带权带花树模板

#include <iostream>

#include <cstdio>

#include <algorithm>

#include <vector>

using namespace std;

typedef long long s64;

const int INF = 2147483647;

const int MaxN = 400;

const int MaxM = 79800;

template <class T>

inline void tension(T &a, const T &b)

{

if (b < a)

a = b;

}

template <class T>

inline void relax(T &a, const T &b)

{

if (b > a)

a = b;

}

template <class T>

inline int size(const T &a)

{

return (int)a.size();

}

inline int getint()

{

char c;

while (c = getchar(), '0' > c || c > '9');

int res = c - '0';

while (c = getchar(), '0' <= c && c <= '9')

res = res \* 10 + c - '0';

return res;

}

const int MaxNX = MaxN + MaxN;

struct edge

{

int v, u, w;

edge(){}

edge(const int &\_v, const int &\_u, const int &\_w)

: v(\_v), u(\_u), w(\_w){}

};

int n, m;

edge mat[MaxNX + 1][MaxNX + 1];

int n\_matches;

s64 tot\_weight;

int mate[MaxNX + 1];

int lab[MaxNX + 1];

int q\_n, q[MaxN];

int fa[MaxNX + 1], col[MaxNX + 1];

int slackv[MaxNX + 1];

int n\_x;

int bel[MaxNX + 1], blofrom[MaxNX + 1][MaxN + 1];

vector<int> bloch[MaxNX + 1];

inline int e\_delta(const edge &e) // does not work inside blossoms

{

return lab[e.v] + lab[e.u] - mat[e.v][e.u].w \* 2;

}

inline void update\_slackv(int v, int x)

{

if (!slackv[x] || e\_delta(mat[v][x]) < e\_delta(mat[slackv[x]][x]))

slackv[x] = v;

}

inline void calc\_slackv(int x)

{

slackv[x] = 0;

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

if (mat[v][x].w > 0 && bel[v] != x && col[bel[v]] == 0)

update\_slackv(v, x);

}

inline void q\_push(int x)

{

if (x <= n)

q[q\_n++] = x;

else

{

for (int i = 0; i < size(bloch[x]); i++)

q\_push(bloch[x][i]);

}

}

inline void set\_mate(int xv, int xu)

{

mate[xv] = mat[xv][xu].u;

if (xv > n)

{

edge e = mat[xv][xu];

int xr = blofrom[xv][e.v];

int pr = find(bloch[xv].begin(), bloch[xv].end(), xr) - bloch[xv].begin();

if (pr % 2 == 1)

{

reverse(bloch[xv].begin() + 1, bloch[xv].end());

pr = size(bloch[xv]) - pr;

}

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

set\_mate(bloch[xv][i], bloch[xv][i ^ 1]);

set\_mate(xr, xu);

rotate(bloch[xv].begin(), bloch[xv].begin() + pr, bloch[xv].end());

}

}

inline void set\_bel(int x, int b)

{

bel[x] = b;

if (x > n)

{

for (int i = 0; i < size(bloch[x]); i++)

set\_bel(bloch[x][i], b);

}

}

inline void augment(int xv, int xu)

{

while (true)

{

int xnu = bel[mate[xv]];

set\_mate(xv, xu);

if (!xnu)

return;

set\_mate(xnu, bel[fa[xnu]]);

xv = bel[fa[xnu]], xu = xnu;

}

}

inline int get\_lca(int xv, int xu)

{

static bool book[MaxNX + 1];

for (int x = 1; x <= n\_x; x++)

book[x] = false;

while (xv || xu)

{

if (xv)

{

if (book[xv])

return xv;

book[xv] = true;

xv = bel[mate[xv]];

if (xv)

xv = bel[fa[xv]];

}

swap(xv, xu);

}

return 0;

}

inline void add\_blossom(int xv, int xa, int xu)

{

int b = n + 1;

while (b <= n\_x && bel[b])

b++;

if (b > n\_x)

n\_x++;

lab[b] = 0;

col[b] = 0;

mate[b] = mate[xa];

bloch[b].clear();

bloch[b].push\_back(xa);

for (int x = xv; x != xa; x = bel[fa[bel[mate[x]]]])

bloch[b].push\_back(x), bloch[b].push\_back(bel[mate[x]]), q\_push(bel[mate[x]]);

reverse(bloch[b].begin() + 1, bloch[b].end());

for (int x = xu; x != xa; x = bel[fa[bel[mate[x]]]])

bloch[b].push\_back(x), bloch[b].push\_back(bel[mate[x]]), q\_push(bel[mate[x]]);

set\_bel(b, b);

for (int x = 1; x <= n\_x; x++)

{

mat[b][x].w = mat[x][b].w = 0;

blofrom[b][x] = 0;

}

for (int i = 0; i < size(bloch[b]); i++)

{

int xs = bloch[b][i];

for (int x = 1; x <= n\_x; x++)

if (mat[b][x].w == 0 || e\_delta(mat[xs][x]) < e\_delta(mat[b][x]))

mat[b][x] = mat[xs][x], mat[x][b] = mat[x][xs];

for (int x = 1; x <= n\_x; x++)

if (blofrom[xs][x])

blofrom[b][x] = xs;

}

calc\_slackv(b);

}

inline void expand\_blossom1(int b) // lab[b] == 1

{

for (int i = 0; i < size(bloch[b]); i++)

set\_bel(bloch[b][i], bloch[b][i]);

int xr = blofrom[b][mat[b][fa[b]].v];

int pr = find(bloch[b].begin(), bloch[b].end(), xr) - bloch[b].begin();

if (pr % 2 == 1)

{

reverse(bloch[b].begin() + 1, bloch[b].end());

pr = size(bloch[b]) - pr;

}

for (int i = 0; i < pr; i += 2)

{

int xs = bloch[b][i], xns = bloch[b][i + 1];

fa[xs] = mat[xns][xs].v;

col[xs] = 1, col[xns] = 0;

slackv[xs] = 0, calc\_slackv(xns);

q\_push(xns);

}

col[xr] = 1;

fa[xr] = fa[b];

for (int i = pr + 1; i < size(bloch[b]); i++)

{

int xs = bloch[b][i];

col[xs] = -1;

calc\_slackv(xs);

}

bel[b] = 0;

}

inline void expand\_blossom\_final(int b) // at the final stage

{

for (int i = 0; i < size(bloch[b]); i++)

{

if (bloch[b][i] > n && lab[bloch[b][i]] == 0)

expand\_blossom\_final(bloch[b][i]);

else

set\_bel(bloch[b][i], bloch[b][i]);

}

bel[b] = -1;

}

inline bool on\_found\_edge(const edge &e)

{

int xv = bel[e.v], xu = bel[e.u];

if (col[xu] == -1)

{

int nv = bel[mate[xu]];

fa[xu] = e.v;

col[xu] = 1, col[nv] = 0;

slackv[xu] = slackv[nv] = 0;

q\_push(nv);

}

else if (col[xu] == 0)

{

int xa = get\_lca(xv, xu);

if (!xa)

{

augment(xv, xu), augment(xu, xv);

for (int b = n + 1; b <= n\_x; b++)

if (bel[b] == b && lab[b] == 0)

expand\_blossom\_final(b);

return true;

}

else

add\_blossom(xv, xa, xu);

}

return false;

}

bool match()

{

for (int x = 1; x <= n\_x; x++)

col[x] = -1, slackv[x] = 0;

q\_n = 0;

for (int x = 1; x <= n\_x; x++)

if (bel[x] == x && !mate[x])

fa[x] = 0, col[x] = 0, slackv[x] = 0, q\_push(x);

if (q\_n == 0)

return false;

while (true)

{

for (int i = 0; i < q\_n; i++)

{

int v = q[i];

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

if (mat[v][u].w > 0 && bel[v] != bel[u])

{

int d = e\_delta(mat[v][u]);

if (d == 0)

{

if (on\_found\_edge(mat[v][u]))

return true;

}

else if (col[bel[u]] == -1 || col[bel[u]] == 0)

update\_slackv(v, bel[u]);

}

}

int d = INF;

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

if (col[bel[v]] == 0)

tension(d, lab[v]);

for (int b = n + 1; b <= n\_x; b++)

if (bel[b] == b && col[b] == 1)

tension(d, lab[b] / 2);

for (int x = 1; x <= n\_x; x++)

if (bel[x] == x && slackv[x])

{

if (col[x] == -1)

tension(d, e\_delta(mat[slackv[x]][x]));

else if (col[x] == 0)

tension(d, e\_delta(mat[slackv[x]][x]) / 2);

}

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

{

if (col[bel[v]] == 0)

lab[v] -= d;

else if (col[bel[v]] == 1)

lab[v] += d;

}

for (int b = n + 1; b <= n\_x; b++)

if (bel[b] == b)

{

if (col[bel[b]] == 0)

lab[b] += d \* 2;

else if (col[bel[b]] == 1)

lab[b] -= d \* 2;

}

q\_n = 0;

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

if (lab[v] == 0) // all unmatched vertices' labels are zero! cheers!

return false;

for (int x = 1; x <= n\_x; x++)

if (bel[x] == x && slackv[x] && bel[slackv[x]] != x && e\_delta(mat[slackv[x]][x]) == 0)

{

if (on\_found\_edge(mat[slackv[x]][x]))

return true;

}

for (int b = n + 1; b <= n\_x; b++)

if (bel[b] == b && col[b] == 1 && lab[b] == 0)

expand\_blossom1(b);

}

return false;

}

void calc\_max\_weight\_match()

{

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

mate[v] = 0;

n\_x = n;

n\_matches = 0;

tot\_weight = 0;

bel[0] = 0;

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

bel[v] = v, bloch[v].clear();

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

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

blofrom[v][u] = v == u ? v : 0;

int w\_max = 0;

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

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

relax(w\_max, mat[v][u].w);

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

lab[v] = w\_max;

while (match())

n\_matches++;

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

if (mate[v] && mate[v] < v)

tot\_weight += mat[v][mate[v]].w;

}

int main()

{

n = getint(), m = getint();

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

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

mat[v][u] = edge(v, u, 0);

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

{

int v = getint(), u = getint(), w = getint();

mat[v][u].w = mat[u][v].w = w;

}

calc\_max\_weight\_match();

printf("%lld\n", tot\_weight);

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

printf("%d ", mate[v]);

printf("\n");

return 0;

}