Notebook/vimrc

```
set nocp ai si noet ts=4 sw=4 sta sm nu ru mouse=a
set tf lz hls is ls=2 ch=2 list lcs=tab:>\ ,trail:. t_Co=256 spell
colo molokai
syn on
imap {<CR> {<CR>}<Esc>O




" nocp      nocompatible (helps fix things)
" ai        autoindent
" si        smartindent
" noet      noexpandtab (don't make tabs into spaces)
" ts=4      tabstop (number of spaces that a tab counts for)
" sw=4      shiftwidth (equivalent number of spaces per indent)
" sta       smarttab
" sm        showmatch
" nu        numbering
" ru        ruler
" rnu       relativenumber
" mouse=a   (click with mouse)
" tf        ttyfast (optimization)
" lz        lazyredraw (optimization)
" confirm   (before quit)####################### Not in use
" hls       hlsearch (highlight)
" is        incsearch
" ls=2      laststatus
" ch=2      cmdheight (cmdbar height)
" list      allow formatting of tabs/trailing space
" lsc       listchars (replace with)
" syn       syntax on
```

Notebook/Header_Large.cpp

```cpp
#include <iostream>
#include <vector>
#include <queue>
#include <stack>
#include <unordered_map>
#include <algorithm>

using namespace std;

typedef long long ll;
typedef unsigned long long ull;
typedef vector<unordered_map<ull, ll> > graph;
```

Notebook/String/kmp.cpp

```cpp
#include <string>
#include <vector>

using namespace std;

vector<int> kmp(string &s, string &p) {
    if (p.empty()) return {};

    vector<int> lps(p.size());
    vector<int> ans;
    for (int i = 1, k = 0; i < p.size(); ++i) {
        while (k > 0 && p[k] != p[i]) k = lps[k-1];
        if (p[k] == p[i]) ++k;
        lps[i] = k;
    }
    for (int i = 0, k = 0; i < s.size(); ++i) {
        while (k > 0 && p[k] != s[i]) k = lps[k-1];
        if (p[k] == s[i]) ++k;
        if (k == p.size()) {
            ans.push_back(i-k+1);
            k = lps[k-1];
        }
    }
    return ans;
}
```

Notebook/String/trie.cpp

```
#include <map>
#include <string>
#include <vector>

using namespace std;

struct trie {
    struct node {
        map<char, int> children;
        int leaf;
        node() = default;
    };

    vector<node> nodes;

    trie() {
        nodes.push_back(node());
    }

    ~trie() = default;

    void insert(string &s) {
        int cur = 0; // root
        for (char c : s) {
            if (!nodes[cur].children.count(c)) {
                nodes[cur].children[c] = nodes.size();
                nodes.push_back(node());
            }
            cur = nodes[cur].children[c];
        }
        if (!nodes[cur].children.count('$')) {
            nodes[cur].children['$'] = nodes.size();
            nodes.push_back(node());
        }
        ++nodes[nodes[cur].children['$']].leaf;
    }

    bool contains(string &s) {
        int cur = 0;
        for (char c : s) {
            if (!nodes[cur].children.count(c)) return false;
            cur = nodes[cur].children[c];
        }
        return nodes[cur].children.count('$');
    }
};
```

Notebook/String/suffixarray.cpp

```
#include <string>
#include <vector>
#include <cmath>
```

```cpp
#include <algorithm>

using namespace std;

struct suffix {
    int rank[2];
    int index;
};

bool cmp(suffix a, suffix b) {
    return a.rank[0] == b.rank[0] ? a.rank[1] < b.rank[1] : a.rank[0] < b.rank[0];
}

struct sarray {
    vector<int> idx, sarr, lcp;
    vector<suffix> suf;
    string t;
    int n;

    sarray(int sz) : idx(sz), suf(sz), sarr(sz), lcp(sz) {}

    void build(string &s) {
        t = s;
        n = s.size();
        for (int i = 0; i < n; ++i) {
            suf[i].index = i;
            suf[i].rank[0] = s[i] - ' ';
            suf[i].rank[1] = (i+1) < n ? s[i+1] - ' ' : -1;
        }
        sort(suf.begin(), suf.begin() + n, cmp);
        for (int len = 4; len < 2*n; len *= 2) {
            int rank = 0, prev_rank = suf[0].rank[0];
            suf[0].rank[0] = rank;
            idx[suf[0].index] = 0;

            for (int i = 1; i < n; ++i) {
                if (suf[i].rank[0] == prev_rank && suf[i].rank[1] == suf[i-1].rank[1
]) {
                    suf[i].rank[0] = rank;
                } else {
                    prev_rank = suf[i].rank[0];
                    suf[i].rank[0] = ++rank;
                }
                idx[suf[i].index] = i;
            }

            for (int i = 0; i < n; ++i) {
                int next = suf[i].index + len/2;
                suf[i].rank[1] = next < n ? suf[idx[next]].rank[0] : -1;
            }
            sort(suf.begin(), suf.begin()+n, cmp);
        }

        for (int i = 0; i < n; ++i) {
            sarr[i] = suf[i].index;
        }
    }

    void build_lcp() {
        fill(lcp.begin(), lcp.begin()+n, 0);
        vector<int> inv(n);
        for (int i = 0; i < n; ++i) {
            inv[sarr[i]] = i;
        }

        int len = 0;
        for (int i = 0; i < n; ++i) {
            if (inv[i] == n-1) {
                len = 0;
                continue;
            }
            int j = sarr[inv[i]+1];
            while (i+len < n && j+len < n && t[i+len] == t[j+len]) ++len;
            lcp[inv[i]] = len;
            if (len > 0) --len;
        }
    }

    int& operator[](int pos) {
        return sarr[pos];
    }
};
```

Notebook/Body_Large.cpp

```
int main() {
    int n; cin >> n;
//  ios_base::sync_with_stdio(false);

//  vector<int> v; v.reserve(n);
//  copy_n(istream_iterator<int>(cin), n, back_inserter(v));

    while(n --> 0) {
        //CODE
    }
}
```

Notebook/reminders

```
* Use long long instead of int
* Use setprecision
* Communicate *all* thoughts (even the trivial ones)
* Explore test cases for patterns
```

Notebook/Dynamic_Programming/Longest_Subseq/subseq.cpp

```
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
typedef pair<ll,ll> pll;

//O(n^2)
int lis(vector<ll>& v){
    vector<int> lis(v.size(), 1);

    for(int i=1; i<v.size(); ++i) for(int j=0; j<i; ++j)
        if(v[j] < v[i] && lis[i] < lis[j] + 1) lis[i] = lis[j] + 1;

    int max_len = 0;
    for(int l : lis) if(max_len < l) max_len = l;
    return max_len;
}

//O(nlogn)
int lis2(vector<ll> v){
    if(v.empty()) return 0;
    vector<int> lis(v.size(), 0);

    int last = 1;
    lis[0] = v[0];
    for(int i=1; i<v.size(); ++i){
        if(v[i] < l[0]) lis[0] = v[i];
        else if(lis[last-1] < v[i]) lis[last++] = v[i];
        else *upper_bound(lis.begin(), lis.begin()+last, v[i]) = v[i];
    }
    return last;
}
```

```
int main(){
    int n; cin >> n;
    vector<ll> v(n);

    for(int i=0; i<n; ++i) cin >> v[i];

    cout << "1: " << lis (v) << endl;
    cout << "2: " << lis2(v) << endl;
}
```

Notebook/Dynamic_Programming/Coin_Change/coinchange.cpp

```
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;

// Note: Required that v is sorted
ll count_ways(vector<ll>& v, ll val){
    if(v.empty()) return 0;

    vector<ll> table(++val, 0);
    table[0] = 1;

    for(int i=0; i<v.size(); ++i) for(int j=v[i]; j<val; ++j)
        table[j] += table[j-v[i]];

    return table[val-1];
}
bool can_make_change(vector<ll>& v, ll val){
    for(ll e : v) if((val %= e) == 0) return true;
    return false;
}

int main(){
    int n, x;
    cin >> n >> x;
    vector<ll> v(n);
    for(int i=0; i<n; ++i) cin >> v[i];

    sort(v.begin(), v.end());
    cout << count_ways(v, x) << endl;
}
```

Notebook/Dynamic_Programming/Knapsack/knapsack.cpp

```
/* Standard Knapsack DP. Runs in n*m time. v holds values and w holds weights. Note
that v and w are 0 indexed but our dp is 1-indexed.
 */

#include <vector>
#include <algorithm>

using namespace std;

int knapsack(vector<int> &v, vector<int> &w) {
    int n = v.size(), m = w.size();
    vector<vector<int>> dp(n+1, vector<int>(m+1));
    for (int i = 1; i <= n; ++i) {
        for (int j = 0; j <= m; ++j) {
            if (j-w[i] < 0) dp[i][j] = dp[i-1][j];
            else dp[i][j] = max(dp[i-1][j], dp[i-1][j-w[i-1]] + v[i-1]);
        }
    }
    return dp[n][m];
}
```

Notebook/Dynamic_Programming/Bitset/subsets_by_size.cpp

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

using namespace std;

inline int next_bit_perm(int v) {
    int t = v | (v - 1);
    return (t + 1) | (((~t & -~t) - 1) >> (__builtin_ctz(v) + 1));
}

void iterate(int n) {
    for (int k = 0; k <= n; ++k) {
        for (int w = (1<<k)-1; w < (1<<n); w = next_bit_perm(w)) {
            // do stuff
        }
    }
}
```

Notebook/Body_Small.cpp

```cpp
int main() {
    int n; cin >> n;
    while(n --> 0) {
        //CODE
    }
}
```

Notebook/Binary_Data/Fenwick_Tree/fenwick.cpp

```cpp
#include <vector>

using namespace std;

struct fenwick {
    vector<int> tree;

    fenwick(int size) : tree(size+1, 0) {}

    void add(int pos, int val) {
        ++pos;
        while (pos < tree.size()) {
            tree[pos] += val;
            pos += (pos & -pos);
        }
    }

    int get(int pos) {
        ++pos;
        int ans = 0;
        while (pos > 0) {
            ans += tree[pos];
            pos -= (pos & -pos);
        }
        return ans;
    }
};
```

Notebook/Binary_Data/Union_Find/union_find.cpp

```
#include <vector>
using namespace std;

struct disjoint_set{
    vector<int> parent;

    disjoint_set(int n) : parent(n) {
        for (int i = 0; i < n; ++i) {
            parent[i] = i;
        }
    }

    int root(int pos) {
        int tmp = pos;
        while (parent[tmp] != tmp) {
            tmp = parent[tmp];
        }
        while (pos != tmp) {
            int next = parent[pos];
            parent[pos] = tmp;
            pos = next;
        }
        return tmp;
    }

    bool find(int p, int q) {
        return root(p) == root(q);
    }

    void merge(int p, int q) {
        int rootp = root(p), rootq = root(q);
        parent[rootp] = rootq;
    }
};
```

Notebook/Binary_Data/Union_Find/onionfind.cpp

```
#include <vector>
using namespace std;

template<class T>
struct disjoint_set{
    vector<T> parent, size;

    disjoint_set(size_t n) : parent(n), size(n, 1){
        for(size_t i=0; i<n; ++i) parent[i] = i;
    }
    T root(T pos){
        while(parent[pos] != pos) {
            parent[pos] = parent[parent[pos]];
            pos = parent[pos];
        }
        return pos;
    }
    // Flatten the set
    // WARNING: once this is done, sets can't be separated
    T squish_root(T pos){
        if(parent[pos] == pos) return pos;
        return parent[pos] = squish_root(parent[pos]);
    }
    // Check if two elements are in the same set
    bool find(T p, T q){
        return root(p) == root(q);
    }
    void onion(T p, T q) {
        T rootp = root(p), rootq = root(q);
        if(size[rootp] > size[rootq]){
            parent[rootq] = rootp;
            size[rootp] += size[rootq];
        }
        else{
            parent[rootp] = rootq;
            size[rootq] += size[rootp];
        }
    }
};
```

Notebook/Binary_Data/LCA/lca.cpp

```
#include <vector>
#include <cmath>

using namespace std;

#define MAXLOG 30

struct tree {
    vector<int> parent, level;
    vector<vector<int>> ancestor;

    tree(int n) : parent(n+1, -1), level(n+1, -1), ancestor(n+1, vector<int>(MAXLOG,
 -1)) {
        level[0] = 0;
    }

    void insert(int i, int par) {
        parent[i] = par;
        level[i] = level[par]+1;
        ancestor[i][0] = par;

        for (int j = 1; (1<<j) < parent.size(); ++j) {
            if (ancestor[i][j-1] != -1) ancestor[i][j] = ancestor[ancestor[i][j-1]][
j-1];
        }
    }

    int lca(int u, int v) {
        if (level[u] < level[v]) swap(u, v);

        int dist = level[u] - level[v];
        while (dist > 0) {
            int rb = log2(dist);
            u = ancestor[u][rb];
            dist -= (1 << rb);
        }

        if (u == v) return u;

        for (int j = log2(parent.size()); j >= 0; --j) {
            if (ancestor[u][j] != -1 && ancestor[u][j] != ancestor[v][j]) {
                u = ancestor[u][j];
                v = ancestor[v][j];
            }
        }

        return parent[u];
    }
};
```

Notebook/Binary_Data/Range_Tree/Templated_Large.cpp

```
#include <functional>

ull MSB(ull x) {
    x |= x >> 0x01;
    x |= x >> 0x02;
    x |= x >> 0x04;
```

```cpp
    x |= x >> 0x08;
    x |= x >> 0x10;
    x |= x >> 0x20;
    return (x >> 1) + 1;
}

template<typename T> struct rtree {
    vector<T> v; T z; function<T(T, T)> c;
    rtree(ull n, function<T(T, T)> m, T x = T()) : v(MSB(n - 1) << 2, x), c(m), z(x)
 {}
    // Set the value at i to x and update the tree
    void set(ull i, T x) {
        for(i += v.size()/2; i; i /= 2) {
            v[i] = x;
            x = c(v[i & ~1], v[i | 1]);
        }
    }
    // Return reference to the value at i
    T& raw(ull i) {
        return v[i + v.size()/2];
    }
    // Update the whole range tree
    void update() {
        for(ull i = v.size()/2 - 1; i; --i) {
            v[i] = c(v[2*i], v[2*i + 1]);
        }
    }
    T get0(ull b, ull s, ull e, ull i, ull j) {
        if(s == i && e == j) return v[b];
        ull m = (s + e)/2;
        if(j <= m) return get0(2*b, s, m, i, j);
        if(i >= m) return get0(2*b + 1, m, e, i, j);
        return c(get0(2*b, s, m, i, m), get0(2*b + 1, m, e, m, j));
    }
    // Return the sum between [i, j)
    T get(ull i, ull j) {
        if(i >= j) return z;
        return get0(1, 0, v.size()/2, i, j);
    }
};

struct irtree : rtree<ll> {
    irtree(ull n) : rtree<ll>(n, [](auto x, auto y) {return x + y;}, 0) {}
};
```

Notebook/Geometry/geom.h

```cpp
#include <vector>
#include <algorithm>

using namespace std;

#define BCK(v, i) ((v)[(v).size() - (i)])

typedef long long ll;

struct pnt {
    ll x, y;
    pnt(ll x = 0, ll y = 0) : x(x), y(y) {}
    pnt operator-() {return pnt(-x, -y);}
    pnt operator+(pnt o) {return pnt(x + o.x, y + o.y);}
    pnt operator-(pnt o) {return *this + (-o);}
    ll sqMag() {return x*x + y*y;}
};

typedef vector<pnt> vpt;

ll cross(pnt a, pnt b) {return a.x*b.y - b.x*a.y;}
ll dot(pnt a, pnt b) {return a.x*b.x + a.y*b.y;}
```

```
bool cnvaCmp(pnt a, pnt b) {
    ll c = cross(a, b);
    return (c == 0) ? (a.sqMag() < b.sqMag()) : (c > 0);
}
bool cartCmp(pnt a, pnt b) {
    return (a.x == b.x) ? (a.y < b.y) : (a.x < b.x);
}

vpt hull(vpt& pts) {
    if(pts.size() < 4) return vpt(pts);
    pnt pvt = pts[0];
    for(pnt pt : pts) if(cartCmp(pt, pvt)) pvt = pt;
    for(pnt& pt : pts) pt = pt - pvt;
    sort(pts.begin(), pts.end(), cnvaCmp);
    for(pnt& pt : pts) pt = pt + pvt;
    vpt h;
    for(pnt pt : pts) {
        while(h.size() > 1 && cross(pt - BCK(h, 1), BCK(h, 1) - BCK(h, 2)) >= 0) h.p
op_back();
        h.push_back(pt);
    }
    return h;
}

//returns 2*area, oriented counterclockwise
ll area2(vpt& poly) {
    pnt opt = BCK(poly, 1);
    ll a = 0;
    for(pnt pt : poly) {
        a += cross(a, opt);
        opt = pt;
    }
    return a;
}
```

Notebook/Geometry/Convex_Hull/Graham_Large.cpp

```
vpt hull(vpt& pts) {
    if(pts.size() < 4) return vpt(pts);
    pnt pvt = pts[0];
    for(pnt pt : pts) if(cartCmp(pt, pvt)) pvt = pt;
    for(pnt& pt : pts) pt = pt - pvt;
    sort(pts.begin(), pts.end(), cnvaCmp);
    for(pnt& pt : pts) pt = pt + pvt;
    vpt h;
    for(pnt pt : pts) {
        while(h.size() > 1 && cross(pt - BCK(h, 1), BCK(h, 1) - BCK(h, 2)) >= 0) h.p
op_back();
        h.push_back(pt);
    }
    return h;
}
```

Notebook/Geometry/Convex_Hull/TopBottom_Large.cpp

```
#include <vector>
#include <iostream>
using namespace std;
typedef long double ld;
#define inf numeric_limits<ld>::max()

struct Point{
    ld x, y;
    friend bool operator==(Point& a, Point& b){
        return a.x == b.x && a.y == b.y;
    }
    ld slope(Point& b){
        return (b.y - y)/(b.x - x);
    }
};
//returns c_hull edge from [it, end)
template<class It> vector<Point> half_hull(It it, It end){
    vector<Point> hull;
    hull.push_back(*it);
    while(it != end && it->x == hull[0].x) ++it;
    if(it == end) return hull;
    hull.push_back(*it);

    while(++it != end){
        if(it->x == hull.back().x) continue;
        while(hull.size() > 1){
            auto back2 = hull[hull.size()-2];
            if(back2.slope(*it) > back2.slope(hull.back())) hull.pop_back();
            else break;
        }
        hull.push_back(*it);
    }
    return hull;
}

vector<Point> convex_hull(vector<Point>& p){
    sort(p.begin(), p.end(), [](Point& a, Point& b){
        return a.x == b.x ? a.y > b.y : a.x < b.x;
    });

    //get two halves of c_hull
    vector<Point> upper = half_hull(p.begin(), --p.end());
    vector<Point> lower = half_hull(p.rbegin(), --p.rend());
    //merge & return
    upper.insert(upper.end(), lower.begin(), lower.end());
    return upper;
}

int main(){
    //vector<Point> pts = {{0,1}, {0,0}, {1,1}, {1,0}};
    vector<Point> pts = {{0,2},{0,0},{2,2,},{2,0},{1,0}};
    vector<Point> c_hull = convex_hull(pts);
    for(Point p : c_hull){
        cout << '(' << p.x << ',' << p.y << ')' << endl;
    }
}
```

Notebook/Geometry/quadtree-problem.cpp

```
#include <iostream>
#include <vector>
#include <queue>

using namespace std;

typedef long long ll;
typedef unsigned long long ull;

struct spnt {
    ull x, y, b;
};

struct reg {
    ull mx, my, Mx, My;
};

typedef vector<spnt> vs;
typedef queue<reg> qr;

bool checkInside(spnt& sp, ull x, ull y) {
    ull dx = x > sp.x ? x - sp.x : sp.x - x;
    ull dy = y > sp.y ? y - sp.y : sp.y - y;
```

```cpp
    return dx*dx*dx + dy*dy*dy <= sp.b;
}

ull outDiff3(ull b, ull e, ull v) {
    --e;
    ull r = 0;
    if(v < b) r = b - v;
    if(v > e) r = v - e;
    return r*r*r;
}

bool inside(ull b, ull e, ull v) {
    return b <= v && v < e;
}

int maxState(int pS, spnt& sp, reg& r) {
    if(pS >= 2) return pS;
    bool c1 = checkInside(sp, r.mx, r.my);
    bool c2 = checkInside(sp, r.mx, r.My - 1);
    bool c3 = checkInside(sp, r.Mx - 1, r.my);
    bool c4 = checkInside(sp, r.Mx - 1, r.My - 1);
    if(c1 && c2 && c3 && c4) return 2;
    if(pS) return pS;
    if(c1 || c2 || c3 || c4) return 1;
    if(inside(r.mx, r.Mx, sp.x) && outDiff3(r.my, r.My, sp.y) <= sp.b) return 1;
    if(inside(r.my, r.My, sp.y) && outDiff3(r.mx, r.Mx, sp.x) <= sp.b) return 1;
    return 0;
}

int main() {
    ull n, k;
    cin >> n >> k;
    vs ps(k);
    while(k --> 0) cin >> ps[k].x >> ps[k].y >> ps[k].b;
    k = ps.size();
    qr qt;
    reg r;
    r.mx = r.my = 0;
    r.Mx = r.My = n + 1;
    qt.push(r);
    ull cnt = 0;
    while(!qt.empty()) {
        r = qt.front();
        qt.pop();
        int s = 0;
        if(r.Mx - r.mx < 2 && r.My - r.my < 2) {
            bool isIn = r.Mx > r.mx && r.My > r.my;
            for(int i = 0; isIn && i < k; ++i)
                isIn = !checkInside(ps[i], r.mx, r.my);
            cnt += isIn;
        } else {
            for(spnt& sp : ps) s = maxState(s, sp, r);
            if(s == 0) cnt += (r.Mx - r.mx)*(r.My - r.my);
            if(s == 1) {
                ull xA, yA;
                reg q1, q2, q3, q4;
                xA = (r.mx + r.Mx)/2;
                yA = (r.my + r.My)/2;
                q1.mx = q3.mx = r.mx;
                q2.Mx = q4.Mx = r.Mx;
                q1.Mx = q2.mx = q3.Mx = q4.mx = xA;
                q1.my = q2.my = r.my;
                q3.My = q4.My = r.My;
                q1.My = q2.My = q3.my = q4.my = yA;
                qt.push(q1);
                qt.push(q2);
                qt.push(q3);
                qt.push(q4);
            }
        }
    }
    cout << cnt << endl;
}
```

Notebook/Geometry/vec2.cpp

```
struct vec2 {
    ll x, y;
    explicit vec2(ll x = 0, ll y = 0) : x(x), y(y) {}
    vec2 operator+(vec2 o) {return vec2(x + o.x, y + o.y);}
    vec2 operator*(ll m) {return vec2(x*m, y*m);}
    vec2 operator-() {return *this * -1;}
    vec2 operator-(vec2 o) {return *this + (-o);}
};
```

Notebook/Number_Theory/Euclidean_Algorithm/extendedEuclid.cpp

```
#include <iostream>
#include <tuple>

using namespace std;

typedef unsigned long long ull;
typedef long long ll;

typedef tuple<ll, ll, ull> tllu;

ull modexp(ull b, ull e, ull m) {
    ull r = 1;
    while(e > 0) {
        if(e % 2 == 1)
            r = r*b % m;
        b = b*b % m;
        e /= 2;
    }
    return r;
}

// a * n + b * m = g = gcd(n, m)
tllu bezout(ull n, ull m) { // return {a, b, g}
    if(m == 0)
        return {1, 0, n};
    ll x, y; ull g;
    tie(x, y, g) = bezout(m, n % m);
    return {y, x - y*(n/m), g};
}

ull modinv(ull a, ull m) {
    ll r = get<0>(bezout(a, m));
    return (r % (ll)m + m) % m;
}

ull gcd(ull a, ull b) {
    return get<2>(bezout(a, b));
}

// r % n = a
// r % m = b
ull crt(ull a, ull b, ull n, ull m) {
    ll x, y; ull M;
    tie(x, y, M) = bezout(n, m);
    ll r = x*n*b + y*m*a;
    M = n*m;
    return (r % (ll)M + M) % M;
}
```

Notebook/Number_Theory/Euclidean_Algorithm/GCD.cpp

```
ull gcd(ull a, ull b) {
    while(b) {
        a %= b;
        swap(a, b);
    }
    return a;
}
```

Notebook/Number_Theory/Euclidean_Algorithm/modinverse.cpp

```
typedef long long ll;

ll modinv(ll a, ll b) {
    ll b0 = b, t, q;
    ll x0 = 0, x1 = 1;
    if (b == 1) return 1;
    while (a > 1) {
        q = a / b;
        t = b, b = a % b, a = t;
        t = x0, x0 = x1 - q * x0, x1 = t;
    }
    if (x1 < 0) x1 += b0;
    return x1;
}
```

Notebook/Number_Theory/Prime_Gen/factor.cpp

```
#include <iostream>
#include <cmath>
#include <map>

using namespace std;

typedef unsigned long long ull;

ull cntFact(ull x, map<ull, ull>& f) {
    ull s = sqrt(x) + 1;
    s = x > s ? s : (x - 1);
    ull c = 1;
    for(; s > 1; --s) if(x % s == 0) {
        c += cntFact(s, f);
        x /= s;
        ull s2 = sqrt(x) + 1;
        s = s > s2 ? s2 : s;
        s = x > s ? s : (x - 1);
        ++s;
    }
    if(f.count(x)) ++f[x];
    else f[x] = 1;
    return c;
}
```

Notebook/Number_Theory/Prime_Gen/sieve.cpp

```
#include <iostream>
#include <vector>

using namespace std;

typedef unsigned long long ull;

void prime_gen(vector<ull>& v, ull mx) {
    vector<bool> sieve(mx + 1);
    for(ull i = 2; i <= mx; ++i) {
        if(sieve[i]) continue;
        v.push_back(i);
        for(ull j = i*i; j <= mx; j += i)
            sieve[j] = true;
    }
}
```

Notebook/Number_Theory/Rational/Rat_Small.cpp

```
ull gcd(ull a, ull b) {
    while(b) {
        a %= b;
        swap(a, b);
    }
    return a;
}

struct rat {
    ll n;
    ull d;
    rat(ll nn = 0, ull dd = 1) : n(nn), d(dd) {
        ull c = gcd(n < 0 ? -n : n, d);
        n /= c;
        d /= c;
    }
    rat operator+(rat o) {return rat(o.d*n + d*o.n, d*o.d);}
    rat operator-() {return rat(-n, d);}
    rat operator-(rat o) {return *this + (-o);}
    rat operator*(rat o) {return rat(n*o.n, d*o.d);}
    rat operator/(rat o) {
        bool sgn = o.n < 0;
        if(sgn) o.n = -o.n;
        return rat(n*o.d*(sgn ? -1 : 1), o.n*d);
    }
    bool operator<(rat o) {return n*o.d < d*o.n;}
};
```

Notebook/Header_Small.cpp

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

using namespace std;

typedef long long ll;
```

Notebook/Graph/SCCs/sccs.cpp

```
#include <bits/stdc++.h>
using namespace std;
typedef unordered_map<int, vector<int>> Graph;
#define pb push_back

Graph transpose(Graph& g){
    Graph t(g.size());
    for(int v=0; v<g.size(); ++v) for(int u : g[v]) t[u].pb(v);
    return t;
}

void dfs(Graph& g, int v, vector<bool>& visited, vector<int>& s){
    visited[v] = true;
    for(int u : g[v]) if(!visited[u]) dfs(g, u, visited, s);
    s.pb(v);
}
```

```cpp
vector<vector<int>> sccs(Graph& g){
    vector<int> s;
    vector<bool> visited(g.size());

    for(int v=0; v<g.size(); ++v)
        if(!visited[v]) dfs(g, v, visited, s);

    Graph t = transpose(g);

    fill(visited.begin(), visited.end(), false);
    vector<vector<int>> sccs;
    while(!s.empty()){
        int v = s.back(); s.pop_back();
        if(!visited[v]){
            vector<int> comp;
            dfs(t, v, visited, comp);
            sccs.pb(comp);
        }
    }
    return sccs;
}
```

Notebook/Graph/SCCs/kosaraju.cpp

```cpp
/* Kosaraju's algorithm - Computes SCCs in O(V+E) time using two DFSs.
 * Returns a DAG of how the SCCs are connected, as well as populating a disjoint set
 with the SCC information.
 */

#include <vector>
#include <map>
#include <stack>

using namespace std;

typedef vector<map<int, int>> graph;

struct disjoint_set{
    vector<int> parent;

    ~disjoint_set() = default;
    disjoint_set() = delete;
    disjoint_set(int n) : parent(n) {
        for (int i = 0; i < n; ++i) {
            parent[i] = i;
        }
    }

    int root(int pos) {
        int tmp = pos;
        while (parent[tmp] != tmp) {
            tmp = parent[tmp];
        }
        while (pos != tmp) {
            int next = parent[pos];
            parent[pos] = tmp;
            pos = next;
        }
        return tmp;
    }

    bool find(int p, int q) {
        return root(p) == root(q);
    }

    void merge(int p, int q) {
        int rootp = root(p), rootq = root(q);
        parent[rootp] = rootq;
    }
};

void dfs(graph &g, stack<int> &s, vector<bool> &vis, int cur) {
    vis[cur] = true;
    for (auto &p : g[cur]) {
        if (!vis[p.first]) dfs(g, s, vis, p.first);
    }
    s.push(cur);
}

void dfs2(graph &g, disjoint_set &ds, vector<bool> &vis, int start, int cur) {
    vis[cur] = true;
    ds.merge(start, cur);
    for (auto &p : g[cur]) {
        if (!vis[p.first]) {
            dfs2(g, ds, vis, start, p.first);
        }
    }
}

graph scc(graph &out, graph &in, disjoint_set &ds) {
    vector<bool> vis(out.size());
    stack<int> s;

    for (int i = 0; i < in.size(); ++i) {
        if (!vis[i]) dfs(in, s, vis, i);
    }

    map<int, int> tag;
    fill(vis.begin(), vis.end(), false);
    int count = 0;
    while (!s.empty()) {
        int cur = s.top(); s.pop();
        if (!vis[cur]) {
            dfs2(out, ds, vis, cur, cur);
            tag[ds.root(cur)] = count;
            ++count;
        }
    }

    graph ans(count);
    for (int i = 0; i < out.size(); ++i) {
        int r1 = ds.root(i);
        for (auto &p : out[i]) {
            int r2 = ds.root(p.first);
            if (r1 != r2) {
                ans[tag[r1]][tag[r2]] = 1;
            }
        }
    }

    return ans;
}
```

Notebook/Graph/Minimum_Spanning_Tree/Prims_Jordans.cpp

```cpp
/**
 * Generate a Minimum Spanning Tree on `g'.
 * Returns the total edge weight of the MST.
 * Preconditions:
 *  parent.size() == g.size();
 *  `g' is bidirectional and connected.
 * Postconditions:
 *  `parent[i]' refers to the parent vertex to `i' in the MST.
 * Complexity:
 *  Time: O(E log E)
 *  Space: O(V + E)
 */
ll PrimsMST(graph& g, vector<ull>& parent) {
    priority_queue<pair<ll, ull>> q;
    vector<ull> vis(g.size());
    ull vert = 0;
    ll w = 0;
    parent[vert] = vert;
    while(true) {
        vis[vert] = true;
        for(auto e : g[vert]) q.push(make_pair(-e.second, e.first));
        while(!q.empty() && vis[q.top().second]) q.pop();
        if(q.empty()) return w;
        auto e = q.top();
        q.pop();
        w -= e.first;
        parent[e.second] = vert;
        vert = e.second;
    }
}
```

Notebook/Graph/Minimum_Spanning_Tree/prim.cpp

```
#include <vector>
#include <queue>
#include <map>

using namespace std;

typedef vector<map<int, int>> graph;

int prim(graph &g) {
    vector<bool> vis(g.size());
    priority_queue<pair<int, int>> pq;
    pq.push({0, 0});
    int ans = 0;
    while (!pq.empty()) {
        auto cur = pq.top(); pq.pop();
        int u, w; tie(w, u) = cur;
        if (vis[u]) continue;
        vis[u] = true;
        ans += -w;
        for (auto adj : g[u]) {
            if (!vis[adj.first]) pq.push({-adj.second, adj.first});
        }
    }
    return ans;
}
```

Notebook/Graph/Topological_Sort/top_sort.cpp

```
/* Toplogical sort algorithm (Kahn's). Takes a graph of in edges and a graph of out_
edges
 * Running time: O(E + V)
 */

#include <vector>
#include <limits>
#include <map>
#include <queue>

using namespace std;

typedef vector<map<int, int>> graph;

bool cycle = false;

vector<int> top_sort(vector<int> &in, graph &out) {
    vector<int> order;
    queue<int> next;
    for (int i = 0; i < in.size(); ++i) {
        if (in[i] == 0) next.push(i);
    }

    while (!next.empty()) {
        int cur = next.front(); next.pop();
        order.push_back(cur);
        for (auto e : out[cur]) {
            int v = e.first;
            --in[v];
            if (in[v] == 0) {
                next.push(v);
            }
        }
    }

    for (int i = 0; i < in.size(); ++i) {
        if (in[i] > 0) {
            cycle = true;
            break;
        }
    }

    return order;
}
```

Notebook/Graph/Max_Flow/mincostmaxflow.cpp

```
#include <vector>
#include <limits>
#include <queue>
#include <cmath>

using namespace std;

struct edge {
    int u, v, cap, cost, flow, orig;
};

typedef vector<vector<int>> graph;

graph g;
vector<edge> edges;

void add_edge(int u, int v, int c, int w) {
    g[u].push_back(edges.size());
    edges.push_back({u, v, c, w, 0, w});
    g[v].push_back(edges.size());
    edges.push_back({v, u, 0, -w, 0, 0});
}

#define INF numeric_limits<int>::max()

// initial potentials using Bellman-Ford
vector<int> potentials(int src) {
    vector<int> pot(g.size(), INF);

    pot[src] = 0;

    for (int i = 0; i < g.size()-1; ++i) {
        for (int u = 0; u < g.size(); ++u) {
            for (int k : g[u]) {
                edge e = edges[k];
                if (pot[u] != INF && e.cap - e.flow > 0 && pot[u] + e.cost < pot[e.v
]) {
                    pot[e.v] = pot[u] + e.cost;
                }
            }
        }
    }
    return pot;
}

// Dijkstra's using cost as distance
bool sssp(vector<int> &dist, vector<int> &parent, int src, int sink) {
    fill(dist.begin(), dist.end(), INF);
    vector<bool> vis(g.size());

    auto cmp = [](pair<int, int> one, pair<int, int> two) { return one.second > two.
second; };
    priority_queue<pair<int, int>, vector<pair<int, int>>, decltype(cmp)> pq(cmp);
    dist[src] = 0;
    parent[src] = -1;
    pq.push(make_pair(src, 0));

    while (!pq.empty()) {
        auto cur = pq.top().first; pq.pop();
        if (vis[cur]) continue;
        vis[cur] = true;
        for (int i : g[cur]) {
            edge e = edges[i];
            if (e.cap - e.flow > 0 && dist[cur] + e.cost < dist[e.v]) {
                dist[e.v] = dist[cur] + e.cost;
                parent[e.v] = i;
                pq.push(make_pair(e.v, dist[e.v]));
            }
        }
    }

    return dist[sink] != INF;
}

// Update "reduced" costs with potentials to avoid negative cycles
void reduce(vector<int> &dist) {
    for (int i = 0; i < edges.size(); ++i) {
        edge &forward = edges[i], &back = edges[i^1];
        if (forward.cap - forward.flow > 0) {
            forward.cost += dist[forward.u] - dist[forward.v];
            back.cost = 0;
        }
    }
}

pair<int, int> min_cost_max_flow(int s, int t) {
    auto dist = potentials(s);
    reduce(dist);
    vector<int> parent(g.size());
    int flow = 0, cost = 0;
    while (sssp(dist, parent, s, t)) {
        reduce(dist);
        int delta = INF;
        for (int cur = t; parent[cur] != -1; cur = edges[parent[cur]].u) {
            edge &e = edges[parent[cur]];
            delta = min(delta, e.cap - e.flow);
        }
        for (int cur = t; parent[cur] != -1; cur = edges[parent[cur]].u) {
            edge &forward = edges[parent[cur]], &back = edges[parent[cur] ^ 1];
            forward.flow += delta;
            back.flow -= delta;
        }
        flow += delta;
    }

    // calculate cost using original costs
    for (int i = 0; i < edges.size(); ++i) {
        edge &e = edges[i];
        cost += e.flow * e.orig;
    }
    return make_pair(flow, cost);
}
```

Notebook/Graph/Max_Flow/pushrelabel.cpp

```cpp
#include <vector>
#include <limits>
#include <map>
#include <queue>

using namespace std;

#define INF numeric_limits<int>::max()

typedef vector<map<int, int>> graph;

int n;
graph g;
vector<int> height, excess;

void push(int u, int v) {
    int d = min(excess[u], g[u][v]);
    g[u][v] -= d;
    g[v][u] += d;
    excess[u] -= d;
    excess[v] += d;
}

void relabel(int u) {
    int d = INF;
    for (auto &p : g[u]) {
        int v, cap;
        tie(v, cap) = p;
        if (cap > 0) d = min(d, height[v]);
    }
    if (d < INF) height[u] = d+1;
}

vector<int> highest(int s, int t) {
    vector<int> ans;
    int h = -1;
    for (int i = 0; i < n; ++i) {
        if (i == s || i == t || excess[i] <= 0) continue;
        if (height[i] > h) {
            ans.clear();
            h = height[i];
        }
        if (height[i] == h) ans.push_back(i);
    }
    return ans;
}

int flow(int s, int t) {
    excess.assign(n, 0);
    excess[s] = INF;
    height.assign(n, 0);
    height[s] = n;
    for (auto &p : g[s]) {
        push(s, p.first);
    }

    vector<int> cur;
    while (!(cur = highest(s, t)).empty()) {
        for (int u : cur) {
            bool pushed = false;
            for (auto &p : g[u]) {
                if (excess[u] == 0) break;
                int v, cap;
                tie(v, cap) = p;
                if (cap > 0 && height[u] == height[v]+1) {
                    push(u, v);
                    pushed = true;
                }
            }
            if (!pushed) {
                relabel(u);
                break;
            }
        }
    }

    return INF - excess[s];
}
```

Notebook/Graph/Max_Flow/dinic.cpp

```
/* Dinic's algorithm - max flow / bipartite matching
 * Running time: O(E*V^2) in general and O(E*sqrt(V)) on bipartite graphs with unit
edge weights
 */

#include <vector>
#include <limits>
#include <map>
#include <queue>

using namespace std;

typedef vector<map<int, int>> graph;
typedef map<int, int>::iterator it;

#define INF numeric_limits<int>::max()

bool bfs(graph &g, vector<int> &level, int s, int t) {
    for (int i = 0; i < g.size(); ++i) {
        level[i] = -1;
    }
    level[s] = 0;

    queue<int> q; q.push(s);

    while (!q.empty()) {
        int u = q.front(); q.pop();
        for (auto e : g[u]) {
            int v = e.first, w = e.second;
            if (level[v] < 0 && w > 0) {
                level[v] = level[u] + 1;
                q.push(v);
            }
        }
    }

    return level[t] >= 0;
}

int dfs(graph &g, vector<int> &level, vector<it> &start, int u, int t, int flow) {
    if (u == t) return flow;

    for (it& e = start[u]; e != g[u].end(); ++e) {
        int v = e->first, w = e->second;
        if (level[v] == level[u]+1 && w > 0) {
            int cur_flow = min(flow, w);
            int temp_flow = dfs(g, level, start, v, t, cur_flow);
            if (temp_flow > 0) {
                g[u][v] -= temp_flow;
                g[v][u] += temp_flow;
                return temp_flow;
            }
        }
    }

    return 0;
}

int dinic(graph &g, int s, int t) {
    int total = 0;
    vector<int> level(g.size());
    vector<it> start(g.size());

    while (bfs(g, level, s, t)) {
        for (int i = 0; i < g.size(); ++i) start[i] = g[i].begin();
        while (int flow = dfs(g, level, start, s, t, INF)) {
            total += flow;
        }
    }

    return total;
}
```

Notebook/Graph/Vertex_Cover/vcover.cpp

```
/* Finds minimum vertex cover in bipartite graph by running Dinic's for maximum matc
hing and then traversing the residual graph in linear time.
 * Running time: O(E*sqrt(V))
 */

#include <vector>
#include <limits>
#include <map>
#include <set>
#include <queue>

using namespace std;

typedef vector<map<int, int>> graph;
typedef map<int, int>::iterator it;

#define INF numeric_limits<int>::max()

bool bfs(graph &g, vector<int> &level, int s, int t) {
    for (int i = 0; i < g.size(); ++i) {
        level[i] = -1;
    }
    level[s] = 0;

    queue<int> q; q.push(s);

    while (!q.empty()) {
        int u = q.front(); q.pop();
        for (auto e : g[u]) {
            int v = e.first, w = e.second;
            if (level[v] < 0 && w > 0) {
                level[v] = level[u] + 1;
                q.push(v);
            }
        }
    }

    return level[t] >= 0;
}

int dfs(graph &g, vector<int> &level, vector<it> &start, int u, int t, int flow) {
    if (u == t) return flow;

    for (it& e = start[u]; e != g[u].end(); ++e) {
        int v = e->first, w = e->second;
        if (level[v] == level[u]+1 && w > 0) {
            int cur_flow = min(flow, w);
            int temp_flow = dfs(g, level, start, v, t, cur_flow);
            if (temp_flow > 0) {
                g[u][v] -= temp_flow;
                g[v][u] += temp_flow;
                return temp_flow;
            }
        }
    }

    return 0;
}

int dinic(graph &g, int s, int t) {
    int total = 0;
    vector<int> level(g.size());
    vector<it> start(g.size());

    while (bfs(g, level, s, t)) {
        for (int i = 0; i < g.size(); ++i) start[i] = g[i].begin();
        while (int flow = dfs(g, level, start, s, t, INF)) {
            total += flow;
        }
    }

    return total;
}

void alternate(graph &g, set<int> &nz, vector<bool> &vis, int cur) {
    vis[cur] = true;
    nz.insert(cur);
    for (auto &p : g[cur]) {
        if (p.second && !vis[p.first]) alternate(g, nz, vis, p.first);
    }
}

set<int> cover(graph &g, int s, int t) {
    dinic(g, s, t);

    set<int> z; // set of unmatched vertices on LHS
    for (int i = 0; i < g.size(); ++i) {
        if (g[s].count(i) && g[s][i]) z.insert(i);
    }

    vector<bool> vis(g.size());
    set<int> nz; // nz = z U {any node reachable via alternating path from a node in
 z}
    for (int i : z) {
        alternate(g, nz, vis, i);
    }

    set<int> cov; // cover = {LHS - nz} U {RHS intersection nz}
    for (int i = 0; i < g.size(); ++i) {
        if (g[s].count(i) && !nz.count(i) || g[i].count(t) && nz.count(i)) {
            cov.insert(i);
        }
    }
    return cov;
}
```

Notebook/Graph/Bipartite_Matching/hopcroft_karp.cpp

```
/* Dinic's algorithm - max flow / bipartite matching
 * Running time: O(E*V^2) in general and O(E*sqrt(V)) on bipartite graphs with unit
edge weights
 */

#include <vector>
#include <limits>
#include <map>
#include <queue>

using namespace std;

typedef vector<map<int, int>> graph;
typedef map<int, int>::iterator it;

#define INF numeric_limits<int>::max()
```

```
bool bfs(graph &g, vector<int> &level, vector<int> &u, vector<int> &pair_u, vector<i
nt> &pair_v) {
    queue<int> q;

    for (int i : u) {
        if (pair_u[i] == -1) {
            level[i] = 0;
            q.push(i);
        } else {
            level[i] = INF;
        }
    }

    while (!q.empty()) {
        int i = q.front(); q.pop();
        if (level[i] < INF) {
            for (auto e : g[i]) {
                int v = e.first;
                if (level[pair_v[v]] == INF) {
                    level[pair_v[v]] = level[i] + 1;
                    q.push(pair_v[v]);
                }
            }
        }
    }

    return level[t] >= 0;
}

int dfs(graph &g, vector<int> &level, vector<it> &start, int u, int t, int flow) {
    if (u == t) return flow;

    for (it& e = start[u]; e != g[u].end(); ++e) {
        int v = e->first, w = e->second;
        if (level[v] == level[u]+1 && w > 0) {
            int cur_flow = min(flow, w);
            int temp_flow = dfs(g, level, start, v, t, cur_flow);
            if (temp_flow > 0) {
                g[u][v] -= temp_flow;
                g[v][u] += temp_flow;
                return temp_flow;
            }
        }
    }

    return 0;
}

int dinic(graph &g, vector<int> &u, vector<int> &v, int s, int t) {
    vector<int> level(g.size());
    vector<int> pair_u(g.size(), -1), pair_v(g.size(), -1);

    int total = 0;
    while (bfs(g, level, s, t)) {
        for (int i : u) {
            if (pair_u[i] == -1 && dfs(g, level, pair_u, pair_v, i)) {
                ++total;
            }
        }
    }

    return total;
}
```

Notebook/Graph/Shortest_Path/floyd_warshall.cpp

```
/* Floyd-Warshall is all-pairs shortest path that *does* handle negative edge weight
s (but not cycles).
 * Running time: O(V^3)
 */

#include <vector>
#include <limits>
#include <map>
#include <queue>

using namespace std;

typedef vector<map<int, int>> graph;

#define INF numeric_limits<int>::max()

vector<vector<int>> dist(graph &g) {
    vector<vector<int>> dist(g.size(), vector<int>(g.size(), INF));

    for (int u = 0; u < g.size(); ++u) {
        dist[u][u] = 0;

        for (auto v : g[u]) {
            dist[u][v.first] = v.second;
        }
    }

    for (int k = 0; k < g.size(); ++k) {
        for (int i = 0; i < g.size(); ++i) {
            for (int j = 0; j < g.size(); ++j) {
                if (dist[i][k] != INF && dist[k][j] != INF && dist[i][j] > dist[i][k
] + dist[k][j]) {
                    dist[i][j] = dist[i][k] + dist[k][j];
                }
            }
        }
    }
    return dist;
}
```

Notebook/Graph/Shortest_Path/bellman_ford.cpp

```
/* Bellman-ford is single-source shortest path that *does* handle negative edge weig
hts.
 * Running time: O(E*V)
 */

#include <vector>
#include <limits>
#include <map>
#include <queue>

using namespace std;

typedef vector<map<int, int>> graph;

#define INF numeric_limits<int>::max()

vector<int> dist(graph &g, int src) {
    vector<int> dist(g.size(), INF);

    dist[src] = 0;

    for (int i = 0; i < g.size()-1; ++i) {
        for (int u = 0; u < g.size(); ++u) {
            for (auto k : g[u]) {
                int v = k.first, w = k.second;
                if (dist[u] != INF && dist[u] + w < dist[v]) {
                    dist[v] = dist[u] + w;
                }
            }
        }
    }
    return dist;
}
```

Notebook/Graph/Shortest_Path/dijkstra.cpp

```
/* Dijkstra's is single-source shortest path that does NOT handle negative edge weig
hts.
 * Running time: O(E*log(E))
 */

#include <vector>
#include <limits>
#include <map>
#include <queue>

using namespace std;

typedef vector<map<int, int>> graph;

#define INF numeric_limits<int>::max()

vector<int> dist(graph &g, int src) {
    vector<int> dist(g.size(), INF);
    vector<bool> vis(g.size());

    auto cmp = [](pair<int, int> one, pair<int, int> two) { return one.second > two.
second; };
    priority_queue<pair<int, int>, vector<pair<int, int>>, decltype(cmp)> pq(cmp);
    dist[src] = 0;
    pq.push(make_pair(src, 0));

    while (!pq.empty()) {
        auto cur = pq.top().first; pq.pop();
        if (vis[cur]) continue;
        vis[cur] = true;
        for (auto adj : g[cur]) {
            if (dist[cur] + adj.second < dist[adj.first]) {
                dist[adj.first] = dist[cur] + adj.second;
                pq.push(make_pair(adj.first, dist[adj.first]));
            }
        }
    }

    return dist;
}
```

Notebook/Graph/Miscellaneous/euler_tour_undirected.cpp

```
/* Hierholzer's algorithm finds an Euler circuit in an undirected ayclic graph.
 * Running time: O(V + E)
 * Input constraints: Every vertex in g must have an even degree.
 */

#include <vector>
#include <algorithm>
#include <map>

using namespace std;

typedef vector<map<int, int>> graph;

vector<int> euler(graph& g) {
    vector<int> circuit;
    vector<int> cur_path = {0};
    while (!cur_path.empty()) {
        int cur = *cur_path.rbegin();
        if (g[cur].empty()) {
            circuit.push_back(cur);
            cur_path.pop_back();
        } else {
            auto p = *g[cur].begin();
            g[cur].erase(p.first);
            g[p.first].erase(cur);
            cur_path.push_back(p.first);
        }
    }
    reverse(circuit.begin(), circuit.end());
    return circuit;
}
```

Notebook/Graph/Miscellaneous/euler_tour_directed.cpp

```
/* Hierholzer's algorithm finds an Euler circuit in a DAG.
 * Running time: O(V + E)
 * Input constraints: Every vertex in g must have in degree == out degree
 */

#include <vector>
#include <algorithm>
#include <map>

using namespace std;

typedef vector<map<int, int>> graph;

void dfs(graph &g, vector<int> &circuit, int cur) {
    for (auto& p : g[cur]) {
        g[cur].erase(p.first);
        dfs(g, circuit, p.first);
    }
    circuit.push_back(cur);
}

vector<int> euler(graph& g) {
    vector<int> circ;
    dfs(g, circ, 0);
    reverse(circ.begin(), circ.end());
    return circ;
}
```