/\*

PROG: travel

LANG: C++

ID: hayk.sa1

\*/

#include <cstdio>

#include <cstring>

#include <set>

using namespace std;

#define inf 1000000000

#define swap(a, b) {a^=b;b^=a;a^=b;}

int n, l;

int e[200010][3];

int \*g[100010], \*w[100010], deg[100010];

// dijkstra with heap

int d[100010], pr[100010];

int h[100010], pl[100010];

void down(int p)

{

int q, lf, rg;

while (1)

{

q = p;

lf = (p<<1)|1;

rg = lf+1;

if (lf < l && d[h[lf]] < d[h[q]])

q = lf;

if (rg < l && d[h[rg]] < d[h[q]])

q = rg;

if (p == q)

return;

swap(h[p], h[q]);

swap(pl[h[p]], pl[h[q]]);

p = q;

}

}

void up(int p)

{

int par;

while (p)

{

par = (p-1)>>1;

if (d[h[p]] >= d[h[par]])

return;

swap(h[p], h[par]);

swap(pl[h[p]], pl[h[par]]);

if (h[3] == 8)

p = p;

p = par;

}

}

void dijkstra()

{

int p, i;

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

{

d[i] = inf;

h[i] = i;

pl[i] = i;

}

l = n;

d[0] = 0;

while (l != 1)

{

p = h[0];

swap(h[0], h[l-1]);

swap(pl[h[0]], pl[h[l-1]]);

l--;

down(0);

for (i = 0; i < deg[p]; i++)

if (d[g[p][i]] > d[p]+w[p][i])

{

d[g[p][i]] = d[p]+w[p][i];

up(pl[g[p][i]]);

}

}

}

////////////////////////////////////////////

int ans[100010];

int \*pc[100010], \*wc[100010], dc[100010]; // i's children, edge weights to children, number of children

struct edge

{

int p, q, l;

} s;

bool operator < (edge a, edge b)

{

return a.l < b.l || a.l == b.l && a.p < b.p || a.l == b.l && a.p == b.p && a.q < b.q;

}

set<edge> dat[100010];

int num[100010], numv;

int id[100010];

int find(int x)

{

if (x != id[x])

id[x] = find(id[x]);

return id[x];

}

void unite(int x, int y)

{

int px=find(x);

int py=find(y);

if (px == py)

return;

if (dat[px].size() < dat[py].size())

{

dat[py].insert(dat[px].begin(), dat[px].end());

dat[px].clear();

id[px] = id[x] = py;

}

else

{

dat[px].insert(dat[py].begin(), dat[py].end());

dat[py].clear();

id[py] = id[y] = px;

}

}

void solve(int p)

{

int fp, q, i;

set<edge>::iterator I;

num[p] = numv++;

for (i = 0; i < dc[p]; i++)

{

solve(pc[p][i]);

unite(p, pc[p][i]);

}

fp = find(p);

for (i = 0; i < deg[p]; i++)

{

q = g[p][i];

if (pr[p] != q)

{

s.p = p;

s.q = q;

if (p > q)

swap(s.p, s.q);

s.l = d[p]+d[q]+w[p][i];

if (num[q] == -1 || num[q] < num[p])

dat[fp].insert(s);

else

{

I = dat[fp].find(s);

if (I != dat[fp].end())

dat[fp].erase(I);

}

}

}

while (dat[fp].size())

{

s = \*dat[fp].begin();

if (num[s.p] <= num[p] || num[s.q] <= num[p])

break;

dat[fp].erase(dat[fp].begin());

}

if (dat[fp].size())

ans[p] = s.l-d[p];

}

int main()

{

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

freopen("travel.out", "w", stdout);

int m, i, j;

// initialization and path computing

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

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

{

scanf("%d%d%d", e[i], e[i]+1, e[i]+2);

deg[--e[i][0]]++;

deg[--e[i][1]]++;

}

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

{

g[i] = new int[deg[i]+1];

w[i] = new int[deg[i]+1];

deg[i] = 0;

}

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

{

g[e[i][0]][deg[e[i][0]]] = e[i][1];

g[e[i][1]][deg[e[i][1]]] = e[i][0];

w[e[i][0]][deg[e[i][0]]++] = e[i][2];

w[e[i][1]][deg[e[i][1]]++] = e[i][2];

}

dijkstra();

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

for (j = 0; j < deg[i]; j++)

if (d[i] == d[g[i][j]]+w[i][j])

pr[i] = g[i][j];

pr[0] = -1;

memset(dc, 0, sizeof(dc));

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

dc[pr[i]]++;

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

{

pc[i] = new int[dc[i]];

wc[i] = new int[dc[i]];

dc[i] = 0;

}

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

for (j = 0; j < deg[i]; j++)

if (d[g[i][j]] == d[i]+w[i][j])

{

pc[i][dc[i]] = g[i][j];

wc[i][dc[i]++] = w[i][j];

}

///////////////////////////////////////////

numv = 0;

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

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

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

id[i] = i;

solve(0);

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

printf("%d\n", ans[i]);

return 0;

}