Summary

Configuration files

.vimrc

```
set nocp
set mouse=a

filetype plugin indent on

syntax on
set smd ls=2 nu sm is nohls bg=dark
set et sw=4 sts=4 ts=8 sta ai ci
set nowb nobk noswf

" compilation shortcuts
noremap  <f9>  <esc>:w<cr>:!g++ -Wall -g % && ./a.out<cr>
noremap! <f9>  <esc>:w<cr>:!g++ -Wall -g % && ./a.out<cr>
noremap  <f10> <esc>:w<cr>:!g++ -Wall -g % && ./a.out < %<.in<cr>
noremap! <f10> <esc>:w<cr>:!g++ -Wall -g % && ./a.out < %<.in<cr>
```

Template

```
#include <cstdio>
#include <cstdlib>
#include <cctype>
#include <cmath>
#include <cstring>
#include <utility>
#include <functional>
#include <algorithm>
#include <string>
#include <vector>
#include <list>
#include <deque>
#include <queue>
#include <stack>
#include <set>
#include <map>
#include <complex>
using namespace std;

#define INF 0x3f3f3f3f
#define PI M_PI
#define mp make_pair

typedef long long ll;
typedef unsigned long long ull;

const double inf = 1.0/0.0;

int cmp_double(double a, double b, double eps = 1e-9) {
    return a + eps > b ? b + eps > a ? 0 : 1 : -1;
}

int main() {
}
```

Graph

Tarjan

Complexity: O(V+E)

```
int n, m;
vector<int> g[MAXN];
int lbl[MAXN], low[MAXN], idx, cnt_scc;
stack<int> st;
bool inSt[MAXN];

void dfs(int v) {
    lbl[v] = low[v] = idx++;
    st.push(v);
    inSt[v] = 1;
    for (vector<int>::iterator it = g[v].begin(); it != g[v].end(); it++) {
        if (lbl[*it] == -1) {
            dfs(*it);
            if (low[*it] < low[v]) {
                low[v] = low[*it];
            }
        } else if (inSt[*it] && lbl[*it] < low[v]) {
            low[v] = lbl[*it];
        }
    }
    if (low[v] == lbl[v]) {
        printf("%d -> ", ++cnt_scc);
        int u;
        do {
            u = st.top();
            st.pop();
            inSt[u] = 0;
            printf("%d; ", u);
        } while (u != v);
        putchar('\n');
    }
}

void tarjan() {
    for (int i = 1; i <= n; i++) {
        lbl[i] = -1;
        inSt[i] = 0;
    }
    idx = cnt_scc = 0;
    for (int i = 1; i <= n; i++)
        if (lbl[i] == -1)
            dfs(i);
}
```

Articulation

Complexity: O(V+E)

```
int n, m;
vector<int> g[MAXN];
int lbl[MAXN], low[MAXN], parent[MAXN], idx;
bool art[MAXN], has_art;

void dfs(int v) {
    int count = 0;
    lbl[v] = low[v] = idx++;
    for (vector<int>::iterator it = g[v].begin(); it != g[v].end(); it++) {
        if (lbl[*it] == -1) {
            parent[*it] = v;
            dfs(*it);
            if (low[*it] < low[v]) {
                low[v] = low[*it];
            } else if (low[*it] >= lbl[v]) {
                count++;
            }
        } else if (*it != parent[v] && lbl[*it] < low[v]) {
            low[v] = lbl[*it];
        }
    }

    if (count > 1 || (lbl[v] != 0 && count > 0)) {
        art[v] = 1;
        has_art = 1;
    }
}

void articulation() {
    for (int i = 1; i <= n; i++) {
        lbl[i] = -1;
        art[i] = 0;
    }

    for (int i = 1; i <= n; i++) {
        if (lbl[i] == -1) {
            idx = 0;
            parent[i] = i;
            dfs(i);
        }
    }
}
```

Bridge

Complexity: O(V+E)

```
int n, m;
vector<int> g[MAXN];
int lbl[MAXN], low[MAXN], parent[MAXN], idx;
bool has_bridge;

void dfs(int v) {
    lbl[v] = low[v] = idx++;
    bool parent_found = 0;
    for (vector<int>::iterator it = g[v].begin(); it != g[v].end(); it++) {
        if (lbl[*it] == -1) {
            parent[*it] = v;
            dfs(*it);
            if (low[*it] < low[v]) {
                low[v] = low[*it];
            } else if (low[*it] == lbl[*it]) {
                printf("%d -> %d\n", v, *it);
                has_bridge = 1;
            }
        } else if (!parent_found && *it == parent[v]) {
            parent_found = 1;
        } else if (lbl[*it] < low[v]) {
            low[v] = lbl[*it];
        }
    }
}

void bridge() {
    for (int i = 1; i <= n; i++) {
        lbl[i] = -1;
    }

    for (int i = 1; i <= n; i++) {
        if (lbl[i] == -1) {
            idx = 0;
            parent[i] = i;
            dfs(i);
        }
    }
}
```

Edmonds-Karp

Complexity: O(V E^2)

```
int n, m, g[MAXN][MAXN];
int parent[MAXN];
bool visited[MAXN];

bool bfs(int s, int t) {
    queue<int> q;
    for (int i = 0; i < n; i++)
        visited[i] = 0;
    visited[s] = 1;
    q.push(s);
    while (!q.empty()) {
        int u = q.front();
        q.pop();
        for (int v = 0; v < n; v++) {
            if (g[u][v] && !visited[v]) {
                parent[v] = u;
                if (v == t)
                    return 1;
                q.push(v);
            }
        }
    }
    return 0;
}

int maxflow(int s, int t) {
    int flow = 0;
    while (bfs(s, t)) {
        int f = INF;
        for (int v = t, u = parent[v]; v != s; v = u, u = parent[v])
            f = min(f, g[u][v]);
        for (int v = t, u = parent[v]; v != s; v = u, u = parent[v]) {
            g[u][v] -= f;
            g[v][u] += f;
        }
        flow += f;
    }
    return flow;
}
```

Hopcroft-Karp

Complexity: O(E sqrt(V))

```
int n, m;
vector<int> g1[MAXN];
int pair_g1[MAXN], pair_g2[MAXM], dist[MAXN];

bool bfs() {
    queue<int> q;
    for (int v = 1; v <= n; v++) {
        if (pair_g1[v] == 0) {
            dist[v] = 0;
            q.push(v);
        } else {
            dist[v] = INF;
        }
    }

    dist[0] = INF;
    while (!q.empty()) {
        int v = q.front();
        q.pop();
        vector<int>::iterator it;
        for (it = g1[v].begin(); it != g1[v].end(); it++) {
            if (dist[pair_g2[*it]] == INF) {
                dist[pair_g2[*it]] = dist[v]+1;
                q.push(pair_g2[*it]);
            }
        }
    }
    return dist[0] != INF;
}

bool dfs(int v) {
    if (v != 0) {
        vector<int>::iterator it;
        for (it = g1[v].begin(); it != g1[v].end(); it++) {
            if (dist[pair_g2[*it]] == dist[v]+1 && dfs(pair_g2[*it])) {
                pair_g2[*it] = v;
                pair_g1[v] = *it;
                return 1;
            }
        }
        dist[v] = INF;
        return 0;
    }
    return 1;
}

int hk() {
    for (int v = 1; v <= n; v++)
        pair_g1[v] = 0;
    for (int v = 1; v <= m; v++)
        pair_g2[v] = 0;

    int matching = 0;
    while (bfs())
        for (int v= 1; v <= n; v++)
            if (pair_g1[v] == 0 && dfs(v))
                matching++;
    return matching;
}
```

Bellman-Ford

Complexity: O(VE)

```
struct Edge {
    int u, v, w;
    Edge () {}
    Edge (int u, int v, int w) : u(u), v(v), w(w) {}
};

int n, m;
Edge e[MAXM];
int dist[MAXN], parent[MAXN];

// return 1 if there is negative cycle
int bellman_ford(int s) {
    for (int i = 1; i <= n; i++)
        dist[i] = INF;
    dist[s] = 0;
    parent[s] = s;
    int flag = 1;
    for (int i = 0; flag && i < n; i++) {
        flag = 0;
        for (int j = 0; j < m; j++)
            if (dist[e[j].u] + e[j].w < dist[e[j].v]) {
                dist[e[j].v] = dist[e[j].u] + e[j].w;
                parent[e[j].v] = e[j].u;
                flag = 1;
            }
    }
    return flag;
}
```

Lowest Common Ancestor

Complexity: < O(N logN), O(logN) >

```
#define MAXN 50000
#define LOGMAXN 16

int n, m, u, v, w;
int ancestor[MAXN][LOGMAXN], parent[MAXN], level[MAXN], dist[MAXN];
vector<pair<int, int> > g[MAXN];

void dfs(int v) {
    vector<pair<int, int> >::iterator it;
    for (it = g[v].begin(); it != g[v].end(); it++) {
        if (it->first != parent[v]) {
            parent[it->first] = v;
            level[it->first] = level[v] + 1;
            dist[it->first] = dist[v] + it->second;
            dfs(it->first);
        }
    }
}

void pre() {
    parent[0] = level[0] = dist[0] = 0;
    dfs(0);
    for (int i = 0; i < n; i++)
        ancestor[i][0] = parent[i];
    for (int j = 1; 1<<j < n; j++)
        for (int i = 0; i < n; i++)
            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 log;
    for (log = 1; 1<<log <= level[u]; log++);
    log--;
    for (int i = log; i >= 0; i--)
        if (level[u] - (1<<i) >= level[v])
            u = ancestor[u][i];
    if (u == v)
        return u;
    for (int i = log; i >= 0; i--)
        if (ancestor[u][i] != ancestor[v][i])
            u = ancestor[u][i], v = ancestor[v][i];
    return parent[u];
}
```

Min-Cost Max-Flow

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

#define MAXC 210
#define MAXG 200

int tc, C1, C2, C, c1, c2, g;
bool net[MAXC][MAXC], visited[MAXC];
int cost[MAXC][MAXC], pi[MAXC], sigma[MAXC];
int p[MAXC];
vector<int> V[MAXC];
const int INF = 1<<20;

bool dijkstra(int s, int t) {
    sigma[t] = INF;
    visited[s] = visited[t] = 0;
    for (int i = 1; i <= C; i++) {
        sigma[i] = INF;
        visited[i] = 0;
    }

    priority_queue<pair<int, int> > PQ;
    PQ.push(make_pair(0, s));
    while (!PQ.empty()) {
        int v = PQ.top().second, w = -PQ.top().first;
        PQ.pop();
        if (!visited[v]) {
            visited[v] = 1;
            vector<int>::iterator it;
            for (it = V[v].begin(); it != V[v].end(); it++) {
                if (net[v][*it] && !visited[*it]) {
                    int ww;
                    if (v < *it)
                        ww = w + (MAXG - cost[v][*it]) + pi[v] - pi[*it];
                    else
                        ww = w + (cost[*it][v] - MAXG) + pi[v] - pi[*it];

                    if (ww < sigma[*it]) {
                        sigma[*it] = ww;
                        PQ.push(make_pair(-ww, *it));
                        p[*it] = v;
                    }
                }
            }
        }
    }
```

```
    if (sigma[t] == INF)
        return 0;
    pi[t] += sigma[t];
    for (int i = 1; i <= C; i++)
        pi[i] += sigma[i];
    return 1;
}

int main() {
    scanf("%d", &tc);
    while (tc--) {
        scanf("%d %d", &C1, &C2);
        C = C1 + C2;
        int s = 0, t = C+1;

        pi[s] = pi[t] = 0;
        for (int i = 1; i <= C; i++) {
            pi[i] = 0;
            for (int j = 1; j <= C; j++) {
                net[i][j] = 0;
            }
        }

        V[s].clear(), V[t].clear();
        for (int i = 1; i <= C1; i++) {
            net[s][i] = 1;
            cost[s][i] = 0;
            V[i].clear();
            V[s].push_back(i);
        }
        for (int i = C1+1; i <= C; i++) {
            net[i][t] = 1;
            cost[i][t] = 0;
            V[i].clear();
            V[i].push_back(t);
        }

        while (scanf("%d %d %d", &c1, &c2, &g), c1 || c2 || g) {
            net[c1][C1+c2] = 1;
            cost[c1][C1+c2] = g;
            cost[C1+c2][c1] = -g;
            V[c1].push_back(C1+c2);
            V[C1+c2].push_back(c1);
        }

        int val = 0, best = 0;
        p[0] = 0;
        while (dijkstra(s, t)) {
            c2 = t, c1 = p[c2];
            while (c1 != c2) {
                val += cost[c1][c2];
                net[c1][c2] = 0;
                net[c2][c1] = 1;
                c2 = c1, c1 = p[c2];
            }
            best = val > best ? val : best;
        }
        printf("%d\n", best);
    }
}
```

String

KMP

Complexity: O(N)

```
int t[MAXS];

void kmp_table(char s[MAXS]) {
    t[0] = -1, t[1] = 0;
    if (!s[1])
        return;
    for (int pos = 2, cnd = 0; s[pos]; ) {
        if (s[pos-1] == s[cnd])
            t[pos++] = ++cnd;
        else if (cnd > 0)
            cnd = t[cnd];
        else
            t[pos++] = 0;
    }
}

int kmp_search(char s1[MAXS], char s2[MAXS]) {
    kmp_table(s2);
    for (int i = 0, j = 0; s1[i+j]; ) {
        if (s2[j] == s1[i+j]) {
            if (!s2[j+1])
                return i;
            j++;
        } else {
            i += j-t[j];
            if (t[j] != -1)
                j = t[j];
            else
                j = 0;
        }
    }
    return -1;
}
```

Aho-Corasick

Complexity: < O(|S|), O(sum(|Si|)), O(|S|) >

```
struct Node {
    map<char, Node*> next;
    Node *fail;
    set<int> wordIds;

    Node () : fail(NULL) {}

    Node* getChild(const char& c) {
        map<char, Node*>::iterator it;
        it = next.find(c);
        if (it != next.end())
            return it->second;
        return NULL;
    }
};

Node *trie;
vector<string> words;

void addWord(const char* word) {
    Node *node = trie, *aux = NULL;
    for (int i = 0; word[i]; i++) {
        aux = node->getChild(word[i]);
        if (aux == NULL) {
            aux = new Node();
            node->next[word[i]] = aux;
        }
        node = aux;
    }
    node->wordIds.insert(words.size());
    words.push_back(word);
}

void init() {
    queue<Node*> q;
    map<char, Node*>::iterator it;

    trie->fail = trie;
    q.push(trie);
    while (!q.empty()) {
        Node *node = q.front();
        q.pop();
        for (it = node->next.begin(); it != node->next.end(); it++) {
            Node *child = it->second;
            char c = it->first;
            q.push(child);
            Node *fail = node->fail;
            while (fail->getChild(c) == NULL && fail != trie)
                fail = fail->fail;
            child->fail = fail->getChild(c);
            if (child->fail == NULL || child->fail == child)
                child->fail = trie;

            child->wordIds.insert(
                child->fail->wordIds.begin(), child->fail->wordIds.end()
            );
        }
    }
}

void search(const char* text) {
    Node *node = trie;
    for (int i = 0; text[i]; i++) {
        while (node->getChild(text[i]) == NULL && node != trie)
            node = node->fail;
        node = node->getChild(text[i]);
        if (node == NULL)
            node = trie;

        set<int>::iterator it;
        for (it = node->wordIds.begin(); it != node->wordIds.end(); it++) {
            // do something with matches
            printf("%s\n", words[*it].c_str());
        }
    }
}
```

Suffix Array and Longest Common Prefix

Complexity: < O(N logN), O(N) >

```
//Output:
// pos  = The suffix array. Contains the n suffixes of str sorted in
//        lexicographical order. Each suffix is represented as a
//        single integer (the position of str where it starts).
// rank = The inverse of the suffix array.
//        rank[i] = the index of the suffix str[i..n) in the pos array.
//        (In other words, pos[i] = k <==> rank[k] = i)
//        With this array, you can compare two suffixes in O(1):
//        Suffix str[i..n) is smaller than str[j..n) iff rank[i] < rank[j]

int n; // length of the string
char str[MAXN];
int rank[MAXN], pos[MAXN], cnt[MAXN], next[MAXN];
bool bh[MAXN], b2h[MAXN];

bool cmp(int a, int b) {
    return str[a] < str[b];
}

void suffix_array() {
    for (int i = 0; i < n; i++)
        pos[i] = i;
    sort(pos, pos+n, cmp);

    for (int i = 0; i < n; i++) {
        bh[i] = (i == 0 || str[pos[i]] != str[pos[i-1]]);
        b2h[i] = 0;
    }

    for (int h = 1; h < n; h <<= 1) {
        int buckets = 0;
        for (int i = 0, j; i < n; i = j) {
            j = i + 1;
            while (j < n && !bh[j])
                j++;
            next[i] = j;
            buckets++;
        }
        if (buckets == n)
            break;

        for (int i = 0; i < n; i = next[i]) {
            cnt[i] = 0;
            for (int j = i; j < next[i]; j++)
                rank[pos[j]] = i;
        }
        cnt[rank[n-h]]++;
        b2h[rank[n-h]] = 1;
        for (int i = 0; i < n; i = next[i]) {
            for (int j = i; j < next[i]; j++) {
                int s = pos[j] - h;
                if (s >= 0) {
                    int head = rank[s];
                    rank[s] = head + cnt[head]++;
                    b2h[rank[s]] = 1;
                }
            }
            for (int j = i; j < next[i]; j++) {
                int s = pos[j] - h;
                if (s >= 0 && b2h[rank[s]]) {
                    for (int k = rank[s] + 1; !bh[k] && b2h[k]; k++)
                        b2h[k] = 0;
                }
            }
        }
        for (int i = 0; i < n; i++) {
            pos[rank[i]] = i;
            bh[i] |= b2h[i];
        }
    }
    for (int i = 0; i < n; i++)
        rank[pos[i]] = i;
}

int height[MAXN];

void getHeight() {
    height[0] = 0;
    for (int i = 0, h = 0; i < n; i++) {
        if (rank[i] > 0) {
            int j = pos[rank[i] - 1];
            while (i + h < n && j + h < n && str[i+h] == str[j+h])
                h++;
            height[rank[i]] = h;
            if (h > 0)
                h--;
        }
    }
}
```

Manacher’s Algorithm

Complexity: O(N)

```
char s[MAXN];
int p[2*MAXN]; // length of the palindrome centered at position (i-1)/2;

void manacher() {
    int m = 0;
    char t[2*MAXN];
    for (int i = 0; s[i]; i++) {
        t[m++] = '#';
        t[m++] = s[i];
        p[i] = 0;
    }
    t[m++] = '#';

    int c = 0, r = 0;
    for (int i = 0; i < m; i++) {
        int i_ = 2 * c - i;
        p[i] = r > i ? min(r-i, p[i_]) : i & 1;
        while (0 <= i-p[i]-1 && i+p[i]+1 < m && t[i-p[i]-1] == t[i+p[i]+1])
            p[i] += 2;
        if (i + p[i] > r) {
            c = i;
            r = i + p[i];
        }
    }
}
```

Z Algorithm

Complexity: O(N)

```
// Input: string s
// Output: z[i], longest common prefix of s with his suffix starting at i

void z_algorithm(char *s, int *z) {
    int l = 0, r = 0;
    for (int i = 0; s[i]; i++) {
        if (i > r) {
            l = r = i;
            for (; s[r] == s[r-l]; r++);
            z[i] = r-- - l;
        }
        else if (z[i-l] < r-i+1) {
            z[i] = z[i-l];
        }
        else {
            l = i;
            for (r++; s[r] == s[r-l]; r++);
            z[i] = r-- - l;
        }
    }
}
```

Dynamic Programming

Optimal Array Multiplication Sequence

Complexity: O(N^3)

```
int n, m[MAXN], c[MAXN][MAXN];

void oams() {
    for (int i = 1; i <= n; i++)
        c[i][i] = 0;

    for (int d = 1; d < n; d++) {
        for (int i = 1; i <= n-d; i++) {
            int j = i+d;
            c[i][j] = INF;
            for (int k = i; k < j; k++)
                c[i][j] = min(c[i][j],
                              c[i][k] + c[k+1][j] + m[i-1]*m[k]*m[j]);
        }
    }
}
```

Optimal Binary Search Tree

Complexity: O(N^3)

```
int n, p[MAXN];
int c[MAXN][MAXN], f[MAXN][MAXN], r[MAXN][MAXN];

void obst() {
    for (int i = 1; i <= n; i++)
        c[i][i-1] = 0;
    c[n+1][n] = 0;

    for (int i = 1; i <= n; i++) {
        c[i][i] = p[i];
        f[i][i] = p[i];
        r[i][i] = i;
    }

    for (int d = 1; d < n; d++) {
        for (int i = 1; i <= n-d; i++) {
            int j = i+d;
            c[i][j] = INF;
            f[i][j] = f[i][j-1] + p[j];
            int rmin = r[i][j-1], rmax = r[i+1][j];
            for (int k = rmin; k <= rmax; k++) {
                int t = c[i][k-1] + c[k+1][j];
                if (t < c[i][j]) {
                    c[i][j] = t;
                    r[i][j] = k;
                }
            }
            c[i][j] += f[i][j];
        }
    }
}
```

Longest Increasing Subsequence

Complexity: O(N logN)

```
int n, m, a[MAXN], b[MAXN], p[MAXN];

void lis() {
    int u, v;
    b[m++] = 0;
    for (int i = 1; i < n; i++) {
        if (a[b[m-1]] < a[i]) {
            p[i] = b[m-1];
            b[m++] = i;
            continue;
        }
        for (u = 0, v = m-1; u < v; ) {
            int c = (u + v)/2;
            if (a[b[c]] < a[i])
                u = c + 1;
            else
                v = c;
        }
        if (a[i] < a[b[u]]) {
            if (u > 0)
                p[i] = b[u-1];
            b[u] = i;
        }
    }
    for (u = m, v = b[m-1]; u--; v = p[v]) {
        b[u] = v;
    }
}
```

Longest Common Increasing Subsequence

Complexity: O(N^2)

```
int n, m, a[MAXN], b[MAXN];
int c[MAXN], prev[MAXN], seq[MAXN];

void lcis() {
    for (int j = 0; j < m; j++)
        c[j] = 0;
    for (int i = 0; i < n; i++) {
        int actual = 0, last = -1;
        for (int j = 0; j < m; j++) {
            if (a[i] == b[j] && actual+1 > c[j]) {
                c[j] = actual+1;
                prev[j] = last;
            } else if (a[i] > b[j] && actual < c[j]) {
                actual = c[j];
                last = j;
            }
        }
    }
    int length = 0, index = -1;
    for (int j = 0; j < m; j++) {
        if (c[j] > length) {
            length = c[j];
            index = j;
        }
    }
    int len = length;
    while (index != -1) {
        seq[--len] = b[index];
        index = prev[index];
    }
    printf("length: %d\n", length);
    for (int i = 0; i < length; i++)
        printf("%d ", seq[i]);
    printf("\n");
}
```

Weighted Activity Selection

Complexity: O(N logN)

```
#include <cstdio>
#include <algorithm>
using namespace std;

#define MAXN 10005

struct Event {
    int b, e, w;
    Event () {}
    Event (int b, int e, int w) : b(b), e(e), w(w) {}
    bool operator< (const Event& o) const {
        if (e != o.e)
            return e < o.e;
        return b < o.b;
    }
};

int n;
Event e[MAXN];

int dp[MAXN];

int main() {
    scanf("%d", &n);
    e[0] = Event(0, 0, 0);
    for (int i = 1; i <= n; i++)
        scanf("%d %d %d", &e[i].b, &e[i].e, &e[i].w);
    sort(e+1, e+n+1);
    dp[0] = 0;
    for (int i = 1; i <= n; i++) {
        int lo = 0, hi = i-1;
        while (lo < hi) {
            int mid = (lo + hi + 1) >> 1;
            if (e[mid].e > e[i].b)
                hi = mid - 1;
            else
                lo = mid;
        }
        dp[i] = max(dp[i-1], e[i].w + dp[lo]);
    }
    printf("Max weight: %d\n", dp[n]);
}
```

Data Structure

Segment Tree with Lazy Propagation

Complexity: < O(N), O(logN) >

```
#define LEFT(x)  (x << 1)
#define RIGHT(x) ((x << 1) + 1)

ll segtree[4*MAXN], lazy[4*MAXN];

void propagate(int node, int lo, int hi) {
    segtree[node] += lazy[node] * (hi-lo+1);
    if (lo != hi) {
        lazy[LEFT(node)] += lazy[node];
        lazy[RIGHT(node)] += lazy[node];
    }
    lazy[node] = 0;
}

void update(int node, int lo, int hi, int i, int j, int val) {
    if (j < lo || hi < i)
        return;

    if (i <= lo && hi <= j) {
        lazy[node] += val;
        return;
    }

    int mid = (lo + hi)/2;
    update(LEFT(node), lo, mid, i, j, val);
    update(RIGHT(node), mid+1, hi, i, j, val);

    propagate(LEFT(node), lo, mid);
    propagate(RIGHT(node), mid+1, hi);
    segtree[node] = segtree[LEFT(node)] + segtree[RIGHT(node)];
}

ll query(int node, int lo, int hi, int i, int j) {
    if (j < lo || hi < i)
        return 0;

    propagate(node, lo, hi);
    if (i <= lo && hi <= j)
        return segtree[node];

    int mid = (lo + hi)/2;
    return query(LEFT(node), lo, mid, i, j) +
           query(RIGHT(node), mid+1, hi, i, j);
}
```

Geometry

Template

```
struct Point {
    double x, y;

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

    Point operator+ (const Point &o) const { return Point(x + o.x, y + o.y); }
    Point operator- (const Point &o) const { return Point(x - o.x, y - o.y); }
    Point operator* (const double &o) const { return Point(x * o, y * o); }
    Point operator/ (const double &o) const { return Point(x / o, y / o); }
    double operator* (const Point &o) const { return x * o.x + y * o.y; }
    double operator% (const Point &o) const { return x * o.y - o.x * y; }

    bool operator== (const Point &o) const {
        return cmp_double(x, o.x) == 0 && cmp_double(y, o.y) == 0;
    }

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

typedef Point Vector;

double abs(Point p) {
    return sqrt(p * p);
}

Vector norm(Vector v) {
    return v / abs(v);
}

double ccw(Point p, Point q, Point r) {
    return (q - p) % (r - p);
}

struct Segment {
    Point p, q;

    Segment() {}
    Segment(Point p, Point q) : p(p), q(q) {}
};
```

```
bool in_segment(Point p, Segment s) {
    double t;
    Vector v = s.q - s.p;
    if (cmp_double(v.x, 0) != 0)
        t = (p.x - s.p.x) / v.x;
    else
        t = (p.y - s.p.y) / v.y;
    return cmp_double(t, 0) >= 0 && cmp_double(t, 1) <= 0 && s.p + v * t == p;
}

struct Line {
    Vector v;
    Point p;
    int a, b, c;

    void init() {
        a = -v.y;
        b = v.x;
        c = a * p.x + b * p.y;
        int d = abs(__gcd(a, __gcd(b, c)));
        if (d != 1)
            a /= d, b /= d, c /= d;
        if (a < 0)
            a = -a, b = -b, c = -c;
        else if (a == 0 && b < 0)
            b = -b, c = -c;
    }

    Line() {}
    Line(Point p, Point q) : v(q-p), p(p) {
        init();
    }
    Line(Segment s) : v(s.q-s.p), p(p) {
        init();
    }

    Point operator() (double t) const { return p + v * t; }

    Vector normal() {
        return Vector(-v.y, v.x);
    }
};

pair<double, double> line_intersection(Line a, Line b) {
    double den = a.v % b.v;
    if (den == 0)
        return make_pair(inf, inf);
    double t = -(b.v % (b.p - a.p)) / den;
    double s = -(a.v % (b.p - a.p)) / den;
    return make_pair(t, s);
}

Point segment_intersection(Segment a, Segment b) {
    Line la = Line(a), lb = Line(b);
    pair<double, double> pdd = line_intersection(la, lb);
    double t = pdd.first, s = pdd.second;
    if (t == inf) {
        if (in_segment(b.p, a))
            return b.p;
        if (in_segment(b.q, a))
            return b.q;
        if (in_segment(a.p, b))
            return a.p;
        if (in_segment(a.q, b))
            return a.q;
        return Point(inf, inf);
    }
    if (cmp_double(t, 0) < 0 || cmp_double(t, 1) > 0)
        return Point(inf, inf);
    if (cmp_double(s, 0) < 0 || cmp_double(s, 1) > 0)
        return Point(inf, inf);
    return la(t);
}

double distPointToLine(Point p, Line l) {
    Vector n = l.normal();
    return (l.p - p) * n / abs(n);
}

struct Circle {
    Point center;
    double radius;

    Circle () {}
    Circle (Point center, double radius) : center(center), radius(radius) {}
};

Point circumcenter(Point p, Point q, Point r) {
    Point a = p - r, b = q - r, c = Point(a*(p+r)/2, b*(q+r)/2);
    return Point(c % Point(a.y, b.y), Point(a.x, b.x) % c)/(a % b);
}

Point incenter(Point p, Point q, Point r) {
    double a = abs(r - q), b = abs(r - p), c = abs(q - p);
    return (p * a + q * b + r * c) / (a + b + c);
}
```

Monotone Chain Convex Hull

Complexity: O(N logN)

```
int n, k;
Point p[MAXN], h[MAXN];

void convex_hull() {
    sort(p, p+n);
    k = 0;
    h[k++] = p[0];
    for (int i = 1; i < n; i++) {
        if (i != n-1 && ccw(p[0], p[n-1], p[i]) >= 0) continue;
        while (k > 1 && ccw(h[k-2], h[k-1], p[i]) <= 0) k--;
        h[k++] = p[i];
    }
    for (int i = n-2, lim = k; i >= 0; i--) {
        if (i != 0 && ccw(p[n-1], p[0], p[i]) >= 0) continue;
        while (k > lim && ccw(h[k-2], h[k-1], p[i]) <= 0) k--;
        h[k++] = p[i];
    }
}
```

Smallest Enclosing Circle

Complexity: O(N^2)

```
bool in_circle(const Circle &c, const Point &p) {
    return cmp_double(abs(c.p - p), c.r) <= 0;
}

int n;
Point p[MAXN];

Circle spanning_circle() {
    random_shuffle(p, p+n);
    Circle c(Point(), -1);
    for (int i = 0; i < n; i++) if (!in_circle(c, p[i])) {
        c = Circle(p[i], 0);
        for (int j = 0; j < i; j++) if (!in_circle(c, p[j])) {
            c = Circle((p[i] + p[j])/2, abs(p[i] - p[j])/2);
            for (int k = 0; k < j; k++) if (!in_circle(c, p[k])) {
                Point o = circumcenter(p[i], p[j], p[k]);
                c = Circle(o, abs(o - p[k]));
            }
        }
    }
    return c;
}
```

Closest Pair of Points

Complexity: O(N logN)

```
#include <cstdio>
#include <cmath>
#include <algorithm>
#include <set>
using namespace std;

struct Point {
    int x, y;
    Point(int x = 0, int y = 0) : x(x), y(y) {}
    Point operator- (const Point &o) const { return Point(x - o.x, y - o.y); }
    int operator* (const Point &o) const { return x * o.x + y * o.y; }
    bool operator< (const Point &o) const {
        return y != o.y ? y < o.y : x < o.x;
    }
};
bool cmpx(const Point &p, const Point &q) {
    return p.x != q.x ? p.x < q.x : p.y < q.y;
}

double abs(const Point &p) {
    return sqrt(p * p);
}

int main() {
    int n;
    Point pnts[MAXN];
    set<Point> box;
    set<Point>::iterator it;
    scanf("%d", &n);
    for (int i = 0; i < n; i++)
        scanf("%d %d", &pnts[i].x, &pnts[i].y);
    sort(pnts, pnts+n, cmpx);
    double best = inf;
    box.insert(pnts[0]);
    for (int i = 1, j = 0; i < n; i++) {
        while (j < i && pnts[i].x - pnts[j].x > best)
            box.erase(pnts[j++]);
        for (it = box.lower_bound(Point(pnts[i].x-best, pnts[i].y-best));
             it != box.end() && it->y <= pnts[i].y + best; it++) {
            best = min(best, abs(pnts[i] - *it));
        }
        box.insert(pnts[i]);
    }
    printf("%.2lf\n", best);
}
```

Math

Sieve, primality, factorization, phi

```
int np, p[MAXP], nf, f[MAXP], e[MAXP];
bool prime[MAXN];

void sieve(int n) {
    memset(prime, 0, sizeof(prime));
    prime[2] = 1;
    for (int i = 3; i <= n; i += 2)
        prime[i] = 1;

    for (int i = 3, lim = sqrt(n); i <= lim; i += 2)
        if (prime[i])
            for (int j = i*i; j <= n; j += i)
                prime[j] = 0;

    np = 0;
    p[np++] = 2;
    for (int i = 3; i <= n; i += 2)
        if (prime[i])
            p[np++] = i;
}

void factor(int n) {
    nf = 0;
    for (int i = 0, lim = sqrt(n); n != 1 && p[i] <= lim; i++) {
        if (n % p[i] == 0) {
            f[nf] = p[i];
            e[nf] = 1;
            n /= p[i];
            while (n % p[i] == 0) {
                e[nf]++;
                n /= p[i];
            }
            nf++;
            lim = sqrt(n);
        }
    }
    if (n != 1) {
        f[nf] = n;
        e[nf] = 1;
        nf++;
    }
}
```

```
int _phi(int n) {
    int ret = 1;
    for (int i = 0, lim = sqrt(n); n != 1 && p[i] <= lim; i++) {
        if (n % p[i] == 0) {
            int pk = p[i];
            n /= p[i];
            while (n % p[i] == 0) {
                pk *= p[i];
                n /= p[i];
            }
            ret *= pk - pk/p[i];
            lim = sqrt(n);
        }
    }
    if (n != 1)
        ret *= n-1;
    return ret;
}

int phi[MAXN];

void build_phi(int n) {
    for (int i = 0; i <= n; i++)
        phi[i] = i;
    for (int i = 2; i <= n; i++) if (phi[i] == i)
        for (int j = i; j <= n; j += i)
            phi[j] = phi[j] / i * (i-1);
}
```

Chinese Remainder Algorithm

```
#include <cstdio>
#include <algorithm>
using namespace std;

const int MAXN = 100010;

typedef pair<int, int> tpii;

struct teq {
    int r, n; // x = r (mod n)
};

int qnt;
teq eqs[MAXN];

tpii egcd(int a, int b) {
    int x = 0, lastx = 1, auxx;
    int y = 1, lasty = 0, auxy;
    while (b) {
        int q = a / b, r = a % b;
        a = b, b = r;
        auxx = x;
        x = lastx - q*x, lastx = auxx;
        auxy = y;
        y = lasty - q*y, lasty = auxy;
    }
    return make_pair(lastx, lasty);
}

int chinese_remainder_algorithm() {
    int beta, sum = 0, n = 1;
    for (int i = 0; i < qnt; i++)
        n *= eqs[i].n;
    for (int i = 0; i < qnt; i++) {
        beta = egcd(eqs[i].n, n/eqs[i].n).second;
        while (beta < 0)
            beta += eqs[i].n;
        sum += (eqs[i].r * beta * n/eqs[i].n) % n;
    }
    return sum;
}

int main() {
    scanf("%d", &qnt);
    for (int i = 0; i < qnt; i++)
        scanf("%d %d", &eqs[i].r, &eqs[i].n);
    printf("%d\n", chinese_remainder_algorithm());
}
```

Shanks Baby-Step Giant-Step Algorithm

```
#define MAXN 100010

int modinv(int a, int n) {
    int b = n, x = 0, lastx = 1, aux;
    while (b) {
        int q = a / b, r = a % b;
        a = b; b = r;
        aux = x;
        x = lastx - q * x, lastx = aux;
    }
    while (lastx < 0)
        lastx += n;
    return lastx;
}

int modpow(int x, int e, int n) {
    int ret = 1;
    while (e) {
        if (e & 1)
            ret = (ret * x) % n;
        x = (x * x) % n;
        e >>= 1;
    }
    return ret;
}

// @param   a   generator of group Z_n
// @param   n   group Z_n
// @return x such that a^x = b (mod n) or -1
int shanks_algorithm(int a, int b, int n) {
    int m = ceil(sqrt(n));
    int table[MAXN];

    for (int i = 0; i < n; i++)
        table[i] = -1;
    int aux = 1;
    for (int j = 0; j < m; j++) {
        table[aux] = j;
        aux = (aux * a) % n;
    }
    aux = modpow(modinv(a, n), m, n);
    for (int i = 0; i < m; i++) {
        if (table[b] != -1)
            return i*m + table[b];
        b = (b * aux) % n;
    }
    return -1;
}
```

FFT

```
typedef complex<long double> pt;

pt tmp[1<<20];
void fft(pt *in, int sz, bool inv) {
    if (sz == 1)
        return;
    for (int i = 0, j = 0, h = sz >> 1; i < sz; i += 2, j++) {
        in[j] = in[i];
        tmp[h+j] = in[i+1];
    }
    for (int i = sz >> 1; i < sz; i++)
        in[i] = tmp[i];
    sz >>= 1;
    pt *even = in, *odd = in + sz;
    fft(even, sz, inv);
    fft(odd, sz, inv);
    long double p = (inv ? -1 : 1) * M_PI / sz;
    pt w = pt(cosl(p), sinl(p)), w_i = 1;
    for (int i = 0; i < sz; i++) {
        pt conv = w_i * odd[i];
        odd[i] = even[i] - conv;
        even[i] += conv;
        w_i *= w;
    }
}
```

Polynomial

```
struct Poly {
    int n;
    double a[MAXN];
    Poly(int n = 0) : n(n) { memset(a, 0, sizeof(a)); }
    Poly(const Poly &o) : n(o.n) { memcpy(a, o.a, sizeof(a)); }
    const double& operator[] (int i) const { return a[i]; }
    double& operator[] (int i) { return a[i]; }
    double operator() (double x) const {
        double ret = 0;
        for (int i = n; i >= 0; i--)
            ret = ret * x + a[i];
        return ret;
    }
    Poly operator+ (const Poly &o) const {
        Poly ret = o;
        for (int i = 0; i <= n; i++)
            ret[i] += a[i];
        ret.n = max(n, o.n);
        return ret;
    }
    Poly operator- (const Poly &o) const {
        Poly ret = o;
        for (int i = 0; i <= n; i++)
            ret[i] -= a[i];
        ret.n = max(n, o.n);
        return ret;
    }
    Poly operator* (const Poly &o) const {
        Poly ret(n + o.n);
        for (int i = 0; i <= n; i++)
            for (int j = 0; j <= o.n; j++)
                ret[i+j] += a[i] * o[j];
        return ret;
    }
};
```

```
Poly fastMult(const Poly &p, const Poly &q) {
    int sz = 1 << (32 - __builtin_clz(p.n + q.n + 1));
    pt pin[sz], qin[sz];
    for (int i = 0; i < sz; i++) {
        if (i <= p.n)
            pin[i] = p[i];
        else
            pin[i] = 0;
        if (i <= q.n)
            qin[i] = q[i];
        else
            qin[i] = 0;
    }
    fft(pin, sz, 0);
    fft(qin, sz, 0);
    for (int i = 0; i < sz; i++)
        pin[i] *= qin[i];
    fft(pin, sz, 1);
    Poly ret(p.n + q.n);
    for (int i = 0; i <= ret.n; i++)
        ret[i] = pin[i].real() / sz;
    while (ret.n > 0 && cmp(ret[ret.n], 0) == 0)
        ret.n--;
    return ret;
}

Poly diff(const Poly &p) {
    Poly ret(p.n-1);
    for (int i = 1; i <= p.n; i++)
        ret[i-1] = i * p[i];
    return ret;
}

pair<Poly, double> ruffini(const Poly &p, double x) {
    if (p.n == 0)
        return make_pair(Poly(), 0);
    Poly ret(p.n-1);
    for (int i = p.n; i > 0; i--)
        ret[i-1] = ret[i] * x + p[i];
    return make_pair(ret, ret[0] * x + p[0]);
}
```

```
/**
 * Find a root in range [lo, hi] assuming that exists only one root in [lo,
hi]
 * pair::second is true if exists a root in the given range or false otherwise
 * pair::first is the root if pair::second is true or 0 if false
 */
pair<double, int> findRoot(const Poly &p, double lo, double hi) {
    if (cmp(p(lo), 0) == 0)
        return make_pair(lo, 1);
    if (cmp(p(hi), 0) == 0)
        return make_pair(hi, 1);
    if (cmp(p(lo), 0) == cmp(p(hi), 0))
        return make_pair(0, 0);
    if (cmp(p(lo), p(hi)) > 0)
        swap(lo, hi);
    while (cmp(lo, hi) != 0) {
        double mid = (lo + hi) / 2;
        double val = p(mid);
        if (cmp(val, 0) == 0)
            lo = hi = mid;
        else if (cmp(val, 0) < 0)
            lo = mid;
        else
            hi = mid;
    }
    return make_pair(lo, 1);
}

/**
 * Return a vector of all real roots with their multiplicity in ascending
order
 */
vector<double> roots(const Poly &p) {
    vector<double> ret;
    if (p.n == 1) {
        ret.push_back(-p[0] / p[1]);
    }
    else {
        vector<double> r = roots(diff(p));
        r.push_back(-MAXX);
        r.push_back(MAXX);
        sort(r.begin(), r.end());
        for (int i = 0, j = 1; j < (int) r.size(); i++, j++) {
            pair<double, int> pr = findRoot(p, r[i], r[j]);
            if (pr.second)
                ret.push_back(pr.first);
        }
    }
    return ret;
}
```

Bignum

```
#include <cstring>
#include <algorithm>
#include <limits>
using namespace std;

typedef long long ll;
typedef unsigned long long ull;

const int MAXD = 1005, DIG = 9, BASE = 1000000000;
const ull BOUND = numeric_limits <ull> :: max() - (ull) BASE * BASE;

struct bignum
{
    int D, digits[MAXD / DIG + 2];
    int sign;

    inline void trim () {
        while (D > 1 && digits[D - 1] == 0)
            D--;
    }

    inline void init (ll x) {
        memset(digits, 0, sizeof(digits));
        D = 0;

        if (x < 0) {
            sign = -1;
            x = -x;
        }
        else {
            sign = 1;
        }

        do {
            digits[D++] = x % BASE;
            x /= BASE;
        } while (x > 0);
    }

    inline bignum (ll x) {
        init(x);
    }

    inline bignum (int x = 0) {
        init(x);
    }

    inline bignum (char *s) {
        memset(digits, 0, sizeof(digits));

        if (s[0] == '-') {
            sign = -1;
            s++;
        }
        else {
            sign = 1;
        }

        int len = strlen(s), first = (len + DIG - 1) % DIG + 1;
        D = (len + DIG - 1) / DIG;

        for (int i = 0; i < first; i++)
            digits[D - 1] = digits[D - 1] * 10 + s[i] - '0';

        for (int i = first, d = D - 2; i < len; i += DIG, d--)
            for (int j = i; j < i + DIG; j++)
                digits[d] = digits[d] * 10 + s[j] - '0';

        trim();
    }

    inline char *str () {
        trim();
        char *buf = new char[DIG * D + 2];
        int pos = 0, d = digits[D - 1];

        if (sign == -1)
            buf[pos++] = '-';

        do {
            buf[pos++] = d % 10 + '0';
            d /= 10;
        } while (d > 0);

        reverse(buf + (sign == -1 ? 1 : 0), buf + pos);

        for (int i = D - 2; i >= 0; i--, pos += DIG)
            for (int j = DIG - 1, t = digits[i]; j >= 0; j--) {
                buf[pos + j] = t % 10 + '0';
                t /= 10;
            }

        buf[pos] = '\0';
        return buf;
    }
```

```
inline bool operator < (const bignum &o) const {
    if (sign != o.sign)
        return sign < o.sign;

    if (D != o.D)
        return sign * D < o.sign * o.D;

    for (int i = D - 1; i >= 0; i--)
        if (digits[i] != o.digits[i])
            return sign * digits[i] < o.sign * o.digits[i];

    return false;
}

inline bool operator > (const bignum &o ) const {
    if (sign != o.sign)
        return sign > o.sign;

    if (D != o.D)
        return sign * D > o.sign * o.D;

    for (int i = D - 1; i >= 0; i--)
        if (digits[i] != o.digits[i])
            return sign * digits[i] > o.sign * o.digits[i];

    return false;
}

inline bool operator == (const bignum &o) const {
    if (sign != o.sign)
        return false;

    if (D != o.D)
        return false;

    for (int i = 0; i < D; i++)
        if (digits[i] != o.digits[i])
            return false;

    return true;
}
```

```
inline bignum operator << (int p) const {
    bignum temp;
    temp.D = D + p;

    for (int i = 0; i < D; i++)
        temp.digits[i + p] = digits[i];

    for (int i = 0; i < p; i++)
        temp.digits[i] = 0;

    return temp;
}

inline bignum operator >> (int p) const {
    bignum temp;
    temp.D = D - p;

    for (int i = 0; i < D - p; i++)
        temp.digits[i] = digits[i + p];

    for (int i = D - p; i < D; i++)
        temp.digits[i] = 0;

    return temp;
}

inline bignum range (int a, int b) const {
    bignum temp = 0;
    temp.D = b - a;

    for (int i = 0; i < temp.D; i++)
        temp.digits[i] = digits[i + a];

    return temp;
}

inline bignum abs () const {
    bignum temp = *this;
    temp.sign = 1;
    return temp;
}
```

```
inline bignum operator + (const bignum &o) const {
    if (sign != o.sign) {
        if (sign == 1)
            return *this - o.abs();
        else
            return o - this->abs();
    }

    bignum sum = o;
    int carry = 0;

    for (sum.D = 0; sum.D < D || carry > 0; sum.D++) {
        sum.digits[sum.D] += (sum.D < D ? digits[sum.D] : 0) + carry;

        carry = 0;
        if (sum.digits[sum.D] >= BASE) {
            sum.digits[sum.D] -= BASE;
            carry = 1;
        }
    }

    sum.D = max(sum.D, o.D);
    sum.trim();
    return sum;
}

inline bignum operator - (const bignum &o) const {
    if (sign != o.sign) {
        if (sign == 1)
            return *this + o.abs();
        else
            return -(this->abs() + o);
    }
    else if (sign == -1) {
        return o.abs() - this->abs();
    }

    bignum diff, temp;

    if (o > *this) {
        diff = o;
        diff.sign = -1;
        temp = *this;
    }
    else {
        diff = *this;
        temp = o;
    }

    for (int i = 0, carry = 0; i < temp.D || carry > 0; i++) {
        diff.digits[i] -= (i < temp.D ? temp.digits[i] : 0) + carry;
        carry = 0;
        if (diff.digits[i] < 0) {
            diff.digits[i] += BASE;
            carry = 1;
        }
    }

    diff.trim();
    return diff;
}

inline bignum operator - () const {
    bignum temp = *this;
    temp.sign = -temp.sign;
    return temp;
}

inline bignum operator * (const bignum &o) const {
    bignum prod = 0;
    ull sum = 0, carry = 0;

    for (prod.D = 0; prod.D < D + o.D - 1 || carry > 0; prod.D++) {
        sum = carry % BASE;
        carry /= BASE;

        for (int j = max(prod.D-o.D+1, 0); j <= min(D-1, prod.D); j++) {
            sum += (ull) digits[j] * o.digits[prod.D - j];

            if (sum >= BOUND) {
                carry += sum / BASE;
                sum %= BASE;
            }
        }

        carry += sum / BASE;
        prod.digits[prod.D] = sum % BASE;
    }

    prod.sign = sign * o.sign;
    prod.trim();
    return prod;
}
```

```cpp
inline double double_div (const bignum &o) const {
    double val = 0, oval = 0;
    int num = 0, onum = 0;

    for (int i = D - 1; i >= max(D - 3, 0); i--, num++)
        val = val * BASE + digits[i];

    for (int i = o.D - 1; i >= max(o.D - 3, 0); i--, onum++)
        oval = oval * BASE + o.digits[i];

    return sign * o.sign * val / oval * (D - num > o.D - onum ? BASE : 1);
}

inline pair<bignum, bignum> divmod (const bignum &o) const {
    if (sign != o.sign) {
        pair<bignum, bignum> p = (this->abs()).divmod(o.abs());
        p.first.sign = -1;
        p.second.sign = sign;
        return p;
    }
    else if (sign == -1) {
        pair<bignum, bignum> p = (this->abs()).divmod(o.abs());
        p.second.sign = sign;
        return p;
    }

    bignum quot = 0, rem = *this, temp;

    for (int i = D - o.D; i >= 0; i--) {
        temp = rem.range(i, rem.D);
        int div = (int) temp.double_div(o);
        bignum mult = o * div;

        while (div > 0 && temp < mult) {
            mult = mult - o;
            div--;
        }

        while (div + 1 < BASE && !(temp < mult + o)) {
            mult = mult + o;
            div++;
        }

        rem = rem - (o * div << i);

        if (div > 0) {
            quot.digits[i] = div;
            quot.D = max(quot.D, i + 1);
        }
    }
        quot.trim();
        rem.trim();
        return make_pair(quot, rem);
    }

    inline bignum operator / (const bignum &o) const {
        return divmod(o).first;
    }

    inline bignum operator % (const bignum &o) const {
        return divmod(o).second;
    }

    inline bignum power (int exp) const {
        bignum p = 1, temp = *this;

        while (exp > 0) {
            if (exp & 1) p = p * temp;
            if (exp > 1) temp = temp * temp;
            exp >>= 1;
        }

        return p;
    }
};

inline bignum gcd (bignum a, bignum b) {
    bignum t;

    while (!(b == 0)) {
        t = a % b;
        a = b;
        b = t;
    }

    return a;
}
```

Useful facts

Erdös-Gallai theorem: A sequence of non-negative integers $d_1 \geq \cdots \geq d_n$ can be represented as the degree sequence of a finite simple graph on n vertices if and only if $d_1 + \cdots + d_n$ is even and $\sum_{i=1}^{k} d_i \leq k(k-1) + \sum_{i=k+1}^{n} \min(d_i, k)$ holds for $1 \leq k \leq n$.

Split graph property: A split graph can be recognized solely from their degree sequence. Let the degree sequence of a graph G be $d_1 \geq \cdots \geq d_n$ and m is the largest value of i such that $d_i \geq i - 1$. Then G if a split graph if and only if $\sum_{i=1}^{m} d_i = m(m-1) + \sum_{i=m+1}^{n} d_i$.

Stirling's approximation: $(n \geq 100)$

$$\ln n! \approx n \ln n - n + \frac{1}{2} \ln(2\pi n)$$

2-SAT: Algorithm for solving Boolean expression in 2-CNF form (example: $(A \vee B) \wedge (B \vee \sim C) \wedge (A \vee C) \wedge (B \vee D)$).

1) Transform each term of conjunctions $(A \vee B)$ into $(\sim A \rightarrow B) \wedge (\sim B \rightarrow A)$
2) Construct graph $G = (V, E)$ such that each literal is a vertex and each implication is an edge
3) Run SCC algorithm. If there is a SCC such that A and $\sim A$ are in it, so the expression cannot be evaluated TRUE. Otherwise, it is possible.

Pick's Theorem: $A = i + \frac{b}{2} - 1$