2-sat

、、、、//具体的建图规则为：

//若 a and b ==1 , !a->a , !b -> b

// a and b ==0 , a->!b , b->!a

// a or b ==1 , !a->b , !b->a

// a or b ==0 , a->!a , b->!b

// a xor b ==1 , a->!b,!b->a,!a->b,b->!a

// a xor b ==0 , a->b,b->a,!a->!b,!b->!a

const int MAX = 1010;

vector <int> G[MAX], rG[MAX], vs;

bool used[MAX];

int cmp[MAX];

char cmd[5];

int V;

void add\_edge(int from, int to) {

G[from].push\_back(to);

rG[to].push\_back(from);

}

void dfs(int v) {

used[v] = true;

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

if (!used[G[v][i]]) dfs(G[v][i]);

}

vs.push\_back(v);

}

void rdfs(int v, int k) {

used[v] = true;

cmp[v] = k;

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

if (!used[rG[v][i]]) {

rdfs(rG[v][i], k);

}

}

}

int scc() {

memset(used, 0, sizeof(used));

vs.clear();

for (int v = 0; v < V; v++) {

if(!used[v]) dfs(v);

}

memset(used, 0, sizeof(used));

int k = 0;

for (int i = vs.size()-1; i >= 0; i--) {

if (!used[vs[i]]) rdfs(vs[i], k++);

}

return k;

}

int main()

{

int is\_false, u, v, c, n, M;

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

V = 2 \* n;

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

G[i].clear();

rG[i].clear();

}

is\_false = 0;

memset(cmp, 0, sizeof(cmp));

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

memset(cmd, 0, sizeof(cmd));

scanf("%d%d%d%s", &u, &v, &c, cmd);

if ((cmd[0] == 'A') && (c == 1)) {

add\_edge(u+n, u);

add\_edge(v+n, v);

}

else if ((cmd[0] == 'A') && (c == 0)) {

add\_edge(u, v+n);

add\_edge(v, u+n);

}

else if ((cmd[0] == 'O') && (c == 1)) {

add\_edge(u+n, v);

add\_edge(v+n, u);

}

else if ((cmd[0] == 'O') && (c == 0)) {

add\_edge(u, u+n);

add\_edge(v, v+n);

}

else if ((cmd[0] == 'X') && (c == 0)) {

add\_edge(u, v);

add\_edge(v, u);

add\_edge(u+n, v+n);

add\_edge(v+n, u+n);

}

else if ((cmd[0] == 'X') && (c == 1)) {

add\_edge(u, v+n);

add\_edge(v, u+n);

add\_edge(v+n, u);

add\_edge(u+n, v);

}

}

scc();

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

if (cmp[i] == cmp[i+n]) {

printf("NO\n");

is\_false = 1;

break;

}

}

if (!is\_false) printf("YES\n");

return 0;

}

Dicnic递归版

const int MAX\_V = 20010;

const int INF = 0x7fffffff;

struct edge {int to, cap, rev;};

vector <edge> G[MAX\_V];

int level[MAX\_V];//距原点的距离

int iter[MAX\_V];//当前弧

void add\_edge(int from, int to, int cap) {

G[from].push\_back((edge){to, cap, G[to].size()});

G[to].push\_back((edge){from, 0, G[from].size()-1});

}

void bfs(int s) {//建立当前残余网络的层次图

memset(level, -1, sizeof(level));//注意是-1

queue<int> q;

level[s] = 0;

q.push(s);

while (!q.empty()) {

int v = q.front();

q.pop();

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

edge &e = G[v][i];

if (e.cap > 0 && level[e.to] < 0) {

level[e.to] = level[v] +1;

q.push(e.to);

}

}

}

}

int dfs(int v, int t, int f) {

if (v == t) return f;//f为当前能传输的最大值

for (int &i = iter[v]; i < G[v].size(); i++) {

edge &e = G[v][i];

if (e.cap > 0 && level[v] < level[e.to]) {//保证离汇点更近（not only+1）

int d = dfs(e.to, t, min(f, e.cap));//min求一个瓶颈

if (d > 0) {

e.cap -= d;

G[e.to][e.rev].cap += d;

return d;

}

}

}

return 0;

}

int max\_flow(int s, int t) {

int flow = 0;

for (;;) {

bfs(s);

if (level[t] < 0) return flow;

memset(iter, 0, sizeof(iter));

int f = dfs(s, t, INF);

while (f > 0) {

flow += f;

f = dfs(s, t, INF);

}

}

}

int main()

{

int N, M, A, B, a, b, c;

scanf("%d%d", &N, &M);

int s = N, t = s + 1;

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

scanf("%d%d", &A, &B);

add\_edge(s, i, A);

add\_edge(i, t, B);

}

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

scanf("%d%d%d", &a, &b, &c);

add\_edge(a-1, b-1, c);

add\_edge(b-1, a-1, c);

}

printf("%d\n", max\_flow(s, t));

return 0;

}

并查集、瓶颈路/树

const int MAX = 200010;

long long int sum[MAX];//可以同时记录很多个值，不仅仅是sum

int p[MAX];

struct edge{int from, to, cost;} E[MAX];

int cmp(const void \*a, const void \*b) {

struct edge \*c = (struct edge \*)a;

struct edge \*d = (struct edge \*)b;

return d->cost - c->cost;

}

int find(int x) {

if (p[x]<0) return x;

return p[x] = find(p[x]);

}

void unite(int x, int y) {//x y不可换

int px = find(x);

int py = find(y);

if (py == px) return;

p[px] += p[py];

p[py] = px;

}

void init(int n) {

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

p[i] = -1;

sum[i] = 0;

}

}

int main()

{

int N;

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

init(N+3);//避免数组爆炸

for (int i = 0; i < N-1; i++) {

scanf("%d%d%d", &E[i].from, &E[i].to, &E[i].cost);

}

qsort(E, N-1, sizeof(E[0]), cmp);//从小到大是a-b

for (int i = 0; i < N-1; i++) {

int pa = find(E[i].from);

int pb = find(E[i].to);

if (pa == pb) continue;

long long int suma = (-1 \* E[i].cost \* p[pb]) + sum[pa];

long long int sumb = (-1 \* E[i].cost \* p[pa]) + sum[pb];

if (suma > sumb) {

unite(pa, pb);

sum[pa] = suma;

}

else {

unite(pb, pa);

sum[pb] = sumb;

}

}

printf("%lld\n", sum[find(1)]);//得到了一棵生成树

}

return 0;

}

差分约束系统

struct Edge{

int from, to, dist;

};

const int MAX\_V = 1010;

const int INF = 0x7fffffff;

int inq[MAX\_V];//是否在队列中

int d[MAX\_V];//s到各点的距离

//int p[MAX\_V];//最短路中的上一条弧 现在没用，后面Dinic有用

int cnt[MAX\_V];//进队次数

vector <int> G[MAX\_V];

vector <Edge> edge;

int n;

void add\_edge(int from, int to, int dist) {

edge.push\_back((Edge){from, to, dist});

int m = edge.size();

G[from].push\_back(m-1);

}

bool bellman\_ford(int s) {

queue <int> q;

memset(inq, 0, sizeof(inq));

memset(cnt, 0, sizeof(cnt));

for (int i = 0; i < n; i++) d[i] = INF;

d[s] = 0;

inq[s] = true;

q.push(s);

while (!q.empty()) {

int u = q.front();

q.pop();

inq[u] = false;

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

Edge &e = edge[G[u][i]];

if (d[u] < INF && d[e.to] > d[u] + e.dist) {

d[e.to] = d[u] + e.dist;

// p[e.to] = G[u][i];

if (!inq[e.to]) {

q.push(e.to);

inq[e.to] = true;

if (++cnt[e.to] > n) return false;

}

}

}

}

return true;

}

int main()

{

// edge.clear();

// for (int i = 0; i < N; i++) G[i].clear();

int L, D, a, b, c;

scanf("%d%d%d", &n, &L, &D);

for (int i = 0; i < L; i++) {//a+c >= b

scanf("%d%d%d", &a, &b, &c);

add\_edge(a-1, b-1, c);

}

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

scanf("%d%d%d", &a, &b, &c);

add\_edge(b-1, a-1, -c);

}

for (int i = 0; i < n-1; i++) {//满足d[i]>=d[i-1]

add\_edge(i+1, i, 0);

}

bool whether = bellman\_ford(0);

if (!whether) printf("-1\n");//有负环

else {

if (d[n-1] - d[0] >= INF) printf("-2\n");//无法到达（无约束）

else printf("%d\n", d[n-1]-d[0]);

}

}

最小树形图

const int INF = 0x7fffffff;

const int MAX\_V = 105;

const int MAX\_E = 1005;

const double eps = 1e-4;

struct node

{

int u, v;

int w;

}edge[MAX\_V \* MAX\_V];

int pre[MAX\_V], id[MAX\_V], vis[MAX\_V];

int in[MAX\_V];//入边的最小权值

int Directed\_MST(int root, int V, int E)

{

int cost = 0;//最小树形图总权值

while(true)

{

//1.找最小入边

for(int i = 0; i < V; i++) in[i] = INF;

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

int u = edge[i].u;

int v = edge[i].v;

if(edge[i].w < in[v] && u != v) {

pre[v] = u;

in[v] = edge[i].w;

if (u == root) son = i;//在连虚根时需要

}

}

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

if(i == root) continue;

if(in[i] == INF) return -1;//除了根以外有点没有入边,则根无法到达它

}

//2.找环

int cnt = 0;

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

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

in[root] = 0;

for(int i = 0; i < V; i++) {//标记每个环

cost += in[i];

int v = i;

while(vis[v] != i && id[v] == -1 && v != root) {

//每个点寻找其前序点，要么最终寻找至根部，要么找到一个环

vis[v] = i;

v = pre[v];

}

if(v != root && id[v] == -1) {//缩点

for(int u = pre[v]; u != v; u = pre[u])

id[u] = cnt;

id[v] = cnt++;

}

}

if(cnt == 0) break; //无环 则break

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

if(id[i] == -1) id[i] = cnt++;//没有被缩点

//3.建立新图

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

//遍历了所有的边,除了被缩点的边外都保留了下来，都是可能的解

int u = edge[i].u;

int v = edge[i].v;

edge[i].u = id[u];

edge[i].v = id[v];

if(edge[i].u != edge[i].v) edge[i].w -= in[v];

//不需要在这里判断自环,更新边权（重新连了一边，要拆掉一边,把之前花费减回来）

}

V = cnt;

root = id[root];

}

return cost;

}

int main()

{

int N, M, r;

while (scanf("%d%d", &N, &M) == 2) {

r = 0;

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

scanf("%d%d%d", &edge[i].u, &edge[i].v, &edge[i].w);

r += edge[i].w;//注意

if (edge[i].u == edge[i].v) edge[i].w = INF;//去除自环

}

r++;

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

edge[i].u = N;

edge[i].v = i-M;//注意

edge[i].w = r;

}

int ans = Directed\_MST(N, N+1, M+N);//注意

if (ans == -1 || ans >= 2\*r) printf("impossible\n");

else printf("%d %d\n", ans-r, son-M);//注意

}

return 0;

}

带花树

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <iostream>

using namespace std;

const int N=510;

int n, head, tail, Start, Finish;

link[N]; //表示哪个点匹配了哪个点

int Father[N]; //这个就是增广路的Father……但是用起来太精髓了

int Base[N]; //该点属于哪朵花

int Q[N];

bool mark[N];

bool map[N][N];

bool InBlossom[N];

bool in\_Queue[N];

int x[N], y[N], m;

void CreateGraph(){

int x,y;

scanf("%d", &n);

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

map[x+1][y+1] = map[y+1][x+1] = 1;

}

void BlossomContract(int x,int y){

fill(mark,mark+n+1,false);

fill(InBlossom,InBlossom+n+1,false);

#define pre Father[link[i]]

int lca,i;

for (i=x;i;i=pre) {i=Base[i]; mark[i]=true; }

for (i=y;i;i=pre) {i=Base[i]; if (mark[i]) {lca=i; break;} } //寻找lca之旅……一定要注意i=Base[i]

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

if (Base[pre]!=lca) Father[pre]=link[i]; //对于BFS树中的父边是匹配边的点，Father向后跳

InBlossom[Base[i]]=true;

InBlossom[Base[link[i]]]=true;

}

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

if (Base[pre]!=lca) Father[pre]=link[i]; //同理

InBlossom[Base[i]]=true;

InBlossom[Base[link[i]]]=true;

}

#undef pre

if (Base[x]!=lca) Father[x]=y; //注意不能从lca这个奇环的关键点跳回来

if (Base[y]!=lca) Father[y]=x;

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

if (InBlossom[Base[i]]){

Base[i]=lca;

if (!in\_Queue[i]){

Q[++tail]=i;

in\_Queue[i]=true; //要注意如果本来连向BFS树中父结点的边是非匹配边的点，可能是没有入队的

}

}

}

void Change(){

int x,y,z;

z=Finish;

while (z){

y=Father[z];

x=link[y];

link[y]=z;

link[z]=y;

z=x;

}

}

void FindAugmentPath(){

fill(Father,Father+n+1,0);

fill(in\_Queue,in\_Queue+n+1,false);

for (int i=1;i<=n;i++) Base[i]=i;

head=0; tail=1;

Q[1]=Start;

in\_Queue[Start]=1;

while (head!=tail){

int x=Q[++head];

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

if (map[x][y] && Base[x]!=Base[y] && link[x]!=y) //无意义的边

if ( Start==y || link[y] && Father[link[y]] ) //精髓地用Father表示该点是否

BlossomContract(x,y);

else if (!Father[y]){

Father[y]=x;

if (link[y]){

Q[++tail]=link[y];

in\_Queue[link[y]]=true;

}

else{

Finish=y;

Change();

return;

}

}

}

}

void Edmonds(){

memset(link,0,sizeof(link));

for (Start=1;Start<=n;Start++)

if (link[Start]==0)

FindAugmentPath();

}

void output(){

fill(mark,mark+n+1,false);

int cnt=0;

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

if (link[i]) cnt++;

printf("%d\n",cnt/2);//注意

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

if (!mark[i] && link[i]){

mark[i]=true;

mark[link[i]]=true;

// printf("%d %dn",i,link[i]);

}

}

int main(){

#ifdef LOCAL

freopen("in.txt","r",stdin);

#endif

CreateGraph();

Edmonds();

output();

return 0;

}