后缀数组模板(sa 数组从 1 到 N，rank 数组从 0 到 N-1，height 数组从 2 到 N)。

```
int sa[nMax], rank[nMax], height[nMax];
int wa[nMax], wb[nMax], wv[nMax], wd[nMax];

int cmp(int *r, int a, int b, int l)
{
    return r[a] == r[b] && r[a+l] == r[b+l];
}

void da(int *r, int n, int m)           //  倍增
算法 r 为待匹配数组  n 为总长度 m 为字符范围
{
    int i, j, 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 *= 2, 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 ++;
        }
    }
}

void calHeight(int *r, int n)            //  求
height 数组。
{
    int i, j, k = 0;
    for(i = 1; i <= n; i ++) rank[sa[i]] = i;
    for(i = 0; i < n; height[rank[i ++]] = k)
    {
        for(k ? k -- : 0, j = sa[rank[i]-1]; r[i+k]
== r[j+k]; k ++);
    }
}

da(  , n+1 ,   ,   )
Cal(   ,   n  )
```

RMQ 版本的后缀数组

```
int wa[maxn],wb[maxn],wv[maxn],Ws[maxn];
int cmp(int *r,int a,int b,int l)
{
    return r[a]==r[b]&&r[a+l]==r[b+l];
}
void da(const char *r,int *sa,int n,int m)
{
    int i,j,p,*x=wa,*y=wb,*t;
    for(i=0; i<m; i++) Ws[i]=0;
    for(i=0; i<n; i++) Ws[x[i]=r[i]]++;
    for(i=1; i<m; i++) Ws[i]+=Ws[i-1];
    for(i=n-1; i>=0; i--) sa[--Ws[x[i]]]=i;
    for(j=1,p=1; p<n; j*=2,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++) Ws[i]=0;
        for(i=0; i<n; i++) Ws[wv[i]]++;
        for(i=1; i<m; i++) Ws[i]+=Ws[i-1];
        for(i=n-1;        i>=0;        i--)
sa[--Ws[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++;
    }
    return;
}
int sa[maxn],Rank[maxn],height[maxn];
//求 height 数组
void calheight(const char *r,int *sa,int n)
{
    int i,j,k=0;
    for(i=1; i<=n; i++) Rank[sa[i]]=i;
    for(i=0; i<n; height[Rank[i++]]=k)
        for(k?k--:0,j=sa[Rank[i]-1];
r[i+k]==r[j+k]; k++);
    return;
}
int dp[maxn][20];
```

```
void Rmq_Init(int n)
{
    int m=floor(log(n+0.0)/log(2.0));
    for(int i=1; i<=n; i++) dp[i][0]=height[i];
    for(int i=1; i<=m; i++)
    {
        for(int j=n; j; j--)
        {
            dp[j][i]=dp[j][i-1];
            if(j+(1<<(i-1))<=n)

dp[j][i]=min(dp[j][i],dp[j+(1<<(i-1))][i-1]);
        }
    }
}
int Rmq_Query(int l,int r)
{
    int a=Rank[l],b=Rank[r];
    if(a>b) swap(a,b);
    a++;
    int m=floor(log(b-a+1.0)/log(2.0));
    return min(dp[a][m],dp[b-(1<<m)+1][m]);
}
```

Rmq_Init(n);

Rmq_Query(l , r) 区间最小值

Spfa(与差分约束)

```
//poj 3169
#include "cstdio"
#include "cstring"
#include "cstdlib"
#include "iostream"
#include "cmath"
#include "queue"
using namespace std;
int N,ML,MD;
struct NODE
{
    int to,len,next;
}edge[200005];
int head[1005],vis[1005],dist[1005],in[1005];
void spfa()
{
    int i,k;
    for(i=0;i<=N;i++)
dist[i]=999999999,vis[i]=0,in[i]=0;
    queue<int> Q;
    while(!Q.empty()) Q.pop();
    Q.push(1);
    dist[1]=0;
    in[1]++;
    while(!Q.empty())
    {
        k=Q.front();
        Q.pop();
        vis[k]=0;
        for(i=head[k];i!=-1;i=edge[i].next)
        {
            if(dist[edge[i].to] > edge[i].len +
dist[k])
            {
                dist[edge[i].to] = edge[i].len +
dist[k];
                if(!vis[edge[i].to])
                {
                    vis[edge[i].to]=1;
                    Q.push(edge[i].to);
                    in[edge[i].to]++;
                    if(in[edge[i].to] > N)
                    {
                        dist[N]=-1;
                        return;
                    }
                }
            }
        }
    }
    if(dist[N] >= 999999999) dist[N]=-2;
}
```

-1 代表有负环，无解。-2 代表无限远~

//求最大距离 a - b < c

//求最小距离 a - b > c

KMP 函数

```
void get_p(int n)    //得到预数组，KMP 主体和这结
构差不多，依题意改一下相关数组即可
{
    int i,j=-1;
    p[0]=-1;
    for(i=1;i<n;i++)
    {
        while(j>-1 && temp[i]!=temp[j+1]) j=p[j];
        if(temp[i] == temp[j+1]) j++;
        p[i]=j;
    }
}
```

球上两点距离公式

```
double calc(NODE a,NODE b)   // x 为经度，y 为纬度（都用弧度表示）
{
       double ans=(D/2)*acos(sin(a.x)*sin(b.x)+cos(a.x)*cos(b.x)*cos(a.y-b.y));
       return ans;
}
```

普通网络流

```
const int maxnode = 1000 + 5;
const int maxedge = 1000 + 5;
const int oo = 1000000000;
int node, src, dest, nedge;
int head[maxnode], point[maxedge], next1[maxedge], flow[maxedge], capa[maxedge];//point[x]==y表示第x条边连接y，head，next为邻接表，flow[x]表示x边的动态值，capa[x]表示x边的初始值
int dist[maxnode], Q[maxnode], work[maxnode];//dist[i]表示i点的等级
void init(int _node, int _src, int _dest){//初始化，node表示点的个数，src表示起点，dest表示终点
    node = _node;
    src = _src;
    dest = _dest;
    for (int i = 0; i < node; i++) head[i] = -1;
    nedge = 0;
}
void addedge(int u, int v, int c1, int c2){//增加一条u到v流量为c1，v到u流量为c2的两条边
    point[nedge] = v, capa[nedge] = c1, flow[nedge] = 0, next1[nedge] = head[u], head[u] = (nedge++);
    point[nedge] = u, capa[nedge] = c2, flow[nedge] = 0, next1[nedge] = head[v], head[v] = (nedge++);
}
bool dinic_bfs(){
    memset(dist, 255, sizeof (dist));
    dist[src] = 0;
    int sizeQ = 0;
    Q[sizeQ++] = src;
    for (int cl = 0; cl < sizeQ; cl++)
        for (int k = Q[cl], i = head[k]; i >= 0; i = next1[i])
            if (flow[i] < capa[i] && dist[point[i]] < 0){
                dist[point[i]] = dist[k] + 1;
                Q[sizeQ++] = point[i];
            }
    return dist[dest] >= 0;
}
int dinic_dfs(int x, int exp){
    if (x == dest) return exp;
    for (int &i = work[x]; i >= 0; i = next1[i]){
        int v = point[i], tmp;
        if (flow[i] < capa[i] && dist[v] == dist[x] + 1 && (tmp = dinic_dfs(v, min(exp, capa[i] - flow[i]))) > 0){
            flow[i] += tmp;
            flow[i^1] -= tmp;
            return tmp;
        }
    }
    return 0;
}
int dinic_flow(){
    int result = 0;
    while (dinic_bfs()){
        for (int i = 0; i < node; i++) work[i] = head[i];
        while (1){
            int delta = dinic_dfs(src, oo);
            if (delta == 0) break;
            result += delta;
        }
    }
    return result;
}
//建图前,运行一遍init();
//加边时，运行addedge(a,b,c,0),表示点a到b流量为c的边建成（注意点序号要从0开始）
//求解最大流运行dinic_flow(),返回值即为答案
```

费用流

```
const int N = 1010;//点
const int M = 2 * 10010;//边
const int inf = 1000000000;
struct Node{//边，点f到点t，流量为c，费用为w
    int f, t, c, w;
}e[M];
int next1[M], point[N], dis[N], q[N], pre[N], ne;//ne为已添加的边数，next，point为邻接表,dis为花费，pre为父亲节点
bool u[N];
void init(){
    memset(point, -1, sizeof(point));
    ne = 0;
}
```

```
void add_edge(int f, int t, int d1, int d2, int w){//f 到 t 的一条边，流量为 d1,反向流量 d2,花费 w,反向边花费-w（可以反悔）
    e[ne].f = f, e[ne].t = t, e[ne].c = d1, e[ne].w = w;
    next1[ne] = point[f], point[f] = ne++;
    e[ne].f = t, e[ne].t = f, e[ne].c = d2, e[ne].w = -w;
    next1[ne] = point[t], point[t] = ne++;
}
bool spfa(int s, int t, int n){
    int i, tmp, l, r;
    memset(pre, -1, sizeof(pre));
    for(i = 0; i < n; ++i)
        dis[i] = inf;
    dis[s] = 0;
    q[0] = s;
    l = 0, r = 1;
    u[s] = true;
    while(l != r) {
        tmp = q[l];
        l = (l + 1) % (n + 1);
        u[tmp] = false;
        for(i = point[tmp]; i != -1; i = next1[i]) {
            if(e[i].c && dis[e[i].t] > dis[tmp] + e[i].w) {
                dis[e[i].t] = dis[tmp] + e[i].w;
                pre[e[i].t] = i;
                if(!u[e[i].t]) {
                    u[e[i].t] = true;
                    q[r] = e[i].t;
                    r = (r + 1) % (n + 1);
                }
            }
        }
    }
    if(pre[t] == -1)
        return false;
    return true;
}
void MCMF(int s, int t, int n, int &flow, int &cost){//起点 s，终点 t，点数 n，最大流 flow，最小花费 cost
    int tmp, arg;
    flow = cost = 0;
    while(spfa(s, t, n)) {
        arg = inf, tmp = t;
        while(tmp != s) {
            arg = min(arg, e[pre[tmp]].c);
            tmp = e[pre[tmp]].f;
        }
        tmp = t;
        while(tmp != s) {
            e[pre[tmp]].c -= arg;
            e[pre[tmp] ^ 1].c += arg;
            tmp = e[pre[tmp]].f;
        }
        flow += arg;
        cost += arg * dis[t];
    }
}
//建图前运行 init()
//节点下标从 0 开始
//加边时运行 add_edge(a,b,c,0,d)表示加一条 a 到 b 的流量为 c 花费为 d 的边（注意花费为单位流量花费）
//特别注意双向边，运行add_edge(a,b,c,0,d),add_edge(b,a,c,0,d)较好，不要只运行一次add_edge(a,b,c,c,d),费用会不对。
//求解时代入 MCMF(s,t,n,v1,v2)，表示起点为 s，终点为 t，点数为 n 的图中，最大流为 v1，最大花费为 v2
```

并查集

```
int parent[];
int root(int p)
{
    if(parent[p]==-1) return p;
    else return  parent[p]=root(parent[p]);
}
void merge(int beg, int end)
{
    beg=root(beg);
    end=root(end);
    parent[end]=beg;
}

int fa[20050],r[20050];
int find(int x)
{
    if(fa[x] == x) return fa[x];
    else
    {
        int ff=fa[x];
        fa[x]=find(fa[x]);
        r[x]=(r[x]+r[ff])&1;
        return fa[x];
    }
}
```

```
void merge(int x,int y,int ff)
{
    int a=find(x),b=find(y);
    fa[b]=a;
    r[b]=(r[x]-r[y]+2+ff)&1;
}
```

强连通

```
struct Node
{
    int to;
    int next;
}edge[50005];
stack<int> sta;
int
head[105],vis[105],low[105],dfn[105],num,index,
inum,gra[105],gro[105];
int get_in[105],get_out[105];
int N;

void dfs(int cur)
{
    low[cur]=dfn[cur]=++index;
    vis[cur]=1;
    sta.push(cur);
    int i,to;
    for(i=head[cur]; i!=-1; i=edge[i].next)
    {
        to=edge[i].to;
        if(dfn[to] == 0)
        {
            dfs(to);
            low[cur]=min(low[cur],low[to]);
        }
        else if(vis[to] == 1)
        {
            low[cur]=min(low[cur],dfn[to]);
        }
    }
    if(low[cur] == dfn[cur])
    {
        inum++;
        while(1)
        {
            int temp=sta.top();
            vis[temp]=0;
            gra[temp]=inum;
            sta.pop();
            if(temp == cur) break;
        }
    }
}

void tarjan()
{
    index=0,inum=0;
    memset(dfn,0,sizeof(dfn));
    memset(low,0,sizeof(low));
    memset(vis,0,sizeof(vis));
    memset(gra,0,sizeof(gra));   //连通分量
    memset(gro,0,sizeof(gro));   //分量内部点个
数
    for(int i=1; i<=N; i++)
    {
        if(!dfn[i])
        {
            dfs(i);
        }
    }
}
```

猪猪的矩阵

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

#define MAX 128
#define MOD 1000000007

using namespace std;

typedef long long i64;

i64 a[MAX][MAX], b[MAX][MAX], c[MAX][MAX],
buff[MAX][MAX], vec[MAX];

void matCpy(i64 a[MAX][MAX], i64 b[MAX][MAX], int
n) {
    int i, j;

    for (i = 0; i < n; ++i) {
        for (j = 0; j < n; ++j) {
            a[i][j] = b[i][j];
        }
    }
}

void norm(i64 a[MAX][MAX], int n) {
    int i, j;
    for (i = 0; i < n; ++i) {
        for (j = 0; j < n; ++j) {
            a[i][j] = (i == j);
        }
    }
}

void matMul(const i64 a[MAX][MAX], const i64 b[MAX][MAX], i64 c[MAX][MAX], int n) {
    int i, j, k;

    for (i = 0; i < n; ++i) {
        for (j = 0; j < n; ++j) {
            for (c[i][j] = k = 0; k < n; ++k) {
                c[i][j] = (c[i][j] + a[i][k] * b[k][j]) % MOD;
            }
        }
    }
}

void matPow(i64 a[MAX][MAX], i64 b, i64 c[MAX][MAX], int n) {
    for (norm(c, n); b; b >>= 1) {
        if (b & 1) {
            matMul(c, a, buff, n);
            matCpy(c, buff, n);
        }
        matMul(a, a, buff, n);
        matCpy(a, buff, n);
    }
}

int main(){
    return 0;
}
```

组合数

```
const int maxm = 100000+10;
ll p[maxm],pinv[maxm];
ll pow_mod(ll x,ll n)
{
    ll res = 1;
    while(n)
    {
        if(n&1) res = res * x %mod;
        x = x * x %mod;
        n >>= 1;
    }
    return res;
}
ll inv(ll x)
{
    return pow_mod(x,mod-2);
}
inline ll C(ll n,ll m)
{
    return p[n]*pinv[m]%mod*pinv[n-m]%mod;
}
//main函数先运行以下
p[0] = 1;pinv[0] = 1;
    for(int i = 1;i < maxm;++i)
    {
        p[i] = p[i-1] * i %mod;
        pinv[i] = inv(p[i]);
    }
```

n中选m个即为C（n,m）

快速读入

```
void reads(int & x)
{
    char c;
    bool neg=false;
    while(((c=getchar())<'0'||c>'9')&&c!='-');
    if(c=='-')
    {
        neg=true;
        while((c=getchar())<'0'||c>'9');
    }
    x=c-'0';
    while(c=getchar(),c>='0'&&c<='9') x=x*10+c-'0';
    if(neg) x=-x;
}
```

朱刘算法

```
#include <iostream>
#include <cstdio>
#include <cmath>
#include <vector>
#include <cstring>
#include <algorithm>
#include <string>
#include <set>
#include <ctime>
#include <queue>
#include <map>
```

```
#include <sstream>

#define  CL(arr,  val)        memset(arr,  val, sizeof(arr))
#define REP(i, n)        for((i) = 0; (i) < (n); ++(i))
#define FOR(i, l, h)     for((i) = (l); (i) <= (h); ++(i))
#define FORD(i, h, l)    for((i) = (h); (i) >= (l); --(i))

const double eps = 1e-6;
const int inf = 10000000;
typedef long long LL;

using namespace std;

const int N = 550;
const int M = 3010;

struct node {
    double x, y;
} point[N];

struct edg {
    int u, v;
    int cost;
} E[M];

int In[N];
int ID[N];
int vis[N];
int pre[N];
int NV, NE;

double SQ(int u, int v) {
    return          sqrt((point[u].x         - point[v].x)*(point[u].x - point[v].x) +
                (point[u].y          - point[v].y)*(point[u].y - point[v].y));
}
void add(int u,int v,int cost)
{
    E[NE].u=u,E[NE].v=v,E[NE++].cost=cost;
}
int Directed_MST(int root) {
    int ret = 0;
    int i, u, v;
    while(true) {
        REP(i, NV)   In[i] = inf;
        REP(i, NE) {    //找最小入边
            u = E[i].u;
            v = E[i].v;
            if(E[i].cost < In[v] && u != v) {
                In[v] = E[i].cost;
                pre[v] = u;
            }
        }
        REP(i, NV) {    //如果存在除root以外的孤立点，则不存在最小树形图
            if(i == root)   continue;
            //printf("%.3lf ", In[i]);
            if(In[i] == inf)    return -1;
        }

        int cnt = 0;
        CL(ID, -1);
        CL(vis, -1);
        In[root] = 0;

        REP(i, NV) {    //找环
            ret += In[i];
            int v = i;
            while(vis[v] != i && ID[v] == -1 && v != root) {
                vis[v] = i;
                v = pre[v];
            }
            if(v != root && ID[v] == -1) {  //重新标号
                for(u = pre[v]; u != v; u = pre[u]) {
                    ID[u] = cnt;
                }
                ID[v] = cnt++;
            }
        }
        if(cnt == 0)    break;
        REP(i, NV) {
            if(ID[i] == -1) ID[i] = cnt++;    //重新标号
        }
        REP(i, NE) {    //更新其他点到环的距离
            v = E[i].v;
            E[i].u = ID[E[i].u];
            E[i].v = ID[E[i].v];
            if(E[i].u != E[i].v) {
                E[i].cost -= In[v];
                }
            }
            NV = cnt;
            root = ID[root];
        }
        return ret;
    }
```

//每次用 add 进行加边，NV、NE（初始化为 0）分别赋值为点数（必须是 0~NV-1）和边数。然后直接运行 Directed_MST（root），返回结果为-1 表示没有最小树形。

LCA

```
#include "cstdio"
#include "cstring"
#include "algorithm"
#include "cmath"
#include "vector"
using namespace std;
#define N 100050  //点数

int f[N],num[1000000],d[N],fa[N],color[N];
int n,m,tt,con;
struct NODE
{
    int x,y;
    int anc;
}lc[N];
int DX[N];
int DPRE[N];
int DI[N];
int FLR;

void dfs(int x, int pre)
{
    int i;
    FLR = 0;
body:
    f[x]=tt++;
    num[con++]=f[x];
    d[f[x]]=con-1;
    fa[f[x]]=f[pre];
    for(i=0;i<v[x].size();i++)
    {
        if(v[x][i] == pre)
            continue;
        //dfs(v[x][i],x);
        DX[FLR] = x;
        DPRE[FLR] = pre;
        DI[FLR] = i;
        FLR++;
        pre = x;
        x = v[x][i];
        i = 0;
        goto body;
doret:
        num[con++]=f[x];
    }

retu:
    FLR--;
    if(FLR<0)
        return;
    x = DX[FLR];
    pre = DPRE[FLR];
    i = DI[FLR];
    goto doret;
}
int dp[500000][20];
void rmqst_init()
{
    int i,j,mm;

    mm = (int)(floor(log((double)con)/log(2.0)));
    for(i=1;i<=con;i++)
    {
        dp[i][0] = num[i];
    }
    for(j=1;j<=mm;j++)
    {
        for(i=1;i<=con-(1<<(j-1));i++)
        {
            dp[i][j]=min(dp[i][j-1],dp[i + (1<<(j-1))][j-1]);
        }
    }
}

//RMQ计算
int rmq_get(int a, int b)
{
    int mm, tmp;
    a=d[a];
    b=d[b];

    if(b < a)
    {
        tmp = a;
        a = b;
        b = tmp;
    }
    mm                                              =
(int)(floor(log((double)(b-a+1))/log(2.0)));
    return min(dp[a][mm],dp[b-(1<<mm)+1][mm]);
}

for(i=1;i<n;i++)  //n个点，n-1条边
        {
            scanf("%d %d",&j,&k);
            v[j].push_back(k);
            v[k].push_back(j);
        }
        tt=1,con=1;
        fa[1]=0,f[0]=0,fa[0]=0;
        dfs(1,0);
        con--;
        rmqst_init();
```

对于给的 i，j 之间的 LCA 为 rmq_get（f[i] ，f[j]）

重复覆盖的 DLX

```
const int maxnode = 3000;
const int MaxM = 55;
const int MaxN = 55;
int K;    //选取限制
struct DLX
{
    int n,m,size;
    int
U[maxnode],D[maxnode],R[maxnode],L[maxnode],Row
[maxnode],Col[maxnode];
    int H[MaxN],S[MaxN];
    int ands,ans[MaxN];
    void init(int _n,int _m)
    {
        n = _n;
        m = _m;
        for(int i = 0;i <= m;i++)
        {
            S[i] = 0;
            U[i] = D[i] = i;
            L[i] = i-1;
            R[i] = i+1;
        }
        R[m] = 0; L[0] = m;
        size = m;
        for(int i = 1;i <= n;i++)
            H[i] = -1;
    }
    void Link(int r,int c)      //记录行列为1的点
    {
        ++S[Col[++size]=c];
        Row[size] = r;
        D[size] = D[c];
        U[D[c]] = size;
        U[size] = c;
        D[c] = size;
        if(H[r] < 0)H[r] = L[size] = R[size] =
size;
        else
        {
            R[size] = R[H[r]];
            L[R[H[r]]] = size;
            L[size] = H[r];
            R[H[r]] = size;
        }
    }
    void remove(int c)
    {
        for(int i = D[c];i != c;i = D[i])
            L[R[i]] = L[i], R[L[i]] = R[i];
    }
    void resume(int c)
    {
        for(int i = U[c];i != c;i = U[i])
            L[R[i]]=R[L[i]]=i;
    }
    bool v[maxnode];
    int f()
    {
        int ret = 0;
        for(int c = R[0];c != 0;c = R[c])v[c] =
true;
        for(int c = R[0];c != 0;c = R[c])
            if(v[c])
            {
                ret++;
                v[c] = false;
                for(int i = D[c];i != c;i = D[i])
                    for(int j = R[i];j != i;j =
R[j])
                        v[Col[j]] = false;
            }
        return ret;

    }
    bool Dance(int d)
    {
        if(d + f() > K)return false;
        if(R[0] == 0)return d <= K;
        int c = R[0];
        for(int i = R[0];i != 0;i = R[i])
            if(S[i] < S[c])
                c = i;
        for(int i = D[c];i != c;i = D[i])
        {
            remove(i);
            for(int j = R[i];j != i;j = R[j])remove(j);
            if(Dance(d+1))return true;
            for(int j = L[i];j != i;j = L[j])resume(j);
            resume(i);
        }
        return false;
    }
};
DLX g;
g.init(m,n);
g.Link(i+1,j+1);
g.Dance(0)
```

精确覆盖的 DLX

```
const int maxnode = 100010;
const int MaxM = 1010;
const int MaxN = 1010;
struct DLX
{
    int n,m,size;
    int U[maxnode],D[maxnode],R[maxnode],L[maxnode],Row[maxnode],Col[maxnode];
    int H[MaxN], S[MaxM];
    int ansd, ans[MaxN];  //统计的可行解
    void init(int _n,int _m)
    {
        n = _n;
        m = _m;
        for(int i = 0;i <= m;i++)
        {
            S[i] = 0;
            U[i] = D[i] = i;
            L[i] = i-1;
            R[i] = i+1;
        }
        R[m] = 0; L[0] = m;
        size = m;
        for(int i = 1;i <= n;i++)
            H[i] = -1;
    }
    void Link(int r,int c)    //记录行列为 1 的点
    {
        ++S[Col[++size]=c];
        Row[size] = r;
        D[size] = D[c];
        U[D[c]] = size;
        U[size] = c;
        D[c] = size;
        if(H[r] < 0)H[r] = L[size] = R[size] = size;
        else
        {
            R[size] = R[H[r]];
            L[R[H[r]]] = size;
            L[size] = H[r];
            R[H[r]] = size;
        }
    }
    void remove(int c)
    {
        L[R[c]] = L[c]; R[L[c]] = R[c];
        for(int i = D[c];i != c;i = D[i])
            for(int j = R[i];j != i;j = R[j])
            {
                U[D[j]] = U[j];
                D[U[j]] = D[j];
                --S[Col[j]];
            }
    }
    void resume(int c)
    {
        for(int i = U[c];i != c;i = U[i])
            for(int j = L[i];j != i;j = L[j])
                ++S[Col[U[D[j]]=D[U[j]]=j]];
        L[R[c]] = R[L[c]] = c;
    }
    //d 为递归深度
    bool Dance(int d)
    {
        if(R[0] == 0)
        {
            ansd = d;
            return true;
        }
        int c = R[0];
        for(int i = R[0];i != 0;i = R[i])
            if(S[i] < S[c])
                c = i;
        remove(c);
        for(int i = D[c];i != c;i = D[i])
        {
            ans[d] = Row[i];
            for(int j = R[i]; j != i;j =
R[j])remove(Col[j]);
            if(Dance(d+1))return true;
            for(int j = L[i]; j != i;j =
L[j])resume(Col[j]);
        }
        resume(c);
        return false;
    }
};

DLX g;
```

优先队列

```
struct cmp1{

    bool operator ()(int &a,int &b){

        return a>b;//最小值优先

    }

};

struct cmp2{

    bool operator ()(int &a,int &b){

        return a<b;//最大值优先

    }

};

priority_queue<int,vector<int>,cmp1>que1;
//最小值优先

priority_queue<int,vector<int>,cmp2>que2;
//最大值优先



//定义结构，使用运算符重载,自定义优先级2

struct number1{

    int x;

    bool operator < (const number1 &a) const
{

        return x>a.x;//最小值优先

    }

};

struct number2{

    int x;

    bool operator < (const number2 &a) const
{

        return x<a.x;//最大值优先

    }

 priority_queue<number1>que5;

 priority_queue<number2>que6;
```