

# Team Reference Document

Encore @ Harbin Institute of Technology

June 26, 2013

# Contents

<b>1 String Processing</b>	<b>1</b>	6.6 nim multiply . . . . .	15
1.1 AC Automaton . . . . .	1	6.7 FFT . . . . .	15
1.2 Suffix Array . . . . .	1	6.8 Romberg . . . . .	16
1.3 Suffix Automaton . . . . .	2	<b>7 Computational Geometry</b>	<b>16</b>
1.4 KMP . . . . .	2	7.1 Formula of Geometry . . . . .	16
1.5 Algorithm Z . . . . .	2	7.2 Float Method . . . . .	17
<b>2 Network Flow</b>	<b>2</b>	7.3 Int Method . . . . .	18
2.1 Max flow . . . . .	2	7.4 Polygon Method . . . . .	18
2.2 Cost flow . . . . .	3	7.5 Circles Method . . . . .	19
<b>3 Data Structure</b>	<b>3</b>	7.6 Scan Line . . . . .	19
3.1 DLX exact cover . . . . .	3	7.7 Spherical Distance . . . . .	19
3.2 DLX fuzzy cover . . . . .	4	7.8 Minimum Circle Cover . . . . .	20
3.3 Partition Tree . . . . .	4	7.9 N-D Volume . . . . .	20
3.4 Leftist Tree . . . . .	4	7.10 Perimeter Of Circles . . . . .	20
3.5 Cartesian Tree . . . . .	5	7.11 Center of Triangle . . . . .	21
3.6 Splay . . . . .	5	7.12 3D Vector Projection . . . . .	21
3.7 SplitTree . . . . .	6	7.13 3D Convexhull . . . . .	21
<b>4 Graph Theory</b>	<b>6</b>	7.14 2D Convexhull . . . . .	22
4.1 2-Satisfiability . . . . .	6	7.15 Points in Triangle . . . . .	22
4.2 Edge Cut . . . . .	7	7.16 N Dimension Cut . . . . .	22
4.3 Vertex Cut . . . . .	7	<b>8 Sevenzero Geometry</b>	<b>23</b>
4.4 Hopcroft Karp . . . . .	7		
4.5 Hungary Algorithm . . . . .	8		
4.6 KM . . . . .	8		
4.7 Stable Marriage . . . . .	8		
4.8 Maximum Clique . . . . .	8		
4.9 Maximal Clique . . . . .	8		
4.10 Euler Circles . . . . .	9		
4.11 Lowest Common Ancestor . . . . .	9		
4.12 Minimum Cut Algorithm . . . . .	9		
4.13 Degree-constrained Spanning Tree . . . . .	9		
4.14 Minimum Directed Tree . . . . .	9		
<b>5 Dynamic Programing</b>	<b>10</b>		
5.1 Mask DP I . . . . .	10		
5.2 Mask DP II . . . . .	10		
5.3 Vetterli Pair on Tree . . . . .	11		
<b>6 Math</b>	<b>11</b>		
6.1 Matrix . . . . .	11		
6.2 Number Theory . . . . .	12		
6.2.1 Phi . . . . .	12		
6.2.2 $a^x == b(modn)$ . . . . .	12		
6.2.3 $x * x == a(modp)$ . . . . .	12		
6.2.4 Miller and Pollard . . . . .	12		
6.2.5 Mod Equation . . . . .	13		
6.3 Fraction . . . . .	13		
6.3.1 $a/b < x/y < c/d$ . . . . .	13		
6.3.2 $x^2 - n * y^2 = 1$ . . . . .	13		
6.3.3 $\sum_{k=0}^{n-1} \lfloor (a + d * k)/m \rfloor$ . . . . .	14		
6.4 Linear Equation . . . . .	14		
6.4.1 Xor Equation . . . . .	14		
6.4.2 Equation in Z . . . . .	14		
6.4.3 Equation in R . . . . .	14		
6.4.4 det . . . . .	15		
6.5 Anti-Nim . . . . .	15		

# 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 = -1;
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 return ans;
24 }

```

## 1.5 Algorithm Z

```

1 #include <cmath>
2 #include <algorithm>
3 #include <stdio>
4 #include <cstring>
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 l1 = pos + next[pos], cur = next[i - pos];
13        if(l1 > i + cur) {
14            next[i] = cur;
15        } else {
16            j = max(l1 - 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 l1 = pos + extend[pos], cur = next[i - pos];
31         if(l1 > i + cur) {
32             extend[i] = cur;
33         } else {
34             j = max(l1 - i, 0);
35             while(str[i + j] == sub[j] && i + j < n && j < m) j++;
36             extend[i] = j;
37             pos = i;
38         }
39     }
}

```

## 2 Network Flow

### 2.1 Max flow

```

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

```

```

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

```

### 2.2 Cost flow

```

1 typedef long long USETYPE;
2 const USETYPE INF = numeric_limits<USETYPE>::max(); //limits>
3 template<typename T = int>
4 class mincost {
5 private:
6     const static int N = 1000;
7     const static int E = 100000;
8     struct edge {
9         int u, v;
10        T cost, cap;
11        edge *nxt;
12    } pool[E], *g[N], *pp, *pre[N];
13    T dist[N];
14
15    bool SPFA(int n, int s, int t) {
16        fill(dist, dist + n, INF);
17        int tail = 0, q[N] = {s};
18        dist[s] = 0;
19        bool vst[N] = {false};
20        vst[s] = true;
21        for(int i = 0; i <= tail; i++) {
22            int u = q[i % n];
23            for (edge *j = g[u]; j != NULL; j = j->nxt) {
24                int v = j->v;
25                if(j->cap && dist[u] != INF && dist[v] > dist[u] + j->
26                    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     };
}

```

## 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 /
11     SQRTSIZE)) * SIZE + k)
12 int maps[ROW][COL], ans[N];
13 char sudoku[SIZE][SIZE];
14 int r[N], l[N], u[N], d[N], c[N], s[N];
15 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] = 1;
40     for (int i = 1; i <= m; i++) {
41         if (i == m) {
42             r[i] = HEAD;
43         } else {
44             r[i] = i + 1;
45         }
46         l[i] = i - 1;
47         c[l] = u[l] = d[i] = i;
48         s[l] = 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         for (int i = 0; i < SIZE; i++) {
135             for (int j = 0; j < SIZE; j++)
136                 putchar(sudoku[i][j]);
137             puts("");
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], n;
8  public:
9      void initialize(int n, int *array) {
10         this->n = n;
11         for (int i = 1; i <= n; i++)
12             sorted[i] = tree[0][i] = array[i];
13         sort(sorted + 1, sorted + n + 1);
14     }
15     void build(int l, int r, int depth) {
16         if (l == r) return;
17         int mid = (l + r) / 2, same = 0, less = 0;
18         for (int i = 1; i <= r; i++)
19             less += (tree[depth][i] < sorted[mid]);
20         same = mid - l + 1 - less;
21         int lpos = l, rpos = mid + 1;
22         for (int i = 1; i <= r; i++) {
23             int w = tree[depth][i];
24             if (w < sorted[mid]) tree[depth + 1][lpos++] = w;
25             else if (w == sorted[mid] && same) {
26                 tree[depth + 1][lpos++] = w;
27                 same--;
28             }
29             else
30                 tree[depth + 1][rpos++] = w;
31             toleft[depth][i] = toleft[depth][l - 1] + lpos - 1;
32         }
33         build(l, mid, depth + 1);
34         build(mid + 1, r, depth + 1);
35     }
36     // ptree.query(l, n, a, b, 0, k) th kth number of [a, b]
37     int query(int l, int R, int l, int r, int depth, int k) {
38         if (l == r) return tree[depth][l];
39         int cnt, mid = (R + L) / 2, tmp1, tmp2;
40         cnt = toleft[depth][r] - toleft[depth][l - 1];
41         if (cnt >= k) {
42             tmp1 = L + toleft[depth][R] - toleft[depth][L - 1];
43             tmp2 = tmp1 + cnt - 1;
44             return query(L, mid, tmp1, tmp2, depth + 1, k);
45         }
46         tmp2 = r + toleft[depth][R] - toleft[depth][r];
47         tmp1 = tmp2 - (r - l - cnt);
48         return query(mid + 1, R, tmp1, tmp2, depth + 1, k - cnt);
49     }
50 }
51 };

```

### 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 <cstdio>
3  #include <cstring>
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         }
133         x->push_up();
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;

```

## 3.7 SplitTree

```

1 #define TREE(x) root[belong[x]], 1, 1, length[belong[x]]
2 #define L seg, id << 1, 1, mid
3 #define R seg, (id << 1) + 1, mid + 1, r
4 const int N = 10011, INF = 0x3f3f3f3f;
5 struct G {
6     int v[N << 1], next[N << 1], w[N << 1], id[N << 1], adj[N], ne;
7     void init() {
8         ne = 2;
9         memset(adj, 0, sizeof(adj));
10    }
11    void addEdge(int a, int b, int c, int d) {
12        v[ne] = b;
13        w[ne] = c;
14        id[ne] = d;
15        next[ne] = adj[a];
16        adj[a] = ne++;
17    }
18 } g;
19 class LCA {
20 private:
21     G* g;
22     int dfn[N], pos[N], st[20][N << 1], len;
23     void dfs(int cur, int p, int d) {
24         pos[cur] = len;
25         dfn[cur] = d;
26         st[0][len++] = cur;
27         for (int i = g->adj[cur]; i; i = g->next[i])
28             if (g->v[i] != p) {
29                 dfs(g->v[i], cur, d + 1);
30                 st[0][len++] = cur;
31             }
32    }
33    void initRMQ() {
34        for (int i = 1, k = 2; k < len; ++i, k <= 1)
35            for (int j = 0; j < len; ++j) {
36                int sk = j + (k >> 1);
37                st[i][j] = st[i - 1][j];
38                if (sk < len && dfn[st[i][j]] > dfn[st[i - 1][sk]])
39                    st[i][j] = st[i - 1][sk];
40            }
41    }
42 public:
43     void init(G &graph, int root) {
44         g = &graph;
45         len = 0;
46         dfs(root, -1, 0);
47         initRMQ();
48    }
49     int query(int x, int y) {
50         x = pos[x], y = pos[y];
51         if (x > y) swap(x, y);
52         int len = y - x + 1, bl;
53         for (bl = -1; len != 0; ++bl, len >= 1);
54         x = st[bl][x];
55         y = st[bl][y - (1 << bl) + 1];
56         return dfn[x] < dfn[y] ? x : y;
57    }
58 } lca;
59 int fa[N], size[N], idv[N], wgt[N];
60 void dfs(int cur) {
61     size[cur] = 1;
62     for (int i = g->adj[cur]; i; i = g->next[i])
63         if (g->v[i] != fa[cur]) {
64             fa[g->v[i]] = cur;
65             wgt[g->v[i]] = g->w[i];
66             idv[g->id[i]] = g->v[i];
67             dfs(g->v[i]);
68             size[cur] += size[g->v[i]]; // notice that
69         }
70 }
71 // begin segment tree
72 int segt[N << 2], length[N], *root[N], *ne;
73 int* newSegTree(int len) {
74     int *res = ne;
75     ne += (len << 2);
76     return res;
77 }
78 int ar[N], belong[N], pos[N];
79 void build(int* seg, int id, int l, int r) {
80     if (l == r) seg[id] = wgt[ar[l]];
81     else {
82         int mid = l + r >> 1;
83         build(L);
84         build(R);
85         seg[id] = max(seg[id << 1], seg[(id << 1) + 1]);
86     }
87 }
88 void insert(int* seg, int id, int l, int r, int x, int t) {
89     if (l == r) seg[id] = t;
90     else {
91         int mid = l + r >> 1;
92         if (x <= mid) insert(L, x, t);
93         else insert(R, x, t);
94         seg[id] = max(seg[id << 1], seg[(id << 1) + 1]);
95     }
96 }
97 int getMax(int *seg, int id, int l, int r, int ll, int rr) {
98     if (ll <= l && r <= rr) return seg[id];
99     else {
100         int mid = l + r >> 1;
101         int res = -INF;
102         if (ll <= mid) res = max(res, getMax(L, ll, rr));
103         if (rr > mid) res = max(res, getMax(R, ll, rr));
104         return res;
105     }
106 }
107 // end segment tree
108 void createTree(int cur, int len) {
109     ar[len] = cur;
110     int maxsize = 0, next = 0;
111     for (int i = g->adj[cur]; i; i = g->next[i])
112         if (g->v[i] != fa[cur] && size[g->v[i]] > maxsize)
113             maxsize = size[g->v[i]], next = g->v[i];
114     if (next) {
115         createTree(next, len + 1);
116         for (int i = g->adj[cur]; i; i = g->next[i])
117             if (g->v[i] != fa[cur] && g->v[i] != next)

```

```

118         createTree(g.v[i], 1);
119     } else {
120         int p = cur;
121         for (int i = 1; i < len; ++i) p = fa[p];
122         root[p] = newSegTree(len);
123         length[p] = len;
124         build(root[p], 1, 1, len);
125         for (int i = 1; i <= len; ++i) {
126             belong[ar[i]] = p;
127             pos[ar[i]] = i;
128         }
129     }
130 }
131 void buildTree() {
132     lca.init(g, 1);
133     fa[1] = -1;
134     wgt[1] = -INF;
135     dfs(1);
136     ne = segt;
137     createTree(1, 1);
138 }
139 int query(int a, int b) {
140     int res = -INF, t;
141     while (a != b) {
142         if (belong[a] == belong[b]) {
143             t = getMax(TREE(b), pos[a] + 1, pos[b]);
144             if (t > res) res = t;
145             break;
146         } else {
147             t = getMax(TREE(b), 1, pos[b]);
148             if (t > res) res = t;
149             b = fa[belong[b]];
150         }
151     }
152     return res;
153 }
154 int main() {
155     int n, cas;
156     for (cin >> cas; cas; --cas) {
157         g.init(); scanf("%d", &n);
158         int a, b, c;
159         rep(1, 1, n) {
160             scanf("%d%d%d", &a, &b, &c);
161             g.addEdge(a, b, c, i);
162             g.addEdge(b, a, c, i);
163         }
164         buildTree(); char s[20];
165         while (scanf("%s", s), s[0] != 'D') {
166             scanf("%d%d", &a, &b);
167             if (s[0] == 'Q') {
168                 c = lca.query(a, b);
169                 printf("%d\n", max(query(c, a), query(c, b)));
170             } else {
171                 int v = idv[a]; // notice that !!!
172                 insert(TREE(v), pos[v], b);
173             }
174         }
175     }
176 }

```

```

49         }while (w != x);
50     }
51 }
52 void toposort(int v) {
53     reach[v] = true;
54     for (edge *i = gscv[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(gscv, 0, sizeof(gscv)); //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], gscv);
66         }
67     }
68 }
69 void becolor(int v) {
70     color[v] = BLUE;
71     for (edge *i = gscv[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 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 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 ~x) & ('y V ~y)
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], *pg, *gscv[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;

```

### 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], *pg, *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     low[x] = dfn[x] = ++depth;
14     int s = adj[x].size(), w, num = 0;
15     REP(i, 0, s) {
16         w = adj[x][i];
17         if (!dfn[w]) {
18             num++;
19             dfs(w, ROOT);
20             low[x] = min(low[w], low[x]);
21             if (x == ROOT && num >= 2)
22                 cnt[x]++;
23             if (x != ROOT && dfn[x] <= low[w])
24                 cnt[x]++;
25         }
26         else low[x] = min(low[x], dfn[w]);
27     }
28 }
29 int solve(int n) {
30     int cc = 0;
31     REP(i, 0, n) {
32         if (dfn[i] == 0) {
33             dfs(i, i);
34             cc++;
35         }
36     }
37     return cc;
38 }
39 int main() {
40     int n, m, x, y;
41     while (scanf("%d %d %d", &n, &m) == 2 && n + m) {
42         initialize(n);
43         REP(i, 0, m) {
44             scanf("%d %d", &x, &y);
45             adj[x].push_back(y);
46             adj[y].push_back(x);
47         }
48         int ans = solve(n);
49         if (m == 0) printf("%d\n", n - 1);
50         else printf("%d\n", ans + *max_element(cnt, cnt + n));
51     }
52     return 0;
53 }

```

### 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         dx[My[v]] = dy[v] + 1;
23         Q.push(My[v]);
24     }
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         }

```

```

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             }
24             else if (slack[y] > t) {
25                 slack[y] = t;
26             }
27         }
28         return false;
29     }
30     double KM() {
31         memset(linky, -1, sizeof(linky));
32         for (int i = 1; i <= nx; i++) lx[i] = -INF;
33         memset(ly, 0, sizeof(ly));
34         for (int i = 1; i <= nx; i++)
35             for (int j = 1; j <= ny; j++)
36                 if (w[i][j] > lx[i]) lx[i] = w[i][j];
37         for (int x = 1; x <= nx; x++) {
38             for (int i = 1; i <= ny; i++) slack[i] = INF;
39             while (true) {
40                 memset(visx, 0, sizeof(visx));
41                 memset(visy, 0, sizeof(visy));
42                 if (find(x)) break;
43                 double d = INF;
44                 for (int i = 1; i <= ny; i++)
45                     if (!visy[i]) d = min(d, slack[i]);
46                 if (d == INF) return -1;
47                 for (int i = 1; i <= nx; i++)
48                     if (visx[i]) lx[i] -= d;
49                 for (int i = 1; i <= ny; i++)
50                     if (visy[i]) ly[i] += d;
51                 else slack[i] -= d;
52             }
53         }
54         int cnt = 0;
55         for (int i = 1; i <= ny; i++)
56             if (linky[i] != -1) cnt++;
57         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 tmpp[N], tmpps = 0, tmpx[N], tmpxs = 0;
14     for(int i = 0; i < ps; i++) {
15         int v = p[i];
16         tmpps = tmpxs = 0;
17         for(int j = i + 1; j < ps; j++) {
18             int u = p[j];
19             if(maps[v][u])

```

```

20             tmpp[tmpps++] = u;
21         }
22         for(int j = 0; j < xs; j++) {
23             int u = x[j];
24             if(maps[v][u])
25                 tmpx[tmpxs++] = u;
26         }
27         CountMaximalClique(tmpp, tmpps, tmpx, 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 Euler Circles

```

1  绪论
2  const int SIZE = 2 * 2000, N = 50;
3  /* Euler degree & connection
4  * fordwn 研究paths the smallest lexicographic path
5  * hoj 1045 John's trip*/
6  struct edge {
7      int v, id;
8      bool operator<(const edge a) const {
9          return id < a.id;
10     }
11 } edges[SIZE];
12 vector<edge> adj[N];
13 int path[SIZE].E, V, S, deg[N], stp;
14 bool vst[SIZE];
15 void dfs(int now) {
16     edge tmp;
17     for (size_t i = 0; i < adj[now].size(); i++) {
18         tmp = adj[now][i];
19         if (!vst[tmp.id] && !vst[tmp.id]) {
20             vst[tmp.id] = vst[tmp.id] = 1;
21             dfs(tmp.v);
22             path[stp++] = tmp.id;
23         }
24     }
25 }
26 void solve() { {
27     for (int i = 0; i < V; i++)
28         sort(adj[i].begin(), adj[i].end());
29     dfs(S);
30     printf("%d", path[stp - 1]);
31     for (int i = stp - 2; i >= 0; i--)
32         printf(" %d", path[i]);
33     putchar('\n');
34 }
35 void initialize(int u, int v) {
36     stp = V = E = S = 0;
37     for (int i = 0; i < N; i++) adj[i].clear();
38     memset(vst, false, sizeof(vst));
39     memset(deg, 0, sizeof(deg));
40     S = min(v, u);
41 }
42 void add_edge(int u, int v, int id, int E) {
43     deg[u]++, deg[v]++;
44     edges[E].v = v, edges[E].id = id;
45     adj[u].push_back(edges[E]);
46     edges[E].v = u, edges[E].id = id;
47     adj[v].push_back(edges[E]);
48 }

```

## 4.11 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 }

```

## 4.12 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 }

```

## 4.13 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 }

```

## 4.14 Minimum Directed Tree

```

1 const int N = 1010, E = N * N;
2 const LL INF = 1000000000000LL;
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                 pre[v] = u;
20             inEdge[v] = edge[i].c;
21         }
22     }
23     REP(i, 0, n) {
24         if (i == root) continue;
25         if (inEdge[i] == INF) return -1;
26     }
27     int now = 0;
28     CC(label, -1), CC(visit, -1);
29     inEdge[root] = 0;
30     REP(i, 0, n) {
31         ans += inEdge[i];
32         int v = i;
33         while (visit[v] != i && label[v] == -1 && v != root) {
34             visit[v] = i;
35             v = pre[v];
36         }
37         if (v != root && label[v] == -1) {
38             for (int u = pre[v]; u != v; u = pre[u])
39                 label[u] = now;
40             label[v] = now++;
41         }
42     }
43     if (now == 0) break;
44     REP(i, 0, n) if (label[i] == -1) label[i] = now++;
45     REP(i, 0, m) {
46         int v = edge[i].v;
47         edge[i].v = label[edge[i].v];
48         edge[i].u = label[edge[i].u];
49         if (edge[i].v != edge[i].u) edge[i].c -= inEdge[v];
50     }
51     root = label[root];
52     n = now;
53 }
54 }
55 return ans;
56 }

```

## 5 Dynamic Programing

### 5.1 Mask DP I

```

1 const int N = 10, TOT = 1000;
2 const int MAXN = 1594323; // 3^13
3 char maps[N][N];
4 int bit3[N] = {}, status[TOT], Hash[MAXN], allS = 0;
5 LL dp[2][TOT];
6 bool check(int s) {
7     int cnt = 0;
8     while(s) {
9         int n = s % 3;
10         if (n == 1) cnt++;
11         if (n == 2) cnt--;
12         if (cnt < 0) return false;
13         s /= 3;
14     }
15     return (cnt == 0);
16 }
17 int getbit(int s, int i) {
18     return s / bit3[i] % 3;
19 }
20 void transfer(LL& dest, LL add) {
21     dest = -1 ? (dest = add) : (dest += add);
22 }
23 LL DP(int n, int m) {
24     int now = 0, pre = 1, state = (1 + 2 * bit3[m - 1]) * 3;
25     CC(dp, 0), dp[0][0] = 1;
26     for (int i = 0; i < n; i++) {
27         for (int j = 0; j < m; j++) {
28             swap(now, pre); CC(dp[now], 0);
29             for (int k = 0, s; s = status[k], s < bit3[m + 1]; k++) {
30                 if (dp[pre][k] == 0) continue;
31                 int l = getbit(s, j), u = getbit(s, j + 1);
32                 int nows = s - 1 * bit3[j] - u * bit3[j + 1];
33                 if (maps[i][j] != '.') {
34                     if (l == 0 && u == 0) {
35                         transfer(dp[now][k], dp[pre][k]);
36                     }
37                 } else if (l == 0 && u == 0) { //build a pair () when can
38                     walk down and right
39                     if (maps[i][j + 1] == '.' && maps[i + 1][j] == '.') {
40                         int nxt = nows + bit3[j] + 2 * bit3[j + 1];
41                         transfer(dp[now][Hash[nxt]], dp[pre][k]);
42                     }
43                 } else if (l == 1 && u == 1) { //merge ((
44                     int cnt = 0;
45                     for (int b = j + 2; b <= m; b++) {
46                         int tmp = getbit(nows, b);
47                         if (tmp == 2) cnt--;
48                         if (tmp == 1) cnt++;
49                         if (cnt == -1) {
50                             transfer(dp[now][Hash[nows - bit3[b]]], dp[pre][k]);
51                             break;
52                         }
53                     }
54                 } else if (l == 2 && u == 2) { //merge ))
55                     int cnt = 0;

```

```

55     for(int b = j - 1; b >= 0; b--) {
56         int tmp = getbit(nows, b);
57         if(tmp == 1) cnt++;
58         if(tmp == 2) cnt--;
59         if(cnt == 1) {
60             transfer(dp[now][Hash[nows + bit3[b]]], dp[pre][k]);
61             break;
62         }
63     }
64     } else if(1 == 2 && u == 1) { //merge } {
65         transfer(dp[now][Hash[nows]], dp[pre][k]);
66     } else if(!1 && u || 1 && !u) {
67         if(maps[i + 1][j] == '.')
68             transfer(dp[now][Hash[nows + (1 + u) * bit3[j]]], dp[pre][k]);
69         if(maps[i][j + 1] == '.')
70             transfer(dp[now][Hash[nows + (1 + u) * bit3[j + 1]]], dp[pre][k]);
71     } else if(1 == 1 && u == 2) { /* only happen last step */
72     }
73     }
74     swap(pre, now); CC[dp[now], 0]; //memset to set illegal state -1
75     for(int k = 0, s; s = status[k], s < bit3[m]; k++)
76         if(dp[pre][k] != -1) dp[now][Hash[s * 3]] = dp[pre][k];
77     }
78     return max(0LL, dp[now][Hash[state]]);
79 }
80 int main() {
81     int n, m;
82     REP(1, 1, N) bit3[i] = bit3[i - 1] * 3;
83     REP(1, 0, bit3[N - 1]) {
84         if(check(1)) {
85             Hash[i] = allS;
86             status[allS++] = i;
87         } else {
88             Hash[i] = -1;
89         }
90     }
91     status[allS] = MAXN;
92     while(scanf("%d %d", &n, &m) == 2 && (n + m)) {
93         CC(maps, 0);
94         REP(1, 0, n) scanf("%s", maps[i]);
95         maps[n][0] = maps[n][m - 1] = '.';
96         printf("%I64d\n", DP(n, m));
97     }
98     return 0;
99 }

```

## 5.2 Mask DP II

```

1  class HashTable {
2  private:
3      const static int SIZE = 1000000, MOD = 10007;
4      struct HashCell {
5          int value, idx;
6          HashCell *mask;
7      } pool[SIZE], *g[MOD], *pp;
8      #define hashFunction(x) ((x) % MOD)
9  public:
10     void clear() {
11         memset(g, 0, sizeof(g));
12         pp = pool;
13     }
14     int find(int x) {
15         int hash = hashFunction(x);
16         for(HashCell *i = g[hash]; i != NULL; i = i->mask) {
17             if(i->value == x) return i->idx;
18         }
19         return -1;
20     }
21     void insert(int x, int idx) {
22         int hash = hashFunction(x);
23         pp->idx = idx, pp->value = x;
24         pp->mask = g[hash];
25         g[hash] = pp++;
26     }
27     HashTable() {
28         const int N = 10;
29         const int STATE_CNT = 1000000;
30         const int INF = 10000000;
31         const int HEX = 10; /* BIT[i] = HEX^i */
32         const int BIT[] = {1, 10, 100, 1000, 10000, 100000, 1000000, 10000000, 100000000, 1000000000};
33         int state[2][STATE_CNT], dp[2][STATE_CNT];
34         int encode(const int s[], const int M) { /* repeat down to keep min-notation */
35             int lab[N], cnt = 0, newS = 0;
36             memset(lab, -1, sizeof(lab));
37             lab[0] = cnt++; /* 0 means not be used */
38             for(int i = M - 1; i >= 0; i--) {
39                 newS *= HEX;
40                 newS += (lab[s[i]] = lab[s[i]] == -1 ? cnt++ : lab[s[i]]);
41             }
42             return newS;
43         }
44         int cnt[N]; /* the number of each plugin */
45         void decode(int src, int* dest, const int M) { /* decode mask code into array */
46             memset(cnt, 0, sizeof(cnt));
47             REP(i, 0, M) {
48                 cnt[src % HEX]++;
49                 dest[i] = src % HEX;
50                 src /= HEX;
51             }
52         }
53         bool isOneBlock(int state) {
54             int last = -1;
55             while(state) {
56                 int now = state % HEX;
57                 if(now != 0 && now != last && last != -1) return false;
58                 if(now != 0) last = now;
59                 state /= HEX;
60             }
61             return true;

```

```

62     }
63     void transfer(int now, int mask, int val, int& newCnt) {
64         int idx = hashTable.find(mask);
65         if(idx != -1) {
66             dp[now][idx] = max(dp[now][idx], val);
67         } else {
68             idx = newCnt;
69             hashTable.insert(mask, newCnt++);
70             state[now][idx] = mask;
71             dp[now][idx] = val;
72         }
73     }
74     int DP(int n, int m, int maps[N][N]) {
75         state[0][0] = dp[0][0] = 0;
76         int ans = -INF, newS[N], now = 0, pre = 1;
77         int oldCnt = 0, newCnt = 1, M = m + 1, mask;
78         REP(1, 0, n) {
79             REP(j, 0, m) {
80                 now ^= 1, pre ^= 1;
81                 oldCnt = 0;
82                 swap(oldCnt, newCnt);
83                 hashTable.clear();
84                 REP(k, 0, oldCnt) {
85                     decode(state[pre][k], newS, M);
86                     if(newS[j] == 0 && newS[j + 1] == 0) {
87                         transfer(now, state[pre][k], dp[pre][k], newCnt);
88                         newS[j] = newS[j + 1] = *max_element(newS, newS + M) + 1;
89                         mask = encode(newS, M);
90                         transfer(now, mask, dp[pre][k] + maps[i][j], newCnt);
91                     } else if(newS[j] == 0 && newS[j + 1]) {
92                         newS[j] = newS[j + 1];
93                         mask = encode(newS, M);
94                         transfer(now, mask, dp[pre][k] + maps[i][j], newCnt);
95                     } if(cnt[newS[j + 1]] > 1) {
96                         newS[j] = newS[j + 1] = 0;
97                         mask = encode(newS, M);
98                         transfer(now, mask, dp[pre][k], newCnt);
99                     }
100                 } else if(newS[j] && newS[j + 1] == 0) {
101                     newS[j + 1] = newS[j];
102                     mask = encode(newS, M);
103                     transfer(now, mask, dp[pre][k] + maps[i][j], newCnt);
104                     if(cnt[newS[j]] > 1) {
105                         newS[j] = newS[j + 1] = 0;
106                         mask = encode(newS, M);
107                         transfer(now, mask, dp[pre][k], newCnt);
108                     }
109                 } else if(newS[j] && newS[j + 1]) {
110                     /* drop current block */
111                     int a = 0, b = 0;
112                     if(cnt[newS[j]] == newS[j + 1]) {
113                         if(cnt[newS[j]] > 2) {
114                             swap(a, newS[j]), swap(b, newS[j + 1]);
115                             int mask = encode(newS, M);
116                             transfer(now, mask, dp[pre][k], newCnt);
117                             swap(a, newS[j]), swap(b, newS[j + 1]);
118                         }
119                     } else {
120                         if(cnt[newS[j]] > 1 && cnt[newS[j + 1]] > 1) {
121                             swap(a, newS[j]), swap(b, newS[j + 1]);
122                             int mask = encode(newS, M);
123                             transfer(now, mask, dp[pre][k], newCnt);
124                             swap(a, newS[j]), swap(b, newS[j + 1]);
125                         }
126                     }
127                     /* merge two block */
128                     int minn = min(newS[j], newS[j + 1]);
129                     for(int b = 0; b <= minn; b++) {
130                         if(newS[b] == newS[j] || newS[b] == newS[j + 1])
131                             newS[b] = minn;
132                     }
133                     mask = encode(newS, M);
134                     transfer(now, mask, dp[pre][k] + maps[i][j], newCnt);
135                 }
136             }
137         }
138         now ^= 1, pre ^= 1;
139         oldCnt = 0;
140         swap(oldCnt, newCnt);
141         hashTable.clear(); /* two different mask can change into one */
142         REP(k, 0, oldCnt) {
143             if(isOneBlock(state[pre][k]))
144                 ans = max(ans, dp[pre][k]);
145             if(state[pre][k] - BIT[m] > 0) {
146                 decode((state[pre][k] - BIT[m]) * 10, newS, M);
147                 if(mask = encode(newS, M)) /* if mask != 0 */
148                     transfer(now, mask, dp[pre][k], newCnt);
149             } else if(state[pre][k] != 0) {
150                 decode(state[pre][k] * 10, newS, M);
151                 if(mask = encode(newS, M))
152                     transfer(now, mask, dp[pre][k], newCnt);
153             }
154         }
155     }
156     return ans;
157 }
158
159 int main() {
160     int n;
161     while(scanf("%d", &n) == 1 && n) {
162         int maps[N][N], ans = -INF;
163         REP(i, 0, n) {
164             REP(j, 0, n) {
165                 scanf("%d", &maps[i][j]);
166                 ans = max(ans, maps[i][j]);
167             }
168         }
169         if(ans <= 0)
170             printf("%d\n", ans);
171         else
172             printf("%d\n", DP(n, n, maps));
173     }
174     return 0;
175 }

```

## 5.3 Veterx Pair on Tree

```

1  const int V = 10000, E = V << 1;
2  struct edge {
3      int v, c;
4      edge *nxt;
5  }pool[E], *g[V], *pp;
6  //initialize and add edge Function
7  int _size[V], vis[V], dist[V], root, maxn;
8  void select(int v, int pre, int tree_size) {
9      int max_sub = 0;
10     _size[v] = 1;
11     for(edge* i = g[v]; i != NULL; i = i->nxt) {
12         if(i->v == pre || vis[i->v]) continue;
13         select(i->v, v, tree_size);
14         _size[v] += _size[i->v];
15         checkmax(max_sub, _size[i->v]);
16     }
17     checkmax(max_sub, tree_size - _size[v]);
18     if(checkmin(maxn, max_sub))
19         root = v;
20 }
21 int _count(int beg, int end, int k) {
22     int ret = 0, lo = beg, hi = end - 1;
23     sort(dist + beg, dist + end);
24     while(lo < hi) {
25         if(dist[hi] + dist[lo] <= k)
26             ret += hi - lo++;
27         else
28             hi--;
29     }
30     return ret;
31 }
32 void dfs(int root, int pre, int len, int& end) {
33     dist[end++] = len, _size[root] = 1;
34     for(edge *i = g[root]; i != NULL; i = i->nxt) {
35         if(i->v == pre || vis[i->v]) continue;
36         dfs(i->v, root, len + i->c, end);
37         _size[root] += _size[i->v];
38     }
39 }
40 int get_sub_solve(int root, int k) {
41     int beg = 0, end = 0, res = 0;
42     for(edge*i = g[root]; i != NULL; i = i->nxt) {
43         if(vis[i->v]) continue;
44         dfs(i->v, root, i->c, end);
45         res += _count(beg, end, k);
46         beg = end;
47     }
48     dist[end++] = 0;
49     res += _count(0, end, k);
50     return res;
51 }
52 //number of path less than k
53 void solve(int n, int k) {
54     queue<int> q;
55     q.push(0);
56     _size[0] = n;
57     CC(vis, 0);
58     int res = 0;
59     while(q.empty() == false) {
60         int now = q.front();
61         q.pop();
62         maxn = numeric_limits<int>::max();
63         select(now, -1, _size[now]);
64         vis[root] = 1;
65         for(edge*i = g[root]; i != NULL; i = i->nxt)
66             if(vis[i->v] == false) q.push(i->v);
67         res += get_sub_solve(root, k);
68     }
69     printf("%d\n", res);
70 }

```

```

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 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)

```

### 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), p - 2, p);
18     for (id[0] = 0, mexp[0] = i = 1; i < m; ++i) {
19         id[i] = i; mexp[i] = mexp[i - 1] * (llong) a % p;
20         if (mexp[i] == b) return i;
21     }
22     stable_sort(id, id + m, log_cmp);
23     sort(mexp, mexp + m);
24     for (i = 0; i < m; ++i) {
25         j = lower_bound(mexp, mexp + m, b) - mexp;
26         if (j < m && mexp[j] == b) return i * m + id[j];
27         b = b * (llong) inv % p;

```

```

28     }
29     return -1;
30 }

```

$a^x = b \pmod{n}$  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
            vu[i];
12        return -1;
13    }
14    void insert(int t, int val) {
15        int key = t&N;
16        v[key] = t;
17        vu[key] = val;
18        next[key] = g[key];
19        g[key] = ne++;
20    }
21 } S;
22
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)
        return i;
36     for (r = 1, res = 0; (tmp = gcd(a, n)) > 1; ++res) {
37         if (b % tmp) return -1;
38         b /= tmp; n /= tmp; r = r * a / tmp % n;
39     }
40     S.init();
41     extend_gcd(r, n, d, x, y);
42     b = (b * x % n + n) % n;
43     int s = (int) ceil(sqrt(n+0.0));
44     for (i = 0, t = 1; i < s; ++i, t = t * a % n) {
45         if (t == b) return i + res;
46         if (S.find(t) == -1) S.insert(t, i);
47         else return -1;
48     }
49     extend_gcd(t, n, d, x, y);
50     x = (x % n + n) % n;
51     for (i = 0; i < s; ++i) {
52         tmp = S.find(b);
53         if (tmp != -1) return i * s + res + tmp;
54         b = b * x % n;
55     }
56     return -1;
57 }

```

### 6.2.3 $x * x == a \pmod{p}$

```

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                 res = (mod_pow(a, (i + 1) / 2, n) * (llong) mod_pow(b, k /
24                     2, n)) % n;
25             }
26             return min(res, n - res); // make that res <= n/2
27         }
28         return -1;
29     }
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 ll mult(ll a, ll b, ll mod) {
2     if (a >= mod) a %= mod;
3     if (b >= mod) b %= mod;
4     if (a <= (1LL<<31) && b <= (1LL<<31)) return a*b%mod;
5
6     ll res = 0;
7     while (b) {
8         if (b&1) {
9             res += a;
10            if (res >= mod) res -= mod;
11        }
12        a <<= 1;
13        if (a >= mod) a -= mod;
14        b >>= 1;
15    }
16    return res;
17 }
18
19 ll gcd(ll a, ll b) {
20     return b ? gcd(b, a%b) : a;
21 }
22
23 /// here is the fast code.
24 ll val[1<<20];
25 ll p_rho(ll n, int limit = 1 << 17) {
26     if (0 == (n&1)) return 2; // must
27     ll d;
28     for (ll c = 1; c < n; ++c) {
29         val[0] = 2;
30         for (int i = 1; i < limit; ++i) {
31             val[i] = (mult(val[i - 1], val[i - 1], n) + c) % n;
32             if (0 == (i & 1)) {
33                 d = gcd(val[i] - val[i>>1] + n, n);
34                 if (d == n) break;
35                 if (d > 1) return d;
36             }
37         }
38     }
39     return n;
40 }
41
42 ll power(ll a, ll b, ll mod) {
43     ll res = 1, t = a;
44     while (b) {
45         if (b&1) res = mult(res, t, mod);
46         t = mult(t, t, mod);
47         b >>= 1;
48     }
49     return res;
50 }
51
52 ll witness(ll a, ll n) {
53     ll b = n - 1;
54     int cnt = 0;
55     while (0 == (b&1)) ++cnt, b >>= 1;
56     ll x = power(a, b, n), y;
57     for (int i = 0; i < cnt; ++i) {
58         y = mult(x, x, n);
59         if (y == 1) {
60             if (x != 1 && x != n - 1) return 0;
61             return 1;
62         }
63         x = y;
64     }
65     return x;
66 }
67
68 bool is_prime(ll n) {
69     if (n == 2) return true;
70     if (n < 2 || (n&1) == 0) return false;
71     int p[5] = {3, 5, 7, 11, 13};
72     for (int i = 0; i < 5; ++i) {
73         if (n == p[i]) return true;
74         if (n % p[i] == 0) return false;
75     }
76
77     for (int i = 0; i < 10; ++i)
78         if (witness(rand() % (n-2) + 2, n) != 1) return false;
79     return true;
80 }
81

```

### 6.2.5 Mod Equation

```

1 #include <cstdio>
2 #include <algorithm>
3 #include <cstring>
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 }

```

```

30 }
31
32 int n;
33 __int64 m[10000], r[10000];
34
35 int main() {
36     while (scanf("%d", &n) == 1) {
37         for (int i = 0; i < n; ++i)
38             scanf("%I64d%I64d", &m[i], &r[i]);
39         printf("%I64d\n", getAns(m, r, n));
40     }
41     return 0;
42 }

```

## 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)*(11)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-squre-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 $\sum_{k=0}^{n-1} \lfloor (a + d * k) / m \rfloor$

```

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 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++]
12            = k;
13        for (j = 0; j < m; ++j) if (j != r && a[j][i])
14            for (k = 0; k < num; ++k) a[j][list[k]] ^= 1;
15        ++r;
16    }
17    for (i = 0; i < m; ++i)
18        if (a[i][nn]) {
19            for (j = 0; j < nn && !a[i][j]; ++j);
20            if (j == nn) return 0; // else x[j] = a[i][nn]/a[i][j];
21        }
22    // the number of free variable = (nn - r)
23    return 1;
24 }

```

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 fracton 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             maxp = A[i][j];
58             indr = i;
59             indc = j;
60         }
61         if (zero(maxp)) return 0;
62         swapA(r, c, indr, indc);
63         pivot(r, c);
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 递归
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 <stdio>
2 #include <iostream>
3 #include <string>
4 #include <cmath>
5 #include <time>
6 #include <stdlib>
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 sgV[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     d2 = nim_mult_power(p,s);
66     d1 = nim_mult_power(p,t);
67     return ((d1*d2) << a) ^ nim_mult_power(m/2,d1);
68 }
69
70
71 int nim_mult(int x,int y) {
72     if (x < y) swap(x,y);
73     if (x < 16) return sg(x,y);
74     int a, m, p, q, s, t, c1, c2, c3;
75     for (a = 1; (1LL << a) <= x; a <= 1);
76     a >>= 1, m = (1<<a);
77     p = x/m, q = x%(m-1);
78     s = y/m, t = y%(m-1);
79     c1 = nim_mult(p,s);
80     c2 = nim_mult(p,t)^nim_mult(q,s);
81     c3 = nim_mult(q,t);
82     return ((c1*c2) << a)^c3^nim_mult_power(m/2,c1);
83 }
84
85 int n;
86 int main() {
87     init();
88     //test();
89     int cas;
90     for (cin >> cas; cas; --cas) {
91         scanf("%d",&n);
92         int res = 0, x, y;
93
94         for (int i = 0; i < n; ++i) {
95             scanf("%d%d",&x,&y);
96             res ^= nim_mult(x,y);
97         }
98         if (res) puts("Have a try, lxhgw.");
99         else puts("Don't waste your time.");
100     }
}

```

## 6.7 FFT

```

1 // hdu 1402
2 #include <stdio>
3 #include <string>
4 #include <math>
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 }

```

```

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]);
40
41         k = len >> 1;
42         while (j >= k) {
43             j -= k;
44             k >>= 1;
45         }
46         if (j < k) j += k;
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[i] - '0' : 0.0;
80         xb[i].r = (i < lb) ? b[i] - '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 = l; i >= 0; i--) putchar(sum[i] + '0');
107         putchar('\n');
108     }
109     return 0;
110 }

```

## 6.8 Romberg

```

1 /*求 f(x) 在区间[aa, bb] 上的积分*/
2 #include <iostream>
3 #include <stdio>
4 #include <math>
5 using namespace std;
6 const int N = 11, MAXR = 17; //迭代次数
7 const double eps = 1e-12; //精度
8 int n;
9 double a[N];
10 double f(double x){
11     double k = 0.0, xx = 1;
12     for(int i = 1; i <= n; ++i){
13         k += a[i] * xx;
14         xx *= x;
15     }
16     return sqrt(1 + k * k);
17 }
18 double Romberg(double aa, double bb){ //aa bb 积分下上限
19     int m, n; //m 控制迭代次数, n 控制复化梯形积分的分点数n=2^m
20     double h, x;
21     double s, q;
22     double ep; //精度要求
23     double y[MAXR]; //为节省空间只需一维数组,
24     //每次循环依次存储 Romberg 计算表的每行元素以供计算下一行算完后更新,,

```

```
25 double p; // p 总是指示待计算元素的前一个元素-> 同一行
26 //迭代初值
27 h = bb - aa;
28 y[0] = 0.5 * h * (f(aa) + f(bb));
29 m = n = 1;
30 ep = eps + 1.0;
31 //迭代计算
32 while (ep >= eps && m < MAXR) {
33     //复化积分公式求 T2n -> Romberg 计算表中的第一列n 初始为1 以后倍增
34     p = 0.0;
35     for (int i = 0; i < n; ++i) //求 Hn{
36         x = aa + (i + 0.5) * h;
37         p += f(x);
38     }
39     p = 0.5 * (y[0] + h * p); //求 T2n = 1/2 (Tn + Hn) 用p 指示
40
41     //求第 m 行元素根据Romberg 计算表本行的前一个元素(p 指示)
42     //和上一行左上角元素 -> y[k - 1] 指示求得
43     s = 1.0;
44     for (int k = 1; k <= m; ++k) {
45         s = pow(4.0, k); //!!!
46         //s *= 4.0;
47         q = (pow(4.0, k) * p - y[k - 1]) / (pow(4.0, k) - 1.0);
48         y[k - 1] = p;
49         p = q;
50     }
51     ep = fabs(q - y[m - 1]);
52     y[m++] = q;
53     n *= 2;
54     h *= 0.5;
55 }
56 return q;
57 }
58 int main()
59 {
60     while(1 == scanf("%d", &n)) {
61         for(int i = 1; i <= n; ++i) {
62             scanf("%lf", a + i);
63             a[i] *= i;
64         }
65         printf("%.2lf\n", Romberg(0.0, 10.0));
66     }
67     return 0;
68 }
```

## 7 Computational Geometry

### 7.1 Formula of Geometry

```
1 三角形
2  :
3  1. 半周长P=(a+b+c)/2
4  2. 面积S=aHa/2=absin(C)/2=sqrt(P(P-a)(P-b)(P-c))
5  3. 中线Ma=sqrt(2(b^2+c^2)-a^2)/2=sqrt(b^2+c^2+2bccos(A))/2
6  4. 角平分线Ta=sqrt(bc((b+c)^2-a^2))/(b+c)=2bccos(A/2)/(b+c)
7  5. 高线Ha=bsin(C)=csin(B)=sqrt(b^2-(a^2+b^2-c^2)/(2a))^2
8  6. 内切圆半径r=S/P=asin(B/2)sin(C/2)/sin((B+C)/2)
9      =4Rsin(A/2)sin(B/2)sin(C/2)=sqrt((P-a)(P-b)(P-c)/P)
10     =Ptan(A/2)tan(B/2)tan(C/2)
11 7. 外接圆半径R=abc/(4S)=a/(2sin(A))=b/(2sin(B))=c/(2sin(C)) 四边形
12
13  :
14 D1, 为对角线D2, 为对角线中点连线M, 为对角线夹角A
15 1. a^2+b^2+c^2+d^2=D1^2+D2^2+4M^2
16 2. S=D1D2sin(A)/2 以下对圆的内接四边形
17 ()
18 3. ac+bd=D1D2
19 4. S=sqrt((P-a)(P-b)(P-c)(P-d)), 为半周长P 正边形
20
21 n: 为外接圆半径
22 R, 为内切圆半径r
23 1. 中心角A=2PI/n
24 2. 内角C=(n-2)PI/n
25 3. 边长a=2sqrt(R^2-r^2)=2Rsin(A/2)=2rtan(A/2)
26 4. 面积S=nar/2=nR^2tan(A/2)=nR^2sin(A)/2=na^2/(4tan(A/2)) 圆
27
28  :
29 1. 弧长l=rA
30 2. 弦长a=2sqrt(2hr-h^2)=2rsin(A/2)
31 3. 弓形高h=r-sqrt(r^2-a^2/4)=r(1-cos(A/2))=atan(A/4)/2
32 4. 扇形面积S1=r^2/2-r^2A/2
33 5. 弓形面积S2=(r^2-a(r-h))/2=r^2(A-sin(A))/2 棱柱
34
35  :
36 1. 体积V=Ah, 为底面积A, 为高h
37 2. 侧面积S=lp, 为棱长l, 为直截面周长p
38 3. 全面积T=S+2A 棱锥
39
40  :
41 1. 体积V=Ah/3, 为底面积A, 为高h 以下对正棱锥
42 ()
43 2. 侧面积S=lp/2, 为斜高l, 为底面周长p
44 3. 全面积T=S+A 棱台
45
46  :
47 1. 体积V=(A1+A2+sqrt(A1A2))h/3, A1, 为上下底面积A2, 为高h 以下对正棱台
48 ()
49 2. 侧面积S=(p1+p2)l/2, p1, 为上下底面周长p2, 为斜高l
50 3. 全面积T=S+A1+A2 圆柱
51
52  :
53 1. 侧面积S=2PIrh
54 2. 全面积T=2PIr(h+r)
55 3. 体积V=PIr^2h 圆锥
56
57  :
58 1. 母线l=sqrt(h^2+r^2)
59 2. 侧面积S=PIrl
60 3. 全面积T=PIr(l+r)
61 4. 体积V=PIr^2h/3 圆台
62
63  :
```

```
64 1. 母线l=sqrt(h^2+(r1-r2)^2)
65 2. 侧面积S=PI(r1+r2)l
66 3. 全面积T=PIr1(l+r1)+PIr2(l+r2)
67 4. 体积V=PI(r1^2+r2^2+r1r2)h/3 球
68
69  :
70 1. 全面积T=4PIr^2
71 2. 体积V=4PIr^3/3 球台
72
73  :
74 1. 侧面积S=2PIrh
75 2. 全面积T=PI(2rh+r1^2+r2^2)
76 3. 体积V=PIh(3(r1^2+r2^2)+h^2)/6 球扇形
77
78  :
79 1. 全面积T=PIr(2h+r0), 为球冠高h, 为球冠底面半径r0
80 2. 体积V=2PIr^2h/3
```

### 7.2 Float Method

```
1 //浮点几何函数库
2 #include <math.h>
3 #define eps 1e-8
4 #define zero(x) (((x)>0?(x):-x)<eps)
5 struct point{double x,y;};
6 struct line{point a,b;};
7 //计算cross product (P1-P0)x(P2-P0)
8 double xmult(point p1,point p2,point p0){
9     return (p1.x-p0.x)*(p2.y-p0.y)-(p2.x-p0.x)*(p1.y-p0.y);
10 }
11 double xmult(double x1,double y1,double x2,double y2,double x0,
12             double y0){
13     return (x1-x0)*(y2-y0)-(x2-x0)*(y1-y0);
14 }
15 //计算dot product (P1-P0).(P2-P0)
16 double dmult(point p1,point p2,point p0){
17     return (p1.x-p0.x)*(p2.x-p0.x)+(p1.y-p0.y)*(p2.y-p0.y);
18 }
19 double dmult(double x1,double y1,double x2,double y2,double x0,
20             double y0){
21     return (x1-x0)*(x2-x0)+(y1-y0)*(y2-y0);
22 }
23 //两点距离
24 double distance(point p1,point p2){
25     return sqrt((p1.x-p2.x)*(p1.x-p2.x)+(p1.y-p2.y)*(p1.y-p2.y));
26 }
27 double distance(double x1,double y1,double x2,double y2){
28     return sqrt((x1-x2)*(x1-x2)+(y1-y2)*(y1-y2));
29 }
30 //判三点共线
31 int dots_inline(point p1,point p2,point p3){
32     return zero(xmult(p1,p2,p3));
33 }
34 int dots_inline(double x1,double y1,double x2,double y2,double x3,
35                 double y3){
36     return zero(xmult(x1,y1,x2,y2,x3,y3));
37 }
38 //判点是否在线段上包括端点,
39 int dot_online_in(point p,line l){
40     return zero(xmult(p,l.a,l.b))&&(!zero(l.a.x-p.x)*(l.b.x-p.x)<eps&&(l.a.y-p.y)*(l.b.y-p.y)<eps);
41 }
42 int dot_online_in(point p,point l1,point l2){
43     return zero(xmult(p,l1,l2))&&(!l1.x-p.x)*(l2.x-p.x)<eps&&(l1.y-p.y)*(l2.y-p.y)<eps;
44 }
45 //判点是否在线段上不包括端点,
46 int dot_online_ex(point p,line l){
47     return dot_online_in(p,l)&&(!zero(p.x-l.a.x)||!zero(p.y-l.a.y))
48         &&(!zero(p.x-l.b.x)||!zero(p.y-l.b.y));
49 }
50 int dot_online_ex(point p,point l1,point l2){
51     return dot_online_in(p,l1,l2)&&(!zero(p.x-l1.x)||!zero(p.y-l1.y))
52         &&(!zero(p.x-l2.x)||!zero(p.y-l2.y));
53 }
54 int dot_online_ex(double x,double y,double x1,double y1,double x2,
55                 double y2){
56     return dot_online_in(x,y,x1,y1,x2,y2)&&(!zero(x-x1)||!zero(y-y1))
57         &&(!zero(x-x2)||!zero(y-y2));
58 }
59 //判两点在线段同侧点在线段上返回,0
60 int same_side(point p1,point p2,line l){
61     return xmult(l.a,p1,l.b)*xmult(l.a,p2,l.b)>eps;
62 }
63 //判两点在线段异侧点在线段上返回,0
64 int opposite_side(point p1,point p2,line l){
65     return xmult(l.a,p1,l.b)*xmult(l.a,p2,l.b)<-eps;
66 }
67 int opposite_side(point p1,point p2,point l1,point l2){
68     return xmult(l1,p1,l2)*xmult(l1,p2,l2)<-eps;
69 }
70 // 点关于直线的对称点// by 1yt
71 // 缺点: 用了斜率
72 // 也可以利用点到直线上的最近点来做. 避免使用斜率. ""
73 point symmetric_point(point p1, point l1, point l2) {
74     point ret;
75     if (l1.x > l2.x - eps && l1.x < l2.x + eps) {
76         ret.x = (2 * l1.x - p1.x);
77         ret.y = p1.y;
78     } else {
79         double k = (l1.y - l2.y) / (l1.x - l2.x);
80         ret.x = (2*k*k*l1.x + 2*k*p1.y - 2*k*l1.y - k*k*p1.x + p1.x) /
81             (1 + k*k);
82         ret.y = p1.y - (ret.x - p1.x) / k;
83     }
84     return ret;
85 }
```

```

83 }
84 //判两直线平行
85 int parallel(line u,line v){
86     return zero((u.a.x-u.b.x)*(v.a.y-v.b.y)-(v.a.x-v.b.x)*(u.a.y-u.b
        .y));
87 }
88 int parallel(point u1,point u2,point v1,point v2){
89     return zero((u1.x-u2.x)*(v1.y-v2.y)-(v1.x-v2.x)*(u1.y-u2.y));
90 }
91 //判两直线垂直
92 int perpendicular(line u,line v){
93     return zero((u.a.x-u.b.x)*(v.a.x-v.b.x)+(u.a.y-u.b.y)*(v.a.y-v.b
        .y));
94 }
95 int perpendicular(point u1,point u2,point v1,point v2){
96     return zero((u1.x-u2.x)*(v1.x-v2.x)+(u1.y-u2.y)*(v1.y-v2.y));
97 }
98 //判两线段相交包括端点和部分重合,
99 int intersect_in(line u,line v){
100     if (!dots_inline(u.a,u.b,v.a)||!dots_inline(u.a,u.b,v.b))
101         return !same_side(u.a,u.b,v)||!same_side(v.a,v.b,u);
102     return dot_online_in(u,a,v)||dot_online_in(u,b,v)||dot_online_in
        (v,a,u)||dot_online_in(v,b,u);
103 }
104 int intersect_in(point u1,point u2,point v1,point v2){
105     if (!dots_inline(u1,u2,v1)||!dots_inline(u1,u2,v2))
106         return !same_side(u1,u2,v1,v2)||!same_side(v1,v2,u1,u2);
107     return dot_online_in(u1,v1,v2)||dot_online_in(u2,v1,v2)||
        dot_online_in(v1,u1,u2)||dot_online_in(v2,u1,u2);
108 }
109 //判两线段相交不包括端点和部分重合,
110 int intersect_ex(line u,line v){
111     return opposite_side(u.a,u.b,v)&&opposite_side(v.a,v.b,u);
112 }
113 int intersect_ex(point u1,point u2,point v1,point v2){
114     return opposite_side(u1,u2,v1,v2)&&opposite_side(v1,v2,u1,u2);
115 }
116 //计算两直线交点注意事先判断直线是否平行,!
117 //线段交点请另外判线段相交时还是要判断是否平行(!)
118 point intersection(line u,line v){
119     point ret=u.a;
120     double t=((u.a.x-v.a.x)*(v.a.y-v.b.y)-(u.a.y-v.a.y)*(v.a.x-v.b.x
        ))
121         /((u.a.x-u.b.x)*(v.a.y-v.b.y)-(u.a.y-u.b.y)*(v.a.x-v.b.x));
122     ret.x+=(u.b.x-u.a.x)*t;
123     ret.y+=(u.b.y-u.a.y)*t;
124     return ret;
125 }
126 point intersection(point u1,point u2,point v1,point v2){
127     point ret=u1;
128     double t=((u1.x-v1.x)*(v1.y-v2.y)-(u1.y-v1.y)*(v1.x-v2.x))
129         /((u1.x-u2.x)*(v1.y-v2.y)-(u1.y-u2.y)*(v1.x-v2.x));
130     ret.x+=(u2.x-u1.x)*t;
131     ret.y+=(u2.y-u1.y)*t;
132     return ret;
133 }
134 //点到直线上的最近点
135 point ptoline(point p,line l){
136     point t=p;
137     t.x+=l.a.y-l.b.y,t.y+=l.b.x-l.a.x;
138     return intersection(p,t,l.a,l.b);
139 }
140 point ptoline(point p,point l1,point l2){
141     point t=p;
142     t.x+=l1.y-l2.y,t.y+=l2.x-l1.x;
143     return intersection(p,t,l1,l2);
144 }
145 //点到直线距离
146 double diaptoline(point p,line l){
147     return fabs(xmult(p,l.a,l.b))/distance(l.a,l.b);
148 }
149 double diaptoline(point p,point l1,point l2){
150     return fabs(xmult(p,l1,l2))/distance(l1,l2);
151 }
152 double diaptoline(double x,double y,double x1,double y1,double x2,
    double y2){
153     return fabs(xmult(x,y,x1,y1,x2,y2))/distance(x1,y1,x2,y2);
154 }
155 //点到线段上的最近点
156 point ptoseg(point p,line l){
157     point t=p;
158     t.x+=l.a.y-l.b.y,t.y+=l.b.x-l.a.x;
159     if (xmult(l.a,t,p)*xmult(l.b,t,p)>eps)
160         return distance(p,l.a)<distance(p,l.b)?l.a:l.b;
161     return intersection(p,t,l.a,l.b);
162 }
163 point ptoseg(point p,point l1,point l2){
164     point t=p;
165     t.x+=l1.y-l2.y,t.y+=l2.x-l1.x;
166     if (xmult(l1,t,p)*xmult(l2,t,p)>eps)
167         return distance(p,l1)<distance(p,l2)?l1:l2;
168     return intersection(p,t,l1,l2);
169 }
170 //点到线段距离
171 double diptoseg(point p,line l){
172     point t=p;
173     t.x+=l.a.y-l.b.y,t.y+=l.b.x-l.a.x;
174     if (xmult(l.a,t,p)*xmult(l.b,t,p)>eps)
175         return distance(p,l.a)<distance(p,l.b)?distance(p,l.a):distance
            (p,l.b);
176     return fabs(xmult(p,l.a,l.b))/distance(l.a,l.b);
177 }
178 double diptoseg(point p,point l1,point l2){
179     point t=p;
180     t.x+=l1.y-l2.y,t.y+=l2.x-l1.x;
181     if (xmult(l1,t,p)*xmult(l2,t,p)>eps)
182         return distance(p,l1)<distance(p,l2)?distance(p,l1):distance(p,
            l2);
183     return fabs(xmult(p,l1,l2))/distance(l1,l2);
184 }
185 //向量以顶点逆时针旋转并放大倍VPanglescale
186 point rotate(point v,point p,double angle,double scale){
187     point ret=p;
188     v.x=p.x,v.y=p.y;
189     p.x=scale*cos(angle);
190     p.y=scale*sin(angle);
191     ret.x+=v.x*p.x-v.y*p.y;
192     ret.y+=v.x*p.y+v.y*p.x;
193     return ret;
194 }

```

## 7.3 Int Method

```

1 //整数几何函数库
2 //注意某些情况下整数运算会出界!
3 #define sign(a) ((a)>0?1:((a)<0?-1:0))
4 struct point{int x,y;};
5 struct line{point a,b;};
6 //计算cross product (P1-P0)x(P2-P0)
7 int xmult(point p1,point p2,point p0){
8     return (p1.x-p0.x)*(p2.y-p0.y)-(p2.x-p0.x)*(p1.y-p0.y);
9 }
10 int xmult(int x1,int y1,int x2,int y2,int x0,int y0){
11     return (x1-x0)*(y2-y0)-(x2-x0)*(y1-y0);
12 }
13 //计算dot product (P1-P0).(P2-P0)
14 int dmult(point p1,point p2,point p0){
15     return (p1.x-p0.x)*(p2.x-p0.x)+(p1.y-p0.y)*(p2.y-p0.y);
16 }
17 int dmult(int x1,int y1,int x2,int y2,int x0,int y0){
18     return (x1-x0)*(x2-x0)+(y1-y0)*(y2-y0);
19 }
20 //判三点共线
21 int dots_inline(point p1,point p2,point p3){
22     return !xmult(p1,p2,p3);
23 }
24 int dots_inline(int x1,int y1,int x2,int y2,int x3,int y3){
25     return !xmult(x1,y1,x2,y2,x3,y3);
26 }
27 //判点是否在线段上包括端点和部分重合,
28 int dot_online_in(point p,line l){
29     return !xmult(p,l.a,l.b)&&(l.a.x-p.x)*(l.b.x-p.x)<=0&&(l.a.y-p.y
        )*(l.b.y-p.y)<=0;
30 }
31 int dot_online_in(point p,point l1,point l2){
32     return !xmult(p,l1,l2)&&(l1.x-p.x)*(l2.x-p.x)<=0&&(l1.y-p.y)*(l2
        .y-p.y)<=0;
33 }
34 int dot_online_in(int x,int y,int x1,int y1,int x2,int y2){
35     return !xmult(x,y,x1,y1,x2,y2)&&(x1-x)*(x2-x)<=0&&(y1-y)*(y2-y)
        <=0;
36 }
37 //判点是否在线段上不包括端点,
38 int dot_online_ex(point p,line l){
39     return dot_online_in(p,l)&&(p.x!=l.a.x||p.y!=l.a.y)&&(p.x!=l.b.x
        ||p.y!=l.b.y);
40 }
41 int dot_online_ex(point p,point l1,point l2){
42     return dot_online_in(p,l1,l2)&&(p.x!=l1.x||p.y!=l1.y)&&(p.x!=l2.
        x||p.y!=l2.y);
43 }
44 int dot_online_ex(int x,int y,int x1,int y1,int x2,int y2){
45     return dot_online_in(x,y,x1,y1,x2,y2)&&(x!=x1||y!=y1)&&(x!=x2||y
        !=y2);
46 }
47 //判两点在直线同侧点在直线上返回,0
48 int same_side(point p1,point p2,line l){
49     return sign(xmult(l.a,p1,l.b))*xmult(l.a,p2,l.b)>0;
50 }
51 int same_side(point p1,point p2,point l1,point l2){
52     return sign(xmult(l1,p1,l2))*xmult(l1,p2,l2)>0;
53 }
54 //判两点在直线异侧点在直线上返回,0
55 int opposite_side(point p1,point p2,line l){
56     return sign(xmult(l.a,p1,l.b))*xmult(l.a,p2,l.b)<0;
57 }
58 int opposite_side(point p1,point p2,point l1,point l2){
59     return sign(xmult(l1,p1,l2))*xmult(l1,p2,l2)<0;
60 }
61 //判两直线平行
62 int parallel(line u,line v){
63     return (u.a.x-u.b.x)*(v.a.y-v.b.y)==(v.a.x-v.b.x)*(u.a.y-u.b.y);
64 }
65 int parallel(point u1,point u2,point v1,point v2){
66     return (u1.x-u2.x)*(v1.y-v2.y)==(v1.x-v2.x)*(u1.y-u2.y);
67 }
68 //判两直线垂直
69 int perpendicular(line u,line v){
70     return (u.a.x-u.b.x)*(v.a.x-v.b.x)==-(u.a.y-u.b.y)*(v.a.y-v.b.y)
        ;
71 }
72 int perpendicular(point u1,point u2,point v1,point v2){
73     return (u1.x-u2.x)*(v1.x-v2.x)==-(u1.y-u2.y)*(v1.y-v2.y);
74 }
75 //判两线段相交包括端点和部分重合,
76 int intersect_in(line u,line v){
77     if (!dots_inline(u.a,u.b,v.a)||!dots_inline(u.a,u.b,v.b))
78         return !same_side(u.a,u.b,v)&&!same_side(v.a,v.b,u);
79     return dot_online_in(u,a,v)||dot_online_in(u,b,v)||dot_online_in
        (v,a,u)||dot_online_in(v,b,u);
80 }
81 int intersect_in(point u1,point u2,point v1,point v2){
82     if (!dots_inline(u1,u2,v1)||!dots_inline(u1,u2,v2))
83         return !same_side(u1,u2,v1,v2)&&!same_side(v1,v2,u1,u2);
84     return dot_online_in(u1,v1,v2)||dot_online_in(u2,v1,v2)||
        dot_online_in(v1,u1,u2)||dot_online_in(v2,u1,u2);
85 }
86 //判两线段相交不包括端点和部分重合,
87 int intersect_ex(line u,line v){
88     return opposite_side(u.a,u.b,v)&&opposite_side(v.a,v.b,u);
89 }
90 int intersect_ex(point u1,point u2,point v1,point v2){
91     return opposite_side(u1,u2,v1,v2)&&opposite_side(v1,v2,u1,u2);
92 }

```

## 7.4 Polygon Method

```

1 #include <stdlib.h>
2 #include <math.h>
3 #define MAXN 1000
4 #define offset 10000
5 #define eps 1e-8
6 #define zero(x) (((x)>0?(x):(x)<0)?<eps>:0)
7 #define _sign(x) ((x)>eps?1:((x)<-eps)?2:0)
8 struct point{double x,y;};
9 struct line(point a,b);
10 double xmult(point p1,point p2,point p0){
11     return (p1.x-p0.x)*(p2.y-p0.y)-(p2.x-p0.x)*(p1.y-p0.y);
12 }
13 //判定凸多边形顶点按顺时针或逆时针给出允许相邻边共线,,
14 int is_convex(int n,point* p){
15     int i,s[3]={1,1,1};
16     for (i=0;i<n&&s[1]|s[2];i++)
17         s[_sign(xmult(p[(i+1)%n],p[(i+2)%n],p[i]))]=0;
18     return s[1]|s[2];
19 }
20 //判定凸多边形顶点按顺时针或逆时针给出不允许相邻边共线,,
21 int is_convex_v2(int n,point* p){
22     int i,s[3]={1,1,1};
23     for (i=0;i<n&&s[0]&&s[1]|s[2];i++)
24         s[_sign(xmult(p[(i+1)%n],p[(i+2)%n],p[i]))]=0;
25     return s[0]&&s[1]|s[2];
26 }
27 //判点在凸多边形内或多边形边上顶点按顺时针或逆时针给出,
28 int inside_convex(point q,int n,point* p){
29     int i,s[3]={1,1,1};
30     for (i=0;i<n&&s[1]|s[2];i++)
31         s[_sign(xmult(p[(i+1)%n],q,p[i]))]=0;
32     return s[1]|s[2];
33 }
34 //判点在凸多边形内顶点按顺时针或逆时针给出在多边形边上返回,,0
35 int inside_convex_v2(point q,int n,point* p){
36     int i,s[3]={1,1,1};
37     for (i=0;i<n&&s[0]&&s[1]|s[2];i++)
38         s[_sign(xmult(p[(i+1)%n],q,p[i]))]=0;
39     return s[0]&&s[1]|s[2];
40 }
41 //判点在任意多边形内顶点按顺时针或逆时针给出,
42 //表示点在多边形边上时的返回值on_edge,为多边形坐标上限offset
43 int inside_polygon(point q,int n,point* p,int on_edge=1){
44     point q2;
45     int i=0,count;
46     while (i<n)
47         for (count=i+0,q2.x=rand()+offset,q2.y=rand()+offset;i<n;i++)
48             if (zero(xmult(q,p[i],p[(i+1)%n]))&&(p[i].x-q.x)*(p[(i+1)%n].x-q.x)<eps&&(p[i].y-q.y)*(p[(i+1)%n].y-q.y)<eps)
49                 return on_edge;
50             else if (zero(xmult(q,q2,p[i])))
51                 break;
52             else if (xmult(q,p[i],q2)*xmult(q,p[(i+1)%n],q2)<-eps&&xmult(p[i],q,p[(i+1)%n])*xmult(p[i],q2,p[(i+1)%n])<-eps)
53                 count++;
54     return count&1;
55 }
56 inline int opposite_side(point p1,point p2,point l1,point l2){
57     return xmult(l1,p1,l2)*xmult(l1,p2,l2)<-eps;
58 }
59 inline int dot_online_in(point p,point l1,point l2){
60     return zero(xmult(p,l1,l2))&&(l1.x-p.x)*(l2.x-p.x)<eps&&(l1.y-p.y)*(l2.y-p.y)<eps;
61 }
62 //判线段在任意多边形内顶点按顺时针或逆时针给出与边界相交返回,,1
63 int inside_polygon(point l1,point l2,int n,point* p){
64     point t[MAXN],tt;
65     int i,j,k=0;
66     if (!inside_polygon(l1,n,p)||!inside_polygon(l2,n,p))
67         return 0;
68     for (i=0;i<n;i++)
69         if (opposite_side(l1,l2,p[i],p[(i+1)%n])&&opposite_side(p[i],p[(i+1)%n],l1,l2))
70             return 0;
71         else if (dot_online_in(l1,p[i],p[(i+1)%n]))
72             t[k++]=l1;
73         else if (dot_online_in(l2,p[i],p[(i+1)%n]))
74             t[k++]=l2;
75         else if (dot_online_in(p[i],l1,l2))
76             t[k++]=p[i];
77     for (i=0;i<k;i++)
78         for (j=i+1;j<k;j++){
79             tt.x=(t[i].x+t[j].x)/2;
80             tt.y=(t[i].y+t[j].y)/2;
81             if (!inside_polygon(tt,n,p))
82                 return 0;
83         }
84     return 1;
85 }
86 point intersection(line u,line v){
87     point ret=u.a;
88     double t=((u.a.x-v.a.x)*(v.a.y-v.b.y)-(u.a.y-v.a.y)*(v.a.x-v.b.x))
89         /((u.a.x-u.b.x)*(v.a.y-v.b.y)-(u.a.y-u.b.y)*(v.a.x-v.b.x));
90     ret.x+=(u.b.x-u.a.x)*t;
91     ret.y+=(u.b.y-u.a.y)*t;
92     return ret;
93 }
94 point barycenter(point a,point b,point c){
95     line u,v;
96     u.a.x=(a.x+b.x)/2;
97     u.a.y=(a.y+b.y)/2;
98     u.b=c;
99     v.a.x=(a.x+c.x)/2;
100    v.a.y=(a.y+c.y)/2;
101    v.b=b;
102    return intersection(u,v);
103 }
104 //多边形重心
105 point barycenter(int n,point* p){
106     point ret,t;
107     double t1=0,t2;
108     int i;
109     ret.x=ret.y=0;
110     for (i=1;i<n-1;i++)
111         if (fabs(t2=xmult(p[0],p[i],p[i+1]))>eps){
112             t=barycenter(p[0],p[i],p[i+1]);
113             ret.x+=t.x*t2;
114             ret.y+=t.y*t2;

```

```

115     t1+=t2;
116 }
117 if (fabs(t1)>eps)
118     ret.x/=t1,ret.y/=t1;
119 return ret;
120 }

```

## 7.5 Circles Method

```

1 #include <math.h>
2 #define eps 1e-8
3 struct point{double x,y;};
4 double xmult(point p1,point p2,point p0){
5     return (p1.x-p0.x)*(p2.y-p0.y)-(p2.x-p0.x)*(p1.y-p0.y);
6 }
7 double distance(point p1,point p2){
8     return sqrt((p1.x-p2.x)*(p1.x-p2.x)+(p1.y-p2.y)*(p1.y-p2.y));
9 }
10 double disptoline(point p,point l1,point l2){
11     return fabs(xmult(p,l1,l2))/distance(l1,l2);
12 }
13 point intersection(point u1,point u2,point v1,point v2){
14     point ret=u1;
15     double t=((u1.x-v1.x)*(v1.y-v2.y)-(u1.y-v1.y)*(v1.x-v2.x))
16         /((u1.x-u2.x)*(v1.y-v2.y)-(u1.y-u2.y)*(v1.x-v2.x));
17     ret.x+=(u2.x-u1.x)*t;
18     ret.y+=(u2.y-u1.y)*t;
19     return ret;
20 }
21 //判直线和圆相交包括相切,
22 int intersect_line_circle(point c,double r,point l1,point l2){
23     return disptoline(c,l1,l2)<r+eps;
24 }
25 //判线段和圆相交包括端点和相切,
26 int intersect_seg_circle(point c,double r,point l1,point l2){
27     double t1=distance(c,l1)-r,t2=distance(c,l2)-r;
28     point t=c;
29     if (t1<eps||t2<eps)
30         return t1>-eps||t2>-eps;
31     t.x+=l1.y-l2.y;
32     t.y+=l2.x-l1.x;
33     return xmult(l1,c,t)*xmult(l2,c,t)<eps&&disptoline(c,l1,l2)-r<eps;
34 }
35 //判圆和圆相交包括相切,
36 int intersect_circle_circle(point c1,double r1,point c2,double r2)
37 {
38     return distance(c1,c2)<r1+r2+eps&&distance(c1,c2)>fabs(r1-r2)-eps;
39 }
40 //计算圆上到点最近点p如,与圆心重合p返回,本身p
41 point dot_to_circle(point c,double r,point p){
42     point u,v;
43     if (distance(p,c)<eps)
44         return p;
45     u.x=c.x+r*fabs(c.x-p.x)/distance(c,p);
46     u.y=c.y+r*fabs(c.y-p.y)/distance(c,p)*((c.x-p.x)*(c.y-p.y)<0?-1:1);
47     v.x=c.x-r*fabs(c.x-p.x)/distance(c,p);
48     v.y=c.y-r*fabs(c.y-p.y)/distance(c,p)*((c.x-p.x)*(c.y-p.y)<0?-1:1);
49     return distance(u,p)<distance(v,p)?u:v;
50 }
51 //计算直线与圆的交点保证直线与圆有交点,
52 //计算线段与圆的交点可用这个函数后判点是否在线段上
53 void intersection_line_circle(point c,double r,point l1,point l2,
54     point& p1,point& p2){
55     point p=c;
56     double t;
57     p.x+=l1.y-l2.y;
58     p.y+=l2.x-l1.x;
59     p=intersection(p,c,l1,l2);
60     t=sqrt((r-r*distance(p,c)*distance(p,c))/distance(l1,l2));
61     p1.x=p.x+(l2.x-l1.x)*t;
62     p1.y=p.y+(l2.y-l1.y)*t;
63     p2.x=p.x-(l2.x-l1.x)*t;
64     p2.y=p.y-(l2.y-l1.y)*t;
65 }
66 //计算圆与圆的交点保证圆与圆有交点圆心不重合,,
67 void intersection_circle_circle(point c1,double r1,point c2,double r2,
68     point& p1,point& p2){
69     point u,v;
70     double t;
71     t=(l1+r1-r2*r2)/distance(c1,c2)/distance(c1,c2)/2;
72     u.x=c1.x+(c2.x-c1.x)*t;
73     u.y=c1.y+(c2.y-c1.y)*t;
74     v.x=u.x+cl.y-c2.y;
75     v.y=u.y-cl.x+c2.x;
76     intersection_line_circle(c1,r1,u,v,p1,p2);
77 }

```

## 7.6 Scan Line

```

1 #include <stdio>
2 #include <set>
3 #include <math>
4 #include <algorithm>
5 using namespace std;
6 const int N = 50010;
7 const double EPS = 1e-8;
8 int X;
9 double a, b;
10 int x[N], y[N], r[N];
11 typedef struct Node{
12     int id;
13     bool tag;
14     bool operator < (const Node& other)const{
15         a = sqrt(1.0*r[id]*r[id]-1.0*(X-x[id])*(X-x[id]));

```

```

16     b = sqrt(1.0*r[other.id]*r[other.id]-1.0*(X-x[other.id])*(X-
17         x[other.id]));
18     a = tag ? y[id]+a : y[id]-a;
19     b = other.tag ? y[other.id]+b : y[other.id]-b;
20     if (fabs(a-b) > EPS) return a < b;
21     else return tag < other.tag;
22 }
23 }Node;
24 typedef struct Event{
25     int id, value;
26     bool in;
27     bool operator < (const Event &other) const{
28         return value < other.value;
29     }
30 }Event;
31 Event arr[2*N];
32 int n;
33 int f[N]={};
34 int main()
35 {
36     Event e;
37     Node node;
38     while (scanf("%d", &n) == 1){
39         set<Node>st;
40         set<Node>::iterator it1, it2;
41         for (int i = 0; i < n; ++i){
42             f[i] = 0;
43             scanf("%d %d", &x[i], &y[i], &r[i]);
44             e.id = i, e.value = x[i]-r[i], e.in = 1;
45             arr[i] = e;
46             e.id = i, e.value = x[i]+r[i], e.in = 0;
47             arr[i+n] = e;
48         }
49         n *= 2;
50         sort(arr, arr+n);
51         for (int i = 0; i < n; ++i){
52             e = arr[i];
53             X = e.value;
54             if (e.in == 0){
55                 node.id = e.id;
56                 node.tag = 0;
57                 st.erase(node);
58                 node.tag = 1;
59                 st.erase(node);
60             }
61             else{
62                 node.id = e.id;
63                 node.tag = 1;
64                 it2 = st.lower_bound(node);
65                 if (it2 == st.begin() || it2 == st.end()){
66                     f[e.id] = 1;
67                 }
68                 else{
69                     it1 = it2;
70                     --it1;
71                     if (it1->id == it2->id) f[e.id] = f[it1->id]+1;
72                     else f[e.id] = max(f[it1->id], f[it2->id]);
73                 }
74                 node.tag = 0;
75                 st.insert(node);
76                 node.tag = 1;
77                 st.insert(node);
78             }
79             int ans = 0;
80             n /= 2;
81             for (int i = 0; i < n; ++i) if (ans < f[i]) ans = f[i];
82             printf("%d\n", ans);
83         }
84         return 0;
85     }
86 }

```

## 7.7 Spherical Distance

```

1  /*
2   * Point 表示点的位置其中
3   * x 表示经度, 表示纬度y
4   * R 为球的半径
5   * dist 返回两点的球面距离
6   */
7  #include<stdio>
8  #include<cmath>
9  const double R = 1000;
10 const double PI = acos(-1);
11 typedef struct Point{
12     double x, y;
13 }Point;
14 double dist(Point a, Point b)
15 {
16     a.x = a.x*2*PI/360;
17     a.y = a.y*2*PI/360;
18     b.x = b.x*2*PI/360;
19     b.y = b.y*2*PI/360;
20     if (fabs(a.x - b.x) < 1e-6) return R*fabs(a.y - b.y);
21     else return R*acos(sin(a.y)*sin(b.y)+cos(a.y)*cos(b.y)*cos(a.x-
22         b.x));
23 }
24 double d[1010][1010];
25 int main()
26 {
27     Point pt[1010];
28     int n;
29     while (scanf("%d", &n) == 1){
30         for (int i = 1; i <= n; i++) scanf("%lf %lf", &pt[i].y, &pt[i].x);
31         for (int i = 1; i <= n; i++){
32             d[i][i] = 0;
33             for (int j = i+1; j <= n; j++){
34                 d[i][j] = d[j][i] = dist(pt[i], pt[j]);
35             }
36             double tmp = 9999999;
37             int mark = 1;
38             for (int i = 1; i <= n; i++){
39                 double max = 0;

```

```

39         for (int j = 1; j <= n; j++) if (max < d[i][j]) max = d[i][j];
40         if (fabs(max-tmp) >= 0.005 && max < tmp) tmp = max, mark = i;
41     }
42     printf("%.2lf %.2lf\n", pt[mark].y, pt[mark].x);
43 }
44 return 0;
45 }

```

## 7.8 Minimum Circle Cover

```

1  #include <stdio.h>
2  #include <math.h>
3  const int maxn = 1005;
4  //const double eps = 1e-6;
5  struct TPoint{
6     double x, y;
7     TPoint operator-(TPoint &a){
8         TPoint pl;
9         pl.x = x - a.x;
10        pl.y = y - a.y;
11        return pl;
12    }
13 };
14 struct TCircle{
15     double r;
16     TPoint centre;
17 };
18 struct TTriangle{
19     TPoint t[3];
20 };
21 TCircle c;
22 TPoint a[maxn];
23 double distance(TPoint pl, TPoint p2){
24     TPoint p3;
25     p3.x = p2.x - pl.x;
26     p3.y = p2.y - pl.y;
27     return sqrt(p3.x * p3.x + p3.y * p3.y);
28 }
29 double triangleArea(TTriangle t){
30     TPoint pl, p2;
31     pl = t.t[1] - t.t[0];
32     p2 = t.t[2] - t.t[0];
33     return fabs(pl.x * p2.y - pl.y * p2.x) / 2;
34 }
35 TCircle circumcircleOfTriangle(TTriangle t){
36     //三角形的外接圆
37     TCircle tmp;
38     double a, b, c, cl, c2;
39     double xA, yA, xB, yB, xC, yC;
40     a = distance(t.t[0], t.t[1]);
41     b = distance(t.t[1], t.t[2]);
42     c = distance(t.t[2], t.t[0]);
43     //根据S = a * b * c / R / 求半径4R
44     tmp.r = a * b * c / triangleArea(t) / 4;
45     xA = t.t[0].x; yA = t.t[0].y;
46     xB = t.t[1].x; yB = t.t[1].y;
47     xC = t.t[2].x; yC = t.t[2].y;
48     cl = (xA * xA + yA * yA - xB * xB - yB * yB) / 2;
49     c2 = (xA * xA + yA * yA - xC * xC - yC * yC) / 2;
50     tmp.centre.x = (cl * (yA - yC) - c2 * (yA - yB)) /
51         ((xA - xB) * (yA - yC) - (xA - xC) * (yA - yB));
52     tmp.centre.y = (cl * (xA - xC) - c2 * (xA - xB)) /
53         ((yA - yB) * (xA - xC) - (yA - yC) * (xA - xB));
54     return tmp;
55 }
56 TCircle MinCircle2(int tce, TTriangle ce){
57     TCircle tmp;
58     if(tce == 0) tmp.r = -2;
59     else if(tce == 1){
60         tmp.centre = ce.t[0];
61         tmp.r = 0;
62     }
63     else if(tce == 2){
64         tmp.r = distance(ce.t[0], ce.t[1]) / 2;
65         tmp.centre.x = (ce.t[0].x + ce.t[1].x) / 2;
66         tmp.centre.y = (ce.t[0].y + ce.t[1].y) / 2;
67     }
68     else if(tce == 3) tmp = circumcircleOfTriangle(ce);
69     return tmp;
70 }
71 void MinCircle(int t, int tce, TTriangle ce){
72     int i, j;
73     TPoint tmp;
74     c = MinCircle2(tce, ce);
75     if(tce == 3) return;
76     for(i = 1; i <= t; i++){
77         if(distance(a[i], c.centre) > c.r){
78             ce.t[tce] = a[i];
79             MinCircle(i - 1, tce + 1, ce);
80             tmp = a[i];
81             for(j = i; j >= 2; j--){
82                 a[j] = a[j - 1];
83             }
84             a[1] = tmp;
85         }
86     }
87 }
88 void run(int n){
89     TTriangle ce;
90     int i;
91     MinCircle(n, 0, ce);
92     printf("%.2lf %.2lf %.2lf\n", c.centre.x, c.centre.y, c.r);
93 }
94 int main()
95 {
96     int n;
97     while(scanf("%d", &n) != 1 && n){
98         for(int i = 1; i <= n; i++){
99             scanf("%lf%lf", &a[i].x, &a[i].y);
100         }
101         run(n);
102     }
103     return 0;
104 }

```

## 7.9 N-D Volume

```

1 #include <stdio>
2 #include <algorithm>
3 using namespace std;
4 typedef long long LL;
5 const int M = 110;
6 const int N = 10;
7 const int MOD = 14121413;
8 typedef struct VT{
9     int a[N], b[N];
10 }VT;
11 VT vt[M];
12 int m, n, size;
13 VT ans[M*M];
14 bool Cross(VT &x, VT &y){
15     for (int i = 0; i < n; ++i) if (x.a[i] >= y.b[i] || x.b[i] <= y
        .a[i]) return 0;
16     return 1;
17 }
18 void Cut (VT &v){
19     int s = size, k1, k2;
20     for (int i = 0; i < s; ++i){
21         if (!Cross(ans[i], v)) continue;
22         for (int j = 0; j < n; ++j){
23             k1 = max(ans[i].a[j], v.a[j]);
24             k2 = min(ans[i].b[j], v.b[j]);
25             if (ans[i].a[j] < k1) ans[size] = ans[i]; ans[size++].b[j]
                = k1;
26             if (k2 < ans[i].b[j]) ans[size] = ans[i]; ans[size++].a[j]
                = k2;
27             ans[i].a[j] = k1, ans[i].b[j] = k2;
28         }
29         ans[i] = ans[--size], --i;
30         if (s > size) s = size;
31     }
32 }
33 int main()
34 {
35     while (scanf("%d %d", &m, &n) == 2){
36         for (int i = 0; i < m; ++i){
37             for (int j = 0; j < n; ++j) scanf("%d", &vt[i].a[j]);
38             for (int j = 0; j < n; ++j) scanf("%d", &vt[i].b[j]);
39         }
40         int ret = 0;
41         for (int i = 0; i < m; ++i){
42             size = 0;
43             ans[size++] = vt[i];
44             for (int j = 0; j < i; ++j) Cut(vt[j]);
45             for (int j = 0; j < size; ++j){
46                 LL tmp = 1;
47                 for (int k = 0; k < n; ++k) tmp = (tmp*(ans[j].b[k]-
                    ans[j].a[k]))%MOD;
48                 ret = (tmp+ret)%MOD;
49             }
50             printf("%d\n", ret);
51         }
52         return 0;
53     }
54 }

```

## 7.10 Perimeter Of Circles

```

1 #include <stdio>
2 #include <vector>
3 #include <algorithm>
4 #include <cmath>
5 using namespace std;
6 const int MAX = 110;
7 const double EPS = 1e-8;
8 const double PI = acos(-1.0);
9 typedef struct Node{
10     double x, y;
11     bool operator <(const Node & other) const {return x < other.x;}
12 }Node;
13 double r[MAX], x[MAX], y[MAX];
14 inline double Dis(double x1, double y1, double x2, double y2){
15     return (x1-x2)*(x1-x2)+(y1-y2)*(y1-y2);
16 }
17 int main()
18 {
19     int n;
20     vector<Node> vt;
21     while (scanf("%d", &n) == 1){
22         for (int i = 0; i < n; ++i) scanf("%lf %lf %lf", &r[i], &x[i]
            , &y[i]);
23         double ans = 0;
24         for (int i = 0; i < n; ++i){
25             vt.clear();
26             Node tmp;
27             bool cover = 0;
28             for (int j = i+1; j < n; ++j){
29                 double dis = Dis(x[i], y[i], x[j], y[j]);
30                 if (dis < (r[i]+r[j])*(r[i]+r[j])+EPS){
31                     if (sqrt(dis)+min(r[i], r[j]) <= max(r[i], r[j])+EPS)
                        {
32                         if (r[i] < r[j]+EPS) {cover = 1; break;}
33                         continue;
34                     }
35                     double angle = atan2(y[j]-y[i], x[j]-x[i]);
36                     if (angle < 0) angle += 2*PI;
37                     double da = acos((r[i]*r[i]+dis-r[j]*r[j])/(2*r[i]*
                        sqrt(dis)));
38                     double a = angle-da, b = angle+da;
39                     if (a < 0){
40                         tmp.x = a+2*PI;
41                         tmp.y = 2*PI;
42                         vt.push_back(tmp);
43                         a = 0;
44                     }
45                     if (b > 2*PI){
46                         tmp.x = 0;
47                         tmp.y = b-2*PI;

```

```

48         vt.push_back(tmp);
49         b = 2*PI;
50     }
51     tmp.x = a, tmp.y = b;
52     vt.push_back(tmp);
53 }
54 }
55 if (cover) continue;
56 sort(vt.begin(), vt.end());
57 double l = 0, a = 0, b = 0;
58 int size = vt.size();
59 for (int k = 0; k < size; ++k){
60     if (vt[k].x <= b){
61         if (vt[k].y > b) b = vt[k].y;
62     }
63     else{
64         l += b-a;
65         a = vt[k].x, b = vt[k].y;
66     }
67 }
68 l += b-a;
69 ans += r[i]*(2*PI-l);
70 }
71 printf("%.3lf\n", ans);
72 }
73 return 0;
74 }

```

## 7.11 Center of Triangle

```

1 #include <stdio>
2 #include <cmath>
3 const double PI = acos(-1.0);
4 double Area(double a, double b, double c){
5     double p = (a+b+c)/2;
6     return sqrt(p*(p-a)*(p-b)*(p-c));
7 }
8 double Fema(double a, double b, double c){
9     double anga = acos((b*b+c*c-a*a)/(2*b*c));
10    double angb = acos((a*a+c*c-b*b)/(2*a*c));
11    double angc = acos((a*a+b*b-c*c)/(2*a*b));
12    if (anga >= PI*2/3) return b+c;
13    if (angb >= PI*2/3) return a+c;
14    if (angc >= PI*2/3) return a+b;
15    double ang = angc+PI/3;
16    return sqrt(a*a+b*b-2*a*b*cos(ang));
17 }
18 double Inner(double a, double b, double c){
19     double s = Area(a, b, c);
20     double r = 2*s/(a+b+c);
21     double anga = acos((b*b+c*c-a*a)/(2*b*c));
22     double angb = acos((a*a+c*c-b*b)/(2*a*c));
23     double angc = acos((a*a+b*b-c*c)/(2*a*b));
24     return r/sin(anga/2)+r/sin(angb/2)+r/sin(angc/2);
25 }
26 double Center(double a, double b, double c){
27     return sqrt(2*a*a+2*b*b-c*c)/3+sqrt(2*a*a+2*c*c-b*b)/3+sqrt(2*c
        *c+2*b*b-a*a)/3;
28 }
29 double Outer(double a, double b, double c){
30     double s = Area(a, b, c);
31     return 3*a*b*c/4/s;
32 }
33 int main()
34 {
35     int a, b, c, t;
36     scanf("%d", &t);
37     while (t--){
38         scanf("%d %d %d", &a, &b, &c);
39         printf("%.3lf %.3lf %.3lf %.3lf\n",
40             Fema(a, b, c), Inner(a, b, c), Center(a, b, c), Outer(a, b,
                c));
41     }
42     return 0;
43 }

```

## 7.12 3D Vector Projection

```

1 cos(a,b) = (x1*x2+y1*y2+z1*z2)/(sqrt(x1*x1+y1*y1+z1*z1)*sqrt(x2*x2
2 +y2*y2+z2*z2));
3 a.b = |a|*cos(a,b)

```

## 7.13 3D Convexhull

```

1 #include <stdio>
2 #include <cmath>
3 #include <algorithm>
4 #include <iostream>
5 using namespace std;
6 const int MAXN = 1111;
7 const double EPS = 1e-6;
8 inline int sgn(double x) {
9     return (x > EPS) - (x < -EPS);
10 }
11 struct P {
12     double x, y, z;
13     P() {}
14     P(double a, double b, double c) :x(a), y(b), z(c) {}
15     P operator - (const P& a) const {
16         return P(x - a.x, y - a.y, z - a.z);
17     }
18     P operator + (const P& a) const {
19         return P(x + a.x, y + a.y, z + a.z);

```

```

20     }
21     double len () {
22         return sqrt(x*x + y*y + z*z);
23     }
24 };
25 double dot(const P& a, const P& b) {
26     return a.x*b.x + a.y*b.y + a.z*b.z;
27 }
28 P det(const P& a, const P& b) {
29     return P(a.y*b.z - a.z*b.y
30             , a.z*b.x - a.x*b.z
31             , a.x*b.y - a.y*b.x);
32 }
33 P cross(const P &a, const P& b, const P& c) {
34     return det(b-a,c-a);
35 }
36 double area (P& a, P& b, P& c) {
37     return cross(a,b,c).len();
38 }
39 double volume (P &u, P& v, P& w, P& p) {
40     return dot(cross(u,v,w), p - u);
41 }
42 bool coplane(P &a, P& b, P& c, P& d) {
43     return sgn(dot(det(c-a,b-a), d-a)) == 0;
44 }
45 struct F {
46     int a,b,c;
47     bool ok;
48     F() {}
49     F(int aa, int bb,int cc, bool k)
50         :a(aa), b(bb), c(cc), ok(k) {}
51 };
52 struct ConvexHull {
53     int n;
54     P p[MAXN];
55     int cnt;
56     F f[MAXN];
57     int to[MAXN][MAXN];
58
59     double dir(F &t, P &u) {
60         return volume(p[t.a],p[t.b],p[t.c],u);
61     }
62     void deal(int t,int a, int b) {
63         int fid = to[a][b];
64         if (!f[fid].ok) return ;
65         if (dir(f[fid], p[t]) > EPS) dfs(t, fid);
66     }
67     else {
68         to[t][b] = to[a][t] = to[b][a] = cnt;
69         f[cnt++] = F(b,a,t,1);
70     }
71 }
72 void dfs(int t,int cur) {
73     f[cur].ok = 0;
74     deal(t, f[cur].b, f[cur].a);
75     deal(t, f[cur].c, f[cur].b);
76     deal(t, f[cur].a, f[cur].c);
77 }
78 void got() {
79     cnt = 0;
80     if (n < 4) return ;
81     random_shuffle(p, p+n);
82     for (int i = 1, j = 1; i < 4; ++i) {
83         bool flag = true;
84         for (; flag && j < n; ++j) {
85             if (i == 1 && p[0] - p[j]).len() > EPS) {
86                 swap(p[i],p[j]); flag = false;
87             } else if (i == 2 && cross(p[0], p[1], p[j]).len() >
88                     EPS) {
89                 swap(p[i],p[j]); flag = false;
90             } else if (i == 3 && !coplane(p[0],p[1],p[2], p[j])) {
91                 swap(p[i],p[j]); flag = false;
92             }
93         }
94         if (flag) return;
95     }
96     F now;
97     for (int i = 0; i < 4; ++i) {
98         now = F(i+1&3, i+2&3, i+3&3, 1);
99         if (dir(now, p[i]) > 0) swap(now.b, now.c);
100         to[now.a][now.b] = to[now.b][now.c] = to[now.c][now.a] =
101             cnt;
102         f[cnt++] = now;
103     }
104     for(int i = 4; i < n; ++i) {
105         for (int j=0; j<cnt; ++j) {
106             if (f[j].ok && dir(f[j], p[i]) > EPS) {
107                 dfs(i,j); break;
108             }
109         }
110     }
111     int tmp = cnt;
112     cnt = 0;
113     for (int i = 0; i < tmp; ++i) if (f[i].ok) f[cnt++] = f[i];
114 }
115 double area() {
116     double res = 0;
117     for (int i=0; i<cnt; ++i) {
118         res += ::area(p[f[i].a], p[f[i].b], p[f[i].c]);
119     }
120     return res*0.5;
121 }
122 double vol() {
123     P o = P(0,0,0);
124     double res = 0;
125     for(int i = 0; i < cnt; ++i) {
126         res += volume(o, p[f[i].a], p[f[i].b], p[f[i].c]);
127     }
128     return fabs(res/6);
129 }
130 bool same(int u,int v) {
131     P a = p[f[u].a], b = p[f[u].b], c = p[f[u].c];
132     return coplane(a,b,c,p[f[v].a])
133         && coplane(a,b,c,p[f[v].b])
134         && coplane(a,b,c,p[f[v].c]);
135 }
136 int face_cnt() {
137     int res = 0;
138     for (int i=0; i<cnt; ++i) {
139         bool t = 1;
140         for (int j=0; j<cnt; ++j) {

```

```

138         if(same(i,j)) t = 0;
139     }
140     res += t;
141 }
142 return res;
143 }
144 P center() {
145     P res(0,0,0), pt = p[f[0].a];
146     double v = 0, t;
147     for (int i=0; i<cnt; ++i) {
148         P a = p[f[i].a], b = p[f[i].b], c = p[f[i].c];
149         t = volume(pt, a, b, c)/6.0;
150         if (t > 0) {
151             res.x += (a.x + b.x + c.x + pt.x)*t;
152             res.y += (a.y + b.y + c.y + pt.y)*t;
153             res.z += (a.z + b.z + c.z + pt.z)*t;
154             v += t;
155         }
156     }
157     res.x /= (4*v), res.y /= (4*v), res.z /= (4*v);
158     return res;
159 }
160 };
161 ConvexHull ch;
162 double get_dis(P & pt) {
163     double res = 1e100;
164     P h;
165     double tmp;
166     for (int i = 0; i < ch.cnt; ++i) {
167         h = cross(ch.p[ch.f[i].a], ch.p[ch.f[i].b], ch.p[ch.f[i].c])
168             ;
169         tmp = fabs(dot(h,pt-ch.p[ch.f[i].a])/h.len());
170         res = min(tmp, res);
171     }
172     return res;
173 }
174 int main() {
175     while (cin >> ch.n) {
176         for (int i = 0; i < ch.n; ++i) {
177             scanf("%lf%lf%lf", &ch.p[i].x, &ch.p[i].y, &ch.p[i].z);
178         }
179         ch.got();
180         P cen = ch.center();
181         printf("%.3lf\n", get_dis(cen));
182     }

```

## 7.14 2D Convexhull

```

1 bool g_cmp(const P& a, const P& b) {
2     if(sig(a.y - b.y) != 0) return a.y < b.y;
3     return a.x < b.x;
4 }
5 //the convexhull is anti-clockwise
6 int graham(P* p, int n, int *ch) {
7     if(n < 2) {
8         ch[0] = 0;
9         return 1;
10    }
11    sort(p, p + n, g_cmp);
12    int len = 0, len0 = 1;
13    for (int i = 0; i < n; ++i) {
14        while(len > len0 && sig(cross(p[ch[len-1]], p[ch[len-2]], p[
15            i])) >= 0) --len;
16        ch[len++] = i;
17    }
18    len0 = len; // notice !!!
19    for (int i = n - 2; i >= 0; --i) {
20        while(len > len0 && sig(cross(p[ch[len-1]], p[ch[len-2]], p[
21            i])) >= 0) --len;
22        ch[len++] = i;
23    }
24    return len - 1;
25 }

```

## 7.15 Points in Triangle

```

1 const double PI = acos(-1.0);
2 const double eps = 1e-9;
3 const int N = 510;
4 struct P {
5     int x, y;
6 } ar[N], br[N];
7 int n,m;
8 double ans;
9 pair<double, int> ap[N];
10 double bp[N];
11 int half[N][N];
12 int below[N][N];
13 double angle[N][N];
14 void pre_process() {
15     for (int i = 0; i < n; ++i) {
16         for (int j = 0; j < n; ++j)
17             angle[i][j] = atan2(ar[j].y - ar[i].y + 0.0, ar[j].x - ar
18                 [i].x + 0.0);
19     }
20     for (int i = 0; i < n; ++i) {
21         for (int j = 0; j < n; ++j) {
22             if (j < i) ap[j] = make_pair(angle[i][j], j);
23             if (j > i) ap[j-1] = make_pair(angle[i][j], j);
24         }
25         sort(ap,ap+n-1);
26         for(int j = 0; j < m; ++j) {
27             bp[j] = atan2(br[j].y - ar[i].y + 0.0, br[j].x - ar[i].x
28                 + 0.0);
29         }
30         sort(bp, bp+m);
31         int beg = 0, end = 0;

```



```

31     for (int j = 0; j < n - 1; ++j) {
32         while(end < m && bp[end] + eps < ap[j].first) ++end;
33         below[i][ap[j].second] = end;
34     }
35     beg = 0, end = 0;
36     for (int j = 0; j < n - 1; ++j) {
37         if (ap[j].first > 0) break;
38         double r = ap[j].first + PI;
39         while (end < m && bp[end] + eps < r) ++end;
40         while (beg < end && bp[beg] + eps < ap[j].first) ++beg;
41         half[i][ap[j].second] = end - beg;
42         half[ap[j].second][i] = m - (end - beg);
43     }
44 }
45
46 int cross(P& o, P& a, P& b) {
47     return (a.x - o.x)*(b.y - o.y) - (b.x - o.x)*(a.y - o.y);
48 }
49
50 int get(int o, int a, int b) { // oa -> ob
51     if (angle[o][a] + eps < angle[o][b]) return below[o][b] - below
52         [o][a];
53     return below[o][b] + m - below[o][a];
54 }
55
56 bool get_Ans() {
57     pre_process();
58     bool has = false;
59     ans = 1e100;
60     for (int i = 0; i < n; ++i) {
61         for (int j = i + 1; j < n; ++j) {
62             for (int k = j + 1; k < n; ++k) {
63                 int a = i, b = j, c = k;
64                 double area = cross(ar[a], ar[b], ar[c]);
65                 if (area < 0) swap(b, c);
66                 int center = get(a, b, c) + get(b, c, a) + get(c, a, b);
67                 center += half[b][a] + half[c][b] + half[a][c];
68                 center -= 2*m;
69                 if (center != 0) {
70                     has = true;
71                     ans = min(ans, fabs(area)*0.5/center);
72                 }
73             }
74         }
75     }
76     return has;
77 }
78
79 int main() {
80     int cas, tcas = 0;
81     for (cin >> cas; cas; --cas) {
82         scanf("%d%d", &n, &m);
83         for (int i = 0; i < n; ++i) scanf("%d%d", &ar[i].x, &ar[i].y);
84         for (int i = 0; i < m; ++i) scanf("%d%d", &br[i].x, &br[i].y);
85         if (get_Ans()) printf("Case %d: %.6lf\n", ++tcas, ans);
86         else printf("Case %d: -1\n", ++tcas);
87     }
88 }

```

## 7.16 N Dimension Cut

```

1 #define N 11
2 #define M 101
3 #define MAX 100001
4 #define MOD 14121413LL
5 struct DIM {
6     __int64 l, u;
7 };
8 struct OBJ {
9     DIM cod[N];
10 };
11 bool inter(OBJ &a, OBJ &b, int n) {
12     for (int i = 0; i < n; ++i)
13         if ((a.cod[i].l - b.cod[i].u)*(a.cod[i].u - b.cod[i].l) >=
14             0)
15             return false;
16     return true;
17 }
18 OBJ rec[MAX];
19 int top;
20 __int64 cut(OBJ obj[M], int m, int n) {
21     top = 0;
22     __int64 ans=0, tmp=1;
23     for (int i = 0; i < m; ++i) {
24         tmp=1;
25         for (int j=0; j<n; j++)
26             tmp=(tmp*(obj[i].cod[j].u-obj[j].cod[j].l))%MOD;
27         ans=(ans+tmp)%MOD;
28         for (int j = top - 1; j >= 0; j--)
29             if (inter(rec[j], obj[i], n)) {
30                 for (int k=0; k<n; k++) {
31                     if (rec[j].cod[k].l<obj[i].cod[k].l) {
32                         rec[top]=rec[j];
33                         rec[top+1].cod[k].u=obj[i].cod[k].l;
34                         rec[j].cod[k].l=obj[i].cod[k].l;
35                     }
36                     if (rec[j].cod[k].u>obj[i].cod[k].u) {
37                         rec[top]=rec[j];
38                         rec[top+1].cod[k].l=obj[i].cod[k].u;
39                         rec[j].cod[k].u=obj[i].cod[k].u;
40                     }
41                 }
42                 tmp=1;
43                 for (int k=0; k<n; k++)
44                     tmp=(tmp*(rec[j].cod[k].u-rec[j].cod[k].l))%MOD;
45                 ans=(ans+tmp)%MOD;
46                 if (ans<0)
47                     ans+=MOD;
48                 --top;
49                 swap(rec[j], rec[top]);
50             }
51         rec[top+1]=obj[i];
52     }
53     return ans;
54 }

```

```

54 int main() {
55     while (scanf("%d%d", &m, &n) != EOF) {
56         OBJ obj[M];
57         for (int i = 0; i < m; ++i) {
58             for (int j = 0; j < n; ++j)
59                 scanf("%I64d", &obj[i].cod[j].l);
60             for (int j = 0; j < n; ++j)
61                 scanf("%I64d", &obj[i].cod[j].u);
62             if (obj[i].cod[j].l > obj[i].cod[j].u)
63                 swap(obj[i].cod[j].l, obj[i].cod[j].u);
64         }
65         printf("%I64d\n", cut(obj, m, n));
66     }
67 }
68

```

## 8 Sevenzero Geometry

```

1 const double PI = acos(-1.0);
2 int sgn(double a) {
3     return (a > EPS) - (a < -EPS);
4 }
5 struct Po {
6     double x, y;
7     Po(double a=0, double b=0) {x=a; y=b;}
8     Po operator - (const Po &a) const {
9         return Po(x - a.x, y - a.y);
10    }
11    Po vect(double a) const { // return a vector of length a
12        a /= sqrt(x*x + y*y);
13        return Po(x * a, y * a);
14    }
15    Po left() const { // rotate 90 degrees
16        return Po(-y, x);
17    }
18 };
19 struct Seg {
20     Po s, e;
21     Seg() {}
22     Seg(Po a, Po b) {s=a; e=b;}
23 };
24 struct Line {
25     double a, b, c;
26     Line(double x=1, double y=-1, double z=0) {a=x; b=y; c=z;}
27     Line(Po p1, Po p2) {
28         int sig=1;
29         a=p2.y-p1.y;
30         if (a<0) {a=-a; sig=-1;}
31         b=sig*(p1.x-p2.x);
32         c=sig*(p1.y*p2.x-p1.x*p2.y);
33     }
34 };
35 double xm(Po a, Po b, Po c) { // (ab)X(ac)
36     return (b.x-a.x)*(c.y-a.y)-(c.x-a.x)*(b.y-a.y);
37 }
38 double dm(Po a, Po b, Po c) { // (ab)*|ac|
39     return (b.x-a.x)*(c.x-a.x)+(b.y-a.y)*(c.y-a.y);
40 }
41 bool posy(Po &a) { // angle sort
42     if (a.y>0 || (a.y==0 && a.x>0))
43         return 1;
44     return 0;
45 }
46 bool cmp(Po a, Po b) { // sgn recommended
47     if (posy(a) != posy(b))
48         return posy(a) > posy(b);
49     return xm(Po(0, 0), a, b) > 0;
50 }
51 Po rotate(Po p, Po p0, double ang) {
52     Po vec=p-p0, ret;
53     ret.x = vec.x * cos(ang) - vec.y * sin(ang);
54     ret.y = vec.x * sin(ang) + vec.y * cos(ang);
55     return ret+p0;
56 }
57 int segcross(Seg a, Seg b) { // 1 normal 2 abnormal
58     double xml, xm2, xm3, xm4;
59     xml=xm(a.s, a.e, b.s);
60     xm2=xm(a.s, a.e, b.e);
61     xm3=xm(b.s, b.e, a.s);
62     xm4=xm(b.s, b.e, a.e);
63     if (xml*xm2<-EPS && xm3*xm4<-EPS)
64         return 1;
65     if (eq(xml, 0) && dm(b.s, a.s, a.e) < EPS) return 2;
66     if (eq(xm2, 0) && dm(b.e, a.s, a.e) < EPS) return 2;
67     if (eq(xm3, 0) && dm(a.s, b.s, b.e) < EPS) return 2;
68     if (eq(xm4, 0) && dm(a.e, b.s, b.e) < EPS) return 2;
69     return 0;
70 }
71 bool Linecross(Line l1, Line l2, Po &p) {
72     double d=l1.a*l2.b-l2.a*l1.b;
73     if (fabs(d)<EPS)
74         return false;
75     p.x=(l2.c+l1.b-l1.c+l2.b)/d;
76     p.y=(l2.a+l1.c-l1.a+l2.c)/d;
77     return true;
78 }
79 double caliper() { // convex caliper
80     if (top<=3)
81         return dis(conv[0], conv[1]);
82     int p=0;
83     double ans=0;
84     for (int i=0; i<top-1; ++i) {
85         while (xm(conv[p], conv[i], conv[i+1]) < xm(conv[(p+1)%top], conv[
86             i], conv[i+1]) + EPS)
87             p=(p+1)%top;
88         ans=max(ans, max(dis(conv[p], conv[i]), dis(conv[p], conv[i+1])));
89     }
90     return ans;
91 }
92 #define N 1510 // -----half plane-----

```



```

94 struct PS {
95     int size;
96     Po p[N];
97     PS() {size=0;}
98     void ins(Po p)
99     {
100         if(size&&eq(p[size-1].x,po.x)&&eq(p[size-1].y,po.y))
101             return;
102         p[size++]=po;
103     }
104 };
105 struct HP {
106     double a,b,c;
107     HP(double x=1,double y=-1,double z=0) {a=x;b=y;c=z;}
108     double ptol(Po p){return a*p.x+b*p.y+c;}
109 };
110 PS cut(Po p1,Po p2,PS ps) { // counterclockwise
111     PS ret;
112     for(int i=0;i<ps.size-1;i++) {
113         double xml=xm(p1,p2,ps.p[i]),xm2=xm(p1,p2,ps.p[i+1]);
114         Po crp;
115         if(xml>EPS&&xm2<-EPS) {
116             ret.ins(ps.p[i]);
117             linecross(Line(p1,p2),Line(ps.p[i],ps.p[i+1]),crp);
118             ret.ins(crp);
119             continue;
120         }
121         if(xml<-EPS&&xm2>EPS) {
122             linecross(Line(p1,p2),Line(ps.p[i],ps.p[i+1]),crp);
123             ret.ins(crp);
124             ret.ins(ps.p[i+1]);
125             continue;
126         }
127         if(xml>-EPS) ret.ins(ps.p[i]);
128         if(xm2>-EPS) ret.ins(ps.p[i+1]);
129     }
130     if(!ret.size) return ret;
131     if(ret.size==1) ret.p[ret.size++]=ret.p[0];
132     ret.ins(ret.p[0]);
133     return ret;
134 }
135 PS hpi(PS ps) {
136     PS ret=ps;
137     for(int i=0;i<ps.size-1;i++)
138         ret=cut(ps.p[i],ps.p[i+1],ret);
139     return ret;
140 }
141 {
142     PS ps;
143     scanf("%d",&n);
144     for(int i=0;i<n;i++)
145         scanf("%lf%lf",&ps.p[i].x,&ps.p[i].y);
146     ps.size=n;
147     for(int i=0;i<ps.size-1;i++)
148         ps.p[ps.size-1]=ps.p[i];
149     swap(ps.p[i],ps.p[ps.size-1]);
150     ps=hpi(ps);
151     double ans=0;
152     for(int i=0;i<ps.size-1;i++)
153         ans+=xm(Po(0,0),ps.p[i],ps.p[i+1]);
154     printf("%.2lf\n",ans/2);
155 }
156
157 //-----circle operator-----
158 // c1's arc is [rp1,rp2] c2's arc is [rp2,rp1]
159 bool inter(Po p1, double r1, Po p2, double r2, Po &rp1, Po &rp2) {
160     double cd = dis(p1, p2);
161     if (sgn(cd - r1 - r2) >= 0) return false; // no inter area //
162     // 'c' is tangency
163     if (sgn(cd - fabs(r2 - r1)) <= 0) return false; // c2 in c1 or
164     // c1 in c2
165     double l = (cd + (r1 * r1 - r2 * r2) / cd) / 2;
166     double h = sqrt(r1 * r1 - l * l);
167     rp1 = p1 + (p2 - p1).vect(l - (p2 - p1).vect(h).left());
168     rp2 = p1 + (p2 - p1).vect(l + (p2 - p1).vect(h).left());
169     return true;
170 }
171 //p1 X p2 > 0 return intersec-points'number
172 int circleLineIntersection(Po cp,double r, Po l1,Po l2,Po& p1,Po&
173 p2) {
174     Po p=cp+(l2-l1).left(),rp;
175     p.x+=l1.y-l2.y;
176     p.y+=l2.x-l1.x;
177     linecross(Line(p,cp),Line(l1,l2),rp);
178     double d=dis(rp,cp);
179     if(sgn(d-r)>0) return 0;
180     if(sgn(d-r)==0)
181     {
182         p1=p2=rp;
183         return 1;
184     }
185     double t=sqrt(r*r-d*d);
186     p1=rp-(rp-cp).left().vect(t);
187     p2=rp+(rp-cp).left().vect(t);
188     return 2;
189 }
190 {
191     inter(p1, r1, p2, r2, rp1, rp2);
192     double d=dis(p1, p2);
193     if (sgn(d - r1 - r2) >= 0)
194     {
195         puts("0.000");
196         continue;
197     }
198     if (sgn(d+r2 - r1)<=0)
199     {
200         printf("%.3lf\n",PI*r2*r2);
201         continue;
202     }
203     if (sgn(d+r1 - r2)<=0)
204     {
205         printf("%.3lf\n",PI*r1*r1);
206         continue;
207     }
208     // Heron's formula is recommended
209     double bow1=r1*r1*acos((r1*r1+d*d-r2*r2)/(2*r1*d))-xm(p1,rp1
210 ,rp2)*0.5;
211     double bow2=r2*r2*acos((r2*r2+d*d-r1*r1)/(2*r2*d))-xm(p2,rp2
212 ,rp1)*0.5;
213     printf("%.3lf\n",bow1+bow2);
214 }
215
216 //-----circle union-----
217 struct Ang {
218     double deg;
219     int dt;
220     Po p;
221     Ang(double d = 0, int t = 0, Po po = Po(0, 0)) {
222         deg = d;
223         dt = t;
224         p = po;
225     }
226     bool operator<(const Ang & a) const {
227         return deg < a.deg;
228     }
229 };
230 double ans[N]; //init ans
231 //n*n*logn
232 void mcu(Po p[N], double r[N], int &n) { // you'd better remove
233     the same circle
234     int rem[N] = {0}, cnt = 0, clude[N] = {0};
235     for (int i = 0; i < n; i++)
236         for (int j = 0; j < n; j++)
237             if (!rem[i] && i != j) {
238                 double d = dis(p[i], p[j]);
239                 //if (sgn(d - r[i] + r[j]) <= 0) rem[j] = 1; // cj in
240                 // ci union optimization
241                 //if (sgn(d + r[i] - r[j]) <= 0) rem[j] = 1; // ci in
242                 // cj inter optimization
243                 if (sgn(d) == 0 && sgn(r[i] - r[j]) == 0) // remove
244                     same
245                     rem[j] = 1;
246                 if (sgn(d + r[i] - r[j]) <= 0 && !rem[j])
247                     clude[i]++;
248             }
249     for (int i = 0; i < n; i++)
250         if (!rem[i]) {
251             p[cnt] = p[i];
252             clude[cnt] = clude[i] + 1;
253             r[cnt++] = r[i];
254         }
255     n = cnt;
256     for (int i = 0; i < n; i++) {
257         Po rp1, rp2;
258         Ang ang[4 * N];
259         cnt = 0;
260         for (int j = 0; j < n; j++)
261             if (i != j) {
262                 if (!inter(p[i], r[i], p[j], r[j], rp1, rp2)) continue
263                 ;
264                 ang[cnt++] = Ang(atan2(rp1.y - p[i].y, rp1.x - p[i].x)
265 , 1, rp1);
266                 ang[cnt++] = Ang(atan2(rp2.y - p[i].y, rp2.x - p[i].x)
267 , -1, rp2);
268                 if (ang[cnt - 2].deg > ang[cnt - 1].deg) { // tangency
269                     attention
270                     ang[cnt++] = Ang(PI, -1, p[i] - Po(r[i], 0));
271                     ang[cnt++] = Ang(-PI, 1, p[i] - Po(r[i], 0));
272                 }
273             }
274     }
275     ang[cnt++] = Ang(-PI, clude[i], p[i] - Po(r[i], 0));
276     ang[cnt++] = Ang(PI, -clude[i], p[i] - Po(r[i], 0));
277     sort(ang, ang + cnt);
278     int sum = 0;
279     for (int j = 0; j < cnt; j++) {
280         if (sum) { //sum = 1 union; sum = n intersec
281             ans[sum] += (ang[j].deg - ang[j - 1].deg) * r[i] * r[i] - xm(p[i],
282 ang[j - 1].p, ang[j].p); // bow
283             ans[sum] += xm(Po(0, 0), ang[j - 1].p, ang[j].p);
284         }
285         sum += ang[j].dt;
286     }
287     }
288     }
289     }
290     }
291     }
292     }
293     }
294     }
295     }
296     }
297     }
298     }
299     }
300     }
301     }
302     }
303     }
304     }
305     }
306     }
307     }
308     }
309     }
310     }
311     }
312     }
313     }
314     }
315     }
316     }
317     }
318     }
319     }
320     }
321     }
322     }
323     }
324     }
325     }
326     }
327     }
328     }
329     }
330     }
331     }
332     }
333     }
334     }
335     }
336     }
337     }
338     }
339     }
340     }
341     }
342     }
343     }
344     }
345     }
346     }
347     }
348     }
349     }
350     }
351     }
352     }
353     }
354     }
355     }
356     }
357     }
358     }
359     }
360     }
361     }
362     }
363     }
364     }
365     }
366     }
367     }
368     }
369     }
370     }
371     }
372     }
373     }
374     }
375     }
376     }
377     }
378     }
379     }
380     }
381     }
382     }
383     }
384     }
385     }
386     }
387     }
388     }
389     }
390     }
391     }
392     }
393     }
394     }
395     }
396     }
397     }
398     }
399     }
400     }
401     }
402     }
403     }
404     }
405     }
406     }
407     }
408     }
409     }
410     }
411     }
412     }
413     }
414     }
415     }
416     }
417     }
418     }
419     }
420     }
421     }
422     }
423     }
424     }
425     }
426     }
427     }
428     }
429     }
430     }
431     }
432     }
433     }
434     }
435     }
436     }
437     }
438     }
439     }
440     }
441     }
442     }
443     }
444     }
445     }
446     }
447     }
448     }
449     }
450     }
451     }
452     }
453     }
454     }
455     }
456     }
457     }
458     }
459     }
460     }
461     }
462     }
463     }
464     }
465     }
466     }
467     }
468     }
469     }
470     }
471     }
472     }
473     }
474     }
475     }
476     }
477     }
478     }
479     }
480     }
481     }
482     }
483     }
484     }
485     }
486     }
487     }
488     }
489     }
490     }
491     }
492     }
493     }
494     }
495     }
496     }
497     }
498     }
499     }
500     }
501     }
502     }
503     }
504     }
505     }
506     }
507     }
508     }
509     }
510     }
511     }
512     }
513     }
514     }
515     }
516     }
517     }
518     }
519     }
520     }
521     }
522     }
523     }
524     }
525     }
526     }
527     }
528     }
529     }
530     }
531     }
532     }
533     }
534     }
535     }
536     }
537     }
538     }
539     }
540     }
541     }
542     }
543     }
544     }
545     }
546     }
547     }
548     }
549     }
550     }
551     }
552     }
553     }
554     }
555     }
556     }
557     }
558     }
559     }
560     }
561     }
562     }
563     }
564     }
565     }
566     }
567     }
568     }
569     }
570     }
571     }
572     }
573     }
574     }
575     }
576     }
577     }
578     }
579     }
580     }
581     }
582     }
583     }
584     }
585     }
586     }
587     }
588     }
589     }
590     }
591     }
592     }
593     }
594     }
595     }
596     }
597     }
598     }
599     }
600     }
601     }
602     }
603     }
604     }
605     }
606     }
607     }
608     }
609     }
610     }
611     }
612     }
613     }
614     }
615     }
616     }
617     }
618     }
619     }
620     }
621     }
622     }
623     }
624     }
625     }
626     }
627     }
628     }
629     }
630     }
631     }
632     }
633     }
634     }
635     }
636     }
637     }
638     }
639     }
640     }
641     }
642     }
643     }
644     }
645     }
646     }
647     }
648     }
649     }
650     }
651     }
652     }
653     }
654     }
655     }
656     }
657     }
658     }
659     }
660     }
661     }
662     }
663     }
664     }
665     }
666     }
667     }
668     }
669     }
670     }
671     }
672     }
673     }
674     }
675     }
676     }
677     }
678     }
679     }
680     }
681     }
682     }
683     }
684     }
685     }
686     }
687     }
688     }
689     }
690     }
691     }
692     }
693     }
694     }
695     }
696     }
697     }
698     }
699     }
700     }
701     }
702     }
703     }
704     }
705     }
706     }
707     }
708     }
709     }
710     }
711     }
712     }
713     }
714     }
715     }
716     }
717     }
718     }
719     }
720     }
721     }
722     }
723     }
724     }
725     }
726     }
727     }
728     }
729     }
730     }
731     }
732     }
733     }
734     }
735     }
736     }
737     }
738     }
739     }
740     }
741     }
742     }
743     }
744     }
745     }
746     }
747     }
748     }
749     }
750     }
751     }
752     }
753     }
754     }
755     }
756     }
757     }
758     }
759     }
760     }
761     }
762     }
763     }
764     }
765     }
766     }
767     }
768     }
769     }
770     }
771     }
772     }
773     }
774     }
775     }
776     }
777     }
778     }
779     }
780     }
781     }
782     }
783     }
784     }
785     }
786     }
787     }
788     }
789     }
790     }
791     }
792     }
793     }
794     }
795     }
796     }
797     }
798     }
799     }
800     }
801     }
802     }
803     }
804     }
805     }
806     }
807     }
808     }
809     }
810     }
811     }
812     }
813     }
814     }
815     }
816     }
817     }
818     }
819     }
820     }
821     }
822     }
823     }
824     }
825     }
826     }
827     }
828     }
829     }
830     }
831     }
832     }
833     }
834     }
835     }
836     }
837     }
838     }
839     }
840     }
841     }
842     }
843     }
844     }
845     }
846     }
847     }
848     }
849     }
850     }
851     }
852     }
853     }
854     }
855     }
856     }
857     }
858     }
859     }
860     }
861     }
862     }
863     }
864     }
865     }
866     }
867     }
868     }
869     }
870     }
871     }
872     }
873     }
874     }
875     }
876     }
877     }
878     }
879     }
880     }
881     }
882     }
883     }
884     }
885     }
886     }
887     }
888     }
889     }
890     }
891     }
892     }
893     }
894     }
895     }
896     }
897     }
898     }
899     }
900     }
901     }
902     }
903     }
904     }
905     }
906     }
907     }
908     }
909     }
910     }
911     }
912     }
913     }
914     }
915     }
916     }
917     }
918     }
919     }
920     }
921     }
922     }
923     }
924     }
925     }
926     }
927     }
928     }
929     }
930     }
931     }
932     }
933     }
934     }
935     }
936     }
937     }
938     }
939     }
940     }
941     }
942     }
943     }
944     }
945     }
946     }
947     }
948     }
949     }
950     }
951     }
952     }
953     }
954     }
955     }
956     }
957     }
958     }
959     }
960     }
961     }
962     }
963     }
964     }
965     }
966     }
967     }
968     }
969     }
970     }
971     }
972     }
973     }
974     }
975     }
976     }
977     }
978     }
979     }
980     }
981     }
982     }
983     }
984     }
985     }
986     }
987     }
988     }
989     }
990     }
991     }
992     }
993     }
994     }
995     }
996     }
997     }
998     }
999     }
1000    }

```