2-sat 1

判断奇环 7

堆优化的SPFA 8

& 2-sat

（1)2-SAT模型：存在n组元素，每组两个元素，且这两个元素互斥。

某些组存在一些“关系”，可以证明，这些关系是对称的，且对称是可传递的。

要求在满足给定关系的情况下在每组元素中选出一个元素的问题称为2-SAT问题。问是否存在即2-SAT判定问题，当然也可以求出一组可行解。

一般有3种题目：判定是否有可行解；最优解（二分）；输出一组可行解

（2) 构图：构图的关键是找“冲突关系”，若a与b冲突，则可以连边a->b',b->a'.

每组2个元素，所以要把多于2个元素的情况压缩，如hdu 1824

（3)是否有可行解判定很简单，缩点后看xi与xi’有没有在同一个强连通子图里就可以了，有则return false。

（4)hdu 1814输出最小字典序

关于输出解，有两种方法，一种是逆向topsort输出任意解（hdu\_1814\_3.cpp)，另一种是暴力dfs，可以输出最小字典序解（hdu\_1814.cpp）。

逆向topsort法：

(0)建立图G的补图G'，下面的讨论都是针对G'的。

(1)开队列Q1，Q2，遇到未染色点，加到Q1里，来一论topsort染色。

(2)topsort-enter:

从Q1里取出元素i，若它此时还是未染色，则染red，把~i加到Q2里

(3)不断地以Q2种的元素~i扩展开来，把~i的所有后继节点都进Q2，并染black

(4)对于之前加到Q1的i点的子节点都入度减1，若入度为0则加到Q1

bool dfs(int u)

{

if(color[hash(u)]) return false;

if(color[u]) return true;

color[u] = 1;

S[top++] = u;

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

int v = edge[k].v;

if(!dfs(v)) return false;

}

return true;

}

bool two-sat()

{

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

if(!color[i] && !color[hash(i)]) {

top = 0;

if(!dfs(i)) {

while(top) color[ S[--top] ] = 0;

if(!dfs(hash(i))) return false;

}

}

return true;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*TARJAN 模板\*\*\*\*\*\*\*\*\*\*\*\*\*

void Tarjan(int now)

{

in\_stack[now] = true;

low[now] = dfn[now] = order++;

S.push(now);

for(int Size = v[now].size(), i = 0; i < Size; i++){

int tmp = v[now][i];

if(in\_stack[tmp]) low[now] = min(low[now], low[tmp]);

else if(dfn[tmp] == -1) {

Tarjan(tmp);

low[now] = min(low[now], low[tmp]);

}

}

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

int tmp;

do {

tmp = S.top();

S.pop();

in\_stack[tmp] = false;

low[tmp] = low[now];

} while(tmp != now);

}

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

(1) 判断是否有解 (AND,OR,XOR) hdu\_4421

/\* 题意：该题给出操作(AND,OR,XOR)和结果(b[i][j]), 判断是否有解

2012长春现场赛的题。

由于不细心，WA了几次。。多是建边的时候粗心

思路：该题给出操作(AND,OR,XOR)和结果(b[i][j])，只要把元素拆成bit形式来做2-SAT就行了。因为\*\*有解就必须每个位bit都有解\*\*，`且每一位bit不互相影响`。如果一下把31位都给拆来建图的话会ME，所以得一位一位地

来，之前我没意识到这一点，原因在没运用到`每一位不互相影响`。

建图时AND->1, 和or->0是重点。问“同组元素连边”有什么意义？达神跟

我解释：如果只是一组元素连边的话没意义，但是有多组元素连边时就有意>义了。举例，A AND B == 1, 连a->a', b->b'. 直观理解为这两组元素一定>得选a'和b'。如果有解取了a和b，则建连通图的时候会把a也连上，同理有b/b'，则这个子连通图会包括a和a'，矛盾。（因为由于对称性，a组和b组肯定

会有至少两条边相连，所以a/a'/b/b'会在同一环里）

什么时候同组元素连边？--还没有比较好的理解...

不知道pre-check有没有必要...

bool build()

{

init();

FOR(row, 0, n) FOR(col, 0, row) {

int bit = (b[row][col] & 1);

int op;

if((row & 1) && (col & 1)) op = OR;

else if(!(row & 1) && !(col & 1)) op = AND;

else op = XOR;

int x = row, y = col, x1 = hash(row), y1 = hash(col);

if(op == OR) {

if(bit == 0) {

v[x1].push\_back(x); v[y1].push\_back(y);

} else {

v[x].push\_back(y1); v[y].push\_back(x1);

}

} else if(op == AND) {

if(bit == 0) {

v[x1].push\_back(y); v[y1].push\_back(x);

} else {

v[x].push\_back(x1); v[y].push\_back(y1);

}

} else if(op == XOR) {

if(bit == 0) {

v[x].push\_back(y); v[y].push\_back(x);

v[x1].push\_back(y1); v[y1].push\_back(x1);

} else {

v[x].push\_back(y1); v[y].push\_back(x1);

v[x1].push\_back(y); v[y1].push\_back(x);

}

}

}

FOR(row, 0, n) FOR(col, 0, row) b[row][col] >>= 1;

\*/

(2) 2-SAT 二分法求最佳 hdu\_3715

/\*

给出一个递归表达式， 问最多能递归到第几层？

思路： 构图：x表0, x'表1

c = 0, 则矛盾式子为a=0,b=0，所以可以连边a->b',b->a'

c = 1, 矛盾式为a=1,b=0或a=0,b=1,可以连边a'->b',b->a,a->b,b'->a

c = 2, 矛盾式为a=1,b=1,可以连a'->b,b'->a

\*/

(3) 2-SAT输出解 输出任意解

/\*

int main()

{

int m, x, y;

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

init();

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

scanf("%d%d", &x, &y);

v[x].push\_back(hash(y)); v[y].push\_back(hash(x));

}

N = 2 \* n;

for(int i = 1; i <= N; i++) if(dfn[i] == -1) Tarjan(i);

bool flag = true;

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

if(belong[i\*2-1] == belong[i\*2]) {

flag = false;

break;

}

if(flag == false) {

printf("NIE\n");

continue;

}

for(int x = 1; x <= N; x++)

for(int Size = v[x].size(), j = 0; j < Size; j++) {

int y = v[x][j];

if(belong[x] == belong[y]) continue;

rv[belong[y]].push\_back(belong[x]);

indegree[belong[y]]++;

}

topsort();

for(int i = 1; i <= N; i++) if(color[belong[i]] == 'R') printf("%d\n", i);

}

return 0;

}

void del(int x)

{

color[x] = 'B';

for(int Size = rv[x].size(), i = 0; i < Size; i++) if(color[rv[x][i]] == 0) del(rv[x][i]);

}

void topsort()

{

for(int i = 1; i <= N; i++) if(indegree[i] == 0) Q.push\_back(i);

while(!Q.empty()) {

int x = Q.front();

Q.pop\_front();

ansv.push\_back(x);

for(int Size = rv[x].size(), i = 0; i < Size; i++) {

int y = rv[x][i];

if(color[y] != 0) continue;

if(--indegree[y] == 0) Q.push\_back(y);

}

}

for(int Size = ansv.size(), i = 0; i < Size; i++) {

int x = ansv[i];

if(color[x] == 0) {

color[x] = 'R';

for(int j = 1; j <= N; j++) if(belong[j] == x && color[belong[hash(j)]] == 0) {

del(belong[hash(j)]);

}

}

}

}

\*/

(5) 2-SAT输出最小字典序解 hdu\_1814

/\* 这里用的是暴力dfs输出解

void solve()

{

for(int i = 1; i <= n; i++) if(color[2\*i-1] == 0) {

ans\_top = 0;

if(!dfs(2\*i-1)) {

for(int j = 0; j < ans\_top; j++) color[ans[j]] = color[hash(ans[j])]= 0;

ans\_top = 0;

dfs(2\*i);

}

}

}

bool dfs(int u)

{

if(color[u] != 0) return color[u] == 1;

color[u] = 1;

color[hash(u)] = -1;

ans[ans\_top++] = u;

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

int v = edge[k].v;

if(!dfs(v)) return false;

}

return true;

}

\*/

/\* 判断奇环 ===================================================\*/

void Find(int u, int c)

{

if(color[u] != -1) return ;

color[u] = c;

for(int k = head[u]; k != -1; k = edge[k].next) Find(edge[k].v, !c);

}

bool dye()

{

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

for(int i = 1; i <= n; i++) if(color[i] == -1) Find(i, 0);

for(int i = 0; i < edge\_num; i++) if(color[edge[i].u] == color[edge[i].v]) return false;

return true;

}

/\* ===========================================================\*/

/\* 堆优化的SPFA ==================================================\*/

typedef pair<int, int> PII;

bool SPFA(int s)

{

memset(times, 0, sizeof(times));

memset(dis, 127, sizeof(dis));

memset(inque, false, sizeof(inque));

dis[s] = 0;

priority\_queue< PII > Q; //默认pair按first增序

Q.push(PII(0, s)); //first表示距离，second才表示标号

while(!Q.empty()) {

int u = Q.top().second;

Q.pop();

inque[u] = false;

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

int v = edge[k].v;

if(dis[v] > dis[u] + edge[k].w) {

dis[v] = dis[u] + edge[k].w;

if(!inque[v]) {

inque[v] = true;

Q.push(PII(dis[v], v));

if(++times[v] > n) return false;

}

}

}

}

return true;

}