目录

dinic.cpp

```
int N, NP, NC, M;
struct Edge
{
    int u, v, cap;
    Edge() {}
    Edge(int u, int v, int cap): u(u), v(v), cap(cap) {}
} es[150 * 150];
int R, S, T;
vector<int> tab[109]; // ± ¯
int dis[109];
int current[109];
void addedge(int u, int v, int cap)
{
    tab[u].push_back(R);
    es[R++] = Edge(u, v, cap); //
    tab[v].push_back(R);
    es[R++] = Edge(v, u, 0); // · ´     -Ì0
    //       ±  ¹      õ½· ´    ± , 2 ^ 1 == 3 ; 3 ^ 1 == 2
}
int BFS()
{
    queue<int> q;
    q.push(S);
    memset(dis, 0x3f, sizeof(dis));
    dis[S] = 0;
    while (!q.empty())
    {
        int h = q.front();
        q.pop();
        for (int i = 0; i < tab[h].size(); i++)
        {
            Edge &e = es[tab[h][i]];
            if (e.cap > 0 && dis[e.v] == 0x3f3f3f3f)
            {
                dis[e.v] = dis[h] + 1;
```

```
                q.push(e.v);
            }
        }
    }
    return dis[T] < 0x3f3f3f3f; // ·µ»      «µ½´
}
int dinic(int x, int maxflow)
{
    if (x == T)
        return maxflow;
    // i = current[x] µ±Ç°»¡ Ż¯
    for (int i = current[x]; i < tab[x].size(); i++)
    {
        current[x] = i;
        Edge &e = es[tab[x][i]];
        if (dis[e.v] == dis[x] + 1 && e.cap > 0)
        {
            int flow = dinic(e.v, min(maxflow, e.cap));
            if (flow)
            {
                e.cap -= flow; //         -½µµ
                es[tab[x][i] ^ 1].cap += flow; // ·´      -
                return flow;
            }
        }
    }
    return 0; //   »µ½    ·
}
int DINIC()
{
    int ans = 0;

    while (BFS()) // ½¨b·
    {
        int flow;
        memset(current, 0, sizeof(current)); // BFSº  µ±    ±Ç°»¡
        while (flow = dinic(S, 0x3f3f3f3f)) //  ´ BFS¿
            ans += flow;
```

```
    }
    return ans;
}
```

dinic.cpp

findnumbersofd$n$_divisors.cpp

```
const int N = 2002;
int f[N],g[N*N];
bool ip[N];
int mu[N],p[N],nt[N];
void init_mu(){
    mu[1]=1;ip[2]=true;p[0]=2;
    for(int i=1;i<N;i+=2)   ip[i]=true;
    for(int i = 3,cnt = 1;i<N;i+=2){
        if(ip[i]){
            p[cnt++] = i;
            mu[i] = -1;
        }
        for(int j = 1,t;j<cnt&&(t= i * p[j])<N;++j){
            ip[t] = false;
            if(i % p[j] == 0)   break;
            mu[t] = -mu[i];
        }
    }
    for(int i=2;i<N;i+=4)  mu[i]=-mu[i>>1];
}
int getsum(int n){
    int sum = 0;
    for(int i=1,j;i<=n;i=j+1){
        j = n/(n/i);
        sum += (j-i+1)*(n/i);
    }
    return sum;
```

```
}
void init(){
    for(int i=0;i<N;++i)        f[i]=getsum(i);     init_mu();
    int last = 1;
    for(int i=2;i<N;++i){
        if(mu[i]==0)    continue;
        nt[last]=i;
        last = i;
    }
    nt[last]=N;
}
void getg(int a,int b,int c){
    int n = a*b;
    for(int i=0;i<=n;++i)   g[i]=0;
    for(int i=1;i<=n&&i<=c;i=nt[i]){ // n^2 \log n
        for(int j=i,t=mu[i]*f[c/i];j<=n;j+=i)   g[j]+=t;
    }
}
LL getf(int a,int b,int d){
    LL res = 0;
    for(int i=1;i<=a;++i){
        for(int j=1;j<=b;++j){
            res+=(a/i)*(b/j)*g[i*j*d];
        }
    }
    return res;
}
int main(){
//        freopen("/Users/dna049/Desktop/AC/in","r",stdin);
    //    freopen("/Users/dna049/Desktop/AC/out","w",stdout);
    init();
    int a,b,c;
    while(~scanf("%d%d%d",&a,&b,&c)){
        getg(a,b,c);
        int ab = min(a,b);
        LL res = 0;
        for(int i=1;i<=ab;i=nt[i]){
            res+=mu[i]*getf(a/i,b/i,i*i);
```

```
        }
        printf("%d\n",int(res%(1<<30)));
    }
    return 0;
}
```

findnumbersofd(n)_divisors.cpp

ISap.cpp

```
class ISap {
  /**
   * Time Complexity: O(N^2*M)
   * Special Case Time Complexity:
   *    If the capacity and flow are integer, the time complexity is
   *    O(min{N^(2/3), M^(1/2)}*M)
   * Improve:
   *    Gap, Multiple Augmented, Manual Stack, Active Label
   * Usage:
   *    init()
   *    {addEdge()}
   *    [preLabel()]
   *    maxflow()
     *  Attention:
     *      AddEdge() will add directed edge defaultly.
     *      If you need undirected edge, please modify it.
   *  preLabel function is not nessary. But will slower 10% if not use.
     *  In hierarchical graph, preLabel function is nessary.
   * */
private :
  static const int N = 1001010, M = 3000010, INF = 1000000001;
public :
  int fir[N], nex[M * 2], son[M * 2], cap[M * 2], flow[M * 2], totEdge;
  int gap[N], dep[N], src, des, totPoint;
```

```
26  void init(int _totPoint) {
      totEdge = 0, totPoint = _totPoint;
28    for(int i = 0; i <= totPoint; ++i)
        gap[i] = dep[i] = 0, fir[i] = -1;
30  }

32  void addEdge(const int u, const int v, const int c) {
        /**
34       * Attention, This function is added directed edge defaultly.
         * */
36    nex[totEdge] = fir[u], son[totEdge] = v, cap[totEdge] = c, flow[totEdge] = 0;
      fir[u] = totEdge++;
38    nex[totEdge] = fir[v], son[totEdge] = u, cap[totEdge] = c, flow[totEdge] = 0;
      fir[v] = totEdge++;
40  }

42  void preLabel() {
      static int que[N], head, tail;
44    head = tail = 0;
      for(int i = 0; i <= totPoint; ++i) dep[i] = totPoint, gap[i] = 0;
46
      que[tail++] = des, dep[des] = 0, ++gap[0];
48    while(head < tail) {
        int u = que[head++];
50      for(int tab = fir[u], v; tab != -1; tab = nex[tab]) {
          if(flow[tab ^ 1] >= cap[tab ^ 1]) continue;
52        if(dep[v = son[tab]] < totPoint) continue;
          ++gap[dep[v] = dep[u] + 1], que[tail++] = v;
54      }
      }
56  }

58  int maxflow() {
      static int cur[N], rev[N];
60    int flows = 0;
      for(int i = 0; i <= totPoint; ++i) cur[i] = fir[i];
62    for(int u = src; dep[u] < totPoint && dep[src] < totPoint; ) {
        if(u == des) {
```

```
64          int nowFlow = INF;
            for(int p = src; p != des; p = son[cur[p]])
66            nowFlow = min(nowFlow, cap[cur[p]] - flow[cur[p]]);
            for(int p = src; p != des; p = son[cur[p]])
68            flow[cur[p]] += nowFlow, flow[cur[p] ^ 1] -= nowFlow;
            flows += nowFlow, u = src;
70        }

72        int tab;
          for(tab = cur[u]; tab != -1; tab = nex[tab])
74          if(cap[tab] > flow[tab] && dep[u] == dep[son[tab]] + 1) break;
          if(tab != -1) {
76          cur[u] = tab, rev[son[tab]] = tab ^ 1;
            u = son[tab];
78        } else {
            if(--gap[dep[u]] == 0) break;
80          cur[u] = fir[u];
            int minDep = totPoint;
82          for(tab = fir[u]; tab != -1; tab = nex[tab])
              if(cap[tab] > flow[tab]) minDep = min(minDep, dep[son[tab]]);
84          ++gap[dep[u] = minDep + 1];
            if(u != src) u = son[rev[u]];
86        }
        }
88      return flows;
      }
90  };
```

ISap.cpp

bignumber.cpp

```
1   class BigNum
    {
3     /*
```

```
  you can use BigNum(x) + BigNum(y) to calculate.
  This code doesn't use referrence when working.
  */
  static const int MAXLENGTH = 10010, M = 100;
  // M -> MOD
public :
  int arr[MAXLENGTH], length;
  // the maximum value -> M * M * MAXLENGTH * MAXLENGTH when calculate *

  BigNum()
  {
    length = 0;
    clr(arr, 0);
  }

  BigNum(string &str) {
    length = 0, clr(arr, 0);
    int x = 0, now = 1, len = sz(str);
    for(int i = len - 1; i >= 0; i--) {
      x = x + (str[i] - '0') * now;
      now *= 10;
      if(now >= M) {
        arr[++length] = x;
        x = 0, now = 1;
      }
    }
    if(x) arr[++length] = x;
    length = max(length, 1);
  }

  inline void operator =(int x)
  {
    length = 0, clr(arr, 0);
    while(x)
    {
      arr[++length] = x % M;
      x /= M;
    }
```

```
    if(!length) length++;
43  }

45  BigNum(int x)
    {
47    *this = x;
    }
49
    inline BigNum operator -(const BigNum& a)
51  {
      BigNum ret;
53    for(int i = 1; i <= length; i++) ret.arr[i] = arr[i] - a.arr[i];
      for(int i = 1; i <= length; i++)
55      if(ret.arr[i] < 0)
        {
57        ret.arr[i + 1]--;
          ret.arr[i] += M;
59      }
      ret.length = max(length, a.length);
61    while(length > 1 && !ret.arr[ret.length]) ret.length--;
      return ret;
63  }

65  inline BigNum operator +(const BigNum& a)
    {
67    BigNum ret;
      ret.length = max(length, a.length);
69    for(int i = 1; i <= ret.length; i++) ret.arr[i] = arr[i] + a.arr[i];
      for(int i = 1; i <= ret.length; i++)
71      if(ret.arr[i] >= M)
        {
73        ret.arr[i + 1]++;
          ret.arr[i] -= M;
75      }
      while(ret.arr[ret.length + 1]) ret.length++;
77    return ret;
    }
79
```

```
inline BigNum operator *(const BigNum& a)
{
  BigNum ret;
  for(int i = 1; i <= length; i++)
    for(int j = 1; j <= a.length; j++)
      ret.arr[i + j - 1] += arr[i] * a.arr[j];
  ret.length = length + a.length - 1;
  for(int i = 1; i <= ret.length; i++)
  {
    ret.arr[i + 1] += ret.arr[i] / M;
    ret.arr[i] %= M;
  }
  while(ret.arr[ret.length + 1])
  {
    ret.length++;
    ret.arr[ret.length + 1] += ret.arr[ret.length] / M;
    ret.arr[ret.length] %= M;
  }
  return ret;
}

inline BigNum operator /(int x)
{
  BigNum ret = *this;
  for(int i = ret.length; i >= 1; i--)
  {
    if(i > 1) ret.arr[i - 1] += (ret.arr[i] % x) * M;
    ret.arr[i] /= x;
  }
  while(ret.length > 1 && !ret.arr[ret.length])
    ret.length--;
  return ret;
}

inline bool operator >(BigNum &a)
{
  if(length != a.length) return length > a.length;
  for(int i = length; i >= 1; i--)
```

```
      if(arr[i] != a.arr[i]) return arr[i] > a.arr[i];
    return false;
  }

  inline void Out()
  {
    printf("%d", (int) arr[length]);
    for(int i = length - 1; i >= 1; i--)
      printf("%02d", (int) arr[i]);
    printf("\n");
  }
} ;
```

bignumber.cpp

MinimumArborescence_ZhuLiu.cpp

```
#define foreach(x, y) for(__typeof((y).begin()) x = (y).begin(); x != (y).end(); ++x)

// points up to 2 * n, edges will up to n * m
const int N = 1010, M = 5000010;
int st[M], ed[M], val[M];
vector<int> valid_edges, valid_points, tpoints, tedges;
int n, m, origin_n, origin_m;
int pre[N], path[N], len, belong[N], deactivate[M], activate[M];
int visit[N];
bool mark[M];

void insert(int u, int v, int w, int deact = -1, int act = -1) {
  st[m] = u, ed[m] = v, val[m] = w;
  deactivate[m] = deact, activate[m] = act;
  ++m;
}

void solve() {
```

```
19  origin_n = n, origin_m = m;

21  for(int i = 0; i < m; ++i) valid_edges.push_back(i);
    for(int i = 0; i < n; ++i) valid_points.push_back(i);
23  int answer = 0;
    int root = 0;
25    bool impossible = false;
    while(true) {
27    foreach(point, valid_points) pre[*point] = -1;
      foreach(e, valid_edges) {
29      int v = ed[*e];
        if(pre[v] == -1 || val[pre[v]] > val[*e]) pre[v] = *e;
31    }
      pre[root] = -1;
33
      foreach(point, valid_points)
35      visit[*point] = -1, belong[*point] = *point;
      tpoints.clear(), tedges.clear();
37    bool flag = false;
      foreach(point, valid_points) {
39      int u = *point;
        if(visit[u] != -1) continue;
41      len = 0;
        for(len = 0; visit[u] == -1; u = st[pre[u]]) {
43        path[len++] = u, visit[u] = *point;
          if(pre[u] == -1) break;
45      }
        if(pre[u] == -1 || visit[u] != *point) {
47        for(int i = 0; i < len; ++i) tpoints.push_back(path[i]);
        } else {
49                int start = 0;
                  while(path[start] != u) tpoints.push_back(path[start++]);
51        flag = true;
          int p = n++;
53        for(int i = start; i < len; ++i) {
            int u = path[i];
55          belong[u] = p;
                    mark[pre[u]] = true, answer += val[pre[u]];
```

```
57          }
            tpoints.push_back(p);
59        }
      }
61    if(!flag) {
            int cnt_nopre = 0;
63      foreach(point, valid_points)
          if(pre[*point] == -1) ++cnt_nopre;
65              else answer += val[pre[*point]], mark[pre[*point]] = true;
            if(cnt_nopre > 1) impossible = true;
67      break;
      }
69    foreach(edge, valid_edges) {
        int u = st[*edge], v = ed[*edge], w = val[*edge];
71      if(belong[u] == belong[v]) continue;
        if(belong[u] == u && belong[v] == v) tedges.push_back(*edge);
73      else {
          insert(
75            belong[u], belong[v],
              belong[v] == v ? w : w - val[pre[v]],
77            belong[v] == v ? -1 : pre[v],
              *edge
79            );
          tedges.push_back(m - 1);
81      }
      }
83
    root = belong[root];
85  valid_points = tpoints, valid_edges = tedges;
  }
87
  for(int i = m - 1; i >= 0; --i)
89  if(mark[i]) {
      if(deactivate[i] != -1) mark[deactivate[i]] = false;
91    if(activate[i] != -1) mark[activate[i]] = true;
    }
93
    if(impossible) puts("-1");
```

```
    else {
        vector<int> ans;
        for(int i = 0; i < origin_m; ++i)
            if(mark[i]) ans.push_back(i + 1);
        sort(ans.begin(), ans.end());
        assert(origin_n - 1 == (int) ans.size());
        printf("%d\n", (int) ans.size());
        for(int i = 0, s = (int) ans.size(); i < s; ++i) {
            printf(i == s - 1 ? "%d\n" : "%d ", ans[i]);
        }
        // printf("%d\n", answer);
    }
}
```

MinimumArborescence_ZhuLiu.cpp

zkwcostflow.cpp

```
#include <cstdio>
#include <cstring>
using namespace std;
const int maxint=~0U>>1;

int n,m,pi1,cost=0;
bool v[550];
struct etype
{
    int t,c,u;
    etype *next,*pair;
    etype(){}
    etype(int t_,int c_,int u_,etype* next_):
        t(t_),c(c_),u(u_),next(next_){}
    void* operator new(unsigned,void* p){return p;}
} *e[550];

```

```
int aug(int no,int m)
{
    if(no==n)return cost+=pi1*m,m;
    v[no]=true;
    int l=m;
    for(etype *i=e[no];i;i=i->next)
        if(i->u && !i->c && !v[i->t])
        {
            int d=aug(i->t,l<i->u?l:i->u);
            i->u-=d,i->pair->u+=d,l-=d;
            if(!l)return m;
        }
    return m-l;
}

bool modlabel()
{
    int d=maxint;
    for(int i=1;i<=n;++i)if(v[i])
        for(etype *j=e[i];j;j=j->next)
            if(j->u && !v[j->t] && j->c<d)d=j->c;
    if(d==maxint)return false;
    for(int i=1;i<=n;++i)if(v[i])
        for(etype *j=e[i];j;j=j->next)
            j->c-=d,j->pair->c+=d;
    pi1 += d;
    return true;
}

int main()
{
    freopen("costflow.in","r",stdin);
    freopen("costflow.out","w",stdout);
    scanf("%d %d",&n,&m);
    etype *Pe=new etype[m+m];
    while(m--)
    {
        int s,t,c,u;
```

```
        scanf("%d%d%d%d",&s,&t,&u,&c);
        e[s]=new(Pe++)etype(t, c,u,e[s]);
        e[t]=new(Pe++)etype(s,-c,0,e[t]);
        e[s]->pair=e[t];
        e[t]->pair=e[s];
    }
    do do memset(v,0,sizeof(v));
    while(aug(1,maxint));
    while(modlabel());
    printf("%d\n",cost);
    return 0;
}
```

zkwcostflow.cpp

makrpointbypolygon.cpp

```
const int N = 610, M = 8010 * 2;
const double EPS = 1e-7;
struct Point {
  int x, y;

  Point() {}
  Point(int x, int y):x(x), y(y) {}

  inline Point operator +(const Point &t) const {
    return Point(x + t.x, y + t.y);
  }

  inline Point operator -(const Point &t) const {
    return Point(x - t.x, y - t.y);
  }

  inline ll operator *(const Point &t) const {
    return x * t.x + y * t.y;
```

```
  }

  inline ll operator %(const Point &t) const {
    return x * t.y - y * t.x;
  }

  inline bool operator <(const Point &t) const {
    return x < t.x || (x == t.x && y < t.y);
  }

  inline void read() {
    scanf("%d%d", &x, &y);
  }
} cap[N], p[M * 2];
double arg[M * 2];
typedef vector<int> Poly;
int n, m;
map<Point, vector<int> > graph;
map<int, int> ofPoly;
int father[N], bel[N];
vector<int> cover[N];
Poly poly[M];
int totPoly, con[M];
double area[M];
vector<int> ans[N];
bool visit[M];
int idx[N];

inline void init() {
  graph.clear(), ofPoly.clear();
  for(int i = 0; i < n; ++i) cover[i].clear();
  for(int i = 0; i < n; ++i) ans[i].clear();
  clr(visit, 0), totPoly = 0;
}

inline int sgn(double x) {
  return x < -EPS ? -1 : x > EPS;
}
```

```
57
inline double getArg(const Point &a) {
59  return atan2(a.y, a.x);
}
61
inline bool cmpByArg(int a, int b) {
63  return arg[a] < arg[b];
}
65
inline bool cmpByArea(int a, int b) {
67  return area[a] < area[b];
}
69
inline bool cmpByOfPolyArea(int x, int y) {
71  return area[bel[x]] < area[bel[y]];
}
73
inline int getNextEdge(int x) {
75  int rank = lower_bound(all(graph[p[x ^ 1]]), x ^ 1, cmpByArg) -
        graph[p[x ^ 1]].begin();
77  return rank ? graph[p[x ^ 1]][rank - 1] : graph[p[x ^ 1]].back();
}
79
inline double getArea(const Poly &n) {
81  double ret = 0.;
  for(int i = 1; i < sz(n) - 1; ++i)
83    ret += (p[n[i]] - p[n[0]]) % (p[n[i + 1]] - p[n[0]]);
  return ret / 2.;
85}

87inline bool pointInPoly(const Point &x, const Poly &n) {
  int cross = 0;
89  for(int i = 0; i < sz(n); ++i) {
    Point v1 = p[n[i]] - x, v2 = p[n[(i + 1) % sz(n)]] - x;
91    int c = sgn(v1 % v2), d = sgn(v1 * v2);
    if(!c && d <= 0) return 0;
93    int d1 = sgn(v1.y), d2 = sgn(v2.y);
    if(c > 0 && d1 <= 0 && d2 > 0) ++cross;
```

```
95      if(c < 0 && d2 <= 0 && d1 > 0) --cross;
    }
97  return cross > 0;
}
99
inline void solve() {
101 // build graph by angle
    for(int i = 0; i < 2 * m; ++i) arg[i] = getArg(p[i ^ 1] - p[i]);
103 for(int i = 0; i < 2 * m; ++i) graph[p[i]].pub(i);
    foreach(it, graph) sort(all(it->second), cmpByArg);
105
    // get all polygons
107 for(int i = 0; i < 2 * m; ++i) {
      if(visit[i]) continue;
109   Poly t;
      t.clear();
111   int now = i;
      do {
113     visit[now] = true, t.pub(now);
        now = getNextEdge(now);
115   } while(now != i);
      area[totPoly] = getArea(t);
117   if(sgn(area[totPoly]) <= 0) continue;
      poly[totPoly] = t;
119   foreach(it, t) ofPoly[*it] = totPoly;
      for(int j = 0; j < n; ++j)
121     if(pointInPoly(cap[j], t)) cover[j].pub(totPoly);
      ++totPoly;
123 }

125 for(int i = 0; i < n; ++i) sort(all(cover[i]), cmpByArea);
    for(int i = 0; i < n; ++i) con[bel[i] = cover[i][0]] = i;
127 for(int i = 0; i < 2 * m; i += 2)
      if(ofPoly.count(i) && ofPoly.count(i ^ 1)) {
129     int x = con[ofPoly[i]], y = con[ofPoly[i ^ 1]];
        // printf("%d %d\n", x, y);
131     ans[x].pub(y), ans[y].pub(x);
      }
```

```
133
  for(int i = 0; i < n; ++i) idx[i] = i;
135 sort(idx, idx + n, cmpByOfPolyArea);
  for(int i = 0; i < n; ++i) {
137   bool flag = true;
    foreach(e, poly[bel[idx[i]]])
139     if(ofPoly.count(*e ^ 1) == 0) {
        flag = false;
141     break;
      }
143   if(!flag) {
      for(int j = i + 1; j < n; ++j)
145     if(pointInPoly(cap[idx[i]], poly[bel[idx[j]]])) {
        ans[idx[i]].pub(idx[j]), ans[idx[j]].pub(idx[i]);
147     break;
        }
149   }
  }
151
  for(int i = 0; i < n; ++i) {
153   sort(all(ans[i]));
    ans[i].erase(unique(all(ans[i])), ans[i].end());
155   printf("%d", sz(ans[i]));
    for(int j = 0; j < sz(ans[i]); ++j)
157     printf(" %d", ans[i][j] + 1);
    puts("");
159 }
}
161
int main() {
163 freopen("a.in", "r", stdin);
  freopen("a.out", "w", stdout);
165 while(scanf("%d%d", &n, &m) == 2 && n + m != 0) {
    init();
167   for(int i = 0; i < n; ++i) cap[i].read();
    for(int i = 0; i < 2 * m; ++i) p[i].read();
169   solve();
  }
```

```
  return 0;
}
```

makrpointbypolygon.cpp

fft.cpp

```
class Complex {
  public :
    double real, image;

    Complex(double real = 0., double image = 0.):real(real), image(image) {}
    Complex(const Complex &t):real(t.real), image(t.image) {}

    Complex operator +(const Complex &t) const {
      return Complex(real + t.real, image + t.image);
    }

    Complex operator -(const Complex &t) const {
      return Complex(real - t.real, image - t.image);
    }

    Complex operator *(const Complex &t) const {
      return Complex(real * t.real - image * t.image,
            real * t.image + t.real * image);
    }
};

const int N = 300010, MOD = 313;
const double PI = acos(-1.);

class FFT {
  /**
   * 1. Need define PI
   * 2. Need define class Complex
```

```
29    * 3. tmp is need for fft, so define a N suffice it
      * 4. dig[30] -> (1 << 30) must bigger than N
31    * */
    private:
33    static Complex tmp[N];
      static int revNum[N], dig[30];
35
    public:
37    static void init(int n) {
        int len = 0;
39      for(int t = n - 1; t; t >>= 1) ++len;
        for(int i = 0; i < n; i++) {
41        revNum[i] = 0;
          for(int j = 0; j < len; j++) dig[j] = 0;
43        for(int idx = 0, t = i; t; t >>= 1) dig[idx++] = t & 1;
          for(int j = 0; j < len; j++)
45          revNum[i] = (revNum[i] << 1) | dig[j];
        }
47    }

49    static int rev(int x) {
        return revNum[x];
51    }

53    static void fft(Complex a[], int n, int flag) {
        /**
55       * flag = 1 -> DFT
         * flag = -1 -> IDFT
57       * */
        for(int i = 0; i < n; ++i) tmp[i] = a[rev(i)];
59      for(int i = 0; i < n; ++i) a[i] = tmp[i];
        for(int i = 2; i <= n; i <<= 1) {
61        Complex wn(cos(2 * PI / i), flag * sin(2 * PI / i));
          for(int k = 0, half = i / 2; k < n; k += i) {
63          Complex w(1., 0.);
            for(int j = k; j < k + half; ++j) {
65            Complex x = a[j], y = w * a[j + half];
              a[j] = x + y, a[j + half] = x - y;
```

```
            w = w * wn;
          }
        }
      }
      if(flag == -1) {
        for(int i = 0; i < n; ++i) a[i].real /= n;
      }
    }

    static void dft(Complex a[], int n) {
      fft(a, n, 1);
    }

    static void idft(Complex a[], int n) {
      fft(a, n, -1);
    }
};
Complex FFT::tmp[N];
int FFT::revNum[N], FFT::dig[30];
```

fft.cpp

FastIO.cpp

```
namespace FastIO {
    /**
     * Defaultly, can deal with negative number.
     * Only for reading integers, modify for specify problems.
     * Throws int exception if nothing to read.
     * */
    const int S = 2000000;
    char s[S], *h = s+S, *t = h;
    inline char getchr(void) {
        if(h == t) {
            if(t != s + S) return -1;
```

```
            t = s + fread(s, 1, S, stdin), h = s;
        }
        return *h++;
    }
    inline int getint(void) throw(int) {
        char c = ' ';
        bool positive = true;
        for (; !isdigit(c); c = getchr()) {
            if(c == -1) throw(-1);
            positive ^= (c == '-');
        }
        int x = 0;
        for (; isdigit(c); c = getchr()) x = x * 10 + c - '0';
        return positive ? x : -x;
    }
}
using FastIO::getint;
```

FastIO.cpp

sustainable01tree.cpp

```
struct Trie {
  /**
   * 1. MAXBIT = max depth
   * 2. rot = The root of the tr[i](ith Trie)
   * 3. interface add(val, w) -> val (value)   w (times)
   *    insert w elements with value val to the trie
   * 4. remember initialize the Trie::Node::tot = 0
   * 5. updata -  change it as you want, it will updata each
   *  node after insert value to its children. In the template,
   *  it updata f
   * */
  static struct Node {
    static int tot;
```

```
14
    int child[2], size;
16
    int f;
18
    inline void init() {
20    child[0] = child[1] = -1, size = f = 0;
    }
22
  } tr[N * M * M];
24
#define child(x, y) tr[x].child[y]
26#define lch(x) child(x, 0)
#define rch(x) child(x, 1)
28#define size(x) (x == -1 ? 0 : tr[x].size)
#define f(x) tr[x].f
30
  int rot;
32
  inline int bruteForce(int u, int v, int d) {
34  int ret = 0;
    while(d >= 0) {
36    int t = size(lch(v)) == 0;
      int _t = t;
38    if(size(child(u, _t)) == 0) _t ^= 1, ret += (1 << d);
      u = child(u, _t), v = child(v, t), --d;
40  }
    return ret;
42}

44inline void updata(int u, int d) {
    f(u) = -1;
46  if(size(u) > 1) {
      f(u) = 0;
48    for(int t = 0; t < 2; ++t) {
        int v = child(u, t);
50      if(size(v) > 1) f(u) = max(f(u), f(v));
        else if(size(v) == 1) {
```

```
52          int xorValue = bruteForce(child(u, t ^ 1), v, d - 1);
            xorValue += (1 << d);
54          f(u) = max(f(u), xorValue);
          }
56      }
    }
58  }

60  inline void addVal(int &x, int val, int w, int d) {
    if(x == -1) tr[x = Node::tot++].init();
62  tr[x].size += w;
    if(d >= 0) addVal(child(x, (val & (1 << d)) > 0), val, w, d - 1);
64
    updata(x, d);
66  }

68  inline void add(int val, int w) {
    addVal(rot, val, w, MAXBIT - 1);
70  }

72  inline void traverse(int x, int now, int d, Trie &pro) {
    if(x == -1) return;
74  if(d < 0) pro.add(now, size(x));
    else {
76    if(size(lch(x)) > 0)
        traverse(lch(x), now, d - 1, pro);
78    if(size(rch(x)) > 0)
        traverse(rch(x), now + (1 << d), d - 1, pro);
80  }
  }
82
  inline void operator +=(Trie &t) {
84  if(size(rot) < size(t.rot)) swap(rot, t.rot);
    t.traverse(t.rot, 0, MAXBIT - 1, *this);
86  }

88  inline int getAnswer() const {
    return f(rot);
```

```
90    }

92  } tr[N];
    int Trie::Node::tot;
94  Trie::Node Trie::tr[N * M * M];
```

sustainable01tree.cpp

printitself.cpp

```
1   #include<iostream>
    #include<string>
3   using namespace std;
    /*************************************
5   * Welcome to visit http://dna049.com
    *************************************/
7   string a[20];
    int main(){
9   a[0]="#include<iostream>";
    a[1]="#include<string>";
11  a[2]="using namespace std;";
    a[3]="/*************************************";
13  a[4]="* Welcome to visit http://dna049.com";
    a[5]="*************************************/";
15  a[6]="string a[20];";
    a[7]="int main(){";
17  a[8]="for(int i=0;i<8;++i) cout<<a[i]<<endl;";
    a[9]="for(int i=0;i<=12;++i) cout<<char(97)<<char(91)<<i<<char(93)<<char(61)<<char(34)<<a[i]<<char(34)<<char(59)<<endl;";
19  a[10]="for(int i=8;i<=12;++i) cout<<a[i]<<endl;";
    a[11]="return 0;";
21  a[12]="}";
    for(int i=0;i<8;++i) cout<<a[i]<<endl;
23  for(int i=0;i<=12;++i) cout<<char(97)<<char(91)<<i<<char(93)<<char(61)<<char(34)<<a[i]<<char(34)<<char(59)<<endl;
    for(int i=8;i<=12;++i) cout<<a[i]<<endl;
25  return 0;
```

```
}
```

printitself.cpp

karpminimumcircuit.cpp

```
#include <cstdio>
#include <cstring>
#include <cstdlib>
#include <cmath>
#include <ctime>
#include <iostream>
#include <algorithm>
#include <set>
#include <map>
#include <queue>
#include <vector>
#include <deque>
using namespace std;
#define INF (1000000001)
#define MLL (10000000000000000001LL)
#define puf push_front
#define pub push_back
#define pof pop_front
#define pob pop_back
#define sz(x) ((int) (x).size())
typedef long long ll;
typedef unsigned long long ull;

template<class T>
inline T read() {
  char ch = ' ';
  T ret = (T) 0;
  bool positive = 1;
  while(!(ch >= '0' && ch <= '9')) {
```

```
    if(ch == '-') positive ^= 1;
    ch = getchar();
  }
  while(ch >= '0' && ch <= '9') {
    ret = ret * ((T) 10) + ((T) (ch - '0'));
    ch = getchar();
  }
  return positive ? ret : -ret;
}

template<class T>
inline void print(T x) {
    static int a[24];
  int n = 0;
    while(x > 0) {
        a[n++] = x % 10;
        x /= 10;
    }
    if(n == 0) a[n++] = 0;
    while(n--) putchar('0' + a[n]);
}

const int N = 110, M = 100 * 100 + 10;
int n, m;
int dis[N][N];
struct EdgeNode {
  int u, v, c;
  EdgeNode() {}
  EdgeNode(int u, int v, int c):u(u), v(v), c(c) {}

  inline void read() {
    scanf("%d%d%d", &u, &v, &c);
  }
} edges[M];

inline void solve() {
  for(int i = 0; i < m; i++) edges[i].u--, edges[i].v--;
```

```
  for(int i = 0; i < n; i++) dis[0][i] = INF;
  dis[0][0] = 0;
  for(int i = 1; i <= n; i++) {
    for(int j = 0; j < n; j++)
      dis[i][j] = INF;

    for(int j = 0; j < m; j++)
      if(dis[i][edges[j].v] > dis[i - 1][edges[j].u] + edges[j].c)
        dis[i][edges[j].v] = dis[i - 1][edges[j].u] + edges[j].c;
  }

  double ans = INF;
  for(int i = 0; i < n; i++) {
    if(dis[n][i] >= INF) continue;
    double now = -INF;
    for(int j = 0; j < n; j++)
      if(dis[j][i] < INF)
        now = max(now, (dis[n][i] - dis[j][i]) / (1. * (n - j)));
    if(now > -INF) ans = min(ans, now);
  }

  printf("%lf\n", ans);
}

int main() {
  freopen("a.in", "r", stdin);
  scanf("%d%d", &n, &m);
  for(int i = 0; i < m; i++) edges[i].read();
  solve();
  return 0;
}
```

karpminimumcircuit.cpp

dancinglinks.cpp

```
struct DLX {
  /**
  1.  node coordinate (1...r, 1...c)
  2.  the order of adding nodes should be
    from up to down, from left to right
  */
  static const int ROWS = 125 * 5 + 10., COLS = 125 + 10;
  static const int N = ROWS * COLS + ROWS + COLS;
  int rows, cols;
  int u[N], d[N], l[N], r[N], rowIndex[N], colIndex[N], nodes;
  int countCols[N];

  inline void init(int _rows, int _cols) {
    rows = _rows, cols = _cols;
    nodes = 0;
    for(int i = 0; i <= cols; i++) {
      int now = nodes++;
      u[now] = d[now] = now;
      l[now] = i == 0 ? cols : i - 1;
      r[now] = i == cols ? 0 : i + 1;
      rowIndex[now] = 0, colIndex[now] = i;
    }
    for(int i = 1; i <= rows; i++) {
      int now = nodes++;
      u[now] = i == 1 ? rows + cols : i - 1 + cols;
      d[now] = i == rows ? 1 + cols : i + 1 + cols;
      l[now] = r[now] = now;
      rowIndex[now] = i, colIndex[now] = 0;
    }
    for(int i = 0; i <= cols; i++) countCols[i] = 0;
  }

  inline void addNode(int x, int y) {
    int now = nodes++;
    u[now] = u[y], d[u[y]] = now;
```

```
    d[now] = y, u[y] = now;
    l[now] = l[cols + x], r[l[cols + x]] = now;
    r[now] = cols + x, l[cols + x] = now;
    countCols[y]++;
    rowIndex[now] = x, colIndex[now] = y;
  }

  inline void del(int x) {
    /**
    the current del node x will be delete first or
    never show as a real node and act as a virtual node
    */
    if(!x) return;
    r[l[x]] = r[x], l[r[x]] = l[x];
    for(int row = d[x]; row != x; row = d[row])
      for(int col = r[row]; col != row; col = r[col]) {
        u[d[col]] = u[col], d[u[col]] = d[col];
        countCols[colIndex[col]]--;
      }
  }

  inline void resume(int x) {
    if(!x) return;
    for(int row = u[x]; row != x; row = u[row])
      for(int col = l[row]; col != row; col = l[col]) {
        u[d[col]] = col, d[u[col]] = col;
        countCols[colIndex[col]]++;
      }
    r[l[x]] = l[r[x]] = x;
  }

  inline void dance(bool *ans, int *size, int used, int remain) {
    /**
    change here
    */
    ans[used + remain] = true;

    int p = 0, maxx = -INF;
```

```
    for(int tab = r[0]; tab != 0; tab = r[tab])
      if(countCols[tab] > maxx)
        maxx = countCols[p = tab];

    if(!p) return;
    del(p);
    for(int row = d[p]; row != p; row = d[row]) {
      for(int col = r[row]; col != row; col = r[col])
        del(colIndex[col]);
      dance(ans, size, used + 1, remain - size[rowIndex[row]]);
      for(int col = l[row]; col != row; col = l[col])
        resume(colIndex[col]);
    }
    resume(p);
  }
} dlk;
```

dancinglinks.cpp

blocklist.cpp

```
struct BlockList {
  /**
  1. indexes -> [1..n].
  2. each block has at least one element after delete.
  3. remenber to updata head, tail after each operation.
  */
  const static int MAXLEN = N * 2;

  int l[MAXLEN], r[MAXLEN], val[MAXLEN];
  int n, block_len, blocks/*, startUse*/;
  int head[MAXLEN], tail[MAXLEN], size[MAXLEN];

  int space[MAXLEN];
```

```
15  inline void init(int len) {
      /**
17    O(len)
      */
19    n = len;
      for(block_len = 2; block_len * block_len <= n; block_len++) ;
21    // block_len = 700;
      for(blocks = 1; blocks * block_len < n; blocks++) ;
23    for(int i = 1; i <= blocks; i++) size[i] = 0;

25    int now = 0;
      size[0] = block_len;
27    for(int i = 1; i <= n; i++) {
        l[i] = i - 1, r[i] = i + 1;
29      val[i] = 0;

31      if(size[now] >= block_len) head[++now] = i;
        size[now]++, tail[now] = i;
33    }
      r[n] = 0;
35    l[0] = r[0] = 0;

37    /*(int i = n + 1; i < MAXLEN - 1; i++) r[i] = i + 1;
      startUse = n + 1;*/
39
      for(int i = 1; i <= blocks; i++)
41      space[i] = 1;
    }
43
    inline int getBlockIndex(int x) {
45    /**
      return index of the block that xth element belongs to.
47    O(sqrt(n))
      */
49    int ret = 1, nowCount = size[1];
      while(ret < blocks && nowCount < x) nowCount += size[++ret];
51    return nowCount < x ? -1 : ret;
    }
```

```
53
  inline int at(int x) {
55  /**
    get xth element of blocklist
57  O(sqrt(n))
    */
59  int bindex = getBlockIndex(x), now = 0;
    if(bindex == -1) return -1;
61  for(int i = 1; i < bindex; i++) now += size[i];
    int ret = head[bindex];
63  for(++now; now < x; ++now) ret = r[ret];
    return ret;
65  }

67  inline int getFirstSpace(int st = 1) {
    int bindex = getBlockIndex(st), x = at(st);
69  while(true) {
      if(val[x] == 0) return st;
71    if(x == tail[bindex]) break;
      x = r[x], st++;
73  }

75  for(int i = bindex + 1; i <= blocks; i++) {
      if(space[i] != -1) return st + space[i];
77    st += size[i];
    }
79  return -1;
  }
81
  inline void calcSpace(int bindex) {
83  space[bindex] = 1;
    int tab = head[bindex];
85  while(true) {
      if(val[tab] == 0) return;
87    if(tab == tail[bindex]) break;
      tab = r[tab], space[bindex]++;
89  }
    space[bindex] = -1;
```

```
91  }
93  inline void set(int pos, int v) {
      /**
95    set pos-th element's value as v
      O(sqrt(n))
97    */
      int x = at(pos), bindex = getBlockIndex(pos);
99    val[x] = v;
101   calcSpace(bindex);
    }
103
    inline void move(int pos, int value) {
105   int x = at(pos), ypos = getFirstSpace(pos);
      int bix = getBlockIndex(pos), biy = getBlockIndex(ypos);
107   int y = at(ypos);
109   if(pos != ypos) {
        if(l[y]) r[l[y]] = r[y];
111     if(r[y]) l[r[y]] = l[y];
        if(tail[biy] == y) tail[biy] = l[y];
113     if(head[biy] == y) head[biy] = r[y];
        if(l[x]) r[l[x]] = y;
115     l[y] = l[x];
        l[x] = y, r[y] = x;
117     if(head[bix] == x) head[bix] = y;
        size[bix]++, size[biy]--;
119
        for(int i = bix + 1; i <= biy; i++) {
121       if(val[tail[i - 1]] == 0) space[i] = 1;
          else if(space[i] != -1) space[i]++;
123       if(space[i - 1] == size[i - 1]) space[i - 1] = -1;
          head[i] = tail[i - 1];
125       tail[i - 1] = l[tail[i - 1]];
          size[i]++, size[i - 1]--;
127     }
      }
```

```
129
    set(pos, value);
131   calcSpace(bix), calcSpace(biy);
  }
133
  inline void getlist(int *ret, int &len) {
135   len = 0;
    for(int i = 1; i <= blocks; i++) {
137     int tab = head[i];
      while(true) {
139       ret[++len] = val[tab];
        if(tab == tail[i]) break;
141       tab = r[tab];
      }
143   }
  }
145 } mylist;
```

blocklist.cpp

treehash.cpp

```
1 const int PRIMESTOT = 7;
  const unsigned int PRIMES[PRIMESTOT] = {
3   313,
    5483,
5   85017,
    451883,
7   6459271,
    37562047,
9   142859339
  };
11
  struct cmpByHashCode {
13  unsigned int *keys;
```

```
  cmpByHashCode(unsigned int *keys):keys(keys) {}
  inline bool operator ()(const int a, const int b) const {
    return keys[a] < keys[b];
  }
};

struct TreeHash {
  /**
   * 1. It will find tree's barycentre firstly and treat it as root.
   * 2. If it has two barycentres, it will add a new nodes between them.
   * 3. Then Hash Every nodes by it subtree's structure.
   * 4. O(n * number of primes).
   * 5. Choose several primes as keys to hash.
   * 6. The number of primes determine the accuracy of hash.
   * */

  static unsigned int factor[N];

  static void prepare(int N) {
    unsigned int tmp[PRIMESTOT];
    for(int i = 0; i < PRIMESTOT; ++i) tmp[i] = PRIMES[i];
    for(int i = 0; i < N; ++i) {
      factor[i] = tmp[i % PRIMESTOT];
      tmp[i % PRIMESTOT] *= tmp[i % PRIMESTOT];
    }
  }

  int head[N], son[N * 2], nex[N * 2], tot;
  int n;
  int rot, bfsList[N], father[N], size[N], maxSubtree[N];
  unsigned int hashCode[N];
  vector<int> child[N];

  inline void init(int m) {
    n = m;
    for(int i = 0; i < n; ++i) head[i] = -1;
    tot = 0;
  }
```

```
53  inline void addEdge(int u, int v) {
      son[tot] = v, nex[tot] = head[u];
55    head[u] = tot++;
    }
57
    inline void bfs(int st) {
59    int len = 0;
      bfsList[len++] = st, father[st] = -1;
61    for(int idx = 0; idx < len; ++idx) {
        int u = bfsList[idx];
63      for(int tab = head[u], v; tab != -1; tab = nex[tab])
          if(father[u] != (v = son[tab]))
65          father[v] = u, bfsList[len++] = v;
      }
67  }

69  inline int getBarycentre() {
      bfs(0);
71
      for(int i = 0; i < n; ++i) size[i] = 0;
73    for(int i = n - 1; i >= 0; --i) {
        int u = bfsList[i];
75      ++size[u];
        if(father[u] != -1) size[father[u]] += size[u];
77
        maxSubtree[u] = n - size[u];
79      for(int tab = head[u], v; tab != -1; tab = nex[tab])
          if(father[u] != (v = son[tab]))
81          maxSubtree[u] = max(maxSubtree[u], size[v]);
      }
83
      int rot = 0;
85    for(int i = 1; i < n; ++i)
        if(maxSubtree[rot] > maxSubtree[i]) rot = i;
87    int anotherRot = -1;
      for(int i = 0; i < n; ++i)
89      if(i != rot && maxSubtree[rot] == maxSubtree[i]) {
```

```
        anotherRot = i;
        break;
      }

    if(anotherRot != -1) {
      int newRot = n++;
      head[newRot] = -1;
      addEdge(newRot, rot), addEdge(newRot, anotherRot);

      for(int tab = head[rot]; tab != -1; tab = nex[tab])
        if(son[tab] == anotherRot) {
          son[tab] = newRot;
          break;
        }
      for(int tab = head[anotherRot]; tab != -1; tab = nex[tab])
        if(son[tab] == rot) {
          son[tab] = newRot;
          break;
        }

      rot = newRot;
    }
    return rot;
  }

  inline int hashTree() {
    int rot = getBarycentre();
    bfs(rot);

    for(int i = n - 1; i >= 0; --i) {
      int u = bfsList[i];

      child[u].clear(), hashCode[u] = 1;
      for(int tab = head[u], v; tab != -1; tab = nex[tab])
        if(father[u] != (v = son[tab])) child[u].push_back(v);
      sort(all(child[u]), cmpByHashCode(hashCode));
      for(int idx = 0; idx < sz(child[u]); ++idx)
        hashCode[u] += factor[idx] * hashCode[child[u][idx]];
```

```
    }
    return rot;
  }
};
unsigned int TreeHash::factor[N];
```

treehash.cpp

countprimes.cpp

```
const int N = 6e6+2;
bool np[N];
int p[N],pi[N];
int getprime(){
    int cnt=0;
    np[0]=np[1]=true;
    pi[0]=pi[1]=0;
    for(int i = 2; i < N; ++i){
        if(!np[i]) p[++cnt] = i;
        for(int j = 1;j <= cnt && i * p[j] < N; ++j) {
            np[i * p[j]] = true;
        }
        pi[i]=cnt;
    }
    return cnt;
}
const int M = 7;
const int PM = 2*3*5*7*11*13*17;
int phi[PM+1][M+1],sz[M+1];
void init(){
    getprime();
    sz[0]=1;
    for(int i=0;i<=PM;++i)  phi[i][0]=i;
    for(int i=1;i<=M;++i){
        sz[i]=p[i]*sz[i-1];
```

```
        for(int j=1;j<=PM;++j){
            phi[j][i]=phi[j][i-1]-phi[j/p[i]][i-1];
        }
    }
}
int sqrt2(LL x){
    LL r = (LL)sqrt(x-0.1);
    while(r*r<=x)   ++r;
    return int(r-1);
}
int sqrt3(LL x){
    LL r = (LL)cbrt(x-0.1);
    while(r*r*r<=x)   ++r;
    return int(r-1);
}
LL Phi(LL x,int s){
    if(s == 0)  return x;
    if(s <= M)  return phi[x%sz[s]][s]+(x/sz[s])*phi[sz[s]][s];
    if(x <= p[s]*p[s])   return pi[x]-s+1;
    if(x <= p[s]*p[s]*p[s] && x< N){
        int s2x = pi[sqrt2(x)];
        LL ans = pi[x]-(s2x+s-2)*(s2x-s+1)/2;
        for(int i=s+1;i<=s2x;++i){
            ans += pi[x/p[i]];
        }
        return ans;
    }
    return Phi(x,s-1)-Phi(x/p[s],s-1);
}
LL Pi(LL x){
    if(x < N)   return pi[x];
    LL ans = Phi(x,pi[sqrt3(x)])+pi[sqrt3(x)]-1;
    for(int i=pi[sqrt3(x)]+1,ed=pi[sqrt2(x)];i<=ed;++i){
        ans -= Pi(x/p[i])-i+1;
    }
    return ans;
}
int main(){
```

```
    init();
    LL n;
    while(scanf("%lld",&n)!=EOF) {
        printf("%lld\n",Pi(n));
    }
    return 0;
}
```

countprimes.cpp

superpower.cpp

```
inline int superPower(int k, int p) {
  /**
   * 1. k^X mod p = k^(X mod phi(p) + phi(p)) mod p  X = k^k^k.....
   * 2. fastpow(x, y, z) x^y mod z
   * */
  if(p == 1) return 0;
  int powers = superPower(k, phi[p]) + phi[p];
  return fastpow(k, powers, p);
}
```

superpower.cpp