **int** LT, **int** next**[]**, **int** B**[])** **{**

**int** i, ind = 0, k = 0; preExKmp**(**T, LT, next**)**;

**while(**ind < LS && ind < LT && T**[**ind**]** == S**[**ind**])** ind++; B**[**0**]** = ind;

**for(**i = 1; i < LS; i++**)** **{** **int** p = k + B**[**k**]** - 1, L = next**[**i - k**]**;

**if((**i - 1**)** + L < p**)** B**[**i**]** = L; **else** **{** ind = max**(**0, p - i + 1**)**;

**while(**ind + i < LS && ind < LT && S**[**ind + i**]** == T**[**ind**])** ind++;

B**[**i**]** = ind; k = i; **}** **}** **}**

## Aho-Corasick Automaton**(**部分代码)

**void** buildAC**()** **{** head = tail = 0; **int** i; node \* p, \* q;

root->fail = root; que**[**tail++**]** = root;

**while(**head < tail**)** **{** p = que**[**head++**]**; q = p->fail;

**for(**i = 0; i < 10; i++**)** **{**

**if(**p->next**[**i**]** != NULL**)** **{**

**if(**p == root**)** p->next**[**i**]**->fail = root;

**else** **{** p->next**[**i**]**->fail = q->next**[**i**]**;

p->next**[**i**]**->is |= q->next**[**i**]**->is; **}**

que**[**tail++**]** = p->next**[**i**]**; **}**

**else{if(**p== root**)**p->next**[**i**]**= root;**else** p->next**[**i**]**= q->next**[**i**]**;**}**

**}**/\***for**\*/ **}**/\***while**\*/ **}**/\*func\*/

**void** query**(**char str**[])** **{** node \* p , \* q; p = root;

**for(int** i = 0, k; str**[**i**]**; i++**)** **{**k = str**[**i**]**-'0'; p = p->next**[**k**]**;

**if(**p->is**)** **{**q = p;**while(**q->is**)** **{**cnt**[**q->lab**]**++;q = q->fail;**}** **}** **}** **}**

## SA

//论文模板, 使用时注意num**[]**有效位为0~n-1, 但是需要将num**[**n**]**=0, 否则RE;另外, 对于模板的处理将空串也处理了,作为rank最小的串, 因此有效串为0~n共, n-1个, 在调用da**()**函数时, 需要调用da**(**num, n + 1, m**)**; 对于sa**[]**, rank**[]**和height**[]**数组都将空串考虑在内, 作为rank最小的后缀! //调用da**(**num, len+1, m**)**;//m为字符个数略大

**int** len, num**[**Maxn**]**, sa**[**Maxn**]**, rank**[**Maxn**]**, height**[**Maxn**]**; //num待处理的串

**int** wa**[**Maxn**]**, wb**[**Maxn**]**, wv**[**Maxn**]**, wd**[**Maxn**]**;

//sa**[**1~n**]**value**(**0~n-1**)**; rank**[**0..n-1**]**value**(**1..n**)**; height**[**2..n**]**

**int** cmp**(int** \*r, **int** a, **int** b, **int** x**)** **{**

return r**[**a**]** == r**[**b**]** && r**[**a + x**]** == r**[**b + x**]**;**}**

**void** da**(int** \*r, **int** n, **int** m**)** **{**//倍增 r为待匹配数组 n为总长度+1 m为字符范围

**int** i, j, k, p, \*x = wa, \*y = wb, \*t;**for(**i = 0; i < m; i++**)** wd**[**i**]** = 0;

**for(**i = 0; i < n; i++**)** wd**[**x**[**i**]** = r**[**i**]]**++;

**for(**i = 1; i < m; i++**)** wd**[**i**]** += wd**[**i - 1**]**;

**for(**i = n - 1; i >= 0; i--**)** sa**[**--wd**[**x**[**i**]]]** = i;

**for(**j = 1, p = 1; p < n; j <<= 1, m = p**)** **{**

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

**for(**i = 0; i < n; i++**)** **if(**sa**[**i**]** >= j**)** y**[**p++**]** = sa**[**i**]** - j;

**for(**i = 0; i < n; i++**)** wv**[**i**]** = x**[**y**[**i**]]**;

**for(**i = 0;i < m;i++**)** wd**[**i**]** = 0; **for(**i = 0; i < n; i++**)** wd**[**wv**[**i**]]**++;

**for(**i = 1; i < m; i++**)** wd**[**i**]** += wd**[**i - 1**]**;

**for(**i = n - 1; i >= 0; i--**)** sa**[**--wd**[**wv**[**i**]]]** = y**[**i**]**;

**for(**t = x, x = y, y = t, p = 1, x**[**sa**[**0**]]** = 0, i = 1; i < n; i++**)**

x**[**sa**[**i**]]** = cmp**(**y, sa**[**i - 1**]**, sa**[**i**]**, j**)** ? p - 1 : p++; **}**

**for(**i = 0, k = 0; i < n; i++**)** rank**[**sa**[**i**]]** = i;

**for(**i = 0; i < n - 1; height**[**rank**[**i++**]]** = k**)**

**for(**k ? k-- : 0, j = sa**[**rank**[**i**]** - 1**]**; r**[**i + k**]** == r**[**j + k**]**; k++**)**; **}**

## 字符串的最小表示

**int** MinRep **(**char S**[]**, **int** L**)** **{int** i = 0, j = 1, k = 0, t;

**while** **(**i < L && j < L && k < L**)** **{** //找不到比它还小的或者完全匹配

t = S**[** **(**i + k**)** % L**]** - S**[** **(**j + k**)** % L**]**;

//t=s**[(**i+k**)** >= L ? i + k - L : i + k**]** - s**[(**j+k**)** >= L ? j + k - L : j+k**]**;

**if** **(**t == 0**)** k++;//相等的话,检测长度加1

**else** **{**//大于的话,s**[**i**]**为首的肯定不是最小表示,最大表示就改<

**if(**t > 0**)** i += k + 1; **else** j += k + 1; **if(**i == j**)** j++; k = 0;**}**

**}** return min **(**i, j**)**;**}**

## DLX

**struct** DLX**{**

**struct** Node**{** Node \*L, \*R, \*U, \*D; **int** col, row;

**}** \*head, \*row**[**Maxn**]**, \*col**[**Maxm**]**, node**[**Maxn \* Maxm**]**;

**int** colsum**[**Maxm**]**, cnt;

/\* dancing link 精确覆盖问题 可以添加迭代加深优化：

\* 1）枚举深度h; \* 2）若当前深度+predeep > h return **false** \*/

/\* **int** predeep**(){bool** vis**[**Maxm**]**;**int** ret = 0;memset**(**vis, 0, sizeof**(**vis**))**;

**for** **(**Node \*p = head->R; p != head; p = p->R**)**

**if** **(**!vis**[**p->col**])** **{** ret ++ ; vis**[**p->col**]** ++ ;

**for** **(**Node \*q = p->D; q != p; q = p->D**)**

**for(**Node \*r = q->R;r != q; r = r->R**)**vis**[**r->col**]** = **true**;

**}** return ret; **}** //\*/

**void** init**(int** mat**[][**Maxm**]**, **int** n, **int** m**)** **{**

cnt = 0; head = &node**[**cnt ++ **]**;memset**(**colsum, 0, sizeof**(**colsum**))**;

**for(int** i = 1; i <= n; i ++ **)** row**[**i**]** = &node**[**cnt ++ **]**;

**for(int** j = 1; j <= m; j ++ **)** col**[**j**]** = &node**[**cnt ++ **]**;

head->D=row**[**1**]**,row**[**1**]**->U = head;head->R = col**[**1**]**,col**[**1**]**->L = head;

head->U=row**[**n**]**,row**[**n**]**->D = head;head->L = col**[**m**]**,col**[**m**]**->R = head;

head->row = head->col = 0;

**for(int** i = 1; i <= n; i ++ **)** **{** **if** **(**i != n**)** row**[**i**]**->D = row**[**i + 1**]**;

**if(**i != 1**)**row**[**i**]**->U= row**[**i - 1**]**;row**[**i**]**->L= row**[**i**]**->R = row**[**i**]**;

row**[**i**]**->row = i, row**[**i**]**->col = 0;**}**

**for(int** i = 1; i <= m; i ++ **)** **{if** **(**i != m**)** col**[**i**]**->R = col**[**i + 1**]**;

**if** **(**i != 1**)** col**[**i**]**->L = col**[**i - 1**]**;

col**[**i**]**->U= col**[**i**]**->D= col**[**i**]**;col**[**i**]**->col = i,col**[**i**]**->row = 0;**}**

**for(int** i = n; i > 0; i -- **)** **for(int** j = m; j > 0; j -- **)**

**if(**mat**[**i**][**j**])** **{** Node \*p = &node**[**cnt ++ **]**;

p->R = row**[**i**]**->R, row**[**i**]**->R->L = p;

p->L = row**[**i**]**, row**[**i**]**->R = p;

p->D = col**[**j**]**->D, col**[**j**]**->D->U = p;

p->U = col**[**j**]**, col**[**j**]**->D = p;

p->row = i; p->col = j; colsum**[**j**]**++; **}**/\***for**\*/ **}**/\*func\*/

**void** remove**(**Node \*c**)** **{** c->L->R = c->R; c->R->L = c->L;

**for(**Node \*p = c->D; p != c; p = p->D**)**

**for(**Node \*q = p->R; q != p; q = q->R**)** **{**

q->U->D = q->D;q->D->U = q->U;colsum**[**q->col**]** -- ;**}}**

**void** resume**(**Node \*c**)** **{**

**for(**Node \*p = c->U; p != c; p = p->U**)**

**for(**Node \*q = p->L; q != p; q = q->L**)** **{**

q->U->D = q; q->D->U = q; colsum**[**q->col**]** ++ ;**}**

col**[**c->col**]**->L->R = col**[**c->col**]**; col**[**c->col**]**->R->L = col**[**c->col**]**; **}**

**int** dfs**(int** deep**){if(**head->R==head**)**return deep;Node \*p, \*q = head->R;

**for(**p = head->R; p != head; p = p->R**)**

**if(**colsum**[**p->col**]** < colsum**[**q->col**])** q = p;

remove**(**q**)**;

**for(**p = q->D; p != q; p = p->D**)** **{**

**for(**Node\* r = p->R; r != p; r = r->R**)**

**if** **(**r->col != 0**)** remove **(**col**[**r->col**])**;

/\*可修改区域\*/ans**[**deep**]** = p->row;/\*------\*/

**int** sta = dfs **(**deep + 1**)**; **if(**sta**)** return sta;

**for(**Node\* r = p->L; r != p; r = r->L**)**

**if(**r->col != 0**)** resume **(**col**[**r->col**])**; **}**

resume**(**q**)**; return **false**; **}**

///\*可重复覆盖

**void** remove**(**Node \*c**)** **{**

**for(**Node \* p = c->D; row**[**p->row**]** != row**[**c->row**]**; p = p->D**)**

p->R->L = p->L; p->L->R = p->R; **}**

**void** resume**(**Node \*c**)** **{**

**for(**Node \* p = c->U; row**[**p->row**]** != row**[**c->row**]**; p = p->U**)**

p->L->R = p->R->L = p; **}**

**int** dfs**(int** deep**)** **{** **if(**head->R == head**)** return deep <= K;

**if(**deep + predeep**()** > K**)** return **false**;Node \*p, \*q = head->R, \*r;

**for(**p = head->R; p != head; p = p->R**)**

**if(**colsum**[**p->col**]** < colsum**[**q->col**])** q = p;

**for(**p = q->D; p != q; p = p->D**)** **{** remove**(**p**)**;

**for(**r = p->R; r != p; r = r->R**)** **if(**r->col != 0**)** remove**(**r**)**;

/\*可修改区域\*/ans**[**deep**]** = p->row;/\*------\*/

**int** sta = dfs**(**deep + 1**)**; **if(**sta**)** return sta;

**for(**r = p->L; r != p; r = r->L**)** **if(**r->col != 0**)** resume**(**r**)**;

resume**(**p**)**; **}** return **false**; **}** //可重复覆盖\*/

**}** dlx;

## Tips-Lquartz

**网络流拓展**:

1.无源汇上下界可行流: 添加附加源汇S,T 对于某边 **(**u,v**)** 在新网络中连边S->v容量B**[**u,v**]**,u->T容量B**[**u,v**]**,u->v容量C**[**u,v**]**-B**[**u,v**]**.最后,一样也是求一下新网络的最大流,判断从附加源点的边,是否都满流即可.求具体的解:根据最前面提出的强制转换方式,边**(**u,v**)**的最终解中的实际流量即为g**[**u,v**]**+B**[**u,v**]**

2.有源汇上下界可行流: 从汇点到源点连一条上限为INF,下限为0的边.按照1.无源汇的上下界可行流一样做即可.改成无源汇后,求的可行流是类似环的,流量即T->S边上的流量.这样做使S,T也流量平衡了.

3.有源汇的上下界最大流: 方法一:2.有源汇上下界可行流中,从汇点到源点的边改为连一条上限为INF,下限为x的边.因为显然x>ans即MIN**(**T->S **)**> MAX**(**S->T**)**,会使求新网络的无源汇可行流无解的（S,T流量怎样都不能平衡）而x<=ans会有解.所以满足二分性质,二分x,最大的x使得新网络有解的即是所求答案原图最大流. 方法二:从汇点T到源点S连一条上限为INF,下限为0的边,变成无源汇的网络.照求无源汇可行流的方法**(**如1**)**,建附加源点S'与汇点T',求一遍S'->T‘的最大流.再把从汇点T到源点S的这条边拆掉.求一次从S到T的最大流即可.**(**关于S',T'的边好像可以不拆?**)(**一定满足流量平衡?**)**表示这方法我也没有怎么理解.

4.有源汇的上下界最小流

方法一:2.有源汇上下界可行流中,从汇点到源点的边改为连一条上限为x,下限为0的边.与3同理,二分上限,最小的x使新网络无源汇可行流有解,即是所求答案原图最小流. 方法二: 照求无源汇可行流的方法(如1)，建附加源点S'与汇点T'，求一遍S'->T‘的最大流. 但是注意这一遍不加汇点到源点的这条边，即不使之改为无源汇的网络去求解. 求完后，再加上那条汇点到源点上限INF的边. 因为这条边下限为0，所以S',T'无影响. 再直接求一遍S'->T'的最大流. 若S’出去的边全满流，T->S边上的流量即为答案原图最小流，否则无解.

**混合欧拉回路判定**:

给出混合图（有有向边,也有无向边）,判断是否存在欧拉回路: 首先是图中的无向边随意定一个方向,然后统计每个点的入度（indeg）和出度（outdeg）,如果存在点（indeg - outdeg）是奇数的话,一定不存在欧拉回路;否则就开始网络流构图：

1,对于有向边,舍弃; 对于无向边,就按照最开始指定的方向建权值为 1 的边;

2,对于入度小于出度的点,从源点连一条到它的边,权值为（outdeg - indeg）/2；出度小于入度的点,连一条它到汇点的权值为（indeg - outdeg）/2 的边;

构图完成,如果满流（求出的最大流值 == 和汇点所有连边的权值之和）,则存在欧拉回路.

**树Hash判定树同构**:

//初始时,给树的每一个节点赋一个随机的权值h**[**i**]**

**int** Hash **(int** j**)** **{** **int** sum = h**[**j + 5000**]**;//这里的j是记录的节点度

//这个巧妙的循环,把子节点的hash值都加给了父节点,作为父节点的hash值

//由于树节点顺序不确定, 因此是子树hash值\*根值的累加

**while** **(**\*p && \*p++ == '0'**)** **{**sum = **(**sum + h**[**j**]** \* Hash **(**j + 1**))** % MOD; **}**

return **(**sum \* sum**)** % 19001; **}**

**RMQ(query)**:

**int** query**(int** l, **int** r**)** **{** //求**[**l, r**]**

**int** k = kk**[**r - l + 1**]**;//预处理 kk**[**i**]** = log2**(**i**)**;

return min**(**st**[**k**][**l**]**, st**[**k**][**r - **(**1<<k**)** + 1**])**;**}**

**斯坦纳树**:

//dp[u][i]表示结点u已经和要连通的结点集合(2^k表示)i连通的最小花费

//初始化将k个点和n个点dp[u][1<<i]初始化为最短路, dp[u][0]=0, 加入队列

//利用spfa求出dp数组 //状态转移分三部分:

// 1. dp[u][su] 利用dp[u][sub] + dp[u][su^sub]更新, sub为su子集

// 2. dp[u][su] 更新相邻的dp[v][sv]

// 3. 将k中不属于su的点与u连接, 利用u, k的最短路

**生成树相关的一些问题**: By 猛犸也钻地 @ 2012.02.24

/\* 度限制生成树 Q: 求一个最小生成树**,**其中V0连接的边不能超过K个或只能刚好K个

1. 去掉所有和V0连接的边**,**对每个连通分量求最小生成树

2. 如果除去点V0外共有T个连通分量**,**且T>K**,**无解

3. 于是现在有一个最小T度生成树**,**然后用dp[V]计算出该点到V0的路径上,权值最大的边是多少**,**再枚举和V0连接的没有使用过的边**,**找出一条边,使得用那条边替换已有的边**,**增加的权值最小**,**不停替换直到V0出度为K \*/

/\* 次小生成树 Q: 求一个次小生成树**,**要求权值之和必须大于等于或严格大于其最小生成树

1. 求最小生成树

2. 找一个根然后dp**,**求出每个点往上走2^L能到达的祖先是谁**,**以及这段路径上的最大边和次大边(如果仅要求大于等于的话就不需要次大边)

3. 枚举没有使用过的边**,**利用上面得到的信息**,**在O(logN)时间内对每条边计算出其能够替换的已有的最大和次大边**,**然后找出最佳替换方式 \*/

/\* 斯坦纳树 Q: 求一个包含指定的K个特殊点的最小生成树**,**其他点不一定在树中

1. 用dp[mask][x]记录树根在点x**,**mask所对应的特殊点集在树中的最小权值之和

2. 将dp[][]初始化为正无穷**,**只有dp[1<<i][Ai]被初始化为0**,**Ai为第i个特殊点

3. 先求出所有点对间最短路**,**然后枚举mask**,**依次做两种转移：

3.1. 枚举x和mask的子集sub**,**合并两棵子树

dp[mask][x]=min(dp[mask][x],dp[sub][x]+dp[mask^sub][x])

3.2. 枚举x和y**,**计算结点从y移动到x的花费

dp[mask][x]=min(dp[mask][x],dp[mask][y]+minDistance(y,x))

在上面的转移中**,**也可以把这些点同时放到队列里**,**用spfa更新最短路 \*/

/\* 生成树计数 Q: 给定一个无权的无向图G**,**求生成树的个数

1. 令矩阵D[][]为度数矩阵**,**其中D[i][i]为结点i的度**,**其他位置的值为0

2. 令矩阵A[][]为邻接矩阵**,**当结点i和j之间有x条边时**,**D[i][j]=D[j][i]=x

3. 令矩阵C=D-A**,**矩阵C'为矩阵C抽去第k行和第k列后的一个n-1阶的子矩阵

其中k可以任意设定**,**构造完C'后**,**生成树的个数即为C'行列式的值 \*/

**匹配问题结论**:

6. 最大独立集 = 顶点数 - 最大匹配数(如果图Ｇ满足二分图条件,用二分图匹配来做)

7. 最小点覆盖 = 最大匹配数

8. 最小路径覆盖 = 顶点数 - 最大匹配数(最少不相交简单路径覆盖有向无环图Ｇ) (PS: 此处注意, 最小路径覆盖是针对有向图而言, 那么将一个点拆开成为i和i'建立二分图)