DinicWithScaling

```
#define pb push_back

struct Dinic{
    struct edge{
        int to, flow, cap;
    };

    const static int N = 555; //count of vertices

    vector<edge> e;
    vector<int> g[N + 7];
    int dp[N + 7];
    int ptr[N + 7];

    void clear(){
        for (int i = 0; i < N + 7; i++) g[i].clear();
        e.clear();
    }

    void addEdge(int a, int b, int cap){
        g[a].pb(e.size());
        e.pb({b, 0, cap});
        g[b].pb(e.size());
        e.pb({a, 0, 0});
    }

    int minFlow, start, finish;

    bool bfs(){
        for (int i = 0; i < N; i++) dp[i] = -1;
        dp[start] = 0;
        vector<int> st;
        int uk = 0;
        st.pb(start);
        while(uk < st.size()){
            int v = st[uk++];
            for (int to : g[v]){
                auto ed = e[to];
                if (ed.cap - ed.flow >= minFlow && dp[ed.to] == -1){
                    dp[ed.to] = dp[v] + 1;
                    st.pb(ed.to);
                }
            }
        }
        return dp[finish] != -1;
    }

    int dfs(int v, int flow){
        if (v == finish) return flow;
        for (; ptr[v] < g[v].size(); ptr[v]++){
            int to = g[v][ptr[v]];
            edge ed = e[to];
            if (ed.cap - ed.flow >= minFlow && dp[ed.to] == dp[v] + 1){
                int add = dfs(ed.to, min(flow, ed.cap - ed.flow));
                if (add){
                    e[to].flow += add;
                    e[to ^ 1].flow -= add;
                    return add;
                }
            }
        }
```

```
        return 0;
    }

    int dinic(int start, int finish){
        Dinic::start = start;
        Dinic::finish = finish;
        int flow = 0;
        for (minFlow = (1 << 30); minFlow; minFlow >>= 1){
            while(bfs()){
                for (int i = 0; i < N; i++) ptr[i] = 0;
                while(int now = dfs(start, (int)2e9 + 7)) flow += now;
            }
        }
        return flow;
    }
} dinic;
```

FFT

```
#define db long double

class cn{
public:
        db x, y;
        cn(){}
        cn(db xx, db yy): x(xx), y(yy) {}
        cn(db xx): x(xx), y(0) {}
        db real() { return x; }
        void operator /= (double f) { x /= f; y /= f; }
};

cn operator + (cn a, cn b) { return cn(a.x + b.x, a.y + b.y); }
cn operator - (cn a, cn b) { return cn(a.x - b.x, a.y - b.y); }
cn operator * (cn a, cn b) { return cn(a.x * b.x - a.y * b.y, a.x * b.y + a.y * b.x);
}

class FFT{
public:
        constexpr const static db pi = acos(-1.0);
        const static int MAX_SIZE = 1 << 21;

        //#define cn complex<db>

        int n;
        cn a[MAX_SIZE * 2 + 7], b[MAX_SIZE * 2 + 7];

        int getReverse(int a, int k){
                int ans = 0;
                for (int i = 0; i < k; i++) if ((a >> i) & 1) ans ^= (1 << (k - i -
1));
                return ans;
        }

        void fft(cn *a, int type){
                int k = -1;
                for (int i = 0; i < 25; i++) if ((n >> i) & 1){
                        k = i;
                        break;
                }
                for (int i = 0; i < n; i++){
                        int j = getReverse(i, k);
                        if (i < j) swap(a[i], a[j]);
                }
```

```
                for (int len = 2; len <= n; len *= 2){
                        cn w(cos(2 * pi / (db)len), sin(2 * pi / (db)len) * type);
                        for (int i = 0; i < n; i += len){
                                cn g = cn(1, 0);
                                for (int j = 0; j < len / 2; j++){
                                        cn x = a[i + j];
                                        cn y = a[i + j + len / 2] * g;
                                        a[i + j] = x + y;
                                        a[i + j + len / 2] = x - y;
                                        g = g * w;
                                }
                        }
                }
                if (type == -1) for (int i = 0; i < n; i++) a[i] /= n;
        }

        vector<int> mult(vector<int> &w1, vector<int> &w2){
                n = 1;
                while(n < w1.size() + w2.size()) n *= 2;
                for (int i = 0; i < w1.size(); i++) a[i] = w1[i];
                for (int i = 0; i < w2.size(); i++) b[i] = w2[i];
                for (int i = w1.size(); i < n; i++) a[i] = 0;
                for (int i = w2.size(); i < n; i++) b[i] = 0;
                fft(a, 1);
                fft(b, 1);
                for (int i = 0; i < n; i++) a[i] = a[i] * b[i];
                fft(a, -1);
                vector<int> ans(n);
                for (int i = 0; i < n; i++) ans[i] = floor((db)a[i].real()
                 + 0.5);
                while(ans.size() && ans.back() == 0) ans.pop_back();
                return ans;
        }
};
```

FlowCirculation

```
#define pb push_back

struct Dinic{
    struct edge{
        int to, flow, cap;
    };

    const static int N = 555; //count of vertices

    vector<edge> e;
    vector<int> g[N + 7];
    int dp[N + 7];
    int ptr[N + 7];

    void clear(){
        for (int i = 0; i < N + 7; i++) g[i].clear();
        e.clear();
    }

    void addEdge(int a, int b, int cap){
                g[a].pb(e.size());
                e.pb({b, 0, cap});
        g[b].pb(e.size());
        e.pb({a, 0, 0});
    }
```

```
    void addCircular(int a, int b, int l, int r) {
        addEdge(S, b, l); //S - source
        addEdge(a, T, l); //T - sink
        addEdge(a, b, r - l);
    }

    int minFlow, start, finish;

    bool bfs(){
        for (int i = 0; i < N; i++) dp[i] = -1;
        dp[start] = 0;
        vector<int> st;
        int uk = 0;
        st.pb(start);
        while(uk < st.size()){
            int v = st[uk++];
            for (int to : g[v]){
                auto ed = e[to];
                if (ed.cap - ed.flow >= minFlow && dp[ed.to] == -1){
                    dp[ed.to] = dp[v] + 1;
                    st.pb(ed.to);
                }
            }
        }
        return dp[finish] != -1;
    }

    int dfs(int v, int flow){
        if (v == finish) return flow;
        for (; ptr[v] < g[v].size(); ptr[v]++){
            int to = g[v][ptr[v]];
            edge ed = e[to];
            if (ed.cap - ed.flow >= minFlow && dp[ed.to] == dp[v] + 1){
                int add = dfs(ed.to, min(flow, ed.cap - ed.flow));
                if (add){
                    e[to].flow += add;
                    e[to ^ 1].flow -= add;
                    return add;
                }
            }
        }
        return 0;
    }

    int dinic(int start, int finish){
        Dinic::start = start;
        Dinic::finish = finish;
        int flow = 0;
        for (minFlow = (1 << 30); minFlow; minFlow >>= 1){
            while(bfs()){
                for (int i = 0; i < N; i++) ptr[i] = 0;
                while(int now = dfs(start, (int)2e9 + 7)) flow += now;
            }
        }
        return flow;
    }
} dinic;
```

NTT

```
class NTT{
public:
        #define db long double
```

```
        #define ll long long
        const static int mod = 998244353;
        const static int root = 646; // 646^(2^20) == 1 (998244353)
        const static int rev_root = 208611436;
        const static int MAX_SIZE = 1 << 21;

        void add(int &a, int b){
                a += b;
                if (a < 0) a += mod;
                if (a >= mod) a -= mod;
        }

        int sum(int a, int b){
                add(a, b);
                return a;
        }

        int mult(int a, int b){
                return a * (ll)b % mod;
        }

        int bp(int a, int k){
                if (k == 0) return 1;
                if (k & 1){
                        return mult(a, bp(a, k - 1));
                } else {
                        int q = bp(a, k >> 1);
                        return mult(q, q);
                }
        }

        int rev(int a){
                return bp(a, mod - 2);
        }

        int n;
        int a[MAX_SIZE * 2 + 7], b[MAX_SIZE * 2 + 7];

        int getReverse(int a, int k){
                int ans = 0;
                for (int i = 0; i < k; i++) if ((a >> i) & 1) ans ^= (1 << (k - i -
1));
                return ans;
        }

        void ntt(int *a, int type){
                int k = -1;
                for (int i = 0; i < 25; i++) if ((n >> i) & 1){
                        k = i;
                        break;
                }
                for (int i = 0; i < n; i++){
                        int j = getReverse(i, k);
                        if (i < j) swap(a[i], a[j]);
                }
                for (int len = 2; len <= n; len *= 2){
                        int w = bp(root, (1 << 20) / len);
                        if (type == -1) w = bp(rev_root, (1 << 20) / len);
                        for (int i = 0; i < n; i += len){
                                int g = 1;
                                for (int j = 0; j < len / 2; j++){
                                        int x = a[i + j];
                                        int y = mult(a[i + j + len / 2], g);
                                        a[i + j] = sum(x, y);
                                        a[i + j + len / 2] = sum(x, mod - y);
                                        g = mult(g, w);
```

```
                                }
                        }
                }
                if (type == -1){
                        int rev_n = rev(n);
                        for (int i = 0; i < n; i++) a[i] = mult(a[i], rev_n);
                }
        }

        vector<int> mult(vector<int> &w1, vector<int> &w2){
                n = 1;
                while(n < w1.size() + w2.size()) n *= 2;
                for (int i = 0; i < w1.size(); i++){
                        a[i] = w1[i];
                        a[i] %= mod;
                        if (a[i] < 0) a[i] += mod;
                }
                for (int i = 0; i < w2.size(); i++){
                        b[i] = w2[i];
                        b[i] %= mod;
                        if (b[i] < 0) b[i] += mod;
                }
                for (int i = w1.size(); i < n; i++) a[i] = 0;
                for (int i = w2.size(); i < n; i++) b[i] = 0;
                ntt(a, 1);
                ntt(b, 1);
                for (int i = 0; i < n; i++) a[i] = mult(a[i], b[i]);
                ntt(a, -1);
                vector<int> ans(n);
                for (int i = 0; i < n; i++) ans[i] = a[i];
                while(ans.size() && ans.back() == 0) ans.pop_back();
                return ans;
        }
};
```

DimasFlows2

```
#include <iostream>
#include <vector>
#include <algorithm>
#include <stdio.h>
#include <sstream>
#include <string>
#include <map>
#include <set>
#include <stdlib.h>
#include <cmath>
#include <math.h>
#include <fstream>
#include <bitset>
#include <time.h>
#include <queue>
#include <cassert>
#include <cstdio>
#define int long long
using namespace std;
struct Edge{int go; int c; int f; int e_cost;};
int n, m, ai, bi, ci;
int parlament_cost;
int number;
vector<int> potentials, d, relax_edge, relax_vertex;
vector<Edge> edges;
vector<vector<int> > houses, bombs, data;
```

```
int K = 1e10;
int INF = 1e15;
void ford_bellman(){
    for (int i=0; i < number; i++){
        d[i] = INF;
    }
    d[0] = 0;
    for (int it=0; it < number; it++){
        for (int i=0; i < number; i++){
            for (int j=0; j < data[i].size(); j++){
                int ed = data[i][j];
                if (edges[ed].f == edges[ed].c) continue;
                if (d[edges[ed].go] > d[i] + edges[ed].e_cost){
                    d[edges[ed].go] = d[i] + edges[ed].e_cost;
                }
            }
        }
    }
}
bool dijkstra(){
    set<pair<int, int> > vertexes;
    for (int i=1; i < number; i++){
        d[i] = INF;
        vertexes.insert(make_pair(INF, i));
    }
    vertexes.insert(make_pair(0, 0));
    d[0] = 0;
    for (int i=0; i < number; i++){
        pair<int, int> p = *vertexes.begin();
        int v = p.second;
        d[v] = p.first;
        vertexes.erase(vertexes.begin());
        for (int j=0; j < data[v].size(); j++){
            int ed = data[v][j];
            if (edges[ed].c == edges[ed].f) continue;
            int when = edges[ed].go;
            int new_d = d[v] + edges[ed].e_cost + potentials[v] - potentials[when];
            if (new_d < d[when]){
                set<pair<int, int> >::iterator it =
vertexes.upper_bound(make_pair(d[when], when-1));
                vertexes.erase(it);
                d[when] = new_d;
                relax_edge[when] = ed;
                relax_vertex[when] = v;
                vertexes.insert(make_pair(d[when], when));
            }
        }
    }
    if (d[number-1] >= INF) return false;
    vector<int> relax_line;
    int nv = number - 1;
    int fl = INF;
    while (nv != 0){
        fl = min(fl, edges[relax_edge[nv]].c - edges[relax_edge[nv]].f);
        relax_line.push_back(relax_edge[nv]);
        nv = relax_vertex[nv];
    }
    for (int i=0; i < relax_line.size(); i++){
        edges[relax_line[i]].f += fl;
        edges[relax_line[i]^1].f -= fl;
    }
    return true;
}
void add_edge(int first, int second, int capacity, int now_cost){
    Edge e1, e2;
    e1 = {second, capacity, 0, now_cost};
```

```
    e2 = {first, 0, 0, -now_cost};
    edges.push_back(e1);
    edges.push_back(e2);
    data[first].push_back(edges.size() - 2);
    data[second].push_back(edges.size() - 1);
}
signed main()
{
    int number;
    for (int i=0; i < n+m+2; i++){
        relax_edge.push_back(0);
        relax_vertex.push_back(0);
        d.push_back(0);
        vector<int> help;
        potentials.push_back(0);
        data.push_back(help);
    }
    ford_bellman();
    while (true){
        for (int i=0; i < number; i++){
            potentials[i] += d[i];
        }
        bool result = dijkstra();
        if (!result) break;
    }
    return 0;
}
```

DimasFlows

```
#include <bits/stdc++.h>
#define int long long
using namespace std;
struct Edge{int go; int c; int f;};
vector<int> where, d, vert;
vector<Edge> edges;
vector<bool> used;
vector<vector<int> > data;
int number, m;
queue<int> q;
int INF = 1e15;
void construct_edge(int u, int v, int c){
    Edge e1 = {v, c, 0};
        Edge e2 = {u, 0, 0};
    edges.push_back(e1);
        edges.push_back(e2);
        data[u].push_back(edges.size() - 2);
        data[v].push_back(edges.size() - 1);
}
int dfs(int vertex, int flow, int maximum){
    if (vertex == number - 1) return flow;
    while (where[vertex] < data[vertex].size()){
        int i = where[vertex];
                int edge_number = data[vertex][i];
        int to = edges[edge_number].go;
        int can = min(edges[edge_number].c - edges[edge_number].f, flow);
        if (can < maximum || d[to] != d[vertex] + 1) {
            where[vertex]++;
            continue;
        }
        int fl = dfs(to, can, maximum);
        if (fl >= maximum){
            edges[edge_number].f += fl;
```

```
            edges[edge_number^1].f -= fl;
            return fl;
        }
        where[vertex]++;
    }
    return 0;
}
void bfs(int maximum){
    while (!q.empty()){
        int vertex = q.front();
        q.pop();
        for (int i=0; i < data[vertex].size(); i++){
                        int edge_number = data[vertex][i];
            int nv = edges[edge_number].go;
            int can = edges[edge_number].c - edges[edge_number].f;
            if (d[nv] == -1 && can >= maximum){
                d[nv] = d[vertex] + 1;
                q.push(nv);
            }
        }
    }
}
void DFS(int vertex){
    used[vertex] = true;
    vert.push_back(vertex);
    for (int i=0; i < data[vertex].size(); i++){
        int e = data[vertex][i];
        if (edges[e].f == edges[e].c) continue;
        if (used[edges[e].go]) continue;
        DFS(edges[e].go);
    }
}
int dinic(){
    int A = 1LL << 60;
    while (A > 0){
        while (true){
            for (int i=0; i < number; i++){
                where[i] = 0;
                d[i] = -1;
            }
            d[0] = 0;
            q.push(0);
            bfs(A);
            if (d[number-1] == -1) break;
            while (true){
                int flow = dfs(0, INF, A);
                if (flow < A) break;
            }
        }
        A /= 2;
    }
}
```

DynamicConvexHullTrick

```
#define ALL(c) (c).begin(),(c).end()
#define IN(x,c) (find(c.begin(),c.end(),x) != (c).end())
#define REP(i,n) for (int i=0;i<(int)(n);i++)
#define FOR(i,a,b) for (int i=(a);i<=(b);i++)
#define INIT(a,v) memset(a,v,sizeof(a))
#define SORT_UNIQUE(c) (sort(c.begin(),c.end()),
c.resize(distance(c.begin(),unique(c.begin(),c.end()))))
template<class A, class B> A cvt(B x) { stringstream ss; ss<<x; A y; ss>>y; return y;
```

```
}

typedef pair<int,int> PII;
typedef long long int64;

#define N 100000

int n;
int64 h[N],w[N];

int64 sqr(int64 x) { return x*x; }

struct line {
        char type;
        double x;
        int64 k, n;
};

bool operator<(line l1, line l2) {
        if (l1.type+l2.type>0) return l1.x<l2.x;
        else return l1.k>l2.k;
}

set<line> env;
typedef set<line>::iterator sit;

bool hasPrev(sit it) { return it!=env.begin(); }
bool hasNext(sit it) { return it!=env.end() && next(it)!=env.end(); }

double intersect(sit it1, sit it2) {
        return (double)(it1->n-it2->n)/(it2->k-it1->k);
}

void calcX(sit it) {
        if (hasPrev(it)) {
                line l = *it;
                l.x = intersect(prev(it), it);
                env.insert(env.erase(it), l);
        }
}

bool irrelevant(sit it) {
        if (hasNext(it) && next(it)->n <= it->n) return true; // x=0 cutoff //useless
        return hasPrev(it) && hasNext(it) && intersect(prev(it),next(it)) <=
intersect(prev(it),it);
}

void add(int64 k, int64 a) {
        sit it;
        // handle collinear line
        it=env.lower_bound({0,0,k,a});
        if (it!=env.end() && it->k==k) {
                if (it->n <= a) return;
                else env.erase(it);
        }
        // erase irrelevant lines
        it=env.insert({0,0,k,a}).first;
        if (irrelevant(it)) { env.erase(it); return; }
        while (hasPrev(it) && irrelevant(prev(it))) env.erase(prev(it));
        while (hasNext(it) && irrelevant(next(it))) env.erase(next(it));
        // recalc left intersection points
        if (hasNext(it)) calcX(next(it));
        calcX(it);
}

int64 query(int64 x) {
```

```
        auto it = env.upper_bound((line){1,(double)x,0,0});
        it--;
        return it->n+x*it->k;
}

int64 g[N];

int64 solve() {
        int64 a=0;
        REP (i,n) a+=w[i];
        g[0]=-w[0];
        FOR (i,1,n-1) {
                add(-2*h[i-1],g[i-1]+sqr(h[i-1]));
                int64 opt=query(h[i]);
                g[i]=sqr(h[i])-w[i]+opt;
        }
        return a+g[n-1];
}
```

FASTIO

```
1. /** Interface */
2.
3. inline int readChar();
4. template <class T = int> inline T readInt();
5. template <class T> inline void writeInt( T x, char end = 0 );
6. inline void writeChar( int x );
7. inline void writeWord( const char *s );
8.
9. /** Read */
10.
11. static const int buf_size = 4096;
12.
13. inline int getChar() {
14.     static char buf[buf_size];
15.     static int len = 0, pos = 0;
16.     if (pos == len)
17.         pos = 0, len = fread(buf, 1, buf_size, stdin);
18.     if (pos == len)
19.         return -1;
20.     return buf[pos++];
21. }
22.
23. inline int readChar() {
24.     int c = getChar();
25.     while (c <= 32)
26.         c = getChar();
27.     return c;
28. }
29.
30. template <class T>
31. inline T readInt() {
32.     int s = 1, c = readChar();
33.     T x = 0;
34.     if (c == '-')
35.         s = -1, c = getChar();
36.     while ('0' <= c && c <= '9')
37.         x = x * 10 + c - '0', c = getChar();
38.     return s == 1 ? x : -x;
39. }
40.
41. /** Write */
42.
```

```
43. static int write_pos = 0;
44. static char write_buf[buf_size];
45.
46. inline void writeChar( int x ) {
47.     if (write_pos == buf_size)
48.         fwrite(write_buf, 1, buf_size, stdout), write_pos = 0;
49.     write_buf[write_pos++] = x;
50. }
51.
52. template <class T>
53. inline void writeInt( T x, char end ) {
54.     if (x < 0)
55.         writeChar('-'), x = -x;
56.
57.     char s[24];
58.     int n = 0;
59.     while (x || !n)
60.         s[n++] = '0' + x % 10, x /= 10;
61.     while (n--)
62.         writeChar(s[n]);
63.     if (end)
64.         writeChar(end);
65. }
66.
67. inline void writeWord( const char *s ) {
68.     while (*s)
69.         writeChar(*s++);
70. }
71.
72. struct Flusher {
73.     ~Flusher() {
74.         if (write_pos)
75.             fwrite(write_buf, 1, write_pos, stdout), write_pos = 0;
76.     }
77. } flusher;
78.
79. /** Example */
```

HalfplaneIntersection

```
#define ld double
struct point{
        ld x, y;
        point() {}
        point(ld x1, ld y1) { x = x1, y = y1; }
        ld operator% (point nxt) const { return x * nxt.y - y * nxt.x; }
        ld operator* (point nxt) const { return x * nxt.x + y * nxt.y; }
        point operator- (point nxt) const { return point(x - nxt.x, y - nxt.y); }
        point operator+ (point nxt) const { return point(x + nxt.x, y + nxt.y); }
};
struct line{
        ld a, b, c;
        point s, t;
        line() {}
        line(point s1, point t1){
                s = s1, t = t1;
                a = t.y - s.y;
                b = s.x - t.x;
                c = (t.x - s.x) * s.y - s.x * (t.y - s.y);
                if ((t - s) % point(a, b) < 0){
                        a = -a, b = -b, c = -c;
                }
        }
```

```
};
const ld BOX = 1e18;
const ld pi = acos(-1.0);
bool equal(point s, point t){
        return (s % t) == 0 && (s * t) > 0;
}
bool cmp(line s, line t){
        if (equal(s.t - s.s, t.t - t.s)){
                if (abs(s.s.x) == BOX) return 0;
                if (abs(t.s.x) == BOX) return 1;
                return (s.t - s.s) % (t.s - s.s) < 0;
        }
        ld val1 = atan2(s.b, s.a);
        ld val2 = atan2(t.b, t.a);
        if (val1 < 0) val1 += pi * 2;
        if (val2 < 0) val2 += pi * 2;
        return val1 < val2;
}

point crossLineLine(line s, line t){
        ld x = (t.c * s.b - s.c * t.b) / (s.a * t.b - s.b * t.a);
        ld y = (t.c * s.a - s.c * t.a) / (s.b * t.a - t.b * s.a);
        return point(x, y);
}
void halfplanesIntersection(vector<line> a){
        //==========BOX=================
        a.pub(line(point(-BOX, -BOX), point(BOX, -BOX)));
        a.pub(line(point(-BOX, BOX), point(-BOX, -BOX)));
        a.pub(line(point(BOX, -BOX), point(BOX, BOX)));
        a.pub(line(point(BOX, BOX), point(-BOX, BOX)));
        //==============================
        sort(all(a), cmp);
        vector<line> q;
        for (int i = 0; i < a.size(); i++){
                if (i == 0 || !equal(a[i].t - a[i].s, a[i - 1].t - a[i - 1].s))
q.pub(a[i]);
        }
        //for (auto c : q){
        //      cout << "Line " << fixed << c.a << ' ' << c.b << ' ' << c.c << endl;
        //}
        vector<int> st;
        for (int it = 0; it < 2; it++){
                for (int i = 0; i < q.size(); i++){
                        while(st.size() > 1){
                                int j = st.back(), k = st[(int)st.size() - 2];
                                if (((q[i].t - q[i].s) % (q[j].t - q[j].s)) == 0)
break;
                                auto pt = crossLineLine(q[i], q[j]);
                                if ((q[k].t - q[k].s) % (pt - q[k].s) > 0) break;
                                st.pop_back();
                        }
                        st.pub(i);
                }
        }
    vector<int> was((int)a.size(), -1);
    bool ok = 0;
    for (int i = 0; i < st.size(); i++){
        int uk = st[i];
        if (was[uk] == -1){
                was[uk] = i;
        } else {
                st = vector<int>(st.begin() + was[uk], st.begin() + i);
                ok = 1;
                break;
        }
    }
```

```
    if (!ok){
                cout << "Impossible", exit(0);
    }
    point ans = point(0, 0);
    for (int i = 0; i < st.size(); i++){
        line l1 = q[st[i]], l2 = q[st[(i + 1) % (int)st.size()]];
        ans = ans + crossLineLine(l1, l2);
    }
    ans.x /= (ld)st.size();
    ans.y /= (ld)st.size();
    for (int i = 0; i < a.size(); i++){
        line l = a[i];
        if ((l.t - l.s) % (ans - l.s) <= 0) cout << "Impossible", exit(0);
    }
    cout << "Possible\n";
    cout.precision(10);
    cout << fixed << ans.x << ' ' << ans.y;
}
```

Hungarian

```
int n, ai;
int matrix[300][300];
vector<int> column_p, string_p, where, minv, strv, where_string;
vector<bool> see;
int INF = 1e15;
int32_t main()
{
    ios_base::sync_with_stdio(false);
    cin >> n;
    for (int i=0; i < n; i++){
        column_p.push_back(0);
        string_p.push_back(0);
        where.push_back(-1);
        where_string.push_back(-1);
        minv.push_back(-1);
        strv.push_back(-1);
        see.push_back(true);
        for (int j=0; j < n; j++){
            cin >> ai;
            matrix[i][j] = ai;
        }
    }
    for (int it=0; it < n; it++){
        vector<int> strings, columns;
        int now_string = it;
        fill(see.begin(), see.end(), true);
        fill(minv.begin(), minv.end(), INF);
        while (true){
            int minimum = INF;
            int mincol = -1;
            strings.push_back(now_string);
            for (int i=0; i < see.size(); i++){
                if (see[i]){
                    if (minv[i] > matrix[now_string][i] - string_p[now_string] -
column_p[i]){
                        minv[i] = matrix[now_string][i] - string_p[now_string] -
column_p[i];
                        strv[i] = now_string;
                    }
                    if (minv[i] < minimum){
                        minimum = minv[i];
                        mincol = i;
```

```
                    }
                }
            }
            for (int i=0; i < strings.size(); i++){
                string_p[strings[i]] += minimum;
            }
            for (int i=0; i < columns.size(); i++){
                column_p[columns[i]] -= minimum;
            }
            for (int i=0; i < n; i++){
                minv[i] -= minimum;
            }
            if (where[mincol] == -1){
                int nc = mincol;
                int str = strv[mincol];
                while (where_string[str] != -1){
                    int col = where_string[str];
                    where[nc] = str;
                    where_string[str] = nc;
                    str = strv[col];
                    nc = col;
                }
                where_string[str] = nc;
                where[nc] = str;
                break;
            }
            else{
                now_string = where[mincol];
                columns.push_back(mincol);
                see[mincol] = false;
            }
        }
    }
    int cost = 0;
    for (int i=0; i < n; i++){
        cost += string_p[i] + column_p[i];
    }
    cout << cost << endl;
    for (int i=0; i < n; i++){
        cout << i + 1 << " " << where_string[i] + 1 << endl;
    }
    return 0;
}
```

MinCostMaxFlow

```
#include <bits/stdc++.h>

using namespace std;

typedef long long ll;
#define mp make_pair
#define pub push_back
#define x first
#define y second
#define all(a) a.begin(), a.end()
#define db double

const int INF = (int)1e9 + 7;

struct edge{
        int to, cap, flow, cost, num;
};
```

```
int sz = 0;
edge e[222222];
vector<int> g[22222];

void addEdge(int v, int to, int cap, int cost, int num){
        g[v].pub(sz);
        e[sz++] = edge{to, cap, 0, cost, num};
        g[to].pub(sz);
        e[sz++] = edge{v, 0, 0, -cost, num};
}


int fb[22222];
pair<int, int> pred[22222];

ll minCostFlow(int needFlow, int start, int finish){
        ll ans = 0;

        while(needFlow){
                for (int i = 0; i < 22222; i++) fb[i] = INF, pred[i] = mp(-1, -1);
                fb[start] = 0;
                vector<int> st;
                int uk = 0;
                st.pub(start);
                while(uk < st.size()){
                        int v = st[uk++];
                        for (int to : g[v]){
                                auto ed = e[to];
                                if (ed.flow < ed.cap && fb[ed.to] > fb[v] + ed.cost){
                                        pred[ed.to] = mp(v, to);
                                        fb[ed.to] = fb[v] + ed.cost;
                                        st.pub(ed.to);
                                }
                        }
                }
                if (fb[finish] == INF){
                        cout << -1;
                        exit(0);
                }

                int canNow = needFlow;
                int v = finish;
                while(1){
                        auto now = pred[v];
                        if (now.x == -1) break;
                        canNow = min(canNow, e[now.y].cap - e[now.y].flow);
                        v = now.x;
                }

                ans += fb[finish] * (ll)canNow;
                v = finish;
                while(1){
                        auto now = pred[v];
                        if (now.x == -1) break;
                        e[now.y].flow += canNow;
                        e[now.y ^ 1].flow -= canNow;
                        v = now.x;
                }
                needFlow -= canNow;
        }

        return ans;
}

int n, m, k;
```

```
bool wasEdge[2222222];
vector<int> q;

void returnPath(int v){
        if (v == n - 1) return;
        for (int to : g[v]){
                auto ed = e[to];
                if (ed.flow == 1 && !wasEdge[ed.num]){
                        q.pub(ed.num);
                        wasEdge[ed.num] = 1;
                        returnPath(ed.to);
                        break;
                }
        }
}

int main() {
        cin >> n >> m >> k;
        for (int i = 0; i < m; i++){
                int v1, v2, cc;
                cin >> v1 >> v2 >> cc;
                v1--; v2--;
                addEdge(v1, v2, 1, cc, i + 1);
                addEdge(v2, v1, 1, cc, i + 1);
        }
        ll ans = minCostFlow(k, 0, n - 1);
        cout.precision(10);
        cout << fixed << (double)ans / k << "\n";
        for (int it = 0; it < k; it++){
                q.clear();
                returnPath(0);
                cout << q.size() << ' ';
                for (int x : q) cout << x << ' ';
                cout << "\n";
        }
}
```

PalindromicTree

```
struct vert{
    int len, suf;
    int to[26];
    vert() { for (int i = 0; i < 26; i++) to[i] = -1; len = -1, suf = -1; }
};
struct palindromeTree{
    vert t[5000007];
    int sz, last;
    string s;
    palindromeTree() { sz = 2; last = 1; t[last].suf = 0; t[last].len = 0; }
    int addChar(char c){
        s += c;
        int p = last;
        while(p != -1 && c != s[(int)s.size() - t[p].len - 2]) p = t[p].suf;

        if (t[p].to[c - 'a'] == -1){
            int now = sz++;
            last = now;
            t[p].to[c - 'a'] = now;
            t[now].len = t[p].len + 2;
            do p = t[p].suf; while(p != -1 && c != s[(int)s.size() - t[p].len - 2]);
            if (p == -1) t[now].suf = 1;
            else t[now].suf = t[p].to[c - 'a'];
            return 1;
```

```
        } else {
            last = t[p].to[c - 'a'];
            return 0;
        }
    }
} t;

int main() {
    string s;
    cin >> s;
    for (int i = 0; i < s.size(); i++){
        cout << t.addChar(s[i]);
    }
}
```

SuffixAutomata

```
using namespace std;
const int K = 2*String_size + 1;
int counter;
int go[K][26];
int last;
int suf[K], len[K];
void add(int number){
    int newlast = counter; len[newlast] = len[last] + 1; int p = last; counter++;
    while (p!=-1 && go[p][number] == -1){
        go[p][number] = newlast;
        p = suf[p];
    }
    if (p == -1){
        suf[newlast] = 0;
    }
    else{
        int q = go[p][number];
        if (len[q] == len[p] + 1){
            suf[newlast] = q;
        }
        else{
            int r = counter; counter ++;
            for (int i=0;i<26;i++){
                go[r][i] = go[q][i];
            }
            suf[r] = suf[q];
            suf[q] = r;
            suf[newlast] = r;
            len[r] = len[p] + 1;
            while (p!=-1 && go[p][number] == q){
                go[p][number] = r;
                p = suf[p];
            }
        }
    }
    last = newlast;
}
int32_t main()
{
    string s;
        cin >> s;
        for (int i=0; i < K; i++){
                suf[i] = -1;
                len[i] = -1;
                for (int j=0; j < 26; j++){
                        go[i][j] = -1;
```

```
                }
        }
        len[0] = -1;
        last = 0;
        counter = 1;
        for (int i=0; i < s.size(); i++){
                add(s[i] - 'a');
        }
    return 0;
}
```

Sufmas

```
#include <bits/stdc++.h>
#define ll long long
using namespace std;
vector <ll> construct(string &s) {
    s += (char) ('a' - 1);
    ll n = s.size();
    vector <ll> suffs(n, 0), classes(n, 0);
    vector <ll> cnt(Q, 0);
    ll last = Q;
    for (ll i = 0; i < n; i++) {
        classes[i] = s[i] - 'a' + 3;
        cnt[classes[i]]++;
    }
    for (ll i = 1; i < Q; i++) {
        cnt[i] += cnt[i - 1];
    }
    for (ll i = 0; i < n; i++) {
        ll w = s[i] - 'a' + 3;
        suffs[cnt[w - 1]++] = i;
    }
    cnt.clear();
    last = 0;
    for (ll i = 0; i < n; i++) {
        if (!i || s[suffs[i - 1]] != s[suffs[i]]) {
            last++;
        }
        classes[suffs[i]] = last;
    }
    cnt.resize(last + 1, 0);
    ll len = 1;
    while (len < n) {
        for (ll i = 0; i < n; i++) {
            cnt[classes[i]]++;
        }
        for (ll i = 1; i <= last; i++) {
            cnt[i] += cnt[i - 1];
        }
        vector <ll> suffs1(n, 0);
        for (ll i = 0; i < n; i++) {
            ll j = (suffs[i] - len + n) % n;
            suffs1[cnt[classes[j] - 1]++] = j;
        }
        suffs = suffs1;
        ll last1 = 0;
        vector <ll> classes1(n, 0);
        for (ll i = 0; i < n; i++) {
            if (!i) {
                last1++;
            } else {
                ll w1 = classes[suffs[i - 1]], w2 = classes[suffs[i]];
```

```
                ll d1 = classes[(suffs[i - 1] + len) % n], d2 = classes[(suffs[i] +
len) % n];
                if (w1 != w2 || d1 != d2) {
                    last1++;
                }
            }
            classes1[suffs[i]] = last1;
        }
        cnt.clear();
        cnt.resize(last1 + 1, 0);
        last = last1;
        classes = classes1;
        len *= 2;
    }
    return suffs;
}

vector <ll> build_lcp(string s, vector <ll> suff) {
    ll n = suff.size();
    vector <ll> rsuff(n, -1);
    for (ll i = 0; i < n; i++) {
        rsuff[suff[i]] = i;
    }
    vector <ll> lcp(n, 0);
    ll pos = rsuff[0];
    assert(pos);
    ll k = suff[pos - 1];
    while (k + lcp[pos] < n && s[lcp[pos]] == s[k + lcp[pos]]) {
        lcp[pos]++;
    }
    for (ll i = 1; i < n - 1; i++) {
        ll q = rsuff[i];
        ll p = rsuff[i - 1];
        lcp[q] = max(lcp[p] - 1, 0LL);
        ll k = suff[q - 1];
        while (max(k, i) + lcp[q] < n && s[k + lcp[q]] == s[i + lcp[q]]) {
            lcp[q]++;
        }
    }
    return lcp;
}
```

XorConvolution

```
#include <bits/stdc++.h>
#define int long long
using namespace std;
const int K = 1<<17;
vector<int> hadamard(vector<int> v){
    for (int step=K; step > 1; step /= 2){
        for (int start=0; start < K; start += step){
            for (int w=0; w < step/2; w++){
                int F = v[start+w] + v[start+step/2+w];
                int S = v[start+w] - v[start+step/2+w];
                v[start + w] = F;
                v[start+step/2+w] = S;
            }
        }
    }
    return v;
}
signed main() {
    ios_base::sync_with_stdio(false);
```

```
    cin.tie(NULL);
    vector<int> f((1<<K)), g((1<<K));
    f = hadamard(f);
    g = hadamard(g);
    for (int i=0; i < K; i++) f[i] *= g[i];
    f = hadamard(f);
    for (int i=0; i < K; i++) f[i] /= K;
    return 0;
}
```

Mincut

```
/*
Для наиболее простой и ясной реализации (с асимптотикой O(n^3)) было выбрано
представление графа в виде матрицы смежности. Ответ хранится в переменных \rm
best\_cost и \rm best\_cut (искомые стоимость минимального разреза и сами вершины,
содержащиеся в нём).

Для каждой вершины в массиве \rm exist хранится, существует ли она, или она была
объединена с какой-то другой вершиной. В списке {\rm v}[i] для каждой сжатой вершины
i хранятся номера исходных вершин, которые были сжаты в эту вершину i.

Алгоритм состоит из n-1 фазы (цикл по переменной \rm ph). На каждой фазе сначала все
вершины находятся вне множества A, для чего массив \rm in\_a заполняется нулями, и
связности w всех вершин нулевые. На каждой из n-{\rm ph} итерации находится вершина
\rm sel с наибольшей величиной w. Если это итерация последняя, то ответ, если надо,
обновляется, а предпоследняя \rm prev и последняя \rm sel выбранные вершины
объединяются в одну. Если итерация не последняя, то \rm sel добавляется в множество
A, после чего пересчитываются веса всех остальных вершин.

Следует заметить, что алгоритм в ходе своей работы "портит" граф \rm g, поэтому, если
он ещё понадобится позже, надо сохранять его копию перед вызовом функции.
*/


const int MAXN = 500;
int n, g[MAXN][MAXN];
int best_cost = 1000000000;
vector<int> best_cut;

void mincut() {
        vector<int> v[MAXN];
        for (int i=0; i<n; ++i)
                v[i].assign (1, i);
        int w[MAXN];
        bool exist[MAXN], in_a[MAXN];
        memset (exist, true, sizeof exist);
        for (int ph=0; ph<n-1; ++ph) {
                memset (in_a, false, sizeof in_a);
                memset (w, 0, sizeof w);
                for (int it=0, prev; it<n-ph; ++it) {
                        int sel = -1;
                        for (int i=0; i<n; ++i)
                                if (exist[i] && !in_a[i] && (sel == -1 || w[i] >
w[sel]))
                                        sel = i;
                        if (it == n-ph-1) {
                                if (w[sel] < best_cost)
                                        best_cost = w[sel],  best_cut = v[sel];
                                v[prev].insert (v[prev].end(), v[sel].begin(),
v[sel].end());
                                for (int i=0; i<n; ++i)
                                        g[prev][i] = g[i][prev] += g[sel][i];
```

```
                                exist[sel] = false;
                        }
                        else {
                                in_a[sel] = true;
                                for (int i=0; i<n; ++i)
                                        w[i] += g[sel][i];
                                prev = sel;
                        }
                }
        }
}
```

SumLine

```
//izban
// sum(i=0..n-1) (a+b*i) div m
ll solve(ll n, ll a, ll b, ll m) {
    if (b == 0) return n * (a / m);
    if (a >= m) return n * (a / m) + solve(n, a % m, b, m);
    if (b >= m) return n * (n - 1) / 2 * (b / m) + solve(n, a, b % m, m);
    return solve((a + b * n) / m, (a + b * n) % m, m, b);
}
```