

Team Reference Document

Encore @ Harbin Institute of Technology

June 25, 2013

Contents

1	String Processing	1
1.1	AC Automaton	1
1.2	Suffix Array	1
1.3	Suffix Automaton	2
1.4	KMP	2
1.5	Algorithm Z	2
2	Network Flow	2
2.1	Max flow	2
2.2	Cost flow	3
3	Data Structure	3
3.1	DLX exact cover	3
3.2	DLX fuzzy cover	4
3.3	Partition Tree	4
3.4	Leftist Tree	4
3.5	Cartesian Tree	5
3.6	Splay	5
4	Graph Theory	6
4.1	2-Satisfiability	6
4.2	Edge Cut	6
4.3	Vertex Cut	6
4.4	Hopcroft Karp	7
4.5	Hungary Algorithm	7
4.6	KM	7
4.7	Stable Marriage	7
4.8	Maximum Clique	7
4.9	Maximal Clique	8
4.10	Lowest Common Ancestor	8
4.11	Minimum Cut Algorithm	8
4.12	Degree-constrained Spanning Tree	8
4.13	Minimum Directed Tree	9
5	Math	9
5.1	Minimum Directed Tree	9
6	Math	9
6.1	Matrix	9
6.2	Number Thoery	9
6.2.1	Phi	9
6.2.2	$a^x == b(modn)$	10
6.2.3	$x * x == a(modp)$	10
6.2.4	Miller and Pollard	10
6.2.5	Get prime in range	11
6.2.6	Mod Equation	11
6.3	Fraction	11
6.3.1	$a/b < x/y < c/d$	11
6.3.2	$x^2 - n * y^2 = 1$	12
6.3.3	$\sum_{k=0}^{n-1} [(a + d * k)/m]$	12
6.4	Linear Equaton	12
6.4.1	Xor Equation	12
6.4.2	Equation in Z	12
6.4.3	Equation in R	13
6.4.4	det	13
6.5	Anti-Nim	13
6.6	nim multiply	13
6.7	FFT	14
6.8	Popynomial	14

1 String Processing

1.1 AC Automaton

```

1 #define code(ch) ((ch) - 'A')
2 const int KIND = 26, MAXN = 3000000;
3 struct node {
4     node* nxt[KIND], *fail;
5     int count, id;
6 } pool[MAXN], *pp, *root, *q[MAXN];
7 node* newNode() {
8     pp->fail = NULL;
9     pp->count = 0;
10    memset(pp->nxt, 0, sizeof(pp->nxt));
11    return pp++;
12 }
13 void initialize() {
14     pp = pool;
15     root = newNode();
16 }
17 void insert(const char* str, int id) {
18     node* now = root;
19     while(*str) {
20         int i = code(*str);
21         now->nxt[i] = now->nxt[i] == 0 ? newNode() : now->nxt[i];
22         now = now->nxt[i];
23         str++;
24     }
25     now->count++, now->id = id;
26 }
27 void buildFail(node*& now, int ith) {
28     if(now == root) now->nxt[ith]->fail = root;
29     node* tmp = now->fail;
30     while(tmp) {
31         if(tmp->nxt[ith] != NULL) {
32             now->nxt[ith]->fail = tmp->nxt[ith];
33             return;
34         }
35         tmp = tmp->fail;
36     }
37     if(tmp == NULL) now->nxt[ith]->fail = root;
38 }
39 void build() {
40     int head = 0, tail = 0;
41     q[tail++] = root;
42     while(head != tail) {
43         node* beg = q[head++];
44         for(int i = 0; i < KIND; i++) {
45             if(beg->nxt[i] == NULL) continue;
46             buildFail(beg, i);
47             q[tail++] = beg->nxt[i];
48         }
49     }
50 }
51 node* goStatus(node* now, int ith) {
52     node* tmp = now;
53     while(now->nxt[ith] == NULL && now != root)
54         now = now->fail;
55     now = now->nxt[ith];
56     return now == NULL ? root : now;
57 }
58 void query(const char* str) {
59     node* p = root, *tmp;
60     int tail = 0;
61     while(*str) {
62         tmp = p = goStatus(p, code(*str));
63         while(tmp != root && tmp->count != -1) {
64             q[tail++] = tmp;
65             tmp->count--;
66             tmp = tmp->fail;
67         }
68         str++;
69     }
70 }

```

1.2 Suffix Array

```

1 const int MAXN = 50001;
2 int sfx[MAXN], temp[MAXN], key[MAXN][2];
3 int _rank[MAXN], bucket[MAXN], height[MAXN];
4 // _rank from 0 to n-1
5 void radixSort(int* in, int n, int idx, int* out) {
6     memset(bucket, 0, sizeof(int) * (n + 1));
7     for(int i = 0; i < n; i++) bucket[key[i][idx]]++;
8     for(int i = 1; i <= n; i++) bucket[i] += bucket[i - 1];
9     for(int i = n - 1; i >= 0; i--) out[--bucket[key[i][idx]]] = in[i];
10 }
11 #define KEY0(i) key[i][0]
12 #define KEY1(i) key[i][1]
13 int cmp(int i, int j) {
14     return KEY0(i) == KEY0(j) ? KEY1(i) < KEY1(j) : KEY0(i) < KEY0(j);
15 }
16 /*text can't contain 0, 0 is used as terminal*/
17 void buildSA(const char* text, int n) {
18     for(int i = 0; i < n; i++)
19         sfx[i] = i, key[i][0] = text[i], key[i][1] = 0;
20     sort(sfx, sfx + n, cmp);
21     for(int i = 0; i < n; i++) key[i][0] = text[sfx[i]];
22     int wid = 1;
23     while(wid < n) {
24         _rank[sfx[0]] = 0;
25         for(int i = 1; i < n; i++)
26             _rank[sfx[i]] = _rank[sfx[i - 1]] + cmp(i - 1, i);
27         for(int i = 0; i < n; i++) {
28             sfx[i] = i;
29             key[i][1] = i + wid < n ? _rank[i + wid] : 0;
30         }
31         radixSort(sfx, n, 1, temp);
32         for(int i = 0; i < n; i++) key[i][0] = _rank[temp[i]];
33         radixSort(temp, n, 0, sfx);
34         for(int i = 0; i < n; i++) key[i][0] = _rank[sfx[i]];

```

```

35     for(int i = 0; i < n; i++)
36         key[i][1] = wid + sfx[i] < n ? _rank[sfx[i] + wid] : 0;
37     wid <= 1;
38 }
39 }
40 void calHeight(const char* text, int* _rank, int n) {
41     //height[i] = lcp(suffix(sa[i - 1]), suffix(sa[i]))
42     for(int i = 0; i < n; i++) _rank[sfx[i]] = i;
43     height[0] = 0;
44     for(int i = 0, k = 0, j; i < n; i++) {
45         if(_rank[i] != 0) {
46             if(k > 0) k--;
47             for (j = sfx[_rank[i] - 1]; text[i + k] == text[j + k]; k++);
48             height[_rank[i]] = k;
49         }
50     }
51 }
52 int RMQ[MAXN][20];
53 //n = len(text), height[0] means nothing
54 void buildRMQ(int n, int* height) {
55     for(int i = 1; i <= n; i++) RMQ[i][0] = height[i - 1];
56     for (int j = 1; j <= log(n + 0.00) / log(2.0); j++)
57         for (int i = 1; i + (1 << j) - 1 <= n; i++)
58             RMQ[i][j] = min(RMQ[i][j - 1], RMQ[i + (1 << (j - 1))][j - 1]);
59 }
60 int queryRMQ(int a, int b) {
61     int len = log(b - a + 1.0) / log(2.0);
62     return min(RMQ[a][len], RMQ[b - (1 << len) + 1][len]);
63 }
64 int queryLCP(int a, int b) {
65     a = _rank[a] + 1, b = _rank[b] + 1;
66     if(a > b) swap(a, b);
67     return queryRMQ(a + 1, b);
68 }

```

1.3 Suffix Automaton

```

1 namespace SAM {
2     const int MAXN = 600000;
3     struct Node {
4         Node *ch[26], *f; int l;
5     } a[MAXN], *root, *acc, *ptr;
6     void Initial() {
7         memset(a, 0, sizeof(a));
8         acc = root = a, ptr = a + 1;
9     }
10    void AddSuffix(int x) {
11        using namespace std;
12        Node * cur = ptr++, *fail = acc;
13        cur->l = acc->l + 1; acc = cur;
14        for(; fail && !fail->ch[x]; fail = fail->f)
15            fail->ch[x] = cur;
16        if(!fail) {
17            cur->f = root;
18        } else if (fail->l + 1 == fail->ch[x]->l) {
19            cur->f = fail->ch[x];
20        } else {
21            Node* r = ptr++, *q = fail->ch[x];
22            *r = *q, r->l = fail->l + 1;
23            cur->f = q->f = r;
24            for(; fail && fail->ch[x] == q; fail = fail->f)
25                fail->ch[x] = r;
26        }
27    }
28    int lcs(const char * src, const char * dest) {
29        Initial();
30        int n = strlen(src), m = strlen(dest), ans = 0, mid = 0;
31        Node * acc = root;
32        for(int i = 0; i < n; i++) {
33            SAM::AddSuffix(src[i] - 'a');
34        }
35        for(int i = 0; i < m; ++i) {
36            int v = dest[i] - 'a';
37            if(acc->ch[v]) {
38                ++mid;
39                acc = acc->ch[v];
40            } else {
41                for(; acc && !acc->ch[v]; acc = acc->f);
42                mid = acc ? acc->l + 1 : 0;
43                acc = acc ? acc->ch[v] : root;
44            }
45            ans = max(ans, mid);
46        }
47        return ans;
48    }
49 }

```

1.4 KMP

```

1 //be careful with mod string and main string
2 void prefix(const char *mode, int *next) {
3     int m = strlen(mode), k = -1, i;
4     next[0] = -1;
5     for (i = 1; i < m; i++) {
6         while (k > -1 && mode[k + 1] != mode[i]) k = next[k];
7         if (mode[k + 1] == mode[i]) k++;
8         next[i] = k;
9     }
10 }
11 int KMP(const char *main, const char *mode) {
12     int n = strlen(main), m = strlen(mode), q = -1, ans = 0;
13     int next[LEN], i;
14     prefix(mode, next);
15     for (i = 0; i < n; i++) {
16         while (q > -1 && mode[q + 1] != main[i]) q = next[q];
17         if (mode[q + 1] == main[i]) q++;
18         if (q == m - 1) {
19             ans++;
20             q = next[q];
21         }
22     }
23 }

```

```

21     }
22 }
23 return ans;
24 }

```

1.5 Algorithm Z

```

1 #include <math>
2 #include <algorithm>
3 #include <stdio>
4 #include <string>
5 using namespace std;
6 void get_suffix(const char* sub, int len, int next[]) {
7     //extend[i] = len(lcp(sub, sub.substr(i)))
8     int pos = 1, j = 0;
9     while(sub[j + 1] == sub[j]) j++;
10    next[0] = len, next[pos] = j;
11    for(int i = 2; i < len; i++) {
12        int ll = pos + next[pos], cur = next[i - pos];
13        if(ll > i + cur) {
14            next[i] = cur;
15        } else {
16            j = max(ll - i, 0);
17            while(sub[i + j] == sub[j] && i + j < len) j++;
18            next[i] = j;
19            pos = i;
20        }
21    }
22 }
23 void extend_kmp(const char* str, int n, const char* sub, int m,
24     int extend[], int next[]) {
25     get_suffix(sub, m, next);
26     int j = 0, pos = 0;
27     while(str[j] == sub[j] && j < n && j < m) j++;
28     extend[0] = j;
29     for(int i = 1; i < n; i++) {
30         int ll = pos + extend[pos], cur = next[i - pos];
31         if(ll > i + cur) {
32             extend[i] = cur;
33         } else {
34             j = max(ll - i, 0);
35             while(str[i + j] == sub[j] && i + j < n && j < m) j++;
36             extend[i] = j;
37             pos = i;
38         }
39     }
40 }

```

2 Network Flow

2.1 Max flow

```

1 const int V = 1010;
2 const int E = V*V*2;
3 const int INF = 1<<29;
4 typedef struct Edge{
5     int v, cap, flow;
6     Edge *next, *re;
7 }Edge;
8 class MaxFlow{
9 public:
10    Edge edge[E], *adj[V], *pre[V], *arc[V];
11    int e, n, d[V], q[V], numb[V];
12    void Init(int x){
13        n = x;
14        for (int i = 0; i < n; ++i) adj[i] = NULL;
15        e = 0;
16    }
17    void Addedge(int x, int y, int f) {
18        edge[e].v = y, edge[e].cap = f, edge[e].next = adj[x], edge[e].
19        re = &edge[e+1]; adj[x] = &edge[e++];
20        edge[e].v = x, edge[e].cap = 0, edge[e].next = adj[y], edge[e].
21        re = &edge[e-1]; adj[y] = &edge[e++];
22    }
23    void Bfs(int v) {
24        int front = 0, rear = 0, r = 0, dis = 0;
25        for (int i = 0; i < n; ++i) d[i] = n, numb[i] = 0;
26        d[v] = 0; ++numb[0];
27        q[rear++] = v;
28        while (front != rear) {
29            if (front == r) ++dis, r = rear;
30            v = q[front++];
31            for (Edge *i = adj[v]; i != NULL; i = i->next) {
32                int t = i->v;
33                if (d[t] == n) d[t] = dis, q[rear++] = t, ++numb[dis];
34            }
35        }
36    }
37    int Maxflow(int s, int t){
38        int ret = 0, i, j;
39        Bfs(t);
40        for (i = 0; i < n; ++i) pre[i] = NULL, arc[i] = adj[i];
41        for (i = 0; i < e; ++i) edge[i].flow = edge[i].cap;
42        i = s;
43        while (d[s] < n) {
44            while (arc[i] && (d[i] != d[arc[i]->v]+1 || !arc[i]->flow))
45                arc[i] = arc[i]->next;
46            if (arc[i]) {
47                j = arc[i]->v;
48                pre[j] = arc[i];
49                i = j;
50                if (i == t) {
51                    int update = INF;
52                    for (Edge *p = pre[t]; p != NULL; p = pre[p->re->v])
53                        checkmin(update, p->flow);
54                    ret += update;
55                }
56            }
57        }
58    }
59 }

```

```

51     for (Edge *p = pre[t]; p != NULL; p = pre[p->re->v]) p->flow
52         -= update, p->re->flow += update;
53     }
54 }
55 else {
56     int min = n - 1;
57     for (Edge *p = adj[i]; p != NULL; p = p->next) if (p->flow)
58         checkmin(min, d[p->v]);
59     if (--numb[d[i]] == 0) return ret;
60     d[i] = min + 1;
61     ++numb[d[i]];
62     arc[i] = adj[i];
63     if (i != s) i = pre[i]->re->v;
64 }
65 return ret;
66 }
67 };

```

2.2 Cost flow

```

1 using namespace std;
2 typedef long long USETYPE;
3 const USETYPE INF = numeric_limits<USETYPE>::max(); //limits>
4 template<typename T = int>
5 class mincost {
6 private:
7     const static int N = 1000;
8     const static int E = 100000;
9     struct edge {
10         int u, v;
11         T cost, cap;
12         edge *nxt;
13     } pool[E], *g[N], *pp, *pre[N];
14     T dist[N];
15
16     bool SPFA(int n, int s, int t) {
17         fill(dist, dist + n, INF);
18         int tail = 0, q[N] = {s};
19         dist[s] = 0;
20         bool vst[N] = {false};
21         vst[s] = true;
22         for (int i = 0; i <= tail; i++) {
23             int u = q[i % N];
24             for (edge *j = g[u]; j != NULL; j = j->nxt) {
25                 int v = j->v;
26                 if (j->cap && dist[u] != INF && dist[v] > dist[u] + j->cost) {
27                     dist[v] = dist[u] + j->cost;
28                     pre[v] = j;
29                     if (!vst[v]) {
30                         tail++;
31                         q[tail % N] = v;
32                         vst[v] = true;
33                     }
34                 }
35             }
36             vst[u] = false;
37         }
38         return dist[t] < INF;
39     }
40 public:
41     #define OP(i) (((i) - pool) ^ 1)
42     void addedge(int u, int v, T cap, T cost) {
43         pp->u = u, pp->v = v;
44         pp->cost = cost, pp->cap = cap;
45         pp->nxt = g[u], g[u] = pp++;
46     }
47     void initialize() {
48         CC(g, 0);
49         pp = pool;
50     }
51     pair<T, T> mincostflow(int n, int s, int t) {
52         T flow = 0, cost = 0;
53         while (SPFA(n, s, t)) {
54             T minf = INF;
55             for (int i = t; i != s; i = pre[i]->u)
56                 minf = min(minf, pre[i]->cap);
57             for (int i = t; i != s; i = pre[i]->u) {
58                 pre[i]->cap -= minf;
59                 pool[OP(pre[i])].cap += minf;
60                 cost += minf * pre[i]->cost;
61             }
62             flow += minf;
63         }
64         return make_pair(flow, cost);
65     }
66 };

```

3 Data Structure

3.1 DLX exact cover

```

1 const int SIZE = 16, SQRTSIZE = 4; //here
2 const int ALLSIZE = SIZE * SIZE, ROW = SIZE * SIZE * SIZE;
3 const int INF = 100000000, COL = SIZE * SIZE * 4;
4 const int N = ROW * COL, HEAD = 0;
5 #define BLOCK(r, c) ((r) * SQRTSIZE + c)
6 #define CROW(r, c, k) ((r) + (c) * SIZE + (k) * SIZE * SIZE)
7 #define ROWCOL(i, j) ((i) * SIZE + (j))
8 #define ROWCOLOR(i, k) (ALLSIZE + (i) * SIZE + k)
9 #define COLCOLOR(j, k) (2 * ALLSIZE + (j) * SIZE + k)
10 #define BLOCKCOLOR(i, j, k) (3 * ALLSIZE + BLOCK((i / SQRTSIZE), (j / SQRTSIZE)) * SIZE + k)
11 int maps[ROW][COL], ans[N];

```

```

12 char sudoku[SIZE][SIZE];
13 int r[N], l[N], u[N], d[N], c[N], s[N];
14 int n, m, ansd, row[N];
15 void resume(const int col) {
16     for (int i = u[col]; i != col; i = u[i]) {
17         for (int j = l[i]; j != i; j = l[j]) {
18             u[d[j]] = j;
19             d[u[j]] = j;
20             s[c[j]]++;
21         }
22     }
23     r[l[col]] = col;
24     l[r[col]] = col;
25 }
26 void cover(const int col) {
27     r[l[col]] = r[col];
28     l[r[col]] = l[col];
29     for (int i = d[col]; i != col; i = d[i]) {
30         for (int j = r[i]; j != i; j = r[j]) {
31             u[d[j]] = u[j];
32             d[u[j]] = d[j];
33             s[c[j]]--;
34         }
35     }
36 }
37 void initialize(int n, int m) {
38     l[HEAD] = m;
39     r[HEAD] = l;
40     for (int i = 1; i <= m; i++) {
41         if (l == m) {
42             r[i] = HEAD;
43         } else {
44             r[i] = i + 1;
45         }
46         l[i] = i - 1;
47         c[i] = u[i] = d[i] = i;
48         s[i] = 0;
49     }
50     int size = m;
51     for (int i = 1; i <= n; i++) {
52         int first = 0;
53         for (int j = 1; j <= m; j++) {
54             if (maps[i - 1][j - 1] == 0) continue;
55             size++;
56             int tmp = u[j];
57             u[j] = size; d[tmp] = size;
58             d[size] = j; u[size] = tmp;
59             if (!first) {
60                 first = size;
61                 l[size] = r[size] = size;
62             } else {
63                 tmp = l[first];
64                 r[tmp] = size;
65                 l[size] = tmp;
66                 l[first] = size;
67                 r[size] = first;
68             }
69             row[size] = i;
70             s[j]++;
71             c[size] = j;
72         }
73     }
74 }
75 bool dfs(int depth) {
76     if (r[HEAD] == HEAD) {
77         ansd = depth;
78         return true;
79     }
80     int minn = INF, v;
81     for (int i = r[HEAD]; i != HEAD; i = r[i]) {
82         if (s[i] < minn) {
83             v = i;
84             minn = s[i];
85         }
86     }
87     cover(v);
88     for (int i = d[v]; i != v; i = d[i]) {
89         for (int j = r[i]; j != i; j = r[j])
90             cover(c[j]);
91         ans[depth] = row[i] - 1;
92         if (dfs(depth + 1))
93             return true;
94         for (int j = l[i]; j != i; j = l[j])
95             resume(c[j]);
96     }
97     resume(v);
98     ans[depth] = -1;
99     return false;
100 }
101
102 int main() {
103     n = ROW;
104     m = COL;
105     while (scanf("%c", &sudoku[0][0]) == 1) {
106         for (int i = 0; i < SIZE; i++) {
107             for (int j = 0; j < SIZE; j++) {
108                 if (i + j) scanf("%c", &sudoku[i][j]);
109             }
110         }
111         memset(maps, 0, sizeof(maps));
112         for (int i = 0; i < SIZE; i++) {
113             for (int j = 0; j < SIZE; j++) {
114                 if (sudoku[i][j] == '-') {
115                     for (int k = 0; k < SIZE; k++) {
116                         maps[CROW(i, j, k)][ROWCOL(i, j)] = 1;
117                         maps[CROW(i, j, k)][ROWCOLOR(i, k)] = 1;
118                         maps[CROW(i, j, k)][COLCOLOR(j, k)] = 1;
119                         maps[CROW(i, j, k)][BLOCKCOLOR(i, j, k)] = 1;
120                     }
121                 } else {
122                     int k = sudoku[i][j] - 'A'; //here
123                     maps[CROW(i, j, k)][ROWCOL(i, j)] = 1;
124                     maps[CROW(i, j, k)][ROWCOLOR(i, k)] = 1;
125                     maps[CROW(i, j, k)][COLCOLOR(j, k)] = 1;
126                     maps[CROW(i, j, k)][BLOCKCOLOR(i, j, k)] = 1;
127                 }
128             }
129             initialize(n, m);
130         }
131         if (dfs(0)) {

```

```

132     for (int i = 0; i < ansd; i++)
133         sudoku[ans[i] % SIZE][ans[i] % ALLSIZE / SIZE] = ans[i]
134             / ALLSIZE + 'A'; //here
135     for (int i = 0; i < SIZE; i++) {
136         for (int j = 0; j < SIZE; j++)
137             putchar(sudoku[i][j]);
138     }
139     puts("");
140 }
141 return 0;
142 }
143 }

```

3.2 DLX fuzzy cover

```

1  const int ROW = 56;
2  const int COL = 56;
3  const int N = ROW * COL, HEAD = 0;
4  const int INF = 1000000000;
5  int maps[ROW][COL], ansq[ROW], row[N];
6  int s[COL], u[N], d[N], l[N], r[N], c[N];
7  void build(int n, int m) {
8      r[HEAD] = 1;
9      l[HEAD] = m;
10     for (int i = 1; i <= m; i++) {
11         l[i] = i - 1;
12         r[i] = (i + 1) % (m + 1);
13         c[i] = d[i] = u[i] = i;
14         s[i] = 0;
15     }
16     int size = m;
17     for (int i = 1; i <= n; i++) {
18         int first = 0;
19         for (int j = 1; j <= m; j++) {
20             if (!maps[i - 1][j - 1]) continue;
21             size++;
22             d[u[j]] = size;
23             u[size] = u[j];
24             d[size] = j;
25             u[j] = size;
26             if (!first) {
27                 first = size;
28                 l[size] = size;
29                 r[size] = size;
30             } else {
31                 l[size] = l[first];
32                 r[size] = first;
33                 r[l[first]] = size;
34                 l[first] = size;
35             }
36             c[size] = j;
37             s[j]++;
38         }
39     }
40 }
41 inline void coverc(int col) {
42     for (int i = d[col]; i != col; i = d[i]) {
43         r[l[i]] = r[i];
44         l[r[i]] = l[i];
45     }
46 }
47 inline void resumec(int col) {
48     for (int i = u[col]; i != col; i = u[i]) {
49         l[r[i]] = i;
50         r[l[i]] = i;
51     }
52 }
53 bool vis[COL];
54 int H() {
55     int cnt = 0;
56     memset(vis, 0, sizeof(vis));
57     for (int i = r[HEAD]; i != HEAD; i = r[i]) {
58         if (vis[i]) continue;
59         cnt++;
60         vis[i] = 1;
61         for (int j = d[i]; j != i; j = d[j])
62             for (int k = r[j]; k != j; k = r[k])
63                 vis[c[k]] = 1;
64     }
65     return cnt;
66 }
67 int cut, nextcut;
68 bool dfs(int dep) {
69     if (!r[HEAD]) return true;
70     int now, minn = ROW;
71     for (int i = r[HEAD]; i != HEAD; i = r[i])
72         if (minn > s[i]) {
73             minn = s[i];
74             now = i;
75         }
76     for (int j = d[now]; j != now; j = d[j]) {
77         //ansq[dep] = row[rp];
78         coverc(j);
79         for (int i = r[j]; i != j; i = r[i])
80             coverc(i);
81         int tmp = dep + 1 + H();
82         if (tmp > cut) nextcut = min(tmp, nextcut);
83         else if (dfs(dep + 1)) return true;
84         for (int i = l[j]; i != j; i = l[i])
85             resumec(i);
86         resumec(j);
87     }
88     return false;
89 }
90 int IDAstar(int n) {
91     cut = H();
92     nextcut = n;
93     memset(vis, 0, sizeof(vis));
94     while (!dfs(HEAD)) {
95         cut = nextcut;
96         nextcut = n;
97     }
98     return cut;
99 }

```

3.3 Partition Tree

```

1  /* NlogN find Kth number in any interval */
2  class partition_tree {
3  private:
4      static const int N = 100005;
5      static const int DEPTH = 20;
6      int tree[DEPTH][N * 4], sorted[N];
7      int toleft[DEPTH][N * 4];
8      int n;
9  public:
10     void initialize(int n, int *array) {
11         this->n = n;
12         for (int i = 1; i <= n; i++)
13             sorted[i] = tree[0][i] = array[i];
14         sort(sorted + 1, sorted + n + 1);
15     }
16     void build(int l, int r, int depth) {
17         if (l == r) return;
18         int mid = (l + r) / 2, same = 0, less = 0;
19         for (int i = l; i <= r; i++)
20             less += (tree[depth][i] < sorted[mid]);
21         same = mid - l + 1 - less;
22         int lpos = l, rpos = mid + 1;
23         for (int i = l; i <= r; i++) {
24             int w = tree[depth][i];
25             if (w < sorted[mid]) tree[depth + 1][lpos++] = w;
26             else if (w == sorted[mid] && same) {
27                 tree[depth + 1][lpos++] = w;
28                 same--;
29             }
30             else
31                 tree[depth + 1][rpos++] = w;
32             toleft[depth][i] = toleft[depth][l - 1] + lpos - 1;
33         }
34         build(l, mid, depth + 1);
35         build(mid + 1, r, depth + 1);
36     }
37     // ptree.query(l, n, a, b, 0, k) th kth number of [a, b]
38     int query(int L, int R, int l, int r, int depth, int k) {
39         if (l == r) return tree[depth][l];
40         int cnt, mid = (R + L) / 2, tmp1, tmp2;
41         cnt = toleft[depth][r] - toleft[depth][l - 1];
42         if (cnt >= k) {
43             tmp1 = L + toleft[depth][l - 1] - toleft[depth][L - 1];
44             tmp2 = tmp1 + cnt - 1;
45             return query(L, mid, tmp1, tmp2, depth + 1, k);
46         } else {
47             tmp2 = r + toleft[depth][R] - toleft[depth][r];
48             tmp1 = tmp2 - (r - l - cnt);
49             return query(mid + 1, R, tmp1, tmp2, depth + 1, k - cnt);
50         }
51     }
52 };

```

3.4 Leftist Tree

```

1  #define DIST(v) ((v == NULL) ? -1 : (v->dist))
2  template<typename T, class Compare = greater<T> >
3  class LeftistTree {
4  private:
5      class node {
6      public:
7          T v;
8          int dist;
9          node *rr, *ll;
10         node() { rr = ll = NULL; dist = 0; }
11         node(T v) { this->v = v; rr = ll = NULL; dist = 0; }
12     };
13     node* root;
14     int s;
15     Compare _compare;
16     node* Merge(node* left, node* right) {
17         if (left == NULL) return right;
18         if (right == NULL) return left;
19         if (_compare(right->v, left->v)) swap(left, right);
20         left->rr = Merge(left->rr, right);
21         if (DIST(left->rr) > DIST(left->ll)) swap(left->ll, left->rr);
22         left->dist = DIST(left->rr) + 1;
23         return left;
24     }
25     void Clear(node*& root) {
26         if (root == NULL) return;
27         Clear(root->ll);
28         Clear(root->rr);
29         delete root;
30         root = NULL;
31     }
32 public:
33     LeftistTree() { root = NULL; s = 0; }
34     ~LeftistTree() { Clear(root); }
35     void Push(T v) {
36         node * newNode = new node(v);
37         root = Merge(newNode, root);
38         s++;
39     }
40     void Clear() { Clear(root); }
41     int Size() { return this->s; }
42     T Top() { return root->v; }
43     void Pop() {
44         node *tmp = root;
45         root = Merge(root->ll, root->rr);
46         delete tmp;
47         s--;
48     }
49     void Merge(LeftistTree<T>& tree) {
50         this->root = Merge(root, tree.root);
51         s += tree.s;
52         tree.root = NULL;
53     }
54 };

```

3.5 Cartesian Tree

```

1 #include <iostream>
2 #include <stdio>
3 #include <string>
4 #include <cmath>
5 #include <algorithm>
6 #include <string>
7 using namespace std;
8 const int N = 100000;
9 struct node {
10     int key, value, id;
11     bool operator < (const nodes oth) const {
12         return key < oth.key;
13     }
14 } nodes[N];
15 /*lt[i] is nodes[i]'s left son, shouldn't sort again*/
16 int lt[N], rt[N], parent[N];
17 void rotate(int i) {
18     while(parent[i] != -1 && nodes[i].value < nodes[parent[i]].value) {
19         rt[parent[i]] = lt[i];
20         if(lt[i] != -1) parent[lt[i]] = parent[i];
21         lt[i] = parent[i];
22         int ff = parent[parent[i]];
23         if(ff != -1) {
24             parent[i] == lt[ff] ? lt[ff] = i : rt[ff] = i;
25         }
26         parent[i] = ff;
27         parent[lt[i]] = i;
28     }
29 }
30 int key[N], value[N], pos[N];
31 void build(int n) {
32     sort(nodes, nodes + n);
33     int rightmost = 0;
34     for(int i = 1; i < n; i++) {
35         pos[nodes[i].id] = i;
36         rt[rightmost] = i;
37         parent[i] = rightmost;
38         rightmost = i;
39         rotate(i);
40     }
41 }
42 #define V(i) (i == -1 ? 0 : nodes[i].id + 1)
43 int main() {
44     int n;
45     while(scanf("%d", &n) == 1) {
46         for(int i = 0; i < n; i++) {
47             scanf("%d %d", &nodes[i].key, &nodes[i].value);
48             nodes[i].id = i;
49             key[i] = nodes[i].key;
50             value[i] = nodes[i].value;
51             lt[i] = rt[i] = parent[i] = -1;
52         }
53         build(n);
54         printf("YES\n");
55         for(int i = 0; i < n; i++) {
56             printf("%d %d %d\n", V(parent[pos[i]]),
57                 V(lt[pos[i]]), V(rt[pos[i]]));
58         }
59     }
60     return 0;
61 }

```

3.6 Splay

```

1 struct node {
2     #define __JUDGE if(tot == 0) return
3     static const int INF = 1000000000;
4     node* ch[2], *pre;
5     int v, minn, tot, delta, flip;
6     node(int v, int tot, node* l, node* r, node* pre)
7         : pre(pre), v(v), minn(v), tot(tot), delta(0), flip(0) {
8         ch[0] = l, ch[1] = r;
9     }
10     inline int min_v() { return minn; }
11     inline int size() { return tot; }
12     void reverse() { __JUDGE; flip ^= 1; }
13     void add(int d) { __JUDGE; minn += d, delta += d, v += d; }
14     void push_down() {
15         __JUDGE;
16         if(delta) {
17             if(ch[0] -> tot) ch[0] -> add(delta);
18             if(ch[1] -> tot) ch[1] -> add(delta);
19         }
20         if(flip) {
21             swap(ch[0], ch[1]);
22             if(ch[0] -> tot) ch[0] -> reverse();
23             if(ch[1] -> tot) ch[1] -> reverse();
24         }
25         flip = delta = 0;
26     }
27     void push_up() {
28         __JUDGE;
29         tot = ch[0] -> size() + ch[1] -> size() + 1;
30         minn = min(v, min(ch[0] -> min_v(), ch[1] -> min_v()));
31     }
32 };
33 class splay_tree {
34 public:
35     splay_tree() {
36         root = null = new node(node::INF, 0, 0, 0, 0);
37     }
38     ~splay_tree() {
39         clear(root);
40         delete null;
41     }
42     // build(0, n + 1, val) make a sequence from 1 to n
43     void build(int l, int r, int* val) {
44         if(l > r) return; // and make sure val[0] = val[1] = INF;
45         build(l, r, root, null, val);
46     }
47     #define centre (root -> ch[1] -> ch[0])

```

```

48     int min_value(int a, int b) {
49         makeInterval(a, b);
50         return centre -> min_v();
51     }
52     void add_value(int a, int b, int value) {
53         makeInterval(a, b);
54         centre -> add(value);
55         splay(centre, null);
56     }
57     void reverse(int a, int b) {
58         if(a == b) return;
59         makeInterval(a, b);
60         centre -> reverse();
61         splay(centre, null);
62     }
63     void revolve(int a, int b, int c) { // c < b - a + 1
64         if(c == 0) return;
65         int len = b - a + 1;
66         reverse(a, a + len - c - 1);
67         reverse(a + len - c, b), reverse(a, b);
68     }
69     void insert(int a, int c) {
70         makeInterval(a + 1, a);
71         centre = new node(c, 1, null, null, root -> ch[1]);
72         root -> ch[1] -> push_up(), root -> push_up();
73         splay(centre, null);
74     }
75     void erase(int a) {
76         makeInterval(a, a);
77         delete centre;
78         centre = null;
79         root -> ch[1] -> push_up(), root -> ch[0] -> push_up();
80     }
81     #undef centre
82     void clear() { clear(root); }
83 private:
84     node* root, * null;
85     void clear(node* now) {
86         if(now == null) return;
87         clear(now -> ch[0]), clear(now -> ch[1]);
88         delete now;
89         now = null;
90     }
91     /* 0: right rotate, 1: left rotate */
92     void rotate(node* x, int type) {
93         node *y = x -> pre;
94         y -> push_down(), x -> push_down();
95         y -> ch[!type] = x -> ch[type];
96         if (x -> ch[!type] != null)
97             x -> ch[!type] -> pre = y;
98         x -> pre = y -> pre;
99         if (y -> pre != null) {
100             if (y -> pre -> ch[1] == y)
101                 y -> pre -> ch[1] = x;
102             else
103                 y -> pre -> ch[0] = x;
104         }
105         x -> ch[!type] = y, y -> pre = x;
106         if (y == root) root = x;
107         y -> push_up(), x -> push_up();
108     }
109     void splay(node* x, node* f) {
110         x -> push_down();
111         while(x -> pre != f) {
112             if (x -> pre -> pre == f) {
113                 if (x -> pre -> ch[0] == x)
114                     rotate(x, 1);
115                 else
116                     rotate(x, 0);
117             } else {
118                 node *y = x -> pre;
119                 node *z = y -> pre;
120                 if (z -> ch[0] == y) {
121                     if (y -> ch[0] == x) // 1
122                         rotate(y, 1), rotate(x, 1);
123                     else // z
124                         rotate(x, 0), rotate(x, 1);
125                 } else {
126                     if (y -> ch[1] == x) // 1
127                         rotate(y, 0), rotate(x, 0);
128                     else // z
129                         rotate(x, 1), rotate(x, 0);
130                 }
131             }
132             x -> push_up();
133         }
134     }
135     void build(int l, int r, node* now, node* pre, int* val) {
136         if(l > r) return;
137         int mid = (l + r) / 2;
138         now = new node(val[mid], 1, null, null, pre);
139         build(l, mid - 1, now -> ch[0], now, val);
140         build(mid + 1, r, now -> ch[1], now, val);
141         now -> push_up();
142     }
143     // the flag node is !not! included, be careful when make
144         interval
145     void findK(int k, node* pre) {
146         node* now = root;
147         while(true) {
148             now -> push_down();
149             int s = now -> ch[0] -> size();
150             if(s == k) break;
151             else if(s > k)
152                 now = now -> ch[0];
153             else {
154                 now = now -> ch[1];
155                 k -= s + 1;
156             }
157         }
158         splay(now, pre);
159     }
160     void makeInterval(int a, int b) {
161         findK(a - 1, null), findK(b + 1, root);
162     }
163 } tree;

```

4 Graph Theory

4.1 2-Satisfiability

```

1  /* 2-sat template node is from 0
2  * i and i^1 is a bool variable(true or false)
3  * conjunctive normal form with 2-sat
4  * x V y == 1 => edge(~x-->y) and edge(~y-->x)
5  * x V y == 0 => (~x V ~y) & (~y V ~x)
6  * x ^ y == (~x V ~y) & (x V y)
7  * x & y == 1 (x V x) & (y V y)
8  * x & y == 0 (~x V ~y) */
9  const int V = 20000, E = 20480 * 4;
10 const int RED = 1, BLUE = 2;
11 struct edge {
12     int v;
13     edge *nxt;
14 } pool[E], *g[V], *pp, *gscc[V];
15 int st[V], top, tms[V], pt;
16 bool reach[V];
17 int dfn[V], low[V], idx[V], sccCnt, depth;
18 int color[V], pre[V];
19 void addedge(int a, int b, edge *g[]) {
20     pp->v = b;
21     pp->nxt = g[a];
22     g[a] = pp++;
23 }
24 void initialize() {
25     memset(reach, 0, sizeof(reach));
26     memset(dfn, 0, sizeof(dfn));
27     memset(g, 0, sizeof(g));
28     top = sccCnt = depth = 0, pp = pool;
29 }
30 void dfs(int x) {
31     st[++top] = x;
32     dfn[x] = low[x] = ++depth;
33     int w;
34     for (edge *i = g[x]; i != NULL; i = i->nxt) {
35         w = i->v;
36         if (reach[w]) continue;
37         else if (dfn[w] == 0) {
38             dfs(w);
39             low[x] = min(low[x], low[w]);
40         }
41         else low[x] = min(low[x], dfn[w]);
42     }
43     if (low[x] == dfn[x]) {
44         sccCnt++;
45         do {
46             w = st[top--];
47             idx[w] = sccCnt - 1;
48             reach[w] = true;
49         } while (w != x);
50     }
51 }
52 void toposort(int v) {
53     reach[v] = true;
54     for (edge *i = gscc[v]; i != NULL; i = i->nxt)
55         if (!reach[i->v]) toposort(i->v);
56     tms[pt++] = v;
57 }
58 void build_regraph(int n) /*anti-graph*/ {
59     memset(gscc, 0, sizeof(gscc)); /*anti-graph scc
60     memset(pre, -1, sizeof(pre)); /*the new node to every scc
61     for (int i = 0; i < n; i++) {
62         if (pre[idx[i]] == -1) pre[idx[i]] = i;
63         for (edge *ptr = g[i]; ptr != NULL; ptr = ptr->nxt) {
64             int w = ptr->v;
65             if (idx[i] != idx[w]) addedge(idx[w], idx[i], gscc);
66         }
67     }
68 }
69 void becolor(int v) {
70     color[v] = BLUE;
71     for (edge *i = gscc[v]; i != NULL; i = i->nxt)
72         if (!color[i->v]) becolor(i->v);
73 }
74 void output(int n) /* Topological Sort */ {
75     memset(color, 0, sizeof(color)); /*color white
76     for (int i = 0; i < pt; i++) {
77         if (!color[tms[i]]) /*color as Topological order*/ {
78             color[tms[i]] = RED;
79             int v = idx[pre[tms[i]] ^ 1];
80             if (color[v] == 0) becolor(v);
81         }
82     }
83     for (int i = 0; i < n; i += 2) {
84         if (color[idx[i]] == RED)
85             printf("%d\n", i + 1);
86         else /*if (color[idx[i ^ 1]] == RED)
87             printf("%d\n", (i ^ 1) + 1);
88     }
89 }
90 bool solve(int n) /*i and i^1 can not be in the same scc */ {
91     for (int i = 0; i < n; i++) if (!reach[i]) dfs(i);
92     for (int i = 0; i < n; i++) if (idx[i] == idx[i ^ 1]) return false;
93     build_regraph(n);
94     pt = 0;
95     memset(reach, 0, sizeof(reach));
96     for (int i = 0; i < sccCnt; i++)
97         if (!reach[i]) toposort(i);
98     reverse(tms, tms + pt);
99     output(n);
100    return true;
101 }
102 int main() {
103     int n, m;
104     while (scanf("%d %d", &n, &m) == 2) {
105         initialize();
106         n *= 2;
107         while (m--) {
108             int a, b;
109             scanf("%d %d", &a, &b);
110             a--, b--;
111             addedge(a, b ^ 1, g);

```

```

112         addedge(b, a ^ 1, g);
113     }
114     if (!solve(n)) printf("NIE\n");
115 }
116     return 0;
117 }

```

4.2 Edge Cut

```

1  /*HOJ2360
2  * idx is new node of the tree
3  * pool should be big enough */
4  const int SIZE = 5000, ROOT = 0, E = 80000;
5  struct edge {
6      int v, id;
7      edge *nxt;
8  } pool[E], *g[SIZE], *pp, *bg[SIZE];
9  stack<int> st;
10 bool flag[E]; /*label the edge in case of multi-edge
11 int depth, ebcc, dfn[SIZE], low[SIZE], idx[SIZE];
12 void initialize() {
13     memset(g, 0, sizeof(g));
14     memset(flag, 0, sizeof(flag));
15     memset(bg, 0, sizeof(bg));
16     memset(dfn, 0, sizeof(dfn));
17     pp = pool, depth = 1, ebcc = 0;
18 }
19 void addedge(int v, int w, edge *g[], int id = 0) {
20     pp->v = w, pp->nxt = g[v];
21     pp->id = id, g[v] = pp++;
22 }
23 void dfs(int v) {
24     st.push(v);
25     dfn[v] = low[v] = depth++;
26     int w, x;
27     for (edge* i = g[v]; i != NULL; i = i->nxt) {
28         w = i->v;
29         if (flag[i->id]) continue;
30         flag[i->id] = true;
31         if (dfn[w]) low[v] = min(low[v], dfn[w]);
32         else {
33             dfs(w);
34             low[v] = min(low[v], low[w]);
35             if (low[w] > dfn[v]) {
36                 ebcc++;
37                 do {
38                     x = st.top();
39                     st.pop();
40                     idx[x] = ebcc;
41                 } while (x != w);
42             }
43         }
44     }
45 }
46 void solve() /*find out the cut and build the tree*/ {
47     dfs(ROOT); /*ROOT = 0 as usual
48     if (!st.empty()) ebcc++;
49     while (!st.empty()) {
50         idx[st.top()] = ebcc;
51         st.pop();
52     }
53 }

```

4.3 Vertex Cut

```

1  /* hoj 1789 Electricity
2  * the graph is not connected
3  * cnt records the number of BBC, it's an cut P if != 0*/
4  const int V = 10000;
5  vector<int> adj[V];
6  int low[V], dfn[V], cnt[V], depth;
7  void initialize(int n) {
8      REP(i, 0, n) adj[i].clear();
9      CC(cnt, 0); CC(dfn, 0);
10     depth = 0;
11 }
12 void dfs(int x, const int ROOT) {
13     {
14         low[x] = dfn[x] = ++depth;
15         int s = adj[x].size(), w, num = 0;
16         REP(i, 0, s)
17         {
18             w = adj[x][i];
19             if (!dfn[w])
20             {
21                 num++;
22                 dfs(w, ROOT);
23                 low[w] = min(low[w], low[x]);
24                 if (x == ROOT && num >= 2)
25                     cnt[x]++;
26                 if (x != ROOT && dfn[x] <= low[w])
27                     cnt[x]++;
28             }
29             else low[x] = min(low[x], dfn[w]);
30         }
31     }
32 }
33 int solve(int n) {
34     {
35         int cc = 0;
36         REP(i, 0, n)
37         {
38             if (dfn[i] == 0)
39             {
40                 dfs(i, i);
41                 cc++;
42             }
43         }
44         return cc;

```

```

45 }
46 int main()
47 {
48     int n, m, x, y;
49
50     while (scanf("%d %d", &n, &m) == 2 && n + m)
51     {
52         initialize(n);
53         REP(i, 0, m)
54         {
55             scanf("%d %d", &x, &y);
56             adj[x].push_back(y);
57             adj[y].push_back(x);
58         }
59         int ans = solve(n);
60         if (m == 0) printf("%d\n", n - 1);
61         else printf("%d\n", ans + *max_element(cnt, cnt + n));
62     }
63     return 0;
64 }

```

4.4 Hopcroft Karp

```

1  const int N = 500, M = 500, INF = 1 << 29;
2  bool g[N][M], chk[M];
3  int Mx[N], My[M], dx[N], dy[M], dis;
4  bool searchP(int n, int m) {
5      queue<int> Q;
6      dis = INF;
7      CC(dx, -1); CC(dy, -1);
8      for (int i = 0; i < n; ++i)
9          if (Mx[i] == -1) {
10             Q.push(i);
11             dx[i] = 0;
12         }
13     while (!Q.empty()) {
14         int u = Q.front();
15         Q.pop();
16         if (dx[u] > dis) break;
17         for (int v = 0; v < m; ++v)
18             if (g[u][v] && dy[v] == -1) {
19                 dy[v] = dx[u] + 1;
20                 if (My[v] == -1) dis = dy[v];
21             }
22         else {
23             dx[My[v]] = dy[v] + 1;
24             Q.push(My[v]);
25         }
26     }
27     return dis != INF;
28 }
29 bool Augment(int u, const int m) {
30     REP(v, 0, m)
31         if (g[u][v] && !chk[v] && dy[v] == dx[u] + 1) {
32             chk[v] = true;
33             if (My[v] != -1 && dy[v] == dis) continue;
34             if (My[v] == -1 || Augment(My[v], m)) {
35                 My[v] = u;
36                 Mx[u] = v;
37                 return true;
38             }
39         }
40     return false;
41 }
42 int MaxMatch(int n, int m) {
43     int ans = 0;
44     CC(Mx, -1); CC(My, -1);
45     while (searchP(n, m)) {
46         CC(chk, false);
47         REP(i, 0, n)
48             if (Mx[i] == -1 && Augment(i, m)) ++ans;
49     }
50     return ans;
51 }

```

4.5 Hungary Algorithm

```

1  /*1. simple maximum match
2  2.min path cover of DAG = |V| - max match
3  define: find some edge cover all the nodes
4  build PXP Bipartite graph do the maximum match
5  3.min path cover of Bipartite graph = max match
6  define : find some point cover all the edge(konig)
7  4.chessBoard is a Bipartite graph,then you know
8  5.max independent set(Bipartite graph)=|V| - max match
9  v is all the point of (set A and set B)
10 6.largest cLOUD(Bipartite graph) = max independent set of
    Complement*/
11 const int V = 201, E = 10000;
12 vector<int> adj[V];
13 int ym[V], chk[V];
14 bool find_path(int x) {
15     FOREACH(adj[x], i) {
16         if (chk[*i]) continue;
17         chk[*i] = true;
18         if (ym[*i] == -1 || find_path(ym[*i])) {
19             ym[*i] = x;
20             return true;
21         }
22     }
23     return false;
24 }
25 int solve(int n) {
26     CC(ym, -1);
27     int res = 0;
28     for (int i = 0; i < n; ++i) {
29         memset(chk, 0, sizeof(chk));
30         if (find_path(i)) res++;
31     }

```

```

32     return res;
33 }

```

4.6 KM

```

1 struct Graph {
2     int ny, nx;
3     double w[N][N];
4     double lx[N], ly[N];
5     int linky[N];
6     int visx[N], visy[N];
7     double slack[N];
8     void init(int nn, int mm) {
9         nx = nn;
10        ny = mm;
11    }
12    bool find(int x) {
13        visx[x] = 1;
14        for (int y = 1; y <= ny; ++y) {
15            if (visy[y]) continue;
16            double t = lx[x] + ly[y] - w[x][y];
17            if (t < eps) {
18                visy[y] = 1;
19                if (linky[y] == -1 || find(linky[y])) {
20                    linky[y] = x;
21                    return true;
22                }
23            } else if (slack[y] > t) {
24                slack[y] = t;
25            }
26        }
27        return false;
28    }
29    double KM() {
30        memset(linky, -1, sizeof(linky));
31        for (int i = 1; i <= nx; ++i) lx[i] = -INF;
32        memset(ly, 0, sizeof(ly));
33        for (int i = 1; i <= nx; ++i)
34            for (int j = 1; j <= ny; ++j)
35                if (w[i][j] > lx[i]) lx[i] = w[i][j];
36        for (int x = 1; x <= nx; ++x) {
37            for (int i = 1; i <= ny; ++i) slack[i] = INF;
38            while (true) {
39                memset(visx, 0, sizeof(visx));
40                memset(visy, 0, sizeof(visy));
41                if (find(x)) break;
42                double d = INF;
43                for (int i = 1; i <= ny; ++i)
44                    if (!visy[i]) d = min(d, slack[i]);
45                if (d == INF) return -1;
46                for (int i = 1; i <= nx; ++i)
47                    if (visx[i]) lx[i] -= d;
48                for (int i = 1; i <= ny; ++i)
49                    if (visy[i]) ly[i] += d;
50                else slack[i] -= d;
51            }
52        }
53        int cnt = 0;
54        for (int i = 1; i <= ny; ++i)
55            if (linky[i] != -1) cnt++;
56        if (cnt != nx) return -1;
57        double tp = 0;
58        for (int i = 1; i <= ny; ++i)
59            if (linky[i] != -1) tp += w[linky[i]][i];
60        return tp;
61    }
62 }g;

```

4.7 Stable Marriage

```

1  /* boy[i][j] gg[i] to mm[j]
2  * girl[i][j] mm[i] to gg[j]*/
3  const int N = 26;
4  const int M = 128;
5  int boy[N][N], girl[N][N];
6  int my[N], mx[N], now[N];
7  void Gale_Shapley(int n) {
8      queue<int> q;
9      for (int i = 0; i < n; ++i) q.push(i);
10     while (!q.empty()) {
11         int i = q.front(); q.pop();
12         int j = now[i]++; mm = boy[i][j];
13         if (my[mm] == -1 || girl[mm][my[mm]] > girl[mm][i]) {
14             if (my[mm] != -1) q.push(my[mm]);
15             my[mm] = i, mx[i] = mm;
16         }
17         else q.push(i);
18     }
19 }
20 char nameB[N], nameG[N];
21 void output(int n) {
22     for (int i = 0; i < n; ++i)
23         printf("%c %c\n", nameB[i], nameG[mx[i]]);
24 }
25 int hashB[M], hashG[M];
26 void initialize() {
27     memset(hashB, 0, sizeof(hashB)); memset(hashG, 0, sizeof(hashG));
28     memset(my, -1, sizeof(my)); memset(now, 0, sizeof(now));
29 }

```

4.8 Maximum Clique


```

1  const int N = 50;
2  int maps[N][N], found, mc, n;
3  int c[N], answer[N], record[N];
4  void dfs(int GraphSize, int *s, int CliqueSize) {
5      if(GraphSize == 0) {
6          if(CliqueSize > mc) {
7              mc = CliqueSize;
8              found = true;
9              copy(record, record + mc, answer);
10         }
11         return ;
12     }
13     for(int i = 0; i < GraphSize; i++) {
14         if(CliqueSize + GraphSize <= mc || c[s[i]] + CliqueSize <= mc)
15             return;
16         int tmps[N], tmpSize = 0;
17         record[CliqueSize] = s[i];
18         for(int j = i + 1; j < GraphSize; j++)
19             if(maps[s[i]][s[j]]) tmps[tmpSize++] = s[j];
20         dfs(tmpSize, tmps, CliqueSize + 1);
21         if(found) return ;
22     }
23 }
24 void initialize() {
25     memset(maps, false, sizeof(maps));
26     mc = 0;
27 }
28 int findMaxClique(int n) {
29     for(int i = n - 1; i >= 0; i--) {
30         found = false;
31         int tail = 0, s[N];
32         for(int j = i + 1; j < n; j++)
33             if(maps[i][j])
34                 s[tail++] = j;
35         record[0] = i;
36         dfs(tail, s, 1);
37         c[i] = mc;
38     }
39     return mc;
40 }

```

4.9 Maximal Clique

```

1  const static int N = 130;
2  int n, maps[N][N], cnt;
3  void CountMaximalClique(int *p, int ps, int *x, int xs) {
4      if(ps == 0) {
5          if(xs == 0) cnt++;
6          return ;
7      }
8      for(int i = 0; i < xs; i++) {
9          int j, v = x[i];
10         for(j = 0; j < ps && maps[p[j]][v]; j++);
11         if(j == ps) return;
12     }
13     int tmpmp[N], tmpmps = 0, tmpxp[N], tmpxs = 0;
14     for(int i = 0; i < ps; i++) {
15         int v = p[i];
16         tmpmps = tmpxs = 0;
17         for(int j = i + 1; j < ps; j++) {
18             int u = p[j];
19             if(maps[v][u])
20                 tmpmp[tmpmps++] = u;
21         }
22         for(int j = 0; j < xs; j++) {
23             int u = x[j];
24             if(maps[v][u])
25                 tmpxp[tmpxs++] = u;
26         }
27         CountMaximalClique(tmpmp, tmpmps, tmpxp, tmpxs);
28         if(cnt > 1000) return;
29         x[xs++] = v;
30     }
31 }
32 int CountMaximalClique() {
33     cnt = 0;
34     int p[N], x[N];
35     for(int i = 0; i < n; i++) p[i] = i;
36     CountMaximalClique(p, n, x, 0);
37     return cnt;
38 }

```

4.10 Lowest Common Ancestor

```

1  const int N = 100000;
2  int father[N], chk[N], dgr[N];
3  vector<vector<int>> adj, query;
4  int set_find(int i) {
5      return father[i] = i == father[i] ? i : set_find(father[i]);
6  }
7  void initialize(int n) {
8      adj.assign(n, vector<int>());
9      query.assign(n, vector<int>());
10     CC(dgr, 0); CC(chk, 0);
11 }
12 void LCA(int u) {
13     father[u] = u;
14     FOREACH(adj[u], i) {
15         LCA(*i), father[*i] = u;
16     }
17     chk[u] = 1;
18     FOREACH(query[u], i) if(chk[*i])
19         printf("%d\n", set_find(*i));
20 }

```

4.11 Minimum Cut Algorithm

```

1  const int V = 501, INF = 100000000, S = 1;
2  int maps[V][V], dist[V], pre;
3  bool vst[V], del[V];
4  void initialize() /* start with 1 */ {
5      memset(del, false, sizeof(del));
6      memset(maps, 0, sizeof(maps));
7  }
8  int maximum_adjacency_search(int t, int n) {
9      for(int i = 1; i <= n; i++)
10         if(!del[i]) dist[i] = maps[S][i];
11     memset(vst, false, sizeof(vst));
12     vst[S] = true;
13     int k = S;
14     for(int j = 1; j <= n - t; j++) {
15         int tmp = -INF;
16         pre = k;
17         for(int i = 1; i <= n; i++)
18             if(!vst[i] && !del[i] && tmp < dist[i]) {
19                 tmp = dist[i];
20                 k = i;
21             }
22         vst[k] = true;
23         for(int i = 1; i <= n; i++)
24             if(!vst[i] && !del[i]) dist[i] += maps[k][i];
25     }
26     return k;
27 }
28 int Stoer_Wagner(int n) {
29     int mcut = INF;
30     for(int i = 1; i < n; i++) {
31         int idx = maximum_adjacency_search(i, n);
32         mcut = min(mcut, dist[idx]);
33         del[idx] = true;
34         for(int i = 1; i <= n; i++) {
35             if(!del[i] && i != pre) {
36                 maps[pre][i] += maps[idx][i];
37                 maps[i][pre] = maps[pre][i];
38             }
39         }
40     }
41     return mcut;
42 }

```

4.12 Degree-constrained Spanning Tree

```

1  const int N = 25, LEN = 15, INF = 1<<29;
2  int dis[N][N] = {}, f[N] = {}, father[N] = {}, n;
3  bool visit[N] = {};
4  bool used[N][N] = {};
5  void Dfs(int last, int v) {
6      visit[v] = 1;
7      if(!father[v]) f[v] = -INF;
8      else f[v] = max(dis[last][v], f[father[v]]);
9      for(int i = 0; i < n; ++i)
10         if(!visit[i] && used[v][i])
11             father[i] = v, Dfs(v, i);
12 }
13 int DegreeLimitMST(int k) {
14     int ret = 0, path[N], group[N] = {}, g = 0, pre[N], degree = 0;
15     memset(used, 0, sizeof(used));
16     for(int i = 1; i < n; ++i)
17         if(!group[i]) {
18             group[i] = ++g;
19             for(int j = 0; j < n; ++j)
20                 path[j] = dis[i][j], pre[j] = i;
21             while(1) {
22                 int tmp = INF, mark = -1;
23                 for(int j = 1; j < n; ++j)
24                     if(!group[j] && path[j] < tmp)
25                         tmp = path[j], mark = j;
26                 if(mark == -1) break;
27                 used[pre[mark]][mark] = 1, used[mark][pre[mark]] = 1;
28                 ret += tmp;
29                 group[mark] = g;
30                 for(int j = 1; j < n; ++j)
31                     if(!group[j] && path[j] > dis[mark][j])
32                         path[j] = dis[mark][j], pre[j] = mark;
33             }
34         }
35     for(int i = 1; i <= g; ++i) {
36         int tmp = INF, mark = -1;
37         for(int j = 1; j < n; ++j)
38             if(group[j] == i && tmp > dis[0][j])
39                 tmp = dis[0][j], mark = j;
40         used[0][mark] = used[mark][0] = 1;
41         ret += tmp;
42         ++degree;
43     }
44     while(degree < k) {
45         memset(visit, 0, sizeof(visit));
46         Dfs(0, 0);
47         int tmp = INF, mark = -1, t;
48         for(int i = 1; i < n; ++i)
49             if(!used[0][i] && dis[0][i] != INF) {
50                 t = ret + dis[0][i] - f[i];
51                 if(tmp > t) tmp = t, mark = i;
52             }
53         if(ret <= tmp) break;
54         ret = tmp;
55         used[0][mark] = used[mark][0] = 1;
56         tmp = f[mark];
57         while(dis[father[mark]][mark] != tmp) mark = father[mark];
58         used[mark][father[mark]] = used[father[mark]][mark] = 0;
59         ++degree;
60     }
61     return ret;
62 }

```

4.13 Minimum Directed Tree

```

1  const int N = 1010, E = N * N;
2  const LL INF = 10000000000LL;
3  template<typename T>
4  struct Edge {
5      int u, v;
6      T c;
7  };
8  Edge<LL> edge[E];
9  int label[N], pre[N], visit[N];
10 template<typename T>
11 T treeGraph(int n, int m, int root, Edge<T>* edge) {
12     int cnt = 0;
13     T inEdge[N], ans = 0;
14     while(true) {
15         fill(inEdge, inEdge + n, INF);
16         REP(i, 0, m) {
17             int u = edge[i].u, v = edge[i].v;
18             if(v != u && edge[i].c < inEdge[v])
19                 {
20                     pre[v] = u;
21                     inEdge[v] = edge[i].c;
22                 }
23         }
24         REP(i, 0, n) {
25             if(i == root) continue;
26             if(inEdge[i] == INF) return -1;
27         }
28         int now = 0;
29         CC(label, -1), CC(visit, -1);
30         inEdge[root] = 0;
31         REP(i, 0, n) {
32             ans += inEdge[i];
33             int v = i;
34             while(visit[v] != i && label[v] == -1 && v != root) {
35                 visit[v] = i;
36                 v = pre[v];
37             }
38             if(v != root && label[v] == -1) {
39                 for(int u = pre[v]; u != v; u = pre[u])
40                     label[u] = now;
41                 label[v] = now++;
42             }
43         }
44         if(now == 0) break;
45         REP(i, 0, n) if(label[i] == -1) label[i] = now++;
46         REP(i, 0, m) {
47             int v = edge[i].v;
48             edge[i].v = label[edge[i].v];
49             edge[i].u = label[edge[i].u];
50             if(edge[i].v != edge[i].u) edge[i].c -= inEdge[v];
51         }
52         root = label[root];
53         n = now;
54     }
55     return ans;
56 }

```

5 Math

5.1 Minimum Directed Tree

```

1  const int N = 1010, E = N * N;
2  const LL INF = 10000000000LL;
3  template<typename T>
4  struct Edge {
5      int u, v;
6      T c;
7  };
8  Edge<LL> edge[E];
9  int label[N], pre[N], visit[N];
10 template<typename T>
11 T treeGraph(int n, int m, int root, Edge<T>* edge) {
12     int cnt = 0;
13     T inEdge[N], ans = 0;
14     while(true) {
15         fill(inEdge, inEdge + n, INF);
16         REP(i, 0, m) {
17             int u = edge[i].u, v = edge[i].v;
18             if(v != u && edge[i].c < inEdge[v])
19                 {
20                     pre[v] = u;
21                     inEdge[v] = edge[i].c;
22                 }
23         }
24         REP(i, 0, n) {
25             if(i == root) continue;
26             if(inEdge[i] == INF) return -1;
27         }
28         int now = 0;
29         CC(label, -1), CC(visit, -1);
30         inEdge[root] = 0;
31         REP(i, 0, n) {
32             ans += inEdge[i];
33             int v = i;
34             while(visit[v] != i && label[v] == -1 && v != root) {
35                 visit[v] = i;
36                 v = pre[v];
37             }
38             if(v != root && label[v] == -1) {
39                 for(int u = pre[v]; u != v; u = pre[u])
40                     label[u] = now;
41                 label[v] = now++;
42             }
43         }
44         if(now == 0) break;
45         REP(i, 0, n) if(label[i] == -1) label[i] = now++;
46         REP(i, 0, m) {
47             int v = edge[i].v;

```

```

48         edge[i].v = label[edge[i].v];
49         edge[i].u = label[edge[i].u];
50         if(edge[i].v != edge[i].u) edge[i].c -= inEdge[v];
51     }
52     root = label[root];
53     n = now;
54 }
55 return ans;
56 }

```

6 Math

6.1 Matrix

```

1 #define rep(i,a,b) for(int i=a;i<b;++i)
2 typedef long long ll;
3
4 const int mod = 1000000;
5 struct Matrix {
6     static const int N = 27;
7     ll v[N][N];
8     int s;
9
10    Matrix(int ss) {
11        s = ss;
12        memset(v, 0, sizeof(v));
13    }
14
15    inline void setE() {
16        rep(i, 0, s) v[i][i] = 1;
17    }
18
19    Matrix operator +(const Matrix & b) {
20        Matrix res = *this;
21        rep(i, 0, s) rep(j, 0, s) {
22            res.v[i][j] += b.v[i][j];
23            if(res.v[i][j] >= mod)
24                res.v[i][j] -= mod;
25        }
26        return res;
27    }
28
29    Matrix operator -(const Matrix & b) {
30        Matrix res = *this;
31        rep(i, 0, s) rep(j, 0, s) {
32            res.v[i][j] -= b.v[i][j];
33            if(res.v[i][j] < 0)
34                res.v[i][j] += mod;
35        }
36        return res;
37    }
38
39    Matrix operator *(const Matrix & b) {
40        Matrix res(s);
41        rep(i, 0, s) rep(j, 0, s) {
42            ll temp = 0;
43            rep(k, 0, s) temp += v[i][k] * b.v[k][j];
44            res.v[i][j] = temp % mod;
45        }
46        return res;
47    }
48
49    Matrix pow(int b) {
50        Matrix res(s), t = *this;
51        res.setE();
52        while(b) {
53            if(b & 1) res = res * t;
54            t = t * t;
55            b >>= 1;
56        }
57        return res;
58    }
59
60    Matrix powS(int b) {
61        Matrix res(s), t = *this, p = t;
62        res.setE();
63        while(b) {
64            if(b & 1) {
65                res = res + t;
66                t = t * p;
67            }
68            t = t + t * p;
69            b >>= 1;
70            p = p * p;
71        }
72        return res;
73    }
74 };

```

6.2 Number Thoery

6.2.1 Phi

```

1 typedef long long LL;
2 const int N = 1000001;
3 int prime[N], np;
4 bool vis[N];
5 LL phi[N];
6
7 void getPhi() {
8     int t;
9     np = 0;
10    memset(vis, 0, sizeof(vis));
11    for(int i = 1; i < N; ++i) phi[i] = i;
12    for(int i = 2; i < N; ++i) {

```

```

13     if (!vis[i]) {
14         prime[np++] = i;
15         phi[i] = i - 1;
16     }
17     for (int j = 0; j < np && (t = i * prime[j]) < N; ++j) {
18         vis[t] = 1;
19         if (i % prime[j] == 0) {
20             phi[t] = phi[i] * prime[j];
21             break;
22         }
23         phi[t] = phi[i] * (prime[j] - 1);
24     }
25 }
26 }

```

6.2.2 $a^x \equiv b \pmod{n}$

$a^x \equiv b \pmod{p}$ p is prime number: Baby-step-gaint-step

```

1 typedef long long llong;
2 int mod_pow(int a, int b, int n) {
3     llong res(1), t(a);
4     while (b) {
5         if (b & 1) res = res * t % n;
6         t = t * t % n; b >>= 1;
7     }
8     return res;
9 }
10 const int N = 50003;
11 int mexp[N], id[N];
12 bool log_cmp(const int a, const int b) { return mexp[a] < mexp[b]
13     ];}
14 // a^x == b(mod p); p is prime and 1 <= a < p; x>=0
15 int mod_log(int a, int b, int p) {
16     if (b == 1) return 0;
17     int i, j, m = (int) ceil(sqrt(p)), inv = mod_pow(mod_pow(a, m, p),
18         p - 2, p);
19     for (id[0] = 0, mexp[0] = i = 1; i < m; ++i) {
20         id[i] = i; mexp[i] = mexp[i - 1] * (llong) a % p;
21         if (mexp[i] == b) return i;
22     }
23     stable_sort(id, id + m, log_cmp);
24     sort(mexp, mexp + m);
25     for (i = 0; i < m; ++i) {
26         j = lower_bound(mexp, mexp + m, b) - mexp;
27         if (j < m && mexp[j] == b) return i * m + id[j];
28         b = b * (llong) inv % p;
29     }
30     return -1;
31 }

```

$a^x \equiv b \pmod{n}$ a, b, n can be any integer. Baby-step-gaint-step

```

1 typedef long long llong;
2 const int N = (1<<14)-1, M = 40000;
3 //spoj 3105
4 struct Hash {
5     int g[N], next[M], v[M], vu[M], ne;
6
7     void init() {
8         ne = 2; memset(g, 0, sizeof(g));
9     }
10     int find(int t) {
11         for (int i = g[t&N]; i; i = next[i]) if (t == v[i]) return
12             vu[i];
13         return -1;
14     }
15     void insert(int t, int val) {
16         int key = t&N;
17         v[ne] = t;
18         vu[ne] = val;
19         next[ne] = g[key];
20         g[key] = ne++;
21     }
22 } S;
23 void extend_gcd(llong a, llong b, llong &d, llong &x, llong &y) {
24     if (b) {
25         extend_gcd(b, a % b, d, y, x);
26         y -= a / b * x;
27     } else d = a, x = 1, y = 0;
28 }
29 int gcd(int a, int b) { return b ? gcd(b, a % b) : a; }
30 // a^x == b(mod n), n is not need to be prime;
31 int mod_log(int a, int b, int n) {
32     b %= n, a %= n;
33     llong t, x, y, d, r, res;
34     int i, tmp;
35     for (i = 0, t = 1 % n; i < 100; ++i, t = t * a % n) if (t == b)
36         return i;
37     for (r = 1, res = 0; (tmp = gcd(a, n)) > 1; ++res) {
38         if (b % tmp) return -1;
39         b /= tmp; n /= tmp; r = r * a / tmp % n;
40     }
41     S.init();
42     extend_gcd(r, n, d, x, y);
43     b = (b * x % n + n) % n;
44     int s = (int) ceil(sqrt(n+0.0));
45     for (i = 0, t = 1; i < s; ++i, t = t * a % n) {
46         if (t == b) return i + res;
47         if (S.find(t) == -1) S.insert(t, i);
48         else return -1;
49     }
50     extend_gcd(t, n, d, x, y);
51     x = (x % n + n) % n;
52     for (i = 0; i < s; ++i) {
53         tmp = S.find(b);
54         if (tmp != -1) return i * s + res + tmp;
55     }
56 }

```

```

54     b = b * x % n;
55 }
56 return -1;
57 }

```

6.2.3 $x * x \equiv a \pmod{p}$

```

1 typedef long long llong;
2 int mod_pow(int a, int b, int mod) {
3     llong res(1), t(a);
4     while (b) {
5         if (b & 1) res = res * t % mod;
6         t = t * t % mod; b >>= 1;
7     }
8     return res;
9 }
10 // x*x == a (mod n) n should be a prime and gcd(a,n) == 1
11 int mod_sqrt(int a, int n) {
12     int res;
13     if (2 == n) return a % n;
14     if (1 == mod_pow(a, (n - 1) / 2, n)) {
15         if (3 == n % 4) res = mod_pow(a, (n + 1) / 4, n);
16         else {
17             int b = 1, k = 0, i = (n - 1) / 2;
18             while (1 == mod_pow(b, (n - 1) / 2, n)) ++b;
19             do {
20                 i /= 2, k /= 2;
21                 if ((0 == mod_pow(a, i, n) * (llong) mod_pow(b, k, n) + 1)
22                     % n) k += (n-1)/2;
23             } while (i % 2 == 0);
24             res = (mod_pow(a, (i + 1) / 2, n) * (llong) mod_pow(b, k /
25                 2, n)) % n;
26         }
27         return min(res, n - res); // make that res <= n/2
28     }
29     return -1;
30 }
31 int x, n;
32 int main() {
33     while (cin >> x >> n) {
34         cout << mod_sqrt(x, n) << endl;
35     }
36 }

```

6.2.4 Miller and Pollard

```

1 #include <cstdio>
2 #include <iostream>
3 #include <cstring>
4 #include <cmath>
5 #include <ctime>
6 #include <cstdlib>
7
8 using namespace std;
9 typedef long long ll;
10
11 ll mult(ll a, ll b, ll mod) {
12     if (a >= mod) a %= mod;
13     if (b >= mod) b %= mod;
14     if (a <= (1LL<<31) && b <= (1LL<<31)) return a*b%mod;
15
16     ll res = 0;
17     while (b) {
18         if (b&1) {
19             res += a;
20             if (res >= mod) res -= mod;
21         }
22         a <<= 1;
23         if (a >= mod) a -= mod;
24         b >>= 1;
25     }
26     return res;
27 }
28
29 ll gcd(ll a, ll b) {
30     return b ? gcd(b, a%b) : a;
31 }
32
33 int cnt1, cnt2, cnt0;
34 ll p_rho(ll n, int limit = 1 << 17) {
35     if (0 == (n&1)) return 2; // must
36     ll x, y, d;
37     for (ll c = 1; c < n; ++c) {
38         x = y = 2;
39         while (true) {
40             x = (mult(x, x, n) + c) % n;
41             y = (mult(y, y, n) + c) % n;
42             y = (mult(y, y, n) + c) % n;
43             d = gcd(y - x + n, n);
44             if (d == n) break;
45             if (d > 1) return d;
46             ++cnt1;
47         }
48     }
49     return n;
50 }
51
52 /// here is the fast code.
53 ll val[1<<20];
54 ll p_rho0(ll n, int limit = 1 << 17) {
55     if (0 == (n&1)) return 2; // must
56     ll d;
57     for (ll c = 1; c < n; ++c) {
58         val[0] = 2;
59         for (int i = 1; i < limit; ++i) {
60             val[i] = (mult(val[i - 1], val[i - 1], n) + c) % n;
61             if (0 == (i & 1)) {
62                 d = gcd(val[i] - val[i>>1] + n, n);
63                 if (d == n) break;
64                 if (d > 1) return d;
65             }
66         }
67     }
68 }

```

```

65     }
66     ++cnt0;
67 }
68 }
69 return n;
70 }
71
72 ll p_rho2(ll n, int limit = 8192) {
73     if (0 == (n&1)) return 2; // must
74     ll x,y,d,i,k;
75     for (ll c = 1; c < n; ++c) {
76         i = 0;
77         k = x = y = 2;
78         while (i++ < limit) {
79             x = (mult(x,x,n) + c) % n;
80             d = gcd(x - y + n, n);
81             if (d == n) break;
82             if (d != n && d > 1) return d;
83             if (i == k) y = x, k <= 1;
84             ++cnt2;
85         }
86     }
87     return n;
88 }
89 }
90
91 ll power(ll a, ll b, ll mod) {
92     ll res = 1, t = a;
93     while (b) {
94         if (b&1) res = mult(res,t,mod);
95         t = mult(t,t,mod);
96         b >>= 1;
97     }
98     return res;
99 }
100
101 ll witness(ll a, ll n) {
102     ll b = n - 1;
103     int cnt = 0;
104     while (0 == (b&1)) ++cnt, b >>= 1;
105     ll x = power(a,b,n), y;
106     for (int i = 0; i < cnt; ++i) {
107         y = mult(x,x,n);
108         if (y == 1) {
109             if (x != 1 && x != n - 1) return 0;
110             return 1;
111         }
112         x = y;
113     }
114     return x;
115 }
116 }
117
118 bool is_prime(ll n) {
119     if (n == 2) return true;
120     if (n < 2 || (n&1) == 0) return false;
121     int p[5] = {3,5,7,11,13};
122     for (int i = 0; i < 5; ++i) {
123         if (n == p[i]) return true;
124         if (n % p[i] == 0) return false;
125     }
126
127     for (int i = 0; i < 10; ++i)
128         if (witness(rand()% (n-2) + 2, n) != 1) return false;
129     return true;
130 }
131
132 int main() {
133     ll n;
134     ll sum0 = 0, sum1 = 0, sum2 = 0;
135     for (int v = 0; v < 1000; ++v) {
136         srand(time(0));
137         for (int i = 3; i < 1000; i+=2) {
138
139             n = rand();
140             if (v % 5 != 0) n = n*n;
141             if (is_prime(n)) {
142                 n = n*n;
143                 cnt0 = cnt1 = cnt2 = 0;
144                 if (p_rho0(n) == n) cout << "ERROR0" << " " << n << endl;
145                 if (p_rho1(n) == n) cout << "ERROR1" << " " << n << endl;
146                 if (p_rho2(n) == n) cout << "ERROR2" << " " << n << endl;
147                 sum0 += cnt0;
148                 sum1 += 3*cnt1;
149                 sum2 += cnt2;
150             } else {
151                 cnt0 = cnt1 = cnt2 = 0;
152                 if (p_rho0(n) == n) cout << "ERROR0" << " " << n << endl;
153                 if (p_rho1(n) == n) cout << "ERROR1" << " " << n << endl;
154                 if (p_rho2(n) == n) cout << "ERROR2" << " " << n << endl;
155                 sum0 += cnt0;
156                 sum1 += 3*cnt1;
157                 sum2 += cnt2;
158             }
159         }
160     }
161     cout << "over" << endl;
162     cout << sum0 << "\n" << sum1 << "\n" << sum2 << endl;
163 }

```

6.2.5 Get prime in range

```

1 //spoj 8586. Prime Generator2
2 #include <stdio>
3 #include <string>
4 #include <algorithm>
5 #include <math>
6 using namespace std;
7 const int N = 46340;
8 bool vis[N];
9 int prime[5000], np;
10
11 void cp() {
12     memset(vis, 0, sizeof(vis));
13     np = 0;

```

```

14     int t;
15     for (int i = 2; i < N; ++i) {
16         if (!vis[i]) prime[np++] = i;
17         for (int j = 0; j < np && (t = i * prime[j]) < N; ++j) {
18             vis[t] = 1;
19             if (i % prime[j] == 0) break;
20         }
21     }
22     //printf("%d\n", np);
23 }
24 int a, b;
25
26 int solve() {
27     int ret = 0, k, t;
28     memset(vis, 0, sizeof(vis));
29     if (2 >= a && 2 <= b) puts("2"); // note that
30     if ((a & 1) == 0) a++;
31     for (int i = 1; i < np && prime[i] <= b; ++i) {
32         if (a % prime[i] == 0) k = a;
33         else {
34             if (a <= b - (prime[i] - (a % prime[i]))) {
35                 k = a + prime[i] - (a % prime[i]);
36                 if ((k & 1) == 0) {
37                     if (k <= b - prime[i])
38                         k += prime[i];
39                     else continue;
40                 } else continue;
41             }
42         }
43         if (k != prime[i] && k <= b) vis[k - a] = 1;
44         t = prime[i]*2;
45         int j = k;
46         while (j <= b - t) {
47             j += t;
48             vis[j - a] = 1;
49         }
50     }
51     for (int i = 0; i <= b - a; i += 2)
52         if (!vis[i] && i + a >= 2) printf("%d\n", a + i);
53 }
54
55 int main() {
56     cp();
57     int cas;
58     bool flag = false;
59     for (scanf("%d", &cas); cas; --cas) {
60         if (flag) puts("");
61         else flag = true;
62         scanf("%d%d", &a, &b);
63         solve();
64     }
65 }

```

6.2.6 Mod Equation

```

1 #include <stdio>
2 #include <algorithm>
3 #include <string>
4 using namespace std;
5
6 // #define __int64 long long
7 void eGcd(__int64 a, __int64 b, __int64 &d, __int64 &x, __int64 &y)
8 {
9     if (!b) {
10         x = 1;
11         y = 0;
12         d = a;
13     } else {
14         eGcd(b, a % b, d, y, x);
15         y -= a / b * x;
16     }
17 }
18
19 __int64 getAns(__int64* m, __int64* r, int n) {
20     __int64 ret = r[0], mod = m[0], t, te, x, y, d;
21     for (int i = 1; i < n; ++i) {
22         t = r[i] - ret;
23         eGcd(mod, m[i], d, x, y);
24         if (t % d) return -1;
25         te = m[i] / d;
26         x = (t / d * x % te + te) % te;
27         ret += mod * x;
28         mod *= te;
29     }
30     return ret;
31 }
32
33 int n;
34 __int64 m[10000], r[10000];
35
36 int main() {
37     while (scanf("%d", &n) == 1) {
38         for (int i = 0; i < n; ++i)
39             scanf("%I64d%I64d", &m[i], &r[i]);
40         printf("%I64d\n", getAns(m, r, n));
41     }
42     return 0;

```

6.3 Fraction

6.3.1 $a/b < x/y < c/d$

```

1 //smallest denominator
2 typedef long long ll;
3 void max_fac(int a,int b,int c,int d,int &x, int &y) {
4     int t = a/b;
5     if ((t + 1)*(ll)d < c) {

```

```

6      x = t + 1, y = 1;
7    } else {
8      max_fac(d, c-t*d, b, a-t*b, y, x);
9      x += t*y;
10   }
11 }

```

6.3.2 $x^2 - n * y^2 = 1$

n is a non-square-number, solve the minimum (x_1, y_1)
all (x_i, y_i) satisfies:

$$x_i + y_i \sqrt{n} = (x_1 + y_1 \sqrt{n})^i$$

$$x_{i+1} = x_1 x_i + n y_1 y_i$$

$$y_{i+1} = x_1 y_i + y_1 x_i$$

```

1 //always need BigInteger
2 typedef long long ll;
3 void getAns(ll &x, ll &y, int n) {
4     ll p0 = 0, p1 = 1, p2;
5     ll q0 = 1, q1 = 0, q2;
6     ll g1 = 0, h1 = 1, g2, h2;
7     ll a0 = (int)(sqrt(n+0.5)), a2 = a0, a3;
8
9     for (int i = 2; ++i) {
10        g2 = a2*h1 - g1;
11        h2 = (n - g2*g2)/h1;
12        a3 = (g2+a0)/h2;
13        p2 = a2*p1+p0;
14        q2 = a2*q1+q0;
15        if (p2*p2-n*q2*q2 == 1) {
16            x = p2;
17            y = q2;
18            return ;
19        }
20        g1 = g2, h1 = h2, a2 = a3;
21        p0 = p1, p1 = p2;
22        q0 = q1, q1 = q2;
23    }
24 }
25
26 /*
27 static BigInteger x, y;
28
29 public static void getAns(int n)
30 {
31     BigInteger p0 = BigInteger.ZERO, p1 = BigInteger.ONE, p2;
32     BigInteger q0 = BigInteger.ONE, q1 = BigInteger.ZERO, q2;
33     BigInteger g1 = BigInteger.ZERO, h1 = BigInteger.ONE, g2, h2;
34     BigInteger a0 = BigInteger.valueOf((int)(Math.sqrt(n + 0.5))),
35         a2 = a0, a3;
36     BigInteger bn = BigInteger.valueOf(n);
37     while (true)
38     {
39         g2 = a2.multiply(h1).subtract(g1);
40         h2 = (bn.subtract(g2.multiply(g2))).divide(h1);
41         a3 = (g2.add(a0)).divide(h2);
42         p2 = a2.multiply(p1).add(p0);
43         q2 = a2.multiply(q1).add(q0);
44         if (p2.multiply(p2).subtract(bn.multiply(q2).multiply(q2)).
45             equals(BigInteger.ONE)) { // notice use equals!!!!
46             x = p2;
47             y = q2;
48             return ;
49         }
50         g1 = g2; h1 = h2; a2 = a3;
51         p0 = p1; p1 = p2;
52         q0 = q1; q1 = q2;
53     }
54 }
55 */

```

6.3.3 $\sum_{k=0}^{n-1} [(a + d * k) / m]$

```

1 typedef long long ll;
2 ll rec(ll a, ll d, ll m, ll n) {
3     ll res = 0;
4     if (a >= m) {
5         res += (a/m)*n;
6         a %= m;
7     }
8     if (d >= m) {
9         res += (d/m)*(n*(n-1)/2);
10        d %= m;
11    }
12
13    if (d == 0) return res;
14    ll top = a + d*n;
15    return res + rec(top%m, m, d, top/m);
16 }

```

6.4 Linear Equaton

6.4.1 Xor Equation

1 1 0 0 = 0 (here when $r = 0, i = 0$)

0 0 1 0 = 1 (here when $r = 1, i = 2$)

0 0 0 1 = 1 (here when $r = 2, i = 3$)

```

1 const int N = 33;
2 int m, nn, num, list[N];
3 // m equations, nn variables
4 int a[N][N];
5
6 int reduce() {
7     int i, j, k, r;
8     for (i = r = 0; i < nn; ++i) {
9         for (j = r; j < m && !a[j][i]; ++j); if (j >= m) continue;
10        if (j > r) for (k = 0; k <= nn; ++k) swap(a[r][k], a[j][k]);
11        for (num = 0, k = i; k <= nn; ++k) if (a[r][k]) list[num++] = k;
12        for (j = 0; j < m; ++j) if (j != r && a[j][i])
13            for (k = 0; k < num; ++k) a[j][list[k]] ^= 1;
14        ++r;
15    }
16    for (i = 0; i < m; ++i)
17        if (a[i][nn]) {
18            for (j = 0; j < nn && !a[i][j]; ++j);
19            if (j == nn) return 0; // else x[j] = a[i][nn]/a[i][j];
20        }
21    // the number of free variable = (nn - r)
22    return 1;
23 }

```

be carefully when use long long and int

```

1 const int N = 33;
2 ll a[N];
3 int m, nn; // nn variables (0 -> nn - 1), m equations
4 inline int bit2(ll v, int n) { return v >> n & 1; }
5 inline void set_bit(ll &v, int n, ll t) { v |= t << n; } // notice v
6 // will be ll
7
8 int reduce() {
9     int i, j, r;
10    for (i=r=0; i < nn; ++i) {
11        for (j=r; j < m && !bit2(a[j], i); ++j); if (j >= m) continue;
12        if (j > r) swap(a[r], a[j]);
13        for (int j = 0; j < m; ++j) if (j != r && bit2(a[j], i)) a[j] ^=
14            a[r];
15        ++r;
16    }
17    for (i = 0; i < m; ++i) if (bit2(a[i], nn)) {
18        if (a[i]^(1LL << nn)); // notice 1LL !!!
19        // x[j] = bit2(a[i], nn) / bit2(a[i], j); j's the first 1
20        // value bit
21        else return 0;
22    }
23    return 1;
24 }

```

6.4.2 Equation in Z

if in Q, change integer to fraction and no clean()

```

1 #include <iostream>
2 #include <algorithm>
3 using namespace std;
4
5 #define rep(i, a, b) for(int i=a; i<b; ++i)
6 typedef long long ll;
7 const int N = 101;
8
9 /* reduce() == 1 has integer value, they are A[i][cols]
10 == 2 has free value, the number is nfree
11 == 0 no value
12 == -1 has fractional value
13 */
14 struct Equation {
15     int rows, cols, nfree, frac, A[N][N];
16
17     void init() {
18         memset(A, 0, sizeof(A));
19     }
20
21     inline int gcd(int a, int b) {
22         return b ? gcd(b, a % b) : a;
23     }
24
25     inline void clean(int *b) {
26         int g = 0;
27         rep(i, 0, cols + 1) if (b[i])
28             if (g) g = gcd(b[i], g);
29             else g = b[i];
30         if (g) rep(i, 0, cols + 1) b[i] /= g;
31     }
32
33     inline void swapRow(int a, int b) {
34         if (a == b) return;
35         rep(i, 0, cols + 1) swap(A[a][i], A[b][i]);
36     }
37
38     inline void pivot(int r, int c) {
39         int u, v;
40         if (A[r][c] == 0) exit(-1);
41         rep(i, 0, rows) if (i != r && A[i][c]) {
42             v = gcd(A[i][c], A[r][c]);
43             u = A[r][c] / v;

```

```

44     v = A[i][c] / v;
45     rep(j, 0, cols + 1) A[i][j] = A[i][j] * u - v * A[r][j];
46     clean(A[i]);
47 }
48 }
49
50 int reduce(int nrow, int ncol) {
51     rows = nrow;
52     cols = ncol;
53     nfree = frac = 0;
54     int r = 0, c = 0, ind = 0;
55     for (; c < cols; ++c, ++r) {
56         for (ind = r; ind < rows && !A[ind][c]; ++ind);
57         if (ind >= rows) {
58             --r;
59             ++nfree;
60             continue;
61         }
62         swapRow(r, ind);
63         pivot(r, c);
64         // this->print();
65     }
66     for (int i = r; i < rows; ++i) if (A[i][cols]) return 0;
67     if (nfree) return 2;
68     for (r = 0; r < rows; ++r) if (A[r][cols]) {
69         for (c = 0; c < cols && !A[r][c]; ++c);
70         if (c == cols) return 0;
71         if (A[r][cols] % A[r][c]) frac = 1;
72         if (!frac) A[r][cols] /= A[r][c]; // get the answer
73     }
74     if (frac) return -1;
75     return 1;
76 }
77
78 void print() {
79     rep(i, 0, rows) {
80         rep(j, 0, cols + 1) cout << A[i][j] << " ";
81         cout << endl;
82     }
83     cout << endl;
84 }
85 };
86
87 Equation test;
88
89 int main() {
90     int n, m, t;
91     while (cin >> n >> m) {
92         rep(i, 0, n) {
93             rep(j, 0, m + 1) cin >> test.A[i][j];
94         }
95         cout << test.reduce(n, m) << endl;
96         test.print();
97     }
98 }

```

6.4.3 Equation in R

```

1 #include <iostream>
2 #include <algorithm>
3 #include <cmath>
4 using namespace std;
5
6 #define rep(i,a,b) for(int i=a;i<b;++i)
7 #define TEST freopen("in","r",stdin);
8 typedef long long ll;
9 const int N = 101;
10 const double eps = 1e-8;
11
12 /* reduce() == 0 has no value
13    == 1 has values, they are A[i][cols]
14 */
15
16
17 struct Equation {
18     double A[N][N];
19     int rows, cols, id[N];
20
21     void init() {
22         memset(A, 0, sizeof(A));
23     }
24
25     inline bool zero(double x) {
26         return fabs(x) < eps;
27     }
28
29     inline void swapA(int r, int c, int ir, int ic) {
30         if (r != ir)
31             rep(i, 0, cols + 1) swap(A[r][i], A[ir][i]);
32         if (c != ic) {
33             rep(i, 0, rows) swap(A[i][c], A[i][ic]);
34             swap(id[c], id[ic]);
35         }
36     }
37
38     inline void pivot(int r, int c) {
39         if (fabs(A[r][c]) < eps) exit(-1);
40         double p;
41         rep(i, 0, rows) if (i != r && fabs(A[i][c]) > eps) {
42             p = A[i][c] / A[r][c];
43             rep(j, 0, cols + 1) A[i][j] -= p * A[r][j];
44         }
45     }
46
47     int reduce(int nrow, int ncol) {
48         rows = nrow;
49         cols = ncol;
50         rep(i, 0, cols) id[i] = i;
51         int r = 0, c = 0, indr = 0, indc = 0;
52         double maxp = 0;
53         for (; c < cols; ++c, ++r) {
54             maxp = 0;
55             rep(i, r, rows) rep(j, c, cols) if (fabs(A[i][j]) > fabs(
56                 maxp)) {

```

```

57         indr = i;
58         indc = j;
59     }
60     if (zero(maxp)) return 0;
61     swapA(r, c, indr, indc);
62     pivot(r, c);
63 }
64 //this->print();
65 rep(i, 0, cols) A[i][0] = A[i][cols] / A[i][i];
66 rep(i, 0, cols) A[id[i]][cols] = A[i][0];
67 return 1;
68 }
69
70 void print() {
71     rep(i, 0, rows) {
72         rep(j, 0, cols + 1) cout << A[i][j] << " ";
73         cout << endl;
74     }
75     cout << endl;
76 }
77 };
78
79 Equation test;
80
81 int main() {
82     int n, m, t;
83     TEST
84     while (cin >> n >> m) {
85         rep(i, 0, n) {
86             rep(j, 0, m + 1) cin >> test.A[i][j];
87         }
88         cout << test.reduce(n, m) << endl;
89         test.print();
90
91         rep(i, 0, m) cout << test.A[i][m] << endl;
92         cout << "Over" << endl;
93     }
94 }

```

6.4.4 det

```

1 错编
2 1 det(ll a[][N],int n) {
3     for(int i=0; i<n; i++)for(int j=0; j<n; j++)a[i][j]%=mod;
4     ll ret=1;
5     for(int i=1; i<n; i++) {
6         for(int j=i+1; j<n; j++)
7             while(a[j][i]) {
8                 ll t=a[i][i]/a[j][i];
9                 for(int k=i; k<n; k++)
10                    a[i][k]=(a[i][k]-a[j][k]*t)%mod;
11                 for(int k=i; k<n; k++)
12                    swap(a[i][k],a[j][k]);
13                 ret=-ret;
14             }
15         if(a[i][i]==0)return 0;
16         ret=ret*a[i][i]%mod;
17     }
18     return (ret+mod)%mod;
19 }

```

6.5 Anti-Nim

Anti-Nim: $res = \oplus_i sg(i)$

$cnt = \sum_i [sg(i) \leq 1]$

first player wins when $(res = 0 \text{ and } cnt = n) \parallel$
 $(res \neq 0 \text{ and } cnt \neq n)$

6.6 nim multiply

$x \otimes y = \text{mex} \{ (x \otimes a) \oplus (b \otimes y) \oplus (a \otimes b) \mid 0 \leq a < x, 0 \leq b < y \}$

```

1 #include <cstdio>
2 #include <iostream>
3 #include <cstring>
4 #include <cmath>
5 #include <ctime>
6 #include <cstdlib>
7 #include <vector>
8 #include <map>
9 #include <algorithm>
10 #include <set>
11 using namespace std;
12 typedef long long ll;
13
14
15 const int N = 1010;
16 const int M = (1<<8);
17
18 bool vis[1<<8];
19 int sg[1<<4][1<<4];
20 int sg(int x,int y) {
21     if (x == 0 || y == 0) return 0;
22     if (sgv[x][y] != -1) return sgv[x][y];
23
24     memset(vis, 0, sizeof(vis));
25     for (int i = 0; i < x; ++i) {
26         vis[sg(i,y)] = 1;
27     }
28

```

```

29     for (int i = 0; i < y; ++i) {
30         vis[sg(x,i)] = 1;
31     }
32
33     for (int xx = 1; xx < x; ++xx) {
34         for (int yy = 1; yy < y; ++yy) {
35             vis[sg(xx,y) ^ sg(x,yy) ^ sg(xx,yy)] = 1;
36         }
37     }
38
39     for (int i = 0; i < M; ++i) if (!vis[i]) return i;
40     return M;
41 }
42
43
44
45
46 void init() {
47     memset(sgv, -1, sizeof(sgv));
48     for (int i = 0; i < 16; ++i) {
49         for (int j = 0; j < 16; ++j) {
50             sgv[i][j] = sg(i,j);
51         }
52     }
53 }
54
55
56
57 int nim_mult_power(int x,int y) { // x is a power of 2
58     //cout << x<<" " << y << endl;
59     if (x < 16) return sg(x,y);
60     int a, m, p, s, t, d1, d2;
61     for (a = 1; (1LL << a) <= x; a <= 1);
62     a >>= 1, m = (1<<a);
63     p = x/m;
64     s = y/m, t = y&(m-1);
65     d1 = nim_mult_power(p,s);
66     d2 = nim_mult_power(p,t);
67     return ((d1^d2) << a) ^ nim_mult_power(m/2,d1);
68 }
69
70 int nim_mult(int x,int y) {
71     if (x < y) swap(x,y);
72     if (x < 16) return sg(x,y);
73     int a, m, p, q, s, t, c1, c2, c3;
74     for (a = 1; (1LL << a) <= x; a <= 1);
75     a >>= 1, m = (1<<a);
76     p = x/m, q = x&(m-1);
77     s = y/m, t = y&(m-1);
78     c1 = nim_mult(p,s);
79     c2 = nim_mult(p,t) ^ nim_mult(q,s);
80     c3 = nim_mult(q,t);
81     return ((c1^c2) << a) ^ c3 ^ nim_mult_power(m/2,c1);
82 }
83
84 int n;
85 int main() {
86     init();
87     //test();
88     int cas;
89     for (cin >> cas; cas; --cas) {
90         scanf("%d",&n);
91         int res = 0, x, y;
92
93         for (int i = 0; i < n; ++i) {
94             scanf("%d%d",&x,&y);
95             res ^= nim_mult(x,y);
96         }
97         if (res) puts("Have a try, lxhgw.");
98         else puts("Don't waste your time.");
99     }
100 }

```

```

40     k = len >> 1;
41     while (j >= k) {
42         j -= k;
43         k >>= 1;
44     }
45     if (j < k) j += k;
46 }
47
48 }
49
50 void fft(Complex *y, int len, double on) // FFT O(nlogn)
51 // if on==1 DFT if on==--1 IDFT
52 {
53     register int h, hh, i, j, k;
54     Complex w,u,t, wn;
55     brc(y, len);
56     for (h = 1, hh = 2; hh <= len; h = hh, hh <= 1) {
57         wn.setValue(cos(on * pi / h), sin(on * pi / h));
58         for (j = 0; j < len; j += hh) {
59             w.setValue(1, 0);
60             for (k = j; k < j + h; k++) {
61                 u = y[k];
62                 t = w * y[k + h];
63                 y[k] = u + t;
64                 y[k + h] = u - t;
65                 w = w*wn;
66             }
67         }
68     }
69     if (on == -1) for (i = 0; i < len; ++i) y[i].r /= len;
70 }
71
72 void multi(char* a, char* b, int* sum, int &len) {
73     int la, lb, i;
74     la = strlen(a);
75     lb = strlen(b);
76     len = 1;
77     while (len < la * 2 || len < lb * 2) len <= 1;
78     for (i = 0; i < len; ++i) {
79         xa[i].r = (i < la) ? a[la - i - 1] - '0' : 0.0;
80         xb[i].r = (i < lb) ? b[lb - i - 1] - '0' : 0.0;
81         xa[i].i = xb[i].i = 0.0;
82     }
83     fft(xa, len, 1); // DFT(a)
84     fft(xb, len, 1); // DFT(b)
85     for (i = 0; i < len; ++i) xa[i] = xa[i] * xb[i]; // a = a*b
86     fft(xa, len, -1); // IDFT(a*b)
87     for (i = 0; i < len; ++i) sum[i] = (int) (xa[i].r + 0.5); //
88     sum = a
89     for (i = 0; i < len; ++i) //
90     {
91         sum[i + 1] += sum[i] / 10;
92         sum[i] %= 10;
93     }
94     len = la + lb - 1;
95     while (sum[len] <= 0 && len > 0) --len;
96 }
97
98 char a[N / 2], b[N / 2];
99 int sum[N]; // result
100
101 int main(void) {
102     int l;
103     register int i;
104     while (scanf("%s%s", a, b) == 2) {
105         multi(a, b, sum, l);
106         for (int i = 1; i >= 0; i--) putchar(sum[i] + '0');
107         putchar('\n');
108         return 0;
109     }

```

6.7 FFT

```

1 // hdu 1402
2 #include <stdio>
3 #include <cstring>
4 #include <cmath>
5 #include <algorithm>
6 #define N 300005
7 #define pi acos(-1.0)
8 using namespace std;
9
10 struct Complex {
11     double r, i;
12
13     Complex(double real = 0.0, double image = 0.0) {
14         r = real;
15         i = image;
16     }
17
18     Complex operator +(const Complex o) {
19         return Complex(r + o.r, i + o.i);
20     }
21
22     Complex operator -(const Complex o) {
23         return Complex(r - o.r, i - o.i);
24     }
25
26     Complex operator *(const Complex o) {
27         return Complex(r * o.r - i * o.i, r * o.i + i * o.r);
28     }
29
30     void setValue(double real = 0.0, double image = 0.0) {
31         r = real;
32         i = image;
33     }
34 } xa[N], xb[N];
35
36 void brc(Complex *y, int len) {
37     register int i, j, k;
38     for (i = 1, j = len >> 1; i < len - 1; ++i) {
39         if (i < j) swap(y[i], y[j]);

```

6.8 Popolynomial

```

1 #include <stdio>
2 #include <cmath>
3 #include <cstdlib>
4 #include <ctime>
5 #include <cstring>
6
7 int flag = 0;
8 typedef long long typec;
9 typec MOD = 999983;
10 const int RANKLIM = 50;
11 const int EQLIM = 10;
12
13 typec gcd(typec a, typec b) {
14     return b ? gcd(b, a % b) : a;
15 }
16
17 typec extendGCD(typec a, typec b, typec& x, typec& y) {
18     if (!b) return x = 1, y = 0, a;
19     typec res = extendGCD(b, a % b, x, y), tmp = x;
20     x = y, y = tmp - (a / b) * y;
21     return res;
22 }
23
24 ///ax = b mod m, be sure that (b, m) = (a, m)
25 typec modEquation(typec a, typec b, typec m) {
26     typec x, y;
27     y = extendGCD(a, m, x, y);
28     while (x < 0) x += m;
29     return (x * (b / y)) % m;
30 }
31
32 class Polynomial {
33 public:
34     Polynomial();
35     void set(int ra, typec* coe);
36     void simplify();
37
38     int getRank() const {
39         return rank;
40     }

```

```

41     typec getValue(typec x) const;
42
43     bool isOne() const {
44         return rank == 0 && coefficient[0];
45     }
46
47     bool isZero() const {
48         return rank == 0 && coefficient[0] == 0;
49     }
50     bool operator==(const Polynomial& a) const;
51
52     bool operator!=(const Polynomial& a) const {
53         return !(*this == a);
54     }
55
56     bool rankEqual(const Polynomial& a) const {
57         return rank == a.rank;
58     }
59
60     bool operator>(const Polynomial& a) const {
61         return rank > a.rank;
62     }
63
64     bool operator<(const Polynomial& a) const {
65         return rank < a.rank;
66     }
67
68     bool operator>=(const Polynomial& a) const {
69         return rank >= a.rank;
70     }
71
72     bool operator<=(const Polynomial& a) const {
73         return rank <= a.rank;
74     }
75     Polynomial & operator+=(const Polynomial&);
76     Polynomial operator+(const Polynomial& const;
77     Polynomial & operator-=(const Polynomial&);
78     Polynomial operator-(const Polynomial& const;
79     Polynomial operator*(const Polynomial& const;
80
81     Polynomial & operator*=(typec ti) {
82         for (int i = 0; i <= rank; i++) coefficient[i] *= ti;
83         return *this;
84     }
85
86     Polynomial operator*(typec ti) const {
87         Polynomial res = *this;
88         return res *= ti;
89     }
90     Polynomial& mulAtRank(typec ti, int ra);
91     Polynomial nMulAtRank(typec ti, int ra) const;
92     Polynomial & operator%=(const Polynomial&);
93     Polynomial operator%(const Polynomial& const;
94     void print() const;
95     void print2() const;
96
97 private:
98     typec coefficient[RANKLIM + 1];
99     int rank;
100 };
101
102 Polynomial::Polynomial() : rank(0) {
103     for (int i = 0; i < RANKLIM + 1; i++)
104         coefficient[i] = 0;
105 }
106
107 void Polynomial::set(int ra, typec* coe) {
108     for (int i = 0; i <= rank; i++)
109         coefficient[i] = 0;
110     rank = ra;
111     for (int i = 0; i <= rank; i++)
112         coefficient[ra - i] = (coe[i] % MOD + MOD) % MOD;
113     while (rank > MOD - 2) {
114         coefficient[rank % (MOD - 1)] += coefficient[rank];
115         coefficient[rank % (MOD - 1)] %= MOD;
116         coefficient[rank--] = 0;
117     }
118     while (coefficient[rank] == 0 && rank) rank--;
119 }
120
121 void Polynomial::simplify() {
122     typec g = coefficient[0];
123     if (g == 0) return;
124     for (int i = 1; i <= rank; i++)
125         g = gcd(g, coefficient[i]);
126     for (int i = 0; i <= rank; i++)
127         coefficient[i] /= g;
128 }
129
130 typec Polynomial::getValue(typec x) const {
131     typec res = 0;
132     for (int i = rank; i >= 0; i++)
133         res *= x, res += coefficient[i], res %= MOD;
134     return res;
135 }
136
137 bool Polynomial::operator==(const Polynomial& a) const {
138     if (rank != a.rank) return false;
139     for (int i = 0; i <= rank; i++)
140         if (coefficient[i] != a.coefficient[i])
141             return false;
142     return true;
143 }
144
145 Polynomial& Polynomial::operator+=(const Polynomial& a) {
146     if (a.rank > rank) rank = a.rank;
147     for (int i = 0; i <= rank; i++)
148         coefficient[i] += a.coefficient[i], coefficient[i] %= MOD;
149     while (coefficient[rank] == 0 && rank) rank--;
150     return *this;
151 }
152
153 Polynomial Polynomial::operator+(const Polynomial& a) const {
154     Polynomial res = *this;
155     return res += a;
156 }
157
158 Polynomial& Polynomial::operator-=(const Polynomial& a) {
159     if (a.rank > rank) rank = a.rank;
160     for (int i = rank; i >= 0; i--) {
161         coefficient[i] -= a.coefficient[i];
162         while (coefficient[i] < 0)
163             coefficient[i] += MOD;
164     }
165     while (coefficient[rank] == 0 && rank) rank--;
166 }
167
168 Polynomial Polynomial::operator*(const Polynomial& a) const {
169     Polynomial res;
170     res.rank = rank + a.rank;
171     if (res.rank > MOD - 2) res.rank = MOD - 2;
172     for (int i = 0; i <= rank; i++)
173         for (int j = 0; j <= a.rank; j++) {
174             res.coefficient[(i + j) % (MOD - 1)] += coefficient[i] *
175                 a.coefficient[j];
176             res.coefficient[(i + j) % (MOD - 1)] %= MOD;
177         }
178     while (res.coefficient[res.rank] == 0 && res.rank) res.rank--;
179     return res;
180 }
181
182 Polynomial Polynomial::mulAtRank(typec ti, int ra) {
183     for (int i = 0; i <= rank; i++)
184         coefficient[rank + ra - i] = (coefficient[rank - i] * ti) %
185             MOD;
186     for (int i = 0; i < ra; i++)
187         coefficient[i] = 0;
188     rank += ra;
189     if (ti == 0) rank = 0;
190     while (rank >= MOD - 1) {
191         coefficient[rank % (MOD - 1)] += coefficient[rank];
192         coefficient[rank % (MOD - 1)] %= MOD;
193         coefficient[rank--] = 0;
194     }
195     return *this;
196 }
197
198 Polynomial Polynomial::nMulAtRank(typec ti, int ra) const {
199     Polynomial res = *this;
200     return res.mulAtRank(ti, ra);
201 }
202
203 Polynomial& Polynomial::operator%=(const Polynomial& divisor) {
204     typec ti;
205     while (*this >= divisor && !(this->isZero())) {
206         ti = modEquation(divisor.coefficient[divisor.rank],
207             coefficient[rank], MOD);
208         *this -= divisor.nMulAtRank(ti, rank - divisor.rank);
209     }
210     return *this;
211 }
212
213 Polynomial Polynomial::operator%(const Polynomial& divisor) const {
214     Polynomial res = *this;
215     return res %= divisor;
216 }
217
218 void Polynomial::print() const {
219     for (int i = rank; i >= 0; i--) {
220         if (coefficient[i] == 0) continue;
221         if (i != rank) printf(" + ");
222         if (!i || coefficient[i] != 1)
223             printf("%lld", coefficient[i]);
224         if (i > 1) printf("x^%d", i);
225     }
226     printf("\n");
227 }
228
229 void Polynomial::print2() const {
230     printf("%d", rank);
231     for (int i = rank; i >= 0; i--)
232         printf(" %lld", coefficient[i]);
233     printf("\n");
234 }
235
236 Polynomial Pgcd(Polynomial a, Polynomial b) {
237     Polynomial swap, zero;
238     while (b != zero) {
239         a %= b;
240         swap = b, b = a, a = swap;
241     }
242     return a;
243 }
244
245 int main() {
246     int n, ra;
247     typec co[100];
248     Polynomial pol[100], g;
249     while (scanf("%d", &n) != EOF) {
250         if (flag) break;
251         for (int i = 0; i < n; i++) {
252             scanf("%d", &ra);
253             for (int j = 0; j <= ra; j++)
254                 scanf("%I64d", co + j);
255             pol[i].set(ra, co);
256         }
257         g = pol[0];
258         for (int i = 1; i < n; i++) {
259             g = Pgcd(pol[i], g);
260             if (g.isOne()) break;
261         }
262         g.simplify();
263         if (g.isOne()) printf("NO\n");
264         else printf("YES\n");
265     }
266     return 0;
267 }

```