ACM-ICPC模板

[图论 4](#_Toc416979337)

[一、 生成树 4](#_Toc416979338)

[斯坦纳树 4](#_Toc416979339)

[最优比例生成树 7](#_Toc416979340)

[度限制最小生成树 8](#_Toc416979341)

[生成树计数 12](#_Toc416979342)

[生成树相关问题 15](#_Toc416979343)

[二、 最短路 16](#_Toc416979344)

[最短路次短路计数 16](#_Toc416979345)

[K短路 17](#_Toc416979346)

[三、 欧拉回路 18](#_Toc416979347)

[基础 18](#_Toc416979348)

[磁鼓欧拉回路(HDU 2894) 18](#_Toc416979349)

[混合欧拉路欧拉回路的判断-->网络流模型 18](#_Toc416979350)

[四、 RMQ 20](#_Toc416979351)

[RMQ实现模板 20](#_Toc416979352)

[五、 最小环&最大环 20](#_Toc416979353)

[最小环 20](#_Toc416979354)

[六、 拓扑排序 21](#_Toc416979355)

[基础知识 22](#_Toc416979356)

[求任意拓扑排序或判断是否可以拓扑排序 23](#_Toc416979357)

[字典序最小解 23](#_Toc416979358)

[有向图拓扑排序计数(状压dp) 25](#_Toc416979359)

[树拓扑排序计数 26](#_Toc416979360)

[关键路径 26](#_Toc416979361)

[七、 LCA 28](#_Toc416979362)

[倍增算法 28](#_Toc416979363)

[Tarjan 29](#_Toc416979364)

[LCA 转 RMQ 30](#_Toc416979365)

[八、 连通性 31](#_Toc416979366)

[点双连通(Tarjan+重建图) 31](#_Toc416979367)

[九、 2 SAT 34](#_Toc416979368)

[综述 34](#_Toc416979369)

[判断可行性 34](#_Toc416979370)

[求任意解 34](#_Toc416979371)

[输出任一解 37](#_Toc416979372)

[输出任意解from XMZhou 40](#_Toc416979373)

[求字典序最小的解 43](#_Toc416979374)

[十、 匹配问题 45](#_Toc416979375)

[基本概念 45](#_Toc416979376)

[HK (O(E\*N^0.5)) 45](#_Toc416979377)

[稳定婚姻问题 47](#_Toc416979378)

[带花树 48](#_Toc416979379)

[一般图最小权匹配 51](#_Toc416979380)

[十一、 最大流 52](#_Toc416979381)

[Dinic 52](#_Toc416979382)

[平面图最小割转最短路 54](#_Toc416979383)

[十二、 费用流 54](#_Toc416979384)

[spfa 54](#_Toc416979385)

[zkw 56](#_Toc416979386)

[十三、 全局最小割 60](#_Toc416979387)

[StoerWagner(求全局最小割) 60](#_Toc416979388)

[K连通块计数 62](#_Toc416979389)

[十四、 网络流拓展 65](#_Toc416979390)

[常见建图方式 65](#_Toc416979391)

[十五、 最小树形图(有向图的最小生成树) 66](#_Toc416979392)

[基础知识 66](#_Toc416979393)

[朱刘算法 66](#_Toc416979394)

[十六、 树的Hash与同构 68](#_Toc416979395)

[树Hash判定树同构 68](#_Toc416979396)

[十七、 最大团 69](#_Toc416979397)

[最大团 69](#_Toc416979398)

[十八、 弦图相关 71](#_Toc416979399)

[弦图判定 71](#_Toc416979400)

[弦图与区间图 By 猛犸也钻地 @ 2012.09.13 72](#_Toc416979401)

[字符串 74](#_Toc416979402)

[一、 Hash 74](#_Toc416979403)

[一般字符串Hash 74](#_Toc416979404)

[字符串区间Hash 74](#_Toc416979405)

[二、 KMP 75](#_Toc416979406)

[函数版 75](#_Toc416979407)

[KMP类 75](#_Toc416979408)

[三、 拓展KMP 76](#_Toc416979409)

[函数版 76](#_Toc416979410)

[四、 Manacher 77](#_Toc416979411)

[Manacher模板 77](#_Toc416979412)

[五、 AC自动机 78](#_Toc416979413)

[普通匹配 78](#_Toc416979414)

[AC自动机+DP 80](#_Toc416979415)

[六、 后缀数组 82](#_Toc416979416)

[论文模板(O(nlogn)) 82](#_Toc416979417)

[七、 字符串最小表示法 83](#_Toc416979418)

[朴素最小表示法 83](#_Toc416979419)

[判定两串是否循环同构 83](#_Toc416979420)

[搜索 84](#_Toc416979421)

[一、 DLX 84](#_Toc416979422)

[精确覆盖 84](#_Toc416979423)

[多重覆盖 87](#_Toc416979424)

[其他 90](#_Toc416979425)

[一、 头文件 90](#_Toc416979426)

[head.cpp 90](#_Toc416979427)

[二、 配置信息 91](#_Toc416979428)

[eclipse 91](#_Toc416979429)

[vimrc 91](#_Toc416979430)

[三、 调试注意事项 92](#_Toc416979431)

[内存大小 92](#_Toc416979432)

[堆栈 92](#_Toc416979433)

[编译器选项 92](#_Toc416979434)

[Java类名要求 92](#_Toc416979435)

[返回值 92](#_Toc416979436)

[ext包 92](#_Toc416979437)

# **图论**

## 生成树

### 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

//hdu 4085

//求斯坦纳森林, 再森林合并为全集

#define MOD 0x3f3f3f3f

int dp[Maxn][MSta];

vector<PII> g[Maxn];

int gg[Maxn][Maxn];

int mp1[Maxn], mp2[Maxn];

int n, m, K;

vector<int> split[Maxk];

int gao[MSta];

int has (int sta) {

int i, a = 0, b = 0, u;

for (i = 0; i < 2 \* K; i++) {

if ((1<<i) & sta) {

u = mp1[i];

if (u <= K) a++;

else b++;

}

}

if (a == b) return true;

else return false;

}

int ans;

queue<PII> que;

int inque[Maxn][MSta];

int doit() {

int i, j, k, u, v, w, su, sv, sub, vv, tp;

int kk = 2 \* K;

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

for (j = 0; j < (1<< kk); j++) dp[i][j] = MOD, inque[i][j] = 0;

dp[i][0] = 0; inque[i][0] = 0;

}

while(!que.empty()) que.pop();

memset(mp2, 0, sizeof(mp2));

for(i = 1, j = 0; i <= K; i++, j++)mp1[j] = i, mp2[i] = 1<<j;

for(i = n - K + 1; i <= n; i++, j++)mp1[j] = i, mp2[i] = 1<<j;

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

for(u = 1; u <= n; u++) {

cmin(dp[u][1<<i], gg[u][mp1[i]]);

inque[u][1<<i] = 1;

que.push(MP(u, 1<<i));

}

}

while(!que.empty()) {

u = que.front().AA; su = que.front().BB; que.pop();

sub = su;

tp = dp[u][su];

do {

sv = sub ^ su;

cmin(tp, dp[u][sub] + dp[u][sv]);

sub = (sub - 1) & su;

}while(sub != su);

if(tp < dp[u][su]) dp[u][su] = tp;

for(j = 0; j < g[u].SZ; j++) {

v = g[u][j].AA; w = g[u][j].BB;

if(su & mp2[v]) continue;

sv = su | mp2[v];

if(dp[v][sv] > dp[u][su] + w) {

dp[v][sv] = dp[u][su] + w;

if(!inque[v][sv]) {

inque[v][sv] = 1;

que.push(MP(v, sv));

}

}

}

for(j = 0; j < kk; j++) {

if(su & (1<<j)) continue;

sv = su | (1<<j);

if(dp[u][sv] > dp[u][su] + gg[u][mp1[j]]) {

dp[u][sv] = dp[u][su] + gg[u][mp1[j]];

if(!inque[u][sv]) {

inque[u][sv] = 1;

que.push(MP(u, sv));

}

}

}

inque[u][su] = 0;

}

for(i = 0; i <= kk; i++) split[i].clear();

for(i = 0; i < (1<<kk); i++) {

split[\_\_builtin\_popcount(i)].PB(i);

gao[i] = MOD;

if(has(i)) {

for(j = 1; j <= n; j++) {

cmin(gao[i], dp[j][i]);

}

}

}

for(i = 1; i <= kk; i++) {

for(j = 0; j <split[i].SZ; j++) {

su = split[i][j];

for(sub = su - 1 & su; sub; sub = sub - 1 & su) {

sv = su ^ sub;

cmin(gao[su], gao[sub] + gao[sv]);

sub = (sub - 1) & su;

}

}

}

return gao[(1<<kk) - 1];

}

int main() {

int i,j,k,u,v,w;

int te;

scanf ("%d", &te);

for (int ca = 1; ca <= te; ca ++) {

scanf ("%d%d%d", &n, &m, &K);

for (i = 1; i <= n; i++) g[i].clear();

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

for (j = 1; j <= n; j++) gg[i][j] = MOD;

gg[i][i] = 0;

}

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

scanf ("%d%d%d", &u, &v, &w);

cmin (gg[u][v], w);

cmin (gg[v][u], w);

}

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

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

if (gg[i][j] < MOD) {

g[i].PB (MP (j, gg[i][j]) );

g[j].PB (MP (i, gg[i][j]) );

}

}

}

for (k = 1; k <= n; k++) {

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

for (j = 1; j <= n; j++) {

cmin (gg[i][j], gg[i][k] + gg[k][j]);

}

}

}

ans = 0;

ans = doit();

if(ans >= MOD) printf("No solution\n");

else printf ("%d\n", ans);

}

return 0;

}

### 最优比例生成树

double prim (double s) {

clr (vis , 0);

double retdis = 0;

int retcost = 0;

vis[1] = 1;

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

lowcost[i] = abs (v[1].h - v[i].h) - s \* dis[1][i];

closest[i] = 1;

}

for (int num = 0 ; num < n - 1 ; num++) {

double minans = inf;

int u;

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

if (vis[i]) continue;

if (lowcost[i] < minans) {

minans = lowcost[i];

u = i;

}

}

vis[u] = 1;

retcost += abs (v[closest[u]].h - v[u].h);

retdis += dis[closest[u]][u];

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

double cal = abs (v[u].h - v[i].h) - s \* dis[u][i];

if (!vis[i] &&cal < lowcost[i]) {

lowcost[i] = cal; closest[i] = u;

}

}

}

return 1.0 \* retcost / retdis;

}

int main() {

double ans = 0;

while (1) {

double tmp = prim (ans);

if (fabs (tmp - ans) < eps) {

break;

}

ans = tmp;

}

printf ("%.3f\n" , ans);

}

### 度限制最小生成树

就是图中的某些点在生成树时有度的限制，找满足这些约束的最小生成树。如果所有点都有度限制，那么这个问题将是NP难题），具体算法如下： 1. 先求出最小 m 度限制生成树：原图中去掉和 V0 相连的所有边（可以事先存两个图， Ray 的方法是一个邻接矩阵，一个邻接表，用方便枚举边的邻接表来构造新图），得到 m 个连通分量，则这 m 个连通分量必须通过 v0 来连接，所以，在图 G 的所有生成树中 dT (v0) ≥m 。也就是说，当 k<m 时，问题无解。对每个连通分量求一次最小生成树（哪个算法都行），对于每个连通分量 V’ ，用一条与 V0 直接连接的最小的边把它与 V0 点连接起来，使其整体成为一个生成树。于是，我们就得到了一个 m 度限制生成树，不难证明，这就是最小 m 度限制生成树。 2. 由最小 m 度限制生成树得到最小 m+1 度限制生成树;

：连接和 V0 相邻的点 v ，则可以知道一定会有一个环出现（因为原来是一个生成树），只要找到这个环上的最大权边（不能与 v0 点直接相连）并删除，就可以得到一个 m+1 度限制生成树，枚举所有和 V0 相邻点 v ，找到替换后，增加权值最小的一次替换 (当然，找不到这样的边时，就说明已经求出) ，就可以求得 m+1 度限制生成树。。如果每添加一条边，都需要对环上的边一一枚举，时间复杂度将比较高（但这个题数据比较小，所以这样也没问题，事实上，直接枚举都能过这个题），这里，用动态规划解决。设 Best (v) 为路径 v0—v 上与 v0 无关联且权值最大的边。定义 father (v) 为 v 的父结点，由此可以得到动态转移方程： Best (v) =max (Best (father (v) ),ω (father (v),v) ) ，边界条件为 Best[v0]=-∞ (因为我们每次寻找的是最大边，所以 -∞ 不会被考虑) ，Best[v’]=-∞| (v0,v’) ∈E (T) 。 3. 当 dT (v0) =k 时停止 (即当 V0 的度为 k 的时候停止) ，但不一定 k 的时候最优。 接下来说一下算法的实现： 采用并查集+ kruskal 代码量还少一点。 首先， 每个连通分量的的最小生成树可以直接用一个循环，循环着 Kruskal 求出。这里利用了联通分量间的独立性，对每个连通分量分别求最小生成树，和放在一起求，毫不影响。而且kruskral算法保证了各连通分量边的有序性。 找最小边的时候，上面讲的动态规划无疑是一种好方法，但是也可以这么做： 先走一个循环，但我们需要逆过来加边，将与v0关联的所有边从小到达排序，然后将各连通分量连接起来，利用并查集可以保证每个连通分量只有一条边与v0相连，由于边已经从小到达排序，故与每个连通分量相连的边就是每个连通分量与v0相连中的最小边。 然后求 m+1 度的最小生成树时，可以直接用 DFS ，最小生成树要一直求到 k 度，然后从中找出一个最优值。 int vis[MAXN];

int lowcost[MAXN];

int best[MAXN];

int pre[MAXN];

int n , k , ans , cnt;

int g[MAXN][MAXN];

int edge[MAXN][MAXN];

int fa[MAXN];

int mark[MAXN];

map <string ,int> m;

void init() {

clr (g , 0x3f);

cnt = 1;

}

void dfs (int s) {

for (int i = 1 ; i <= cnt ; i++) {

if (mark[i] && edge[s][i]) {

mark[i] = 0;

fa[i] = s;

dfs (i);

}

}

}

int prim (int s) {

clr (mark , 0);

int sum , pos;

vis[s] = mark[s] = 1;

sum = 0;

for (int i = 1 ; i <= cnt ; i++) {

lowcost[i] = g[s][i];

pre[i] = s;

}

for (int i = 1 ; i <= cnt ; i++) {

pos = -1;

for (int j = 1 ; j <= cnt ; j++) {

if (!vis[j] && !mark[j]) {

if (pos == -1 || lowcost[pos] > lowcost[j]) {

pos = j;

}

}

}

if (pos == -1) break;

vis[pos] = mark[pos] = 1;

edge[pre[pos]][pos] = edge[pos][pre[pos]] = 1;

sum += g[pre[pos]][pos];

for (int j = 1 ; j <= cnt ; j++) {

if (!vis[j] && !mark[j]) {

if (lowcost[j] > g[pos][j]) {

lowcost[j] = g[pos][j];

pre[j] = pos;

}

}

}

}

int minedge = inf;

int root = -1;

for (int i = 1 ; i <= cnt ; i++) {

if (mark[i] && g[i][1] < minedge) {

minedge = g[i][1];

root = i;

}

}

mark[root] = 0;

dfs (root);

fa[root] = 1;

return sum + minedge;

}

int Best (int s) {

if (fa[s] == 1) return -1;

if (best[s] != -1) return best[s];

int tmp = Best (fa[s]);

if (tmp != -1 && g[fa[tmp]][tmp] > g[fa[s]][s]) {

best[s] = tmp;

} else best[s] = s;

return best[s];

}

void solve() {

clr (vis , 0);

clr (fa , -1);

clr (edge , 0);

vis[1] = 1;

int num = 0;

ans = 0;

for (int i = 1 ; i <= cnt ; i++) {

if (!vis[i]) {

num++;

ans += prim (i);

}

}

int minadd , change;

for (int i = num + 1 ; i <= k && i <= cnt; i ++) {

clr (best , -1);

for (int j = 1 ; j <= cnt ; j++) {

if (fa[j] != 1 && best[j] == -1) {

Best (j);

}

}

minadd = inf;

for (int j = 1 ; j <= cnt ; j++) {

if (g[1][j] != inf && fa[j] != 1) {

int a = best[j];

int b = fa[best[j]];

int tmp = g[1][j] - g[a][b];

if (minadd > tmp) {

minadd = tmp;

change = j;

}

}

}

if (minadd >= 0) break;

ans += minadd;

int a = best[change];

int b = fa[best[change]];

g[a][b] = g[b][a] = inf;

fa[a] = 1;

g[1][a] = g[a][1] = g[change][1];

g[1][change] = g[change][1] = inf;

}

}

int main() {

m.clear();

m["Park"] = 1;

init();

scanf ("%d" , &n);

string str1 , str2;

int v;

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

cin >> str1 >> str2;

scanf ("%d" , &v);

int a = m[str1];

int b = m[str2];

if (!a) {

m[str1] = a = ++cnt;

}

if (!b) {

m[str2] = b = ++cnt;

}

if (g[a][b] > v) g[a][b] = g[b][a] = v;

}

scanf ("%d" , &k);

solve();

printf ("Total miles driven: %d\n" , ans);

19

return 0;

}

### 生成树计数

Matrix-Tree定理(Kirchhoff矩阵-树定理)

Kirchhoff矩阵C[G]=D[G]-A[G] (度数矩阵-邻接矩阵)

求C[G]任意一个n-1阶主子式的行列式(取出第i行和第i列)

//spoj 104 Highways

//朴素求解, ans是double用printf("%0.0f\n",ans)输出成int

//返回是否有解,若有解ans\*=a[i][i]中

#define eps 1e-10

#define fabs(x) ((x)>eps?(x):-(x))

#define ZERO(x) (fabs(x) > eps ? 0 : 1)

int gauss\_tpivot (int n, double a[][Maxn]) {

int i, j, k, p;

double maxp, t;

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

if(ZERO(a[k][k])) {

for(i = k + 1; i < n; i++) if(!ZERO(a[i][i])) break;

if(i >= n) return 0;

for(j = k; j < n; j++) {

swap(a[k][j], a[i][j]);

}

}

maxp = a[k][k];

for (j = k + 1; j < n; j++) {

a[k][j] /= maxp;

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

a[i][j] -= a[i][k] \* a[k][j];

}

}

return 1;

}

int main() {

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

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

a[i][j] = b[i][j] = 0;

}

}

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

scanf("%d%d", &u, &v);

if(u == v) continue;

u--; v--;

a[u][u]++; a[v][v]++;

b[u][v] = b[v][u] = 1;

}

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

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

a[i][j] = a[i][j] - b[i][j];

}

}

double ans = 0;

if(gauss\_tpivot(n - 1, a)) {

ans = 1;

for(i = 0; i < n - 1; i++) ans \*= a[i][i];

}

else ans = 0;

printf("%0.0f\n", ans);

}

//hdu 4305 MOD = 10007

//求n阶方阵的行列式

//MST计数调用detMod(n-1, a, MOD)

LL detMod(int n, LL a[][Maxn], LL MOD) {

LL ret = 1;

int i, j, k, t;

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

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

a[i][j] = (a[i][j] + MOD) % MOD;

}

}

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

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

while(a[j][i]) {

t = a[i][i] / a[j][i];

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

a[i][k] = (a[i][k] - a[j][k] \* t) % MOD;

swap(a[i][k], a[j][k]);

}

ret = -ret;

}

}

if(a[i][i] == 0) return 0;

ret = ret \* a[i][i] % MOD;

}

return (ret + MOD) % MOD;

}

int main() {

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

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

a[i][j] = b[i][j] = 0;

}

}

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

scanf("%d%d", &u, &v);

if(u == v) continue;

u--; v--;

a[u][u]++; a[v][v]++;

b[u][v]++; b[v][u]++;

}

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

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

a[i][j] = a[i][j] - b[i][j];

}

}

double ans = 0;

if(gauss\_tpivot(n - 1, a)) {

ans = 1;

for(i = 0; i < n - 1; i++) ans \*= a[i][i];

}

else ans = 0;

printf("%0.0f\n", ans);

}

### 生成树相关问题

// 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 \*/

## 最短路

### 最短路次短路计数

void dij() {

clr (cnt , 0);

clr (vis , 0);

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

dis[i][0] = inf;

dis[i][1] = inf;

}

dis[st][0] = 0;

cnt[st][0] = 1;

for (int i = 1 ; i <= 2 \* n ; i++) {

int tmp = inf , tx = -1 , flag;

for (int j = 1 ; j <= n ; j++) {

if (!vis[j][0] && dis[j][0] < tmp) {

tmp = dis[j][0];

tx = j;

flag = 0;

} else

if (!vis[j][1] && dis[j][1] < tmp) {

tmp = dis[j][1];

tx = j;

flag = 1;

}

}

if (tx == -1) break;

13

vis[tx][flag] = 1;

for (int j = head[tx] ; ~j ; j = e[j].next) {

int v = e[j].v , c = e[j].c;

if (dis[tx][flag] + c < dis[v][0]) {

dis[v][1] = dis[v][0];

cnt[v][1] = cnt[v][0];

dis[v][0] = dis[tx][flag] + c;

cnt[v][0] = cnt[tx][flag];

} else

if (dis[tx][flag] + c == dis[v][0]) {

cnt[v][0] += cnt[tx][flag];

} else

if (dis[tx][flag] + c == dis[v][1]) {

cnt[v][1] += cnt[tx][flag];

} else

if (dis[tx][flag] + c < dis[v][1]) {

dis[v][1] = dis[tx][flag] + c;

cnt[v][1] = cnt[tx][flag];

}

}

}

}

### K短路

练手: POJ2449 UVA10740

提高: SGU145 SGU314

最短路+A\*

f[x] = g[x] + h[x]

其中h[x]以从t到x的最短路

g[x]为从s到x的当前路径长度

YEN算法

MPS算法

最短路树+回溯

带环K短路

无环K短路

不重复k短路

费用流~.~

## 欧拉回路

### 基础

判断无向图欧拉路和欧拉回路的条件

判断有向图欧拉路和欧拉回路的条件

注意图的连通性的影响

fleury算法(和一般dfs的差别???)

### 磁鼓欧拉回路(HDU 2894)

n个触头, 建图以(1<<(n-1))个结点, 建1<<n条有向边, 从最大结点出发跑欧拉回路

### 混合欧拉路欧拉回路的判断-->网络流模型

(相关题目：pku1637，zju1992，hdu3472, hdu1956)

给出一张混合图（有有向边，也有无向边），判断是否存在欧拉回路。

首先是对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（indeg - outdeg）是奇数的话，一定不存在欧拉回路；

如果所有点的入度和出度之差都是偶数，那么就开始网络流构图：

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），那么存在欧拉回路，否则不存在。

建图基础:

//outdeg出度, indeg入度, !w双向边

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

scanf("%d%d%d", &u, &v, &w);

outdeg[u]++;

indeg[v]++;

if(!w) {

adde(u, v, 1);

}

}

int flag = true;

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

if(abs(indeg[i] - outdeg[i]) & 1) {

flag = false;

break;

}

}

if(flag) {

S = 0; T = n + 1; sums = sumt = 0;

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

if(outdeg[i] > indeg[i]) {

adde(S, i, (outdeg[i] - indeg[i]) / 2);

sums += (outdeg[i] - indeg[i]) / 2;

}

if(indeg[i] > outdeg[i]) {

adde(i, T, (indeg[i] - outdeg[i]) / 2);

sumt += (indeg[i] - outdeg[i]) / 2;

}

}

int flow = dinic(S, T, T + 1);

if(flow != sums) flag = false;

}

## RMQ

### RMQ实现模板

验题 POJ3264

struct RMQ {

for(j = 1, k = 0; (1<<j) <= n; j++, k++)

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

st[j][i] = min(st[j - 1][i], st[j - 1][i + (1<<k)]);

int query(int l, int r) {

// int k = 31 - \_\_builtin\_clz(r - l + 1);

// int k = kk[r - l + 1];

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

}

};

## 最小环&最大环

### 最小环

验题: POJ1734

//floyd算法

//g[][] 边

//f[][] 最短路

//pre[][] 记录路径

int MinCircle() {

int i, j, k, p;

int mc = MOD;

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

for(j = 1; j <= n; ++j) {

f[i][j] = g[i][j];

if(g[i][j] < MOD && i != j) {

pre[i][j] = j; //记录初始长度为1的路径

}

else {

pre[i][j] = -1;

}

}

}

for(k = 1; k <= n; ++k) {

for(i = 1; i < k ; ++i) {

for(j = i + 1; j < k; ++j) {

int tp = g[k][i] + f[i][j] + g[j][k];

if(tp < mc) { //更新最小环和路径

mc = tp;

for(p = i, len = 0; p != -1; p = pre[p][j], len++) {

path[len] = p;

}

path[len++] = k;

}

}

}

for(i = 1; i <= n; ++i) { //普通floyd过程

for(j = 1; j <= n; ++j) {

if(f[i][j] > f[i][k] + f[k][j]) {

f[i][j] = f[i][k] + f[k][j];

pre[i][j] = pre[i][k];

}

}

}

}

return mc;

}

未验题:

//http://www.cnblogs.com/Yz81128/archive/2012/08/15/2640940.html

//dijkstra

//求一条边的两段的结点之间最短路(不包含这条边)和这条边之和

//O(m\*m\*logn)

## 拓扑排序

### 基础知识

一、拓扑排序

什么是拓扑排序？简单地说，由某个集合上的一个偏序得到该集合上的一个全序，这个操作称之为拓扑排序。

AVO网（Activity On Vertex Network），也称作顶点表示活动的网，就是用顶点表示活动，用弧表示活动间的优先关系的有向图。

拓扑排序是对AOV网构造一个线性序列，使所有的优先关系在此序列中得以体现，换句话说，就是在AVO网中，若Vi是Vj的前驱，那么在拓扑序列中，Vi必须排在Vj之前。

拓扑排序的步骤如下：

1、在有向图中选取一个没有前驱的顶点，并输出；

2、从图中删除该顶点和所有以它为尾的弧；

3、重复1和2，直到全部顶点均被输出为止。

（注：弧尾指弧的起始点，弧头是指弧的终点）

一个有向无环图的拓扑序列是不唯一的。

二、关键路径

与AVO网相对应的是AOE网（Activity On Edge）即边表示活动的网。

所谓AOE网，也称作“用表示活动的网”，就是用顶点表示事件，用弧表示活动，弧上的权值表示该活动的持续时间的有向图。

关键路径：是指AOE网上，完成工程的最短时间是从开始点到完成点的最长路径长度，路径长度最长的路径叫做关键路径。

分析关键路径的目的是辨别哪些是关键活动，以便争取提高关键活动的工效，缩短整个工期。

事件Vi的最早发生时间Ve(i)中从顶点V到Vi的最长路径长度。

活动Ai的最迟开始时间V(i)是该活动的终点所表示的事件最迟发生时间与该活动的所需时间之差。

一个活动的最迟开始时间减去最早开始时间之差定义为完成该活动的时间余量，而最迟开始时间等于最早开始时间的活动叫做关键活动。

求关键路径的算法描述：

(1) 输入e条弧<j,k>，建立AOE网的存储结构。

(2) 从源点v1出发，令ve(1)=0,求 ve(j) 2<=j<=n。

(3) 从汇点vn出发，令vl(n)=ve(n),求 vl(i) 1<=i<=n-1。

(4) 根据各顶点的ve和vl值，求每条弧s（活动）的最早开始时间e(s)和最晚开始时间l(s)，其中e(s)=l(s)的为关键活动。

求关键路径的算法分析

（1） 求关键路径必须在拓扑排序的前提下进行，有环图不能求关键路径；

（2） 只有缩短关键活动的工期才有可能缩短工期；

（3） 若一个关键活动不在所有的关键路径上，减少它并不能减少工期；

（4） 只有在不改变关键路径的前提下，缩短关键活动才能缩短整个工期。

### 求任意拓扑排序或判断是否可以拓扑排序

队列实现

dfs实现

记录各点完成访问的时刻(完成时间),用DFS遍历一次整个图,得出各结点的完成时间,然后按完成时间倒序排列就得到了图的拓扑序列

/\* 拓扑排序 O(e)

\* 确保是有向无环图！

\* 结果逆序存放在sta中！

\* \*/

VI ve[Maxn];

bool vis[Maxn];

int sta[Maxn], top;

void dfsTopo(int u) {

vis[u] = true;

for (int i = 0; i < ve[u].size(); i ++ )

if (!vis[ve[u][i]])

dfsTopo(ve[u][i]);

sta[top ++ ] = u;

}

void Toposort(int n) {

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

top = 0;

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

if (!vis[i]) dfsTopo(i);

}

### 字典序最小解

一般直接将队列改为优先队列即可(要求无重复标号!!!)

HDU1285

区分字典序最小和标号小的尽量靠前 特殊题目要求: POJ3687

int n, m;

struct node {

int u, v, next;

}e[Maxm];

int tot, last[Maxn];

priority\_queue<int > Q;

vector<int> ans, out;

int deg[Maxn];

void adde(int u, int v) {

e[tot].u = u; e[tot].v = v; e[tot].next = last[u]; last[u] = tot++;

}

int main() {

int i, j, u, v, te;

scanf("%d", &te);

while(te--) {

scanf("%d%d", &n, &m);

tot = 0;

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

last[i] = -1;

deg[i] = 0;

}

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

scanf("%d%d", &u, &v);

adde(v, u);

deg[u]++;

}

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

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

if(deg[i] == 0) Q.push(i);

}

ans.clear();

out.clear();

while(!Q.empty()) {

u = Q.top();

Q.pop();

ans.PB(u);

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

v = e[j].v;

deg[v]--;

if(deg[v] == 0) Q.push(v);

}

}

if(ans.SZ < n) {

printf("-1\n");

}

else {

reverse(ans.BG, ans.ED);

out = ans;

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

out[ans[i] - 1] = i + 1;

}

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

if(i) printf(" ");

printf("%d", out[i]);

}

printf("\n");

}

}

return 0;

}

/\*

2

5 4

5 1

4 2

1 3

2 3

10 5

4 1

8 1

7 8

4 1

2 8

ans:

2 4 5 3 1 逆向建图

5 1 6 2 7 8 3 4 9 10 没有判重边的话就输出 -1

\*/

有重复标号字典序最小???

### 有向图拓扑排序计数(状压dp)

验题: ZOJ1346 HDU4917

// dp[status] 当前图中结点01表示, 初始化只剩一个点dp[1<<j]=1

// deg[] 结点度数

// g[][] vector存边

// group[] 将当前块结点映射到group的0~now位置

// now当前块节点数

// 各个连通块求dp值, 乘上组合数C[n][now], 没处理一块n-=now

int doit(int status) {

if(dp[status] != -1) return dp[status];

LL sum = 0;

int i, j, u, v;

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

u = group[i];

if(deg[u] == 0) {

deg[u] = -1;

for(j = 0; j < g[u].SZ; j++) {

v = g[u][j];

deg[v]--;

}

sum += doit(status ^ (1<<i));

for(j = 0; j < g[u].SZ; j++) {

v = g[u][j];

deg[v]++;

}

deg[u] = 0;

}

}

return (dp[status] = sum % MOD);

}

### 树拓扑排序计数

计算公式

n!/node[i] (1 < i <= n)

n表示树的总节点数, node[i] 表示以i为根的子树的节点数

### 关键路径

求关键路径：

1. 关键节点：

正向拓扑序， 求每个点最早到达时间early[i] = max(early[i], early[j] + edge[k]);

利用汇点的early值，反向拓扑求每个点迟到达时间late[i] = min(late[i], late[j] - edge[k]);

关键节点early == late

PS：

最早开始时间可以理解为是必须等到之前的任务完成才能做

最迟开始时间可以理解为是必须为后面的任务留出足够的时间

2. 关键路径 ：

源点到汇点的最长路（可能有多条）

## LCA

### 倍增算法

//改bfs

//验题: POJ1330,

//LCA O(nlogn)

//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

struct node {

int u, v, l, next;

}e[Maxm];

int tot, last[Maxn];

void adde(int u, int v) {

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

e[tot].next = last[u]; last[u] = tot++;

}

int depth[Maxn], fa[STEP][Maxn];

int n, m;

void initLCA() {

int i, j;

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

depth[i] = -1;

last[i] = -1;

for(j = 0; j < STEP; j++) fa[j][i] = -1;

}

tot = 0;

}

void dfsLCA(int u) {

int j, v;

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

v = e[j].v;

if(depth[v] == -1) {

fa[0][v] = u;

depth[v] = depth[u] + 1;

dfsLCA(v);

}

}

}

int getLCA (int x, int y) {

int i, dif = abs(depth[x] - depth[y]);

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

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

if ((1 << i) & dif) {

dif -= (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

验题: HDU2586

//Tarjan离线LCA

//e存放树边

//q存放query

struct GRAPH {

struct node {

int u, v, n, next;

}adj[Maxm];

int tot, last[Maxn];

void init(int n) {

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

tot = 0;

}

void adde(int u, int v, int n) {

adj[tot].u = u; adj[tot].v = v; adj[tot].n = n;

adj[tot].next = last[u]; last[u] = tot++;

}

}e, q;

int fa[Maxn], lca[Maxn], ans[Maxn];

int visit[Maxn];

int getfa(int x) {

if(x == fa[x]) return x;

else return (fa[x] = getfa(fa[x]));

}

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

0: (1)

/ \

1: (2) (7)

/ \ \

2: (3) (4) (8)

/ \

3: (5) (6)

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

## 连通性

### 点双连通(Tarjan+重建图)

#define STEP 20

struct GRAPH {

struct node {

int u, v, next;

}adj[Maxm];

int tot, last[Maxn];

void init(int n) {

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

tot = 0;

}

void adde(int u, int v) {

// cout << "adde " << u << " " << v << endl;

adj[tot].u = u; adj[tot].v = v;

adj[tot].next = last[u]; last[u] = tot++;

}

}e, e1;

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

int lcnt[Maxm];

int nindex, ncnt;

stack<int> sta;

int n, m, q;

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]);

}

}

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

// if(child

}

set<PII> S;

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);

}

}

int visit[Maxm];

void gao(int u) {

int j, v;

for(j = e1.last[u]; j != -1; j = e1.adj[j].next) {

v = e1.adj[j].v;

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

visit[v] = 1;

gao(v);

}

}

}

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);

}

}

}

void solve() {

int i, j;

memset(dfn, 0, sizeof(dfn));

memset(low, 0, sizeof(low));

memset(iscut, -1, sizeof(iscut));

memset(color, 0, sizeof(color));

memset(belong, -1, sizeof(belong));

ncnt = nindex = 0;

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;

}

}

buildGraph();

}

## 2 SAT

### 综述

综述：每个条件的形式都是x[i]为真/假或者x[j]为真/假， 每个x[i]拆成2\*i和2\*i+1两个点，分别表示x[i]为真，x[i] 为假；加的每一条边之间的关系是and

模型一：两者（A，B）不能同时取（但可以两个都不选） 说明：A 为假或 B 为假 那么选择了 A 就只能选择 B’，选择了 B 就只能选择 A’ 连边 A→B’，B→A’

模型二：两者（A，B）不能同时不取（但可以两个都选） 说明：A 为真或 B 为真 那么选择了 A’就只能选择 B，选择了 B’就只能选择 A 连边 A’→B，B’→A

模型三：两者（A，B）要么都取，要么都不取 说明：...... 那么选择了 A，就只能选择 B，选择了 B 就只能选择 A，选择了 A’就只能选择 B’， 选择了 B’就只能选择 A’ 连边 A→B，B→A，A’→B’，B’→A’

模型四：两者（A，A’）必取A 那么，那么，该怎么说呢？先说连边吧。 连边 A’→A

模型五（补充） ：两者（A，B）两个必须不相同，即要么选A，要么选B 逻辑表达：A||B 非 A||非 B 连边：A 为真或 B 为真： A’--->B B’--->A;

A 为假或 B 为假: A-->B’ B-->A 说明：A 或 B，非 A 或非 B，前者表示两者至少有一个 true，后者表示至少有一个 false

### 判断可行性

建图，求强连通，判断任意对象的两部是否属于同一强连通：

### 求任意解

方法是，对原图求一次强连通分量，然后看每组中的两个点是否属于同一个强连通分量，如果存在这种情况，那么无解

然后对于缩点后的图G'，我们将G'中所有边转置。进行拓扑排序。

对于缩点后的所有点，我们先预处理求出所有冲突顶点。例如缩点后Ai所在强连通分支的ID为id[ Ai] ，同理~Ai在 id[~Ai], 所以冲突顶点

conflict[ id[Ai] ]=conflict[ id[~Ai] ];

同理conflict[ id[~Ai] ]=conflict[ id[Ai] ];

设缩点后有Nscc个点。

然后对拓扑序进行染色，初始化所有点color均为未着色

顺序遍历得到的拓扑序列，对于未着色的点x，将x染成红色，同时将所有与x矛盾的点conflic[x]染成蓝色。

2-sat的一组解就等价于所有缩点后点颜色为红色的点，也就是color[ id[i] ]=RED的所有点

缩点

　　Tarjan算法缩点，将所有的边反过来（ 为什么？后面有嗯 ）

判可行

　　枚举集合的两个元素，看其是否处于不同的块内，若否的话则给出不可行信息

记录矛盾

　　这里所说的矛盾不是题中给出的两个人之间有仇恨，那样的边是实际存在，我们这里说的矛盾是指若两个块分别含有两个对立节点，也就是说一个集合的两个元素分布在了两个不同的块中，那么这两个块就是矛盾的，即不可能同时被选择，这样一种矛盾关系是不存在于边中的，是不依赖于输入的数据的，我们要找到与一个块对立的块，并把它们保存下来。

拓扑排序

　　将缩点后的图进行拓扑排序（排的是块而不是节点）

构造方案

　　按照拓扑序列的顺序，依次访问所有块，若一个块未被标记，将其标记为“选择”，不传递“选择”标记，将被选块的对立块标记为“不选择”，将其“不选择”标记沿着边正向传递。（ 不是逆着边么？哼，图已经被反过来了，你到底有没有认真看呐！囧 ）

/\*2-sat\*/

#define Maxn 5010

vector<VI> e, ne;

int dfn[Maxn], low[Maxn], block[Maxn], sta[Maxn], top, tsp, N;

int sta1[Maxn], rid[Maxn], top1; //rid 新旧图映射，sta1, top1 拓扑排序，输出路径

/\*初始化 top = tsp = N = 0 和 block为 -1, dfn 为 0\*/

void tarjan(int u, int root) {

dfn[u] = low[u] = ++ tsp;

sta[top ++ ] = u;

for (int i = 0; i < e[u].size(); i ++ ) {

int v = e[u][i];

if (!dfn[v]) tarjan(v, root);

if (dfn[v] < dfn[root]) continue;

low[u] = min(low[u], low[v]);

}

if (low[u] == dfn[u]) {

int v;

do {

v = sta[ -- top];

block[v] = N;

} while (v != u);

rid[N] = u;

sta1[top1 ++ ] = N;

N ++ ;

}

}

int color[Maxn];

void colorBlue(int u) {

color[u] = 2;

for (int i = 0; i < ne[u].size(); i ++ ) {

int v = ne[u][i];

if (!color[v])

colorBlue(v);

}

}

bool twoSat(int n) {

memset(dfn, 0, sizeof(dfn));

memset(block, -1, sizeof(block));

top1 = top = N = tsp = 0;

for (int i = 0; i < n \* 2; i ++ )

if (!dfn[i])

tarjan(i, i);

for (int i = 0; i < n \* 2; i += 2 )

if (block[i] == block[i ^ 1])

return 0;

//return 1;

/\*建立新图\*/

ne.clear();

for (int i = 0; i < N; i ++ ) {

VI p;

ne.PB(p);

}

for (int i = 0; i < n \* 2; i ++ )

for (int j = 0; j < e[i].size(); j ++ )

if (block[i] != block[e[i][j]])

ne[block[e[i][j]]].PB(block[i]); //反向建图

memset(color, 0, sizeof(color));

for (int i = 0; i < top1; i ++ ) {

int x = sta1[i];

if (!color[x]) {

color[x] = 1;

x = block[rid[x] ^ 1];

colorBlue(x);

}

}

return 1;

}

### 输出任一解

//POJ3648

#define Maxn 411

#define Maxm 411111

/\*

\* SCC 使用图struct

\*/

struct GRAPH {

struct node {

int u, v, next;

}adj[Maxm];

int tot, last[Maxn];

void init(int n) {

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

tot = 0;

}

void adde(int u, int v) {

// cout << "adde " << u << "-->" << v << endl;

adj[tot].u = u; adj[tot].v = v;

adj[tot].next = last[u]; last[u] = tot++;

}

}e, e1;

int dfn[Maxn], low[Maxn], belong[Maxn], instack[Maxn], ncnt, nindex;

stack<int> sta;

int rid[Maxn], sta1[Maxn], top1; //rid 新旧图映射，sta1拓扑排序，输出路径

void Tarjan(int u) {

int j, v;

dfn[u] = low[u] = nindex++;

sta.push(u);

instack[u] = 1;

for(j = e.last[u]; ~j; j = e.adj[j].next) {

v = e.adj[j].v;

if(-1 == dfn[v]) {

Tarjan(v);

if(low[u] > low[v]) low[u] = low[v];

}

else if(instack[v] && dfn[v] < low[u]) {

low[u] = dfn[v];

}

}

if(dfn[u] == low[u]) {

do {

v = sta.top();

sta.pop();

instack[v] = 0;

belong[v] = ncnt;

}while(u != v);

rid[ncnt] = u;

sta1[top1++] = ncnt;

ncnt++;

}

}

int color[Maxn];

void colorBlue(int u) {

int j, v;

color[u] = 2;

for(j = e1.last[u]; j != -1; j = e1.adj[j].next) {

v = e1.adj[j].v;

if(!color[v]) colorBlue(v);

}

}

void buildGraph() {

int i, j, u, v, x, y;

e1.init(ncnt);

for(j = 0; j < e.tot; j++) {

x = belong[u = e.adj[j].u];

y = belong[v = e.adj[j].v];

if(x != y) {

e1.adde(y, x);//反向建图

}

}

}

int twoSat(int n) {

int i, u, v, flag = 0;

memset(dfn, -1, sizeof(dfn));

memset(low, -1, sizeof(low));

memset(instack, 0, sizeof(instack));

memset(belong, -1, sizeof(belong));

ncnt = 0; //缩点, 新点[0,ncnt)

nindex = 1;

top1 = 0;

for(i = 0; i < 2\*n; i++) {

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

while(!sta.empty()) sta.pop();

Tarjan(i);

}

}

// for(i = 0; i < 2 \* n; i++) cout << i << " ~ " << belong[i] << endl;

for(i = 0; i < 2 \* n; i += 2) {

if(belong[i] == belong[i ^ 1]) return false;

}

//建新图

buildGraph();

memset(color, 0, sizeof(color));

for(i = 0; i < top1; i++) {

u = sta1[i];

if(!color[u]) {

color[u] = 1;

v = belong[rid[u] ^ 1];

colorBlue(v);

}

}

return 1;

}

int n, m;

char cu, cv;

int main() {

int i, j, u, v, w, x, y;

//freopen("", "r", stdin);

//freopen("", "w", stdout);

while(scanf("%d%d", &n, &m) != EOF) {

if(n == 0 && m == 0) break;

e.init(4 \* n);

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

scanf("%d%c%d%c", &u, &cu, &v, &cv);

if(cu == 'w') u <<= 1;

else u=(u<<1)|1;

if(cv == 'w') v <<= 1;

else v=(v<<1)|1;

if(u != (v ^ 1)) {

e.adde(u, v ^ 1);

e.adde(v, u ^ 1);

}

}

e.adde(0, 1);

if(!twoSat(n)) {

printf("bad luck\n");

}

else {

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

if(i > 1)printf(" ");

if(color[belong[i<<1]] == 1) {

printf("%dh", i);

}

else printf("%dw", i);

}

printf("\n");

}

}

return 0;

}

/\*

2 3

0w 1h

1w 0h

0h 1h

10 6

3h 7h

5w 3w

7h 6w

8w 3w

7h 3w

2w 5h

0 0

\*/

### 输出任意解from XMZhou

struct Edge {

int v,next;

} e[MAXE];

int head[MAXN],num;

void init() {

num = 0;

memset (head,-1,sizeof (head) );

}

void addedge (int u,int v) {

e[num].v = v;

e[num].next = head[u];

head[u] = num++;

}

int Low[MAXN],DFN[MAXN],Stack[MAXN],Belong[MAXN];//Belong数组的值1~scc

int Index,top;

int scc;

bool Instack[MAXN];

void Tarjan (int u) {

int v;

Low[u] = DFN[u] = ++Index;

Stack[top++] = u;

Instack[u] = true;

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

v = e[i].v;

if ( !DFN[v] ) {

Tarjan (v);

if (Low[u] > Low[v]) Low[u] = Low[v];

} else

if (Instack[v] && Low[u] > DFN[v]) Low[u] = DFN[v];

}

if (Low[u] == DFN[u]) {

scc++;

do {

v = Stack[--top];

Instack[v] = false;

Belong[v] = scc;

} while (v != u);

}

}

bool solvable (int n) { //n是总个数,需要选择一半

memset (DFN,0,sizeof (DFN) );

memset (Instack,false,sizeof (Instack) );

Index = scc = top = 0;

for (int i = 0; i < n; i++) if (!DFN[i]) Tarjan (i);

for (int i = 0; i < n; i += 2) {

if (Belong[i] == Belong[i^1]) return false; //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* //拓扑排序求任意一组解部分

}

return true;

}

queue<int>q1,q2;

vector<vector<int> > dag;//缩点后的逆向DAG图

char color[MAXN];//染色，为'R'是选择的

int indeg[MAXN];//入度

int cf[MAXN];

void solve (int n) {

dag.assign (scc+1,vector<int>() );

memset (indeg,0,sizeof (indeg) );

memset (color,0,sizeof (color) );

for (int u = 0; u < n; u++) for (int i = head[u]; i != -1; i = e[i].next) {

int v = e[i].v;

if (Belong[u] != Belong[v]) {

dag[Belong[v]].push\_back (Belong[u]);

indeg[Belong[u]]++;

}

}

for (int i = 0; i < n; i += 2) {

cf[Belong[i]] = Belong[i^1];

cf[Belong[i^1]] = Belong[i];

}

while (!q1.empty() ) q1.pop();

while (!q2.empty() ) q2.pop();

for (int i = 1; i <= scc; i++) if (indeg[i] == 0) q1.push (i);

while (!q1.empty() ) {

int u = q1.front();

q1.pop();

if (color[u] == 0) {

color[u] = 'R';

color[cf[u]] = 'B';

}

int siz = dag[u].size();

for (int i = 0; i < siz; i++) {

indeg[dag[u][i]]--;

if (indeg[dag[u][i]] == 0) q1.push (dag[u][i]);

}

}

}

int change (char s[]) {

int ret = 0;

int i = 0;

while (s[i]>='0' && s[i]<='9') {

ret \*= 10;

ret += s[i]-'0';

i++;

}

if (s[i] == 'w') return 2\*ret;

else return 2\*ret+1;

}

int main() {

int n,m;

char s1[10],s2[10];

while (scanf ("%d%d",&n,&m) == 2) {

if (n == 0 && m == 0) break;

init();

while (m--) {

scanf ("%s%s",s1,s2);

int u = change (s1);

int v = change (s2);

addedge (u^1,v);

addedge (v^1,u);

}

addedge (1,0);

if (solvable (2\*n) ) {

solve (2\*n);

for (int i = 1; i < n; i++) { //注意这一定是判断

color[Belong[ if (color[Belong[2\*i]] == 'R') printf ("%dw",i);

else printf ("%dh",i);

if (i < n-1) printf (" ");

else printf ("\n");

}

} else printf ("bad luck\n");

}

return 0;

}

### 求字典序最小的解

也就是说，沿着一条路径，如果一个点被选择了，那么这条路径以后的点都将被选择，那么，出现不可行的情况就是，存在一个党派两个党员都被选择了，那么，我们只需要枚举一下就可以了，数据不大，答案总是可以出来的。

这么一个简单的算法，放在第一个说，看来你已经知道它不是重点了，但是，若题目要求的是字典序最小的方案的话，那就只能选择这个算法了。

//求字典序最小解frome XMZhou

struct Edge {

int v,next;

} e[MAXE];

int head[MAXN],tot;

void init() {

tot = 0;

memset (head,-1,sizeof (head) );

}

void addedge (int u,int v) {

e[tot].v = v;

e[tot].next = head[u];

head[u] = tot++;

}

bool vis[MAXN];//染色标记，为true表示选择

int S[MAXN],top;//栈

bool dfs (int u) {

if (vis[u^1]) return false;

if (vis[u]) return true;

vis[u] = true;

S[top++] = u;

for (int i = head[u]; i != -1; i = e[i].next) if (!dfs (e[i].v) ) return false;

return true;

}

bool Twosat (int n) {

memset (vis,false,sizeof (vis) );

for (int i = 0; i < n; i += 2) {

if (vis[i] || vis[i^1]) continue;

top = 0;

if (!dfs (i) ) {

while (top) vis[S[--top]] = false;

if (!dfs (i^1) ) return false;

}

}

return true;

}

int main() {

int n,m;

int u,v;

while (scanf ("%d%d",&n,&m) == 2) {

init();

while (m--) {

scanf ("%d%d",&u,&v);

u--;

v--;

addedge (u,v^1);

addedge (v,u^1);

}

if (Twosat (2\*n) ) {

for (int i = 0; i < 2\*n; i++) if (vis[i]) printf ("%d\n",i+1);

} else printf ("NIE\n");

}

return 0;

}

## 匹配问题

### 基本概念

1. 最大匹配

1. dfs

2. bfs

3. Hopcroft-Karp

4. 利用贪心构造最初解再利用匈牙利增广路调整

5. 最大流

2. 最优完备匹配

1. KM( O(n^4)

2. KM + (O(n^3)) PS: 好像没有这么低的复杂度

3. 费用流

3. 稳定婚姻问题

4. 多重匹配

建立最大流模型解决

5.多重最优匹配

建立费用流模型解决

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, 否则算法复杂度很高...)

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, )

### HK (O(E\*N^0.5))

验题: HDU 2389 spoj 4206(n=50000,m=150000)

/\*

\* HK O(E\*V^0.5) 最大匹配

\* 左半图 0~n-1

\* 右半图 n~n+m-1

\*/

struct edgenode {

int v, next;

}e[Maxm];

int last[Maxn], tot;

int n, m, match = 0;

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

queue<int> Q;

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

void adde(int u, int v) {

e[tot].v = v; e[tot].next = last[u]; last[u] = tot++;

}

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;

// match += preHungary();

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;

}

### 稳定婚姻问题

延迟认可算法(Gale-Shapley算法)

int mx[Maxn], my[Maxm]; //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];

int cur[Maxn];

int n, m;

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) { //y没有匹配

mx[i] = v; my[v] = i;

}

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 {//i比以前的差, i找下一个y

que.push(i);

}

}

}

### 带花树

### 一般图最小权匹配

可将求一般图最小权完美匹配的问题转化为求最大权完美匹配的问题，在杜端甫编著的《运筹图论》和龚劬编的《图论与网络最优化算法》中均介绍了求一般图最大权完美匹配的算法，可参考。

## 最大流

### Dinic

//dinic O(m\*n^2)

const int MOD = 1000000007;

#define Maxn 10000

#define Maxm 100000

struct node {

int u, v, c, next;

}e[Maxm];

int tot, last[Maxn];

int cur[Maxn], dist[Maxn], que[Maxn], sta[Maxn], top, head, tail;

void adde(int u, int v, int c, int c1) {

e[tot].u = u; e[tot].v = v; e[tot].c = c; e[tot].next = last[u]; last[u] = tot++;

e[tot].u = v; e[tot].v = u; e[tot].c = c1; e[tot].next = last[v]; last[v] = tot++;

}

bool bfs(int s, int t, int n) {

int i, j, u, v;

for(i = 0; i < n; i++) dist[i] = MOD;

dist[s] = 0;

head = tail = 0;

que[tail++] = s;

while(head < tail) {

u = que[head++];

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

if(e[j].c == 0) continue;

v = e[j].v;

if(dist[v] > dist[u] + 1) {

dist[v] = dist[u] + 1;

if(v == t) return true;

que[tail++] = v;

}

}

}

return false;

}

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;

for(i = top - 1; i >= 0; i--) {

tp = min(tp, e[sta[i]].c);

}

maxflow += tp;

for(i = top - 1; i >= 0; i--) {

e[sta[i]].c -= tp;

e[sta[i] ^ 1].c += tp;

if(e[sta[i]].c == 0) top = i;

}

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;

}

}

}

return maxflow;

}

\*\*“有提交就会有奇迹”的dinic\*\*

dinic是有技巧的，以下是我喜欢的19行版dinic (by 雁过留声)

//这是数组以及结构定义，不计算行数

int level[NMax+2],queue[NMax+2];

long long Min(long long a,long long b){

return a>b?b:a;

}

struct edge{

long long e,f;

edge \*next,\*opt;

}\*E[NMax];

//正文

int mkLevel(){

for (int i=0;i<N;i++)level[i]=-1;

int bot,x;

level[queue[0]=0]=0;bot=1;

for (int top=0;top<bot;top++){x=queue[top];

for (edge \*p=E[x];p;p=p->next)if (p->f && level[p->e]==-1)

level[queue[bot++]=p->e]=level[x]+1;

}

return level[N-1]!=-1;

}

long long extend(int x,long long alpha){

if (x==N-1)return alpha;

long long r,t;r=0;

for (edge \*p=E[x];p;p=p->next)if (p->f && level[p->e]==level[x]+1)

t=extend(p->e,Min(p->f,alpha-r)),p->f-=t,p->opt->f+=t,r+=t;

if (!r)level[x]=-1;

return r;

}

void Dinic(){while (mkLevel())extend(0,(long long)1<<60);}

### 平面图最小割转最短路

## 费用流

### spfa

验题: 3680

const LL MOD = 0x3f3f3f3fLL;

const LL INF = (1LL<<55);

struct node {

int u, v, next;

LL c, w;

}e[Maxm];

int tot, last[Maxn];

LL dist[Maxn];

LL cost, flow;

int pre[Maxn];

bool visit[Maxn];

queue<int> Q;

void adde(int u, int v, LL c, LL w){

e[tot].u = u; e[tot].v = v; e[tot].c = c; e[tot].w = w; e[tot].next = last[u]; last[u] = tot++;

e[tot].u = v; e[tot].v = u; e[tot].c = 0; e[tot].w = -w; e[tot].next = last[v]; last[v] = tot++;

}

bool spfa(int s, int t, int n) {

int i;

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

dist[i] = INF;

visit[i] = 0;

pre[i] = -1;

}

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

Q.push(s);

visit[s] = true;

dist[s] = 0;

pre[s] = -1;

while(!Q.empty()) {

int u = Q.front();

visit[u] = false;

Q.pop();

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

if(e[j].c > 0 && dist[u] + e[j].w < dist[e[j].v]) {

dist[e[j].v] = dist[u] + e[j].w;

pre[e[j].v] = j;

if(!visit[e[j].v]) {

Q.push(e[j].v);

visit[e[j].v] = true;

}

}

}

}

if(dist[t] == INF) return false;

else return true;

}

LL ChangeFlow(int t) {

LL det = MOD;

int u = t;

while(~pre[u]) {

u = pre[u];

det = min(det, e[u].c);

u = e[u].u;

}

u = t;

while(~pre[u]) {

u = pre[u];

e[u].c -= det;

e[u ^ 1].c += det;

u = e[u].u;

}

return det;

}

LL MinCostFlow(int s, int t, int n) {

LL mincost, maxflow;

mincost = maxflow = 0;

while(spfa(s, t, n)) {

LL det = ChangeFlow(t);

mincost += det \* dist[t];

maxflow += det;

}

cost = mincost;

flow = maxflow;

return mincost;

}

### zkw

验题: 3680

zkw+deque

#define MOD 0x3f3f3f3f

int flow, cost, value;

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

deque<int> Q;

void adde(int u, int v, int c, int w) {

e[tot].u = u; e[tot].v = v; e[tot].c = c; e[tot].w = w; e[tot].next = last[u]; last[u] = tot++;

e[tot].u = v; e[tot].v = u; e[tot].c = 0; e[tot].w = -w; e[tot].next = last[v]; last[v] = tot++;

}

int Aug(int u, int m) {

if(u == des) {

cost += value \* m;

flow += m;

return m;

}

visit[u] = true;

int l = m;

int j, v, c, w;

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]) {

int 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, c, w, del;

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

dist[src] = 0;

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

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));

}

}

zkw+priority\_queue

#define MOD 0x3f3f3f3f

int flow, cost, value;

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

priority\_queue<pair<int, int> > Q;

void adde(int u, int v, int c, int w) {

e[tot].u = u; e[tot].v = v; e[tot].c = c; e[tot].w = w; e[tot].next = last[u]; last[u] = tot++;

e[tot].u = v; e[tot].v = u; e[tot].c = 0; e[tot].w = -w; e[tot].next = last[v]; last[v] = tot++;

}

int Aug(int u, int m) {

if(u == des) {

cost += value \* m;

flow += m;

return m;

}

visit[u] = true;

int l = m;

int j, v, c, w;

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]) {

int 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, c, w, del, d;

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

dist[src] = 0;

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

Q.push(MP(0, src));

while(!Q.empty()) {

u = Q.top().BB;

d = -Q.top().AA;

Q.pop();

if(d != dist[u]) continue;

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 = d + w) < dist[v]) {

dist[v] = del;

Q.push(MP(-del, 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));

}

}

## 全局最小割

### StoerWagner(求全局最小割)

验题:POJ2914

//朴素最小割

//注意节点下标0~n-1

int comb[Maxn];

int edge[Maxn][Maxn], g[Maxn][Maxn], node[Maxn];

int S, T, minCut;

int top, sta[Maxn];

int maxi;

vector<int> parta, partb;

vector<int> belong[Maxn];

int Search (int n) {

int i, j, u;

int vis[Maxn], wet[Maxn];

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

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

int minCut = 0, temp = -1, top = 0;

int maxi;

S = -1, T = -1;

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;

}

int StoerWagner (int n) {

int ans = MOD, i, j, cur;

memset(comb, 0, sizeof(comb));

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

node[i] = i;

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

k = n - i + 1;

cur = Search (k);

if (cur < ans) {

ans = cur;

}

if (ans == 0) return ans;

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];

}

}

}

return ans;

}

int n, m;

int main() {

int i, j, u, v, w;

while (scanf ("%d%d", &n, &m) != EOF) {

memset(edge, 0, sizeof(edge));

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

int u, v, z;

scanf ("%d%d%d", &u, &v, &z);

edge[u][v] += z;

edge[v][u] += z;

}

printf ("%d\n", StoerWagner (n) );

}

return 0;

}

### K连通块计数

HDU 4654

int n, m, k;

//K连通块计数

//注意节点下标0~n-1

int comb[Maxn];

int edge[Maxn][Maxn], g[Maxn][Maxn], node[Maxn];

int S, T, minCut;

int top, sta[Maxn];

int maxi;

vector<int> parta, partb;

vector<int> belong[Maxn];

int Search (int n) {

int i, j, u;

int vis[Maxn], wet[Maxn];

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

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

int minCut = 0, temp = -1, top = 0;

int maxi;

S = -1, T = -1;

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;

}

int StoerWagner (vector<int> & li) {

int ans = MOD, i, j, k,cur, n = li.SZ, u, v;

int used[Maxn];

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]];

}

}

int cur = StoerWagner(li);

if(cur >= k) return 1;

vector<int> a(parta), b(partb);

return dfs(a) + dfs(b);

}

int main() {

int i, j, u, v, w;

while (scanf ("%d%d%d", &n, &m, &k) != EOF) {

memset(g, 0, sizeof(g));

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

scanf ("%d%d", &u, &v);

u--; v--;

g[u][v] += 1;

g[v][u] += 1;

}

vector<int> li;

for(i = 0; i < n; i++) li.PB(i);

printf ("%d\n", dfs(li));

}

return 0;

}

## 网络流拓展

### 常见建图方式

B[u,v]表示(u,v)流量的下限，C[u,v]表示(u,v)流量的上限, F[u,v]表示(u,v)的流量,

g[u,v]表示F[u,v]-B[u,v] 显然 0<=g[u,v]<=C[u,v]-B[u,v]

1.无源汇的可行流 :

我们要想办法转换为有源汇的最大流问题.

考虑流量都为g[,]且容量为C[,]-B[,]的网络，貌似有点接近最后的转换方式了，

为了不忽略B[,]这一条件，我们把g[,]最后强制加上B[,].

但会发现一个致命漏洞，加上后就未必满足流量平衡了！

对于这个有两种办法解决..

一种方法是 添加附加源汇S,T 对于某点 u, 设 M(u)=sigma(B[i,u])-sigma(B[u,j]) ，则根据流量平衡条件有 M(u)同时等于 sigma(g[u,j])-sigma(g[i,u]) 若M(u)<0，即sigma(g[u,j]) < sigma(g[i,u]) 进入u的流量比从u 出去的多， 所以 u -> T 连容量为 -(sigma(B[i,u])-sigma(B[u,j]) ) 的边同理. M(u)>0时，即 S->u 连容量为 sigma(B[i,u])-sigma(B[u,j]) 的边. 然后再 对于任意边(i,u)/(u,j) 连一条 C[u,v]-B[u,v]的边. 这样 只需对新的网络求一遍最大流即可. 若出附加源点的边都满流即是存在可行流，反之不然.满流的必要条件是显然的. 不满流不能保证加上B[,]后流量平衡. 前面都白费了.

另一种方法相对简单.其实类似，本质相同.仍添加附加源汇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] 可以这样理解，边S->v : 求的时候直接从S流过来的流量值B[u,v], 与最终解中边(u,v)强制加上的从 u流过来的流量B[u,v]，对v点的流量平衡条件的影响 实质等价. 边u->T同理. 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.

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]

为什么这种方法只适用于无源汇上下界可行流？

本质上是因为S,T并不满足流量平衡，而上述的方法都是考虑到每点的流量平衡而建的. 但有些时候貌似还是可以出正确解. 至于有没有什么解决方法，下次再想想吧~【标记下】

例题 ZOJ 2314 / SGU 194 Reactor Cooling http://acm.sgu.ru/problem.php?contest=0&problem=194

2.有源汇的上下界可行流

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. 这样做显然使S,T也变得流量平衡了.

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？）（这样一定满足流量平衡？）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，否则若不全满流即无解.

和求3.有源汇的上下界最大流过程相反，感性理解是:

首先明确，我们的方法是通过加边转化成对任一点都有流量平衡的无源汇的网络，进行求解.

即最终解只能是加上边后，求的无源汇可行流，即T->S这边上的流量. 不改成无源汇的直接求的解是未必正确的，在（1）中已经提到.

然后，因为第一遍做的时候并无这条边，所以S->T的流量在第一遍做的时候都已经尽力往其他边流了. 于是加上T->S这条边后，都是些剩余的流不到其他边的流量. 从而达到尽可能减少T->S这边上的流量的效果，即减小了最终答案.

感觉上第一遍做的既然是不改成无源汇直接求的，应该是错误的？

这里不是错误的. 首先我们的解都是按照第二遍所求的而定，其次这里这样做本质是延迟对T->S这条边的增流.

## 最小树形图(有向图的最小生成树)

### 基础知识

hdu4009 hdu2121 poj3164 UVa11865

本题为不是固定根的最小树形图，我们可以虚拟出一根来，然后在把这个根跟每个点相连，相连的点可以设为无穷大，或者设为所有边和大一点，比如为r，然后就可以利用最小树形图进行计算了，计算出的结果减去r,如果比r还大就可以认为通过这个虚拟节点我们连过原图中两个点，即原图是不连通的，我们就可以认为不存在最小树形图。关于输出最小根也挺简单，在找最小入弧时，如果这条弧的起点是虚拟根，那么这条弧的终点就是要求的根

### 朱刘算法

验题:POJ3164 HDU2121

/\*

\* 最小树形图（根固定） O(VE)

\* 有向图最小生成树

\* 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; 求根, 则求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, 因为要考虑重新标号建图的统一

\* mytype 根据实际情况确定

\*/

typedef int mytype;

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

mytype inv[Maxn];

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];

}

}

## 树的Hash与同构

### 树Hash判定树同构

int h[11000];

char str1[3100], str2[3100];

char \*p;

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) ) % 19001;

}

return (sum \* sum) % 19001;

}

inline void init() {

//为每一个树根节点给定一个随机权值

for (int i = 0; i < 10000; i++)

h[i] = (rand() %19901);

}

int main() {

int T;

scanf ("%d", &T);

init();

while (T--) {

scanf ("%s%s", str1, str2);

p = str1;

int a = Hash (1);

p = str2;

int b = Hash (1);

if (a == b) {

puts ("same");

} else {

puts ("different");

}

}

return 0;

}

## 最大团

### 最大团

bool am[100][100];

int ans;

int c[100];

int U[100][100];

int n;

bool dfs(int rest,int num) {

if (!rest) {

if (num>=ans)

return 1;

else

return 0;

}

int pre=-1;

for (int i=0; i<rest && rest-i+num>=ans; i++) {

int idx=U[num][i];

if (num+c[idx]<ans)

return 0;

int nrest=0;

for (int j=i+1; j<rest; j++)

if (am[idx][U[num][j]])

U[num+1][nrest++]=U[num][j];

if (dfs(nrest,num+1))

return 1;

}

return 0;

}

int main() {

while (scanf("%d",&n),n) {

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

for (int j=0; j<n; j++)

scanf("%d",&am[i][j]);

ans=0;

for (int i=n-1; i>=0; i--) {

int rest=0;

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

if (am[i][j])

U[0][rest++]=j;

ans+=dfs(rest,0);

c[i]=ans;

}

printf("%d\n",ans);

}

return 0;

}

## 弦图相关

### 弦图判定

//ZOJ 1015 fishing net

//简单的弦图判定

int adj[Maxn][Maxn];

int n, m;

int 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

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;

}

int main() {

int i, j, u, v, w;

while(scanf("%d%d", &n, &m) != EOF) {

if(!n && !m) break;

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

for(j = 1; j <= n; j++) {

adj[i][j] = 0;

}

}

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

scanf("%d%d", &u, &v);

adj[u][v] = adj[v][u] = 1;

}

if(getList() && check()) {

printf("Perfect\n");

}

else {

printf("Imperfect\n");

}

printf("\n");

}

return 0;

}

### 弦图与区间图 By 猛犸也钻地 @ 2012.09.13

/\* 相关定义 //

1. 子图：原图点集的子集+原图边集的子集

2. 诱导子图：原图点集的子集+这些点在原图中所连出的边集

3. 团：原图的一个子图，且是完全图

4. 极大团：此团不是其他团的子集

5. 最大团：点数最多的团 -> 团数

6. 最小染色：用最少的颜色给点染色使相邻点颜色不同 -> 色数

7. 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

8. 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

9. 弦：连接环中不相邻的两个点的边

10.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)

11.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

// 重点内容 //

12.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

13.弦图各类算法：

最小染色算法 -> 按照完美消除序列，从后向前，依次染上可以染的最小颜色

最大独立集算法 -> 按照完美消除序列，从前向后，能选则选

最小团覆盖算法 -> 求出最大独立集，记为{p1,p2,..,pk}

则{N(p1)+{p1},N(p2)+{p2},..,N(pk)+{pk}}即为解

16.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 \*/

# **0字符串**

## **Hash**

### 一般字符串Hash

LL strhash (char \*str) {

int i;

LL c, ret = 0;

for(i= 0, ret = 0; str[i]; i++) {

c = str[i] - 'a';

ret = ((ret << 5) + ret) ^ c;

}

return ret;

}

17,37, 79, 163, 331, 673,1361,2729,5471,10949,

21911,43853, 87719, 175447, 350899,701819,1403641,2807303,5614657, 11229331,

22458671, 44917381, 89834777, 179669557, 359339171,

718678369, 1437356741, 2147483647

### 字符串区间Hash

//prePow预处理MOD^x的值, 可以在程序最开始做!

prePow[0] = 1;

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

prePow[i] = prePow[i - 1] \* MOD;

}

//存储在H数组中, 利用溢出进行Hash

void preHash(int len, char str[], LL H[]) {

int i;

H[0] = str[0];

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

H[i] = (H[i - 1] \* MOD + str[i]);

}

}

//求区间[l,r]的Hash值

LL HASH(int l, int r, LL H[], LL prePow[]) {[l,r], base 0

LL ret = 0;

if(l) ret = H[l - 1] \* prePow[r - l + 1];

return (H[r] - ret);

}

## KMP

### 函数版

char S[Maxn], T[Maxn];

int next[Maxn], is[Maxn];

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

int i, j;

next[0]=-1; next[1] = 0;

for (i = 1, j = 0; i < LT; ) {

while (j != -1 && T[i] != T[j]) j = next[j];

i++; j++;

next[i] = j;

}

}

void KMP (char S[], int LS, char T[], int LT, int next[]) {

int i, j;

for(i = 0; i < LS; i++) is[i] = 0;

for (i = 0, j = 0; i < LS; i++) {

while (j != -1 && S[i] != T[j]) j = next[j];

j++;

if (j == LT) {

is[i - LT + 1] = 1;

j = next[j];

}

}

}

### KMP类

struct KMP {//调用init(s(待匹配串),t(模式串));适用多次KMP

char S[Maxn], T[Maxn];

int next[Maxn], is[Maxn];

int LT, LS;

void init(char s[], char t[]) {

int i;

LT = strlen(t);

LS = strlen(s);

for(i = 0; i <= LT; i++) T[i] = t[i];

for(i = 0; i <= LS; i++) S[i] = s[i];

}

void getnext() {

int i, j;

next[0]=-1; next[1] = 0;

for (i = 1, j = 0; i < LT; ) {

while (j != -1 && T[i] != T[j]) j = next[j];

i++; j++;

next[i] = j;

}

}

void kmp() {

int i, j;

for(i = 0; i < LS; i++) is[i] = 0;

for (i = 0, j = 0; i < LS; i++) {

while (j != -1 && S[i] != T[j]) j = next[j];

j++;

if (j == LT) {

is[i - LT + 1] = 1;

j = next[j];

}

}

}

}kmp;

## 拓展KMP

### 函数版

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;

}

}

}

## Manacher

### Manacher模板

//HDU\_3068

//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;

int mx = 0;

int id;

for (i = len; i < Maxn; i++)

str[i] = 0; //没有这一句有问题。。就过不了ural1297，比如数据：ababa aba

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;

}

}

}

int main() {

int i, ans;

while (scanf ("%s", s) != EOF) {

len = strlen (s);

init(s, str, len);

Manacher(str, p, len);

ans = 0;

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

if (p[i] > ans)

ans = p[i];

printf ("%d\n", ans - 1);

}

return 0;

}

## AC自动机

### 普通匹配

struct node {

struct node \* fail;

struct node \* next[10];

bool is;

int lab;

}Tree[Maxn], \* root, \* que[Maxn];

int tot, head, tail;

node \* newNode() {

node \* p = &Tree[tot++];

p->is = false;

p->lab = 0;

p->fail = NULL;

for(int i = 0; i < 10; i++) p->next[i] = NULL;

return p;

}

void Insert(char str[], int n) {

node \* p = root;

for(int i = 0, k; str[i]; i++) {

k = str[i] - '0';

if(p->next[k] == NULL) {

p->next[k] = newNode();

}

p = p->next[k];

}

p->is = true;

p->lab = n;

}

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];

}

}

}

}

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;

}

}

}

}

### AC自动机+DP

struct node {

struct node \* next[26];

struct node \* fail;

int is;

int v;

}A[Maxn], \*root, \*que[Maxn];

int head, tail;

char str[Maxn];

int tot, ans, n;

node \* newNode() {

node \* p = &A[tot++];

p->fail = NULL;

p->is = false;

p->v = -MOD;

for(int i = 0; i <26; i++) p->next[i] = NULL;

return p;

}

void Insert(node \* root, char str[], int l, int r) {

int i, k;

for(i = l; i < r; i++) {

k = str[i] - 'a';

if(root->next[k] == NULL) root->next[k] = newNode();

root = root->next[k];

}

root->is = 1;

}

void buildAC() {

int k;

head = tail = 0;

que[tail++] = root;

root->fail = root;

node \* p, \* q;

while(head < tail) {

p = que[head++]; q = p->fail;

for(k = 0; k < 26; k++) {

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

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

else {

p->next[k]->fail = q->next[k];

p->next[k]->is |= q->next[k]->is;

}

que[tail++] = p->next[k];

}

else {

if(p == root) p->next[k] = root;

else p->next[k] = q->next[k];

}

}

}

}

void query(int n, int l, int r, int v0) {

int i, k, v;

node \* p, \* q;

p = root;

v = v0;

for(i = l; i < r; i++) {

k = str[i] - 'a';

p = p->next[k];

if(p->is) {

q = p;

while(q->is) {

cmax(v, v0 + q->v);

q = q->fail;

}

}

}

cmax(p->v, v);

cmax(ans, v);

}

## 后缀数组

### 论文模板(O(nlogn))

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!

//O(nlogn)

//调用da(num, len+1, m);//m为字符个数略大

int len;

int num[Maxn]; //待处理的串

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

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

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) { // 倍增算法 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++);

}

}

## 字符串最小表示法

### 朴素最小表示法

ZOJ2006 ZOJ1729 HDU 3374

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);

}

### 判定两串是否循环同构

# 搜索

## DLX

### 精确覆盖

验题: POJ3074

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];

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

int ret = 0;

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;

memset(colsum, 0, sizeof (colsum) );

head = &node[cnt ++ ];

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] ++ ;

}

}

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;

}

} dlx;

int mat[Maxn][Maxm];

int mem[Maxn][3]; //记录每行代表哪一格填几

//数独填充(x,y)=v

void addline(int x, int y, int v) {

int i, j;

n++;

mem[n][0] = x;

mem[n][1] = y;

mem[n][2] = v;

for(i = 0; i < Maxm; i++) mat[n][i] = 0;

mat[n][x \* 9 + y + 1] = 1;

mat[n][81 + x \* 9 + v] = 1;

mat[n][162 + y \* 9 + v] = 1;

mat[n][243 + (3 \* (x / 3) + y / 3) \* 9 + v] = 1;

}

### 

### 多重覆盖

验题 HDU3498, HDU5046

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];

Node \* p, \*q, \*r;

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

int ret = 0;

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

if(!vis[p->col]) {

ret++;

vis[p->col]++;

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

for(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;

int i, j;

Node \* p;

memset(colsum, 0, sizeof(colsum));

head = &node[cnt++];

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

for(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(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(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(i = n; i > 0; i--) {

for(j = m; j > 0; j--) {

if(mat[i][j]) {

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]++;

}

}

}

}

void remove(Node \*c) {

Node \* p;

for(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) {

Node \* p;

for(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;

}

} dlx;

# 其他

## 头文件

### head.cpp

//#pragma comment(linker,"/STACK:102400000,102400000")

#include <bits/stdc++.h>

#include <cstdio>

#include <cstring>

#include <cstdlib>

#include <cmath>

#include <cassert>

#include <iostream>

#include <sstream>

#include <algorithm>

#include <string>

#include <vector>

#include <set>

#include <map>

#include <queue>

#include <stack>

#include <bitset>

using namespace std;

typedef long long LL;

typedef pair<int, int> PII;

#define PB push\_back

#define MP make\_pair

#define AA first

#define BB second

#define SZ size()

#define BG begin()

#define OP begin()

#define ED end()

#define SQ(x) ((x)\*(x))

#define cmax(x, y) x = max(x, y)

#define cmin(x, y) x = min(x, y)

int main() {

int i, j, u, v, w;

//freopen("", "r", stdin);

//freopen("", "w", stdout);

return 0;

}

## 配置信息

### eclipse

Eclipse中默认是输入"."后出现自动提示，用于类成员的自动提示，可是有时候我们希望它能在我们输入类的首字母后就出现自动提示，可以节省大量的输入时间（虽然按alt + /会出现提示，但还是要多按一次按键，太麻烦了）。 从Window -> preferences -> Java -> Editor -> Content assist -> Auto-Activation下，我们可以在"."号后面加入我们需要自动提示的首字幕，比如"ahiz"。 然后我们回到Eclipse的开发环境，输入"a"，提示就出现了。但是我们可以发现，这个Auto-Activation下的输入框里最多只能输入5个字母，也许是Eclipse的开发人员担心我们输入的太多会影响性能，但计算机的性能不用白不用，所以我们要打破这个限制。其实上面都是铺垫，制造一下气氛，以显得我们下面要做的事情很牛似的，其实不然，一切都很简单。嘿嘿 :)

在"."后面随便输入几个字符，比如"abij"，然后回到开发环境，File -> export -> general -> preferences -> 选一个地方保存你的首选项，比如C:"a.epf用任何文本编辑器打开a.epf，查找字符串“abij”，找到以后，替换成“abcdefghijklmnopqrstuvwxyz”，总之就是你想怎样就怎样！！然后回到Eclipse，File -> import -> general -> preferences -> 导入刚才的a.epf文件。此时你会发现输入任何字幕都可以得到自动提示了。爽！！！最后：自动提示弹出的时间最好改成100毫秒以下，这样会比较爽一点，不然你都完事了，自动提示才弹出来:)，不过也要看机器性能。

FileWriter fileWriter=new FileWriter("c:\\Result.txt");

int [] a=new int[]{11112,222,333,444,555,666};

for (int i = 0; i < a.length; i++) {

fileWriter.write(String.valueOf(a[i])+" ");

}

### vimrc

## 调试注意事项

### 内存大小

全局数组大小

函数内数组大小

编译时间是否算入总时间

### 堆栈

堆栈大小

无参递归层次

能否开栈

### 编译器选项

-O1 ?

-O2 ?

I64d OR lld

### Java类名要求

Main ?

### 返回值

CE

AC

WA

RE

PE

TLE

MLE

SF

### ext包

ext/pb\_ds/assoc\_container.hpp

ext/pb\_ds/priority\_queue.hpp

ext/rope

ext/hash\_map

ext/hash\_set