bool cw(const point &a, const point &b, const point &c) {return (b.first - a.first) \* (c.second - a.second) - (b.second - a.second) \* (c.first - a.first) < 0; }

vector<point> **graham**(vector<point> p) {

int n = p.size();

if (n <= 1)

return p;

int k = 0;

sort(p.begin(), p.end());

vector<point> q(n \* 2);

for (int i = 0; i < n; q[k++] = p[i++])

for (; k >= 2 && !cw(q[k - 2], q[k - 1], p[i]); --k);

for (int i = n - 2, t = k; i >= 0; q[k++] = p[i--])

for (; k > t && !cw(q[k - 2], q[k - 1], p[i]); --k);

q.resize(k - 1 - (q[0] == q[1]));

return q;}

class **DSU** {

int ro[MAX\_N] , ra[MAX\_N];

init(int n)

FOR(i , 0 , n) ro[i] = I; ra[i] = 1;

int findRoot(int i)

return (ro[i] == i ? i : (ro[i] = findRoot(ro[i])));

int isSameSet(int x , int y)

return (findRoot(x) == findRoot(y));

void unionSet(int x , int y)

if (!isSameSet(x , y))

int u = findRoot(x);

int v = findRoot(y);

if (ra[u] > ra[v]) ro[v] = u;

else ro[u] = v; if (ra[u] == ra[v]) ra[v]++;

class **fenwicktree** {

int ft[MAX\_N];

init(int n) FOR(i , 1 , n) ft[i] = 0;

int rsq(int b)

int sum = 0;

FOR(; b ; b -= (b & (-b))) sum += ft[b];

return sum;

int RSQ(int a , int b)

return rsq(b) - (a == 1 ? 0 : rsq(a - 1)) ;

void adjust(int k , int v , int n)

for (; k < n ; k += (k & (-k))) ft[k] += v;

**Max matching**

**int cnt = 1;**

bool dfs(int x)

if (visited[x] == cnt)

return 0;

visited[x] = cnt;

REP (j, 0, adj[x].size())

int i = adj[x][j];

if (!Right[i] || dfs(Right[i])){

Left[x] = i;

Right[i] = x;

return 1;

return 0;

int matching()

int run = 1;

while (run)

run = 0;

cnt++;

FOR(i, 1, n){

if (Left[i] == 0 && dfs(i) == true) run = 1;

**Max matching with cost**

#define N 1003

#define f1(i,n) for (int i=1; i<=n; i++)

#define f2(i,a,u) for (int i=0,u; u=a[i]; i++)

int n;

int c[N][N];

int f[N], d[N], Decision[N];

int Assigned[N], Visited[N];

vector<int> Left, Right;

queue<int> qu;

bool assignable(int &a, int b){ if (a == 0) a = b; else return false; return true; }

bool minimize(int &a, int b){ if (a > b) a = b; else return false; return true; }

bool maximize(int &a, int b){ if (a < b) a = b; else return false; return true; }

int bfs\_next(){

while (qu.size()){

int u = qu.front(); qu.pop();

f2(i, Right, v){

if (c[u][v] + f[u] + f[v] == 0)

if (assignable(Visited[v], u)){

if (!Assigned[v]) return v;

Visited[Assigned[v]] = v;

qu.push(Assigned[v]);

}

if (minimize(d[v], c[u][v] + f[u] + f[v])) Decision[v] = u;

}

}

return 0;

}

int bfs\_first(int Start){

while (qu.size()) qu.pop();

f1(i, 2 \* n) Visited[i] = 0;

f2(i, Right, u) { d[u] = c[Start][u] + f[Start] + f[u]; Decision[u] = Start; }

Visited[Start] = -1, qu.push(Start);

return bfs\_next();

}

bool adjust(){

int Delta = 0x11112222;

f2(i, Right, u) if (!Visited[u]) minimize(Delta, d[u]);

if (Delta >= 0x11112222) return false;

assert(Delta != 0);

f2(i, Left, u) if (Visited[u]) f[u] -= Delta;

f2(i, Right, u) if (Visited[u]) f[u] += Delta;

f2(i, Right, u)

if (!Visited[u]) { d[u] -= Delta; if (d[u] == 0) qu.push(Decision[u]); }

return true;

}

void enlarge(int u){

while (u > 0){

int y = u, x = Visited[y];

u = Assigned[x];

Assigned[x] = y;

Assigned[y] = x;

}

}

map<int, int> Map[N];

int force(int Pos, int Used){

int Answer = 0x11112222;

if (Pos == n + 1) return 0;

if (Map[Pos].count(Used)) return Map[Pos][Used];

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

if ((1 << i - 1) & ~Used)

minimize(Answer, force(Pos + 1, Used | (1 << i - 1)) + c[Pos][i + n]);

return Map[Pos][Used] = Answer;

}

main(){

cin >> n;

srand(n \* 1000);

f1(i, n) f1(j, n) {

//cin >> c[i][j+n];

c[i][j + n] = rand() % 100 \* 10;

cout << c[i][j + n] << (j == n ? "\n" : " ");

}

f1(i, n) Left.push\_back(i); Left.push\_back(0);

f1(i, n) Right.push\_back(i + n); Right.push\_back(0);

f2(i, Left, u) {

int x = bfs\_first(u);

while (x == 0){

if (!adjust()) break;

else x = bfs\_next();

}

enlarge(x);

}

int Answer = 0;

f1(i, n) Answer += c[i][Assigned[i]];

cout << Answer << endl;

cout << force(1, 0) << endl;

cin.ignore(2);

}

**KMP**

**// a, b index from 1**

m = strlen(a + 1), n = strlen(b + 1);

Prev[0] = -1;

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

Prev[i] = 0;

for (int j = Prev[i - 1]; j != -1; j = Prev[j])

if (b[j + 1] == b[i]) {

Prev[i] = j + 1;

Break; }

int u = 0;

for (int i = 1; i <= m; i++) {

while (u != 0 && b[u + 1] != a[i]) u = Prev[u];

if (b[u + 1] == a[i]) u++;

if (u == n) printf("%d ", i - n + 1); }

**Table prime**

bool nonpr[N];

int prime[N];

int nPrime;

void eratos(int n)

int i, j;

nonpr[0] = nonpr[1] = true;

for (i = 2; i <= n; i++)

if (!nonpr[i])

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

nonpr[j] = true;

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

if (!nonpr[i]) prime[++nPrime] = i;

ii **extended\_gcd**(long a, long b){

ii q, s;

if (b==0) return ii(1, 0);

q = ii(a/b, a%b);

s = extended\_gcd(b, q.Y);

return ii(s.Y, s.X-q.X\*s.Y);

**Max Flow**

bool bfs()

{ int u, v, i, k;

queue<int> Q;

Q.push(source);

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

while (!Q.empty())

{ u = Q.front();

Q.pop();

k = adj[u].size();

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

{ v = adj[u][i];

if (par[v] == -1 && dist[u][v] > 0) {

par[v] = u; Q.push(v);

if (v == sink) return true;

}}}

return false;

}

int max\_flow()

{ int ret = 0;

while (bfs()) {

flow = INF;

path\_update(sink);

ret += flow;

}}

void path\_update(int v)

{

int u = par[v];

flow = min(flow, dist[u][v]);

if (u != source)

path\_update(u);

dist[u][v] -= flow;

dist[v][u] += flow;

return;

}

**Max\_Flow with minimum distance**

bool minimize(int &a, int b){ if (a > b) a = b; else return false; return true; }

int n, m;

vector<ii> a[12309];

int c[123][123];

int f[123][123];

ii d[12309];

bool inqueue[12309];

bool **fb**(int start, int target) {

queue<int> qu;

int u, i, v;

for (i = 1; i <= n; i++) { inqueue[i] = false; d[i].X = 1000111000; }

qu.push(start); inqueue[start] = true; d[start].X = 0;

while (qu.size()) {

u = qu.front(); qu.pop(); inqueue[u] = false;

for (i = 0; v = a[u][i].Y; i++)

if (c[u][v] > f[u][v])

if (minimize(d[v].X, d[u].X + (f[u][v] >= 0 ?

a[u][i].X : -a[u][i].X))) {

d[v].Y = u;

if (!inqueue[v])

qu.push(v); inqueue[v] = true;

}

}

if (d[target].X < 1000111000) return true;

else return false;

}

int **enlarge**(int start, int target, int delta, int &answer) {

int i;

for (i = target; i != start; i = d[i].Y)

if (f[d[i].Y][i] < 0)

minimize(delta, -f[d[i].Y][i]);

else

minimize(delta, c[d[i].Y][i] - f[d[i].Y][i]);

for (i = target; i != start; i = d[i].Y) {

f[d[i].Y][i] += delta;

f[i][d[i].Y] -= delta; }

answer += delta \* d[target].X;

return delta;

}

**main**() {

int i, j, p, q, w;

int r = 0, capacity = 0;

while (scanf("%d%d", &n, &m) > 0) {

for (i = 1; i <= n; i++) a[i].clear();

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

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

f[i][j] = c[i][j] = 0;

for (i = 1; i <= m; i++) {

scanf("%d%d%d", &p, &q, &w);

a[p].push\_back(ii(w, q));

a[q].push\_back(ii(w, p));

c[p][q] += 1;

c[q][p] += 1; }

for (i = 1; i <= n; i++) a[i].push\_back(ii());

scanf("%d%d", &capacity, &p);

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

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

c[i][j] \*= p;

r = 0;

while (fb(1, n))

if (capacity == 0) break;

else capacity -= enlarge(1, n, capacity, r);

if (capacity != 0) printf("Impossible.\n");

else printf("%d\n", r);

} }

**Prim**

int n, m, d[N];

vector<int> a[N], b[N];

int prim(int u) {

int Sum = 0;

priority\_queue<ii> qu;

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

qu.push(ii(0, u)); d[u] = 0;

while (qu.size()) {

ii Pop = qu.top(); qu.pop();

int u = Pop.second, du = -Pop.first;

if (du != d[u]) continue;

Sum += d[u]; d[u] = 0;

for (int i = 0; int v = a[u][i]; i++)

if (d[v] > b[u][i]) {

d[v] = b[u][i];

qu.push(ii(-d[v], v));}

} return Sum; }

main() {

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

for (int i = 1; i <= m; i++) {

int x, y, z;

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

a[x].push\_back(y);

b[x].push\_back(z);

a[y].push\_back(x);

b[y].push\_back(z);

}

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

a[i].push\_back(0);

cout << prim(1) << endl;

}

**Suffix Array**

const int N = 200005;

int n, sa[N], ra[N], rb[N], G;

char a[N];

bool cmp(int x, int y) {

if (ra[x] != ra[y]) return ra[x] < ra[y];

return ra[x + G] < ra[y + G];

}

main() {

scanf("%s", a + 1);

n = strlen(a + 1);

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

{

sa[i] = i; ra[i] = a[i];

}

for (G = 1; G <= n; G \*= 2) {

sort(sa + 1, sa + n + 1, cmp);

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

rb[sa[i]] = rb[sa[i - 1]] + cmp(sa[i - 1], sa[i]);

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

ra[i] = rb[i];

if (ra[sa[n]] == n) break;

}

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

printf("%d\n", sa[i] - 1);

}

**Z Function**

#define N 500005

int n, z[N];

char a[N];

void make\_z(char a[], int n, int F[]) {

int L = -1, R = -1; F[0] = n;

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

if (i > R) {

L = i; R = i - 1;

while (R < n - 1 && a[R + 1] == a[R - L + 1]) R++;

F[i] = R - L + 1;

} else if (F[i - L] < R - i + 1) F[i] = F[i - L];

else {

L = i;

while (R < n - 1 && a[R + 1] == a[R - L + 1]) R++;

F[i] = R - L + 1;

}}}}

main(){

gets(a); n = strlen(a);

make\_z(a, n, z);

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

printf("%d ", z[i]);

printf("\n");

}

**Convex hull and size**

struct Point {

long long x, y;

bool operator < (const Point &v) const { return x == v.x ? y < v.y : x < v.x; }

long long cross(const Point &p, const Point &q) const { return (p.x - x) \* (q.y - y) - (p.y - y) \* (q.x - x); }

} p[N], poly[N];

int n;

void enter() {

scanf("%d", &n);

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

scanf("%lld%lld", &p[i].x, &p[i].y);

}

long long size(Point poly[], int k) {

long long S = (poly[k - 1].x - poly[0].x) \* (poly[k - 1].y + poly[0].y);

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

S += (poly[i - 1].x - poly[i].x) \* (poly[i - 1].y + poly[i].y);

return S;

printf("%lld\n", S);

}

void solve() {

sort(p, p + n); int k = 0;

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

while (k >= 2 && poly[k - 2].cross(poly[k - 1], p[i]) <= 0) --k;

poly[k++] = p[i];

}

for (int i = n - 2, t = k + 1; i >= 0; --i) {

while (k >= t && poly[k - 2].cross(poly[k - 1], p[i]) <= 0) --k;

poly[k++] = p[i];

}

printf("%lld\n", size(poly, k));

}

int main() { enter(); solve(); return 0; }

**Circumscribed circle**

typedef pair<double, double> point;

typedef pair<point, double> circle;

#define X first

#define Y second

point operator + (point a, point b) { return point(a.X + b.X, a.Y + b.Y); }

point operator - (point a, point b) { return point(a.X - b.X, a.Y - b.Y); }

point operator / (point a, double x) { return point(a.X / x, a.Y / x); }

double abs(point a) { return sqrt(a.X\*a.X + a.Y\*a.Y); }

point center\_from(double bx, double by, double cx, double cy) {

double B = bx\*bx + by\*by, C = cx\*cx + cy\*cy, D = bx\*cy - by\*cx;

return point((cy\*B - by\*C) / (2 \* D), (bx\*C - cx\*B) / (2 \* D));

}

circle circle\_from(point A, point B, point C) {

point I = center\_from(B.X - A.X, B.Y - A.Y, C.X - A.X, C.Y - A.Y);

return circle(I + A, abs(I));

}

const int N = 100005;

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

point a[N];

circle f(int n, vector<point> T) {

if (T.size() == 3 || n == 0) {

if (T.size() == 0) return circle(point(0, 0), -1);

if (T.size() == 1) return circle(T[0], 0);

if (T.size() == 2) return circle((T[0] + T[1]) / 2, abs(T[0] - T[1]) / 2);

return circle\_from(T[0], T[1], T[2]);

}

random\_shuffle(a + 1, a + n + 1);

circle Result = f(0, T);

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

if (abs(Result.X - a[i]) > Result.Y + 1e-9) {

T.push\_back(a[i]);

Result = f(i - 1, T);

T.pop\_back();

}

return Result;

}

main() {

scanf("%d", &n);

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

scanf("%d%d", &x[i], &y[i]);

a[i] = point(x[i], y[i]);

}

circle C = f(n, vector<point>());

(cout << fixed).precision(2);

cout << 2 \* C.Y << endl;

}

**Chặt tam phân**

LL=minX, RR=maxX;

ML=(LL+LL+RR)/3, MR=(LL+RR+RR)/3;

while (LL!=ML) and (ML!=MR) and (MR!=RR) do

if f(ML)>f(MR) then LL=ML;

else RR=MR:

ML=(LL+LL+RR)/3, MR=(LL+RR+RR)/3;

print (LL+RR)/2;

**Check Bipartie Graph**

#define f1(i,n) for (int i=1; i<=n; i++)

int n, m;

vector<int> a[12309];

bool Invalid = false;

bool Visited[12309];

int Color[12309];

void visit(int u) {

Visited[u] = true;

for (int i = 0, v; v = a[u][i]; i++){

if (!Visited[v]){

Color[v] = 3 - Color[u];

visit(v); }

else {

if (Color[v] == Color[u])

Invalid = true;

}}}

main(){

cin >> n >> m;

f1(i, m) {

int p, q;

cin >> p >> q;

a[p].push\_back(q);

a[q].push\_back(p);

}

f1(i, n) a[i].push\_back(0);

f1(i, n){

if (!Visited[i]){

Color[i] = 1; visit(i);

}

}

if (Invalid) cout << "Invalid" << endl;

else {

f1(i, n) cout << Color[i] << " "; cout << endl;

}

}

**Segment Tree**

int T[10230997];

int n, m;

int max(int a, int b){ return a > b ? a : b; }

int query(int node, int ll, int rr, int u, int v){

if (v<ll or rr<u or ll>rr) { return -1000111000; }

if (u <= ll and ll <= rr and rr <= v) return T[node];

return max(

query(node \* 2, ll, (ll + rr) / 2, u, v),

query(node \* 2 + 1, (ll + rr) / 2 + 1, rr, u, v)

);

}

void update(int node, int ll, int rr, int i, int v){

if (i<ll or i>rr or ll>rr) return;

if (ll != rr){

update(node \* 2, ll, (ll + rr) / 2, i, v);

update(node \* 2 + 1, (ll + rr) / 2 + 1, rr, i, v);

T[node] = max(T[node \* 2], T[node \* 2 + 1]);

}

else T[node] = v;

}

main(){

int p, q; char c;

init();

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

while (m--){

scanf(" %c%d%d", &c, &p, &q);

if (c == 'u' or c == 'U') update(1, 1, n, p, q);

if (c == 'q' or c == 'Q') printf("%d\n", query(1, 1, n, p, q));

}

}