

ACM/ICPC at Wuhan University

FreeRadiant STL

Created by team members Li Jiaqi, Han Shuai and Jin Yi

目录

Graph.....	2
Maxflow.....	2
MinCostMaxFlow.....	3
NonbipartiteMaxMatch.....	4
KM.....	5
无源无汇有上下界网络流判定可行流存在.....	5
N^3 MinCut.....	6
最小树形图.....	7
Gabow.....	8
Control.....	8
DataStructure.....	10
Splay.....	10
LeftistTree.....	12
SuffixArray.....	13
Aho-Corasick.....	14
Computational Geometry.....	14
CircleTangent.....	14
Hp Intersection.....	15
Intersection of Two Triangles.....	16
Intersection of Triangle and Circle.....	17
Point_in_poly.....	18
Convex_poly_area_union.....	18
Circle_area_union.....	19
Plane_cross.....	21
Others.....	22
Extended_GCD.....	22
PrimeChecker.....	23
BigInteger.....	23
Vimrc.....	25

Graph

Maxflow

```

1 struct Graph {
2     struct Adj {
3         int v, c, b;
4         Adj(int _v, int _c, int _b):
5             v(_v), c(_c), b(_b) {}
6         Adj() {}
7     };
8     int n, S, T, h[maxn], cnt[maxn];
9     vector<Adj> adj[maxn];
10    void clear() {
11        for (int i = 0; i < n; ++i) {
12            adj[i].clear();
13        }
14        n = 0;
15    }
16    void insert(int u, int v, int c, int d = 0) {
17        get_max(n, max(u, v) + 1);
18        adj[u].push_back(Adj(v, c, adj[v].size()));
19        adj[v].push_back(Adj(u, c * d, adj[u].size() - 1));
20    }
21    int maxflow(int _S, int _T) {
22        S = _S, T = _T;
23        fill(h, h + n, 0);
24        fill(cnt, cnt + n, 0);
25        int flow = 0;
26        while (h[S] < n) {
27            flow += dfs(S, maxint);
28        }
29        return flow;
30    }
31    int dfs(int u, int flow) {
32        if (u == T) {
33            return flow;
34        }
35        int minh = n - 1, ct = 0;
36        for (vector<Adj>::iterator it = adj[u].begin(); flow && it !=
adj[u].end(); ++it) {
37            if (it->c) {
38                if (h[it->v] + 1 == h[u]) {
39                    int k = dfs(it->v, min(it->c, flow));
40                    if (k) {
41                        it->c -= k;

```

```

42         adj[it->v][it->b].c += k;
43         flow -= k;
44         ct += k;
45     }
46     if (h[S] >= n) {
47         return ct;
48     }
49 }
50 get_min(minh, h[it->v]);
51 }
52 }
53 if (ct) {
54     return ct;
55 }
56 if (--cnt[h[u]] == 0) {
57     h[S] = n;
58 }
59 h[u] = minh + 1;
60 ++cnt[h[u]];
61 return 0;
62 }
63 };

```

MinCostMaxFlow

```

1 struct Graph{
2     struct Adj {
3         int v, c, w, b;
4         Adj(int _v, int _c, int _w, int _b):v(_v), c(_c), w(_w), b(_b) {};
5     }*st[maxn];
6     vector<Adj> adj[maxn];
7     int n;
8     void clear() {
9         for (int i = 0; i < n; ++i) {
10             adj[i].clear();
11         }
12         n = 0;
13     }
14     void insert(int u, int v, int c, int w, int d = 0) {
15         get_max(n, max(u, v) + 1);
16         adj[u].push_back(Adj(v, c, w, adj[v].size()));
17         adj[v].push_back(Adj(u, 0, -w, adj[u].size() - 1));
18         if (d) {
19             adj[v].push_back(Adj(u, c, w, adj[u].size()));
20             adj[u].push_back(Adj(v, 0, -w, adj[v].size() - 1));
21         }

```

```

22     }
23     pair<int, int> mcmf(int S, int T) {
24         int d;
25         int flow = 0, cost = 0;
26         while ((d = bell(S, T))) {
27             flow += d;
28             for (int v = T; v != S; v = adj[st[v]->v][st[v]->b].v) {
29                 cost += st[v]->w * d;
30                 st[v]->c -= d;
31                 adj[st[v]->v][st[v]->b].c += d;
32             }
33         }
34         return make_pair(flow, cost);
35     }
36     int bell(int S, int T) {
37         int d[maxn], bfs[maxn], hash[maxn];
38         fill(hash, hash + n, 0);
39         fill(d, d + n, maxint);
40         hash[S] = 1; d[S] = 0; bfs[0] = S;
41         for (int s = 0, t = 1; s != t; hash[bfs[s]] = 0, s = NEXT(s + 1, n))
42         {
43             int v = bfs[s];
44             for (vector<Adj>::iterator it = adj[v].begin(); it != adj[v].end();
45                 ++it) {
46                 if (it->c > 0 && d[v] + it->w < d[it->v]) {
47                     d[it->v] = d[v] + it->w;
48                     st[it->v] = &(*it);
49                     if (hash[it->v] == 0) {
50                         hash[it->v] = 1;
51                         bfs[t] = it->v;
52                         t = NEXT(t + 1, n);
53                     }
54                 }
55             }
56             if (d[T] == maxint) {
57                 return 0;
58             }
59             int ans = maxint;
60             for (int v = T; v != S; v = adj[st[v]->v][st[v]->b].v) {
61                 get_min(ans, st[v]->c);
62             }
63             return ans;
64         }
65     }
66 };

```

```

1 struct Graph {
2     int n, match[maxn];
3     bool adj[maxn][maxn];
4     void clear() {
5         memset(adj, 0, sizeof(adj));
6         n = 0;
7     }
8     void insert(const int &u, const int &v) {
9         get_max(n, max(u, v) + 1);
10        adj[u][v] = adj[v][u] = 1;
11    }
12    int max_match() {
13        memset(match, -1, sizeof(match));
14        int ans = 0;
15        for (int i = 0; i < n; ++i) {
16            if (match[i] == -1) {
17                ans += bfs(i);
18            }
19        }
20        return ans;
21    }
22    int Q[maxn], pre[maxn], base[maxn];
23    bool hash[maxn];
24    bool in_blossom[maxn];
25    int bfs(int p) {
26        memset(pre, -1, sizeof(pre));
27        memset(hash, 0, sizeof(hash));
28        for (int i = 0; i < n; ++i) {
29            base[i] = i;
30        }
31        Q[0] = p;
32        hash[p] = 1;
33        for (int s = 0, t = 1; s < t; ++s) {
34            int u = Q[s];
35            for (int v = 0; v < n; ++v) {
36                if (adj[u][v] && base[u] != base[v] && v != match[u]) {
37                    if (v == p || (match[v] != -1 && pre[match[v]] != -1)) {
38                        int b = contract(u, v);
39                        for (int i = 0; i < n; ++i) {
40                            if (in_blossom[base[i]]) {
41                                base[i] = b;
42                                if (hash[i] == 0) {
43                                    hash[i] = 1;

```

```

44                                Q[t++] = i;
45                            }
46                        }
47                    }
48                } else if (pre[v] == -1) {
49                    pre[v] = u;
50                    if (match[v] == -1) {
51                        argument(v);
52                        return 1;
53                    } else {
54                        Q[t++] = match[v];
55                        hash[match[v]] = 1;
56                    }
57                }
58            }
59        }
60        return 0;
61    }
62    void argument(int u) {
63        while (u != -1) {
64            int v = pre[u];
65            int k = match[v];
66            match[u] = v;
67            match[v] = u;
68            u = k;
69        }
70    }
71    void change_blossom(int b, int u) {
72        while (base[u] != b) {
73            int v = match[u];
74            in_blossom[base[v]] = in_blossom[base[u]] = true;
75            u = pre[v];
76            if (base[u] != b) {
77                pre[u] = v;
78            }
79        }
80    }
81    int contract(int u, int v) {
82        memset(in_blossom, 0, sizeof(in_blossom));
83        int b = find_base(base[u], base[v]);
84        change_blossom(b, u);
85        change_blossom(b, v);
86        if (base[u] != b) {
87            pre[u] = v;
88        }
89        if (base[v] != b) {

```

```

91     pre[v] = u;
92 }
93 return b;
94 }
95 int find_base(int u, int v) {
96     bool in_path[maxn] = {};
97     while (true) {
98         in_path[u] = true;
99         if (match[u] == -1) {
100             break;
101         }
102         u = base[pre[match[u]]];
103     }
104     while (!in_path[v]) {
105         v = base[pre[match[v]]];
106     }
107     return v;
108 }
109 };

```

KM

```

1 struct Graph {
2     int w[maxn][maxn], lx[maxn], ly[maxn], matx[maxn], maty[maxn], n;
3     bool fx[maxn], fy[maxn];
4     void clear() {
5         memset(w, 0, sizeof(w));
6         n = 0;
7     }
8     void insert(int u, int v, int c) {
9         get_max(n, max(u + 1, v + 1));
10        w[u][v] = c;
11    }
12    int match() {
13        memset(ly, 0, sizeof(ly));
14        for (int i = 0; i < n; ++i) {
15            lx[i] = -maxint;
16            for (int j = 0; j < n; ++j) {
17                get_max(lx[i], w[i][j]);
18            }
19        }
20        memset(matx, -1, sizeof(matx));
21        memset(maty, -1, sizeof(maty));
22        for (int i = 0; i < n; ++i) {
23            memset(fx, false, sizeof(fx));
24            memset(fy, false, sizeof(fy));

```

```

25        if (!dfs(i)) {
26            --i;
27            int p = maxint;
28            for (int k = 0; k < n; ++k) {
29                if (fx[k] == true) {
30                    for (int j = 0; j < n; ++j) {
31                        if ((fy[j] == false)) {
32                            get_min(p, lx[k] + ly[j] - w[k][j]);
33                        }
34                    }
35                }
36            }
37            for (int j = 0; j < n; ++j) {
38                ly[j] += fy[j] * p;
39            }
40            for (int k = 0; k < n; ++k) {
41                lx[k] -= fx[k] * p;
42            }
43        }
44    }
45    int ans = 0;
46    for (int i = 0; i < n; ++i) {
47        ans += w[maty[i]][i];
48    }
49    return ans;
50 }
51 bool dfs(int u) {
52     fx[u] = 1;
53     for (int v = 0; v < n; ++v) {
54         if (lx[u] + ly[v] == w[u][v] && fy[v] == false) {
55             fy[v] = true;
56             if (maty[v] == -1 || dfs(maty[v])) {
57                 matx[u] = v;
58                 maty[v] = u;
59                 return true;
60             }
61         }
62     }
63     return false;
64 }
65 };

```

无源无汇有上下界网络流判定可行流存在

```

1 bool bfs(int ss, int tt) {
2     memset(level, -1, sizeof(level));

```

```

3     que[0] = tt;
4     level[tt] = n;
5     for (int h = 0, t = 0; h <= t; h++) {
6         int u = que[h];
7         for (int i = 0; i < edge[u].size(); i++)
8             if (level[edge[u][i].v] == -1 &&
edge[edge[u][i].v][edge[u][i].b].f > 0) {
9                 level[edge[u][i].v] = level[u] - 1;
10                t++;
11                que[t] = edge[u][i].v;
12            }
13    }
14    return (level[ss] != -1);
15 }
16 void solve()
17 {
18     if (dinic(0, n - 1) != s) {
19         printf("NO\n");
20         return ;
21     }
22     for (int i = 1; i < n - 1; i++)
23         for (int j = 0; j < edge[i].size(); j++)
24             if (edge[i][j].index > -1)
25                 ans[edge[i][j].index] += cap[i][j] - edge[i][j].f;
26 }
27

```

N^3 MinCut

```

1 int n, m;
2 int adj[210][210];
3 int f[210], bestf[210];
4 int find(int v) {
5     if (v != bestf[v]) {
6         return bestf[v] = find(bestf[v]);
7     }
8     return v;
9 }
10 int bestff;
11 int v[210], w[210];
12 bool a[210];
13 vector<pair<int, int> > res;
14 int mincut() {
15     int ans = maxint;
16     for (int i = 0; i < n; ++i) {
17         for (int j = 0; j < n; ++j) {

```

```

18             adj[i][j] = -adj[i][j];
19         }
20     }
21     for (int i = 0; i < n; ++i) {
22         f[i] = v[i] = i;
23     }
24     for (int t = n; t > 1; --t) {
25         a[0] = true;
26         for (int i = 1; i < t; ++i) {
27             a[i] = false;
28             w[i] = adj[v[0]][v[i]];
29         }
30         int prev = v[0];
31         for (int i = 1; i < t; ++i) {
32             int zj = -1;
33             for (int j = 1; j < t; ++j) {
34                 if (a[j] == false && (zj == -1 || w[j] > w[zj])) {
35                     zj = j;
36                 }
37             }
38             a[zj] = true;
39             if (i == t - 1) {
40                 if (w[zj] < ans) {
41                     ans = w[zj];
42                     copy(f, f + n, bestf);
43                     bestff = v[zj];
44                 }
45                 for (int k = 0; k < t; ++k) {
46                     adj[v[k]][prev] = adj[prev][v[k]] += adj[v[zj]][v[k]];
47                 }
48                 f[v[zj]] = prev;
49                 v[zj] = v[t - 1];
50                 break;
51             }
52             prev = v[zj];
53             for (int j = 1; j < t; ++j) {
54                 if (a[j] == false) {
55                     w[j] += adj[v[zj]][v[j]];
56                 }
57             }
58         }
59     }
60     bool flag[210] = {};
61     for (int i = 0; i < n; ++i) {
62         if (find(i) == bestff) {
63             flag[i] = true;
64         }

```

```

65     }
66     res.clear();
67     for (int i = 0; i < n; ++i) {
68         for (int j = i + 1; j < n; ++j) {
69             if (flag[i] == flag[j] && save[i][j]) {
70                 res.push_back(make_pair(i, j));
71             }
72         }
73     }
74     return ans;
75 }

```

最小树形图

```

1 //最小树形图 这个版本求的是欧几里德最小树形图,根为1
2 #include <stdio.h>
3 #include <string.h>
4 #include <math.h>
5 const int MAXV=101,MAXE=10001;
6 const double MAXDOUBLE=1e20;
7 double gh[MAXV][MAXV],//原始有向图,顶点从1开始编号
8 mincost[MAXV],ans;
9 int prev[MAXV],stack[MAXV],id[MAXV],mark[MAXV],numv,nume,cnt,top,sta,
10 x[MAXV],y[MAXV];//顶点的x,y坐标
11 bool scanned[MAXV];
12 void Combine() {
13     int i,j,now,x;
14     double t;
15     now=stack[sta];
16     for (i=sta;i<top;i++) {
17         x=stack[i];
18         ans+=mincost[x];
19         id[x]=now;
20         for (j=1;j<=numv;j++) {
21             if (gh[j][x]!=-1) {
22                 t=gh[j][x]-mincost[x];
23                 if (t<gh[j][now] || gh[j][now]==-1) gh[j][now]=t;
24             }
25             if ((gh[x][j]!=-1 && gh[x][j]<gh[now][j]) || gh[now][j]==-1)
gh[now][j]=gh[x][j];
26         }
27     }
28     for (i=2;i<=numv;i++) if (id[i]==i) prev[i]=id[prev[i]];
29     mincost[now]=MAXDOUBLE;
30     for (i=1;i<=numv;i++) if (i!=now && id[i]==i && gh[i][now]!=-1 &&
gh[i][now]<mincost[now]) mincost[now]=gh[i][now],prev[now]=i;

```

```

31 }
32 bool Find_Circle() {
33     int i,temp,mark[MAXV];
34     memset(scanned,0,sizeof(scanned));
35     for (i=numv;i>1;i--) {
36         if (id[i]!=i || scanned[i]) continue;
37         memset(mark,0,sizeof(mark));
38         top=1;
39         temp=i;
40         while (temp!=1 && !mark[temp] && !scanned[temp]) {
41             scanned[temp]=1;
42             mark[temp]=top;
43             stack[top++]=temp;
44             temp=prev[temp];
45         }
46         if (mark[temp]) {
47             sta=mark[temp];
48             return 1;
49         }
50     }
51     return 0;
52 }
53 void DFS(int v) {
54     cnt++;
55     scanned[v]=1;
56     for (int i=1;i<=numv;i++) if (!scanned[i] && gh[v][i]!=-1) DFS(i);
57 }
58 void Minimum_Arborescence () {
59     while (Find_Circle()) Combine();
60     for (int i=2;i<=numv;i++) if (i==id[i]) ans+=mincost[i];
61 }
62 int main () {
63     int i,j,v1,v2,val;
64     while (scanf("%d%d",&numv,&nume)!=EOF) {
65         for (i=1;i<=numv;i++) for (j=1;j<=numv;j++) gh[i][j]=-1;
66         memset(scanned,0,sizeof(scanned));
67         for (i=1;i<=numv;i++) scanf("%d%d",&x[i],&y[i]);
68         for (i=0;i<nume;i++) {
69             scanf("%d%d",&v1,&v2);
70             gh[v1][v2]=sqrt((x[v1]-x[v2])*(x[v1]-x[v2])+(y[v1]-y[v2])*(y[v1]-y[v2]));
71         }
72         cnt=0;
73         DFS(1);
74         if (cnt<numv) printf("poor snoopy\n");//最小树形图不存在
75     }
76     id[1]=1;

```

```

77         for (i=2;i<=numv;i++) {
78             id[i]=i;
79             mincost[i]=MAXDOUBLE;
80             for (j=1;j<=numv;j++) if (i!=j && gh[j][i]!=-1 &&
gh[j][i]<mincost[i]) mincost[i]=gh[j][i],prev[i]=j;
81         }
82         ans=0;
83         Minimum_Arborescence();
84         printf("%.21f\n",ans); //答案保留两位小数
85     }
86 }
87 return 0;
88 }

```

Gabow

```

1 int sc[maxn],pre[maxn],path[maxn],s[maxn];
2 int cnt0,cnt1,lens,lenp;
3 void dfs(int w)
4 {
5     pre[w]=cnt0++;
6     s[lens++]=w;    path[lenp++]=w;
7     for(_adj*pt=adj[w];pt;pt=pt->next)
8         if(pre[pt->v]==-1) dfs(pt->v);
9     else
10         if(sc[pt->v]==-1)
11             while(pre[path[lenp-1]]>pre[pt->v]) lenp--;
12     if(path[lenp-1]!=w) return ;
13     else lenp--;
14     do sc[s[--lens]]=cnt1; while(s[lens]!=w);
15     cnt1++;
16 }
17 void memset()
18 {
19     memset(adj,0,sizeof(adj));
20     memset(sc,-1,sizeof(sc));
21     memset(pre,-1,sizeof(pre));
22     lens=lenp=0;
23     cnt0=0;
24     cnt1=0;
25 }

```

Control

```

1 class Graph {

```

```

2     public:
3         static const int maxn = 2600 / 2;
4         static const int maxm = 51;
5         static const int mmax = maxn * maxm + maxn + maxm + 100;
6         int n, m;
7         int adj[maxn][maxm];
8         void init() {
9             n = m = 0;
10            memset(adj, 0, sizeof(adj));
11        }
12        void insert(int u, int v) {
13            get_max(n, u + 1);
14            get_max(m, v + 1);
15            adj[u][v] = 1;
16        }
17        int find_ans() {
18            build_dlx();
19            for (int ans = 0; ans <= n; ++ans) {
20                if (dfs(0, ans)) {
21                    return ans;
22                }
23            }
24            return -1;
25        }
26        private:
27            int head;
28            int U[mmax], D[mmax], L[mmax], R[mmax],
29                CN[mmax], RN[mmax];
30            void addUD(const int &a, const int &h) {
31                U[a] = h;
32                D[a] = D[h];
33                U[D[h]] = a;
34                D[h] = a;
35                CN[a] = h;
36            }
37            void addLR(const int &a, const int &h) {
38                L[a] = h;
39                R[a] = R[h];
40                L[R[h]] = a;
41                R[h] = a;
42                RN[a] = h;
43            }
44            void add(const int &k, const int &r, const int &c) {
45                addUD(k, c);
46                addLR(k, r);
47            }
48            void remove(const int &k) {

```



```

49     for (int j = R[k]; j != k; j = R[j]) {
50         for (int i = D[j]; i != j; i = D[i]) {
51             L[R[i]] = L[i];
52             R[L[i]] = R[i];
53         }
54         D[U[j]] = D[j];
55         U[D[j]] = U[j];
56     }
57 }
58 void unremove(const int &k) {
59     for (int j = L[k]; j != k; j = L[j]) {
60         D[U[j]] = j;
61         U[D[j]] = j;
62         for (int i = U[j]; i != j; i = U[i]) {
63             L[R[i]] = i;
64             R[L[i]] = i;
65         }
66     }
67 }
68 void build_dlx() {
69     head = mmax - 1;
70     U[head] = D[head] = L[head] = R[head] = head;
71     int cnt = 0;
72     for (int i = 0; i < m; ++i) {
73         U[cnt] = D[cnt] = cnt;
74         addLR(cnt++, head);
75     }
76     for (int i = 0; i < n; ++i) {
77         L[cnt] = R[cnt] = cnt;
78         addUD(cnt++, head);
79     }
80     for (int i = 0; i < n; ++i) {
81         for (int j = 0; j < m; ++j) {
82             if (adj[i][j]) {
83                 addLR(cnt, m + i);
84                 addUD(cnt, j);
85                 cnt++;
86             }
87         }
88     }
89 }
90 }
91 int h() {
92     int hash[maxm] = {};
93     int ans = 0;
94     for (int c = R[head]; c != head; c = R[c]) {
95         if (hash[c] == 0) {

```

```

96         hash[c] = 1;
97         ++ans;
98         for (int j = D[c]; j != c; j = D[j]) {
99             for (int i = R[j]; i != j; i = R[i]) {
100                 if (CN[i] != head) {
101                     hash[CN[i]] = 1;
102                 }
103             }
104         }
105     }
106     return ans;
107 }
108 bool dfs(int k, int lim) {
109     if (k + h() > lim) {
110         return false;
111     }
112     if (R[head] == head) {
113         return true;
114     }
115     int c;
116     c = R[head];
117     L[R[c]] = L[c];
118     R[L[c]] = R[c];
119     for (int i = D[c]; i != c; i = D[i]) {
120         L[R[i]] = L[i];
121         R[L[i]] = R[i];
122     }
123     for (int i = D[c]; i != c; i = D[i]) {
124         remove(RN[i]);
125         if (dfs(k + 1, lim)) {
126             return true;
127         }
128         unremove(RN[i]);
129     }
130     for (int i = U[c]; i != c; i = U[i]) {
131         L[R[i]] = i;
132         R[L[i]] = i;
133     }
134     L[R[c]] = c;
135     R[L[c]] = c;
136     return false;
137 }
138 }
139 };

```

Splay

```

1  const int MaxN = 200000 + 1;
2
3  struct treeNode {
4      int l, r, p, s;
5      int Key, Min, Add;
6      bool Reverse;
7  };
8
9  treeNode tree[MaxN];
10 int N, M, Root;
11
12 inline void Pass(int t) {
13     int l = tree[t].l, r = tree[t].r;
14     if (tree[t].Add) {
15         tree[l].Add += tree[t].Add;
16         tree[r].Add += tree[t].Add;
17         tree[t].Key += tree[t].Add;
18         tree[t].Min += tree[t].Add;
19         tree[t].Add = 0;
20     }
21     if (tree[t].Reverse) {
22         swap(tree[t].l, tree[t].r);
23         tree[l].Reverse ^= 1;
24         tree[r].Reverse ^= 1;
25         tree[t].Reverse = false;
26     }
27 }
28
29 inline void Update(int t) {
30     tree[t].Min = tree[t].Key;
31     int l = tree[t].l, r = tree[t].r;
32     if (l) tree[t].Min = min(tree[t].Min, tree[l].Min + tree[l].Add);
33     if (r) tree[t].Min = min(tree[t].Min, tree[r].Min + tree[r].Add);
34     tree[t].Min += tree[t].Add;
35     tree[t].s = tree[l].s + tree[r].s + 1;
36 }
37
38 inline void Left(int x) {

```

```

39     int y = tree[x].p;
40     int z = tree[y].p;
41     if (tree[z].l == y) tree[z].l = x;
42     else tree[z].r = x;
43     tree[x].p = z;
44     tree[y].p = x;
45     tree[y].r = tree[x].l;
46     tree[x].l = y;
47     tree[tree[y].r].p = y;
48     Update(y);
49 }
50
51 inline void Right(int x) {
52     int y = tree[x].p;
53     int z = tree[y].p;
54     if (tree[z].l == y) tree[z].l = x;
55     else tree[z].r = x;
56     tree[x].p = z;
57     tree[y].p = x;
58     tree[y].l = tree[x].r;
59     tree[x].r = y;
60     tree[tree[y].l].p = y;
61     Update(y);
62 }
63
64 int stack[MaxN];
65
66 inline void Splay(int x) {
67     int top = 0;
68     for (int t = x; t; t = tree[t].p)
69         stack[++top] = t;
70     for (int i = top; i >= 1; --i) Pass(stack[i]);
71     while (tree[x].p) {
72         int y = tree[x].p;
73         int z = tree[y].p;
74         if (z) {
75             if (tree[z].l == y) {
76                 if (tree[y].l == x) {
77                     Right(y); Right(x);
78                 }
79             }
80             else {
81                 Left(x); Right(x);
82             }
83         }
84     }
85 }

```

```

82     }
83     else {
84         if (tree[y].l == x) {
85             Right(x); Left(x);
86         }
87         else {
88             Left(y); Left(x);
89         }
90     }
91 }
92 else {
93     if (tree[y].l == x) Right(x);
94     else Left(x);
95 }
96 }
97 Update(x);
98 Root = x;
99 }
100
101 inline int Select(int K) {
102     int t = Root;
103     while (t) {
104         Pass(t);
105         if (tree[tree[t].l].s + 1 >= K) {
106             if (tree[tree[t].l].s < K) break;
107             t = tree[t].l;
108         } else {
109             K -= tree[tree[t].l].s + 1;
110             t = tree[t].r;
111         }
112     }
113     Splay(t);
114     return t;
115 }
116
117 inline void ADD() {
118     int P, Q, D;
119     scanf("%d %d %d", &P, &Q, &D);
120     P = Select(P); Q = Select(Q);
121     int L = tree[P].l, R = tree[Q].r;
122     tree[Q].Add += D;
123     tree[L].Add -= D;
124     tree[R].Add -= D;

```

```

125 }
126
127 inline void REVERSE() {
128     int P, Q;
129     scanf("%d %d", &P, &Q);
130     P = Select(P); Q = Select(Q);
131     int L = tree[P].l, R = tree[Q].r;
132     tree[P].l = tree[Q].r = 0;
133     tree[Q].Reverse ^= 1;
134     Splay(P);
135     tree[Q].l = L; tree[L].p = Q;
136     tree[P].r = R; tree[R].p = P;
137     Splay(Q);
138 }
139
140 inline void REVOLVE() {
141     int P, Q, T;
142     scanf("%d %d %d", &P, &Q, &T);
143     T %= (Q - P + 1);
144     if (!T) return;
145     int I = Select(Q - T);
146     P = Select(P); Q = Select(Q);
147     int L = tree[P].l, R = tree[Q].r;
148     tree[P].l = tree[Q].r = 0;
149     Splay(P); Splay(I);
150     int J = tree[I].r;
151     tree[I].r = 0; tree[J].p = 0;
152     tree[I].p = Q; tree[Q].r = I;
153     while (Pass(J), tree[J].l) J = tree[J].l;
154     tree[J].l = L; tree[L].p = J;
155     tree[I].r = R; tree[R].p = I;
156     Splay(J);
157 }
158
159 inline void INSERT() {
160     int x, P;
161     scanf("%d %d", &P, &x);
162     tree[++N].Key = x;
163     if (P) {
164         P = Select(P);
165         int Q = tree[P].r;
166         tree[P].r = N; tree[N].p = P;
167         tree[N].r = Q; tree[Q].p = N;

```

```

168     }
169     else {
170         P = Select(1);
171         tree[N].r = P; tree[P].p = N;
172     }
173     Splay(N);
174 }
175
176 inline void DELETE() {
177     int P;
178     scanf("%d", &P);
179     P = Select(P);
180     int Q = tree[P].l;
181     tree[Q].p = 0;
182     if (!Q) {
183         Root = tree[P].r;
184         tree[tree[P].r].p = 0;
185         return;
186     }
187     if (!tree[P].r) return;
188     while (Pass(Q), tree[Q].r) Q = tree[Q].r;
189     tree[Q].r = tree[P].r;
190     tree[tree[P].r].p = Q;
191     Splay(Q);
192 }
193
194 inline void MIN() {
195     int P, Q;
196     scanf("%d %d", &P, &Q);
197     P = Select(P); Q = Select(Q);
198     int L = tree[P].l, R = tree[Q].r;
199     tree[P].l = tree[Q].r = 0;
200     Splay(P);
201     printf("%d\n", tree[P].Min);
202     tree[P].l = L; tree[Q].r = R;
203     Splay(Q);
204 }
205
206 int main() {
207     for (int i = 1; i <= N; ++i) {
208         tree[i].p = i - 1; tree[i].r = i + 1;
209     }
210     Root = 1;

```

```

211     tree[N].r = 0;
212     for (int i = N; i >= 1; --i)
213         Update(i);
214 }

```

LeftistTree

```

1 struct Tree {
2     static const int maxn = 1000000;
3     typedef int type_name;
4     static int d[maxn], l[maxn], r[maxn], len;
5     static type_name key[maxn];
6     int root;
7     void init() {
8         root = -1;
9     }
10    void insert(const type_name &k) {
11        root = merge(root, new_node(k));
12    }
13    const type_name& min() {
14        return key[root];
15    }
16    void erase_min() {
17        root = merge(l[root], r[root]);
18    }
19    void merge(const Tree& t) {
20        root = merge(root, t.root);
21    }
22    int new_node(const type_name &k) {
23        key[len] = k;
24        l[len] = r[len] = d[len] = -1;
25        return len++;
26    }
27    int merge(int a, int b) {
28        if (a == -1) {
29            return b;
30        }
31        if (b == -1) {
32            return a;
33        }
34        if (key[b] < key[a]) {
35            swap(a, b);
36        }
37        r[a] = merge(r[a], b);
38        if (d[r[a]] > d[l[a]]) {
39            swap(l[a], r[a]);

```

```

40     }
41     if (r[a] == -1) {
42         d[a] = 0;
43     } else {
44         d[a] = d[r[a]] + 1;
45     }
46     return a;
47 }
48 };
49 int Tree::d[maxn], Tree::l[maxn], Tree::r[maxn], Tree::len;
50 Tree::type_name Tree::key[maxn];

```

SuffixArray

```

1 //suffix array (da)
2 #define maxn 1000001
3 int wa[maxn],wb[maxn],wv[maxn],ws[maxn];
4 int cmp(int *r,int a,int b,int l)
5 {
6     return r[a]==r[b] && r[a+l]==r[b+l];
7 }
8 void da(int *r,int *sa,int n,int m)
9 {
10     int i, j, p, *x=wa, *y=wb, *t;
11     for(i = 0; i < m; i++) ws[i] = 0;
12     for(i = 0; i < n; i++) ws[x[i]] = r[i]++;
13     for(i = 1; i < m; i++) ws[i] += ws[i-1];
14     for(i = n-1; i >= 0; i--) sa[--ws[x[i]]] = i;
15     for(j = 1, p = 1; p < n; j *= 2, m = p)
16     {
17         for(p = 0, i = n-j; i < n; i++) y[p++] = i;
18         for(i = 0; i < n; i++) if(sa[i] >= j) y[p++] = sa[i]-j;
19         for(i = 0; i < n; i++) wv[i] = x[y[i]];
20         for(i = 0; i < m; i++) ws[i] = 0;
21         for(i = 0; i < n; i++) ws[wv[i]] ++;
22         for(i = 1; i < m; i++) ws[i] += ws[i-1];
23         for(i = n-1; i >= 0; i--) sa[--ws[wv[i]]] = y[i];
24         for(t=x, x=y, y=t, p=1, x[sa[0]]=0, i=1; i < n; i++){
25             x[sa[i]] = cmp(y,sa[i-1],sa[i],j) ? p-1 : p++;
26         }
27     }
28 }
29 int rank[maxn], h[maxn];

```

```

30 void calheight(int *r,int *sa,int n)
31 {
32     int i, j, k = 0;
33     for(i = 0; i < n; i++) rank[sa[i]] = i;
34     for(i = 0; i < n; h[rank[i++]] = k){
35         for(k ? k-- : 0, j = sa[rank[i]-1]; r[i+k] == r[j+k]; k++);
36     }
37 }
38 int l2[maxn], best[20][maxn];
39 void initRMQ(int n)
40 {
41     int i, j;
42     for(l2[0] = -1, i = 1; i <= n; i++){
43         l2[i] = ((i&(i-1))==0) ? l2[i-1]+1 : l2[i-1];
44     }
45     for(i = 1; i <= n; i++) best[0][i] = h[i];
46     for(i = 1; i <= l2[n]; i++)
47         for(j = 1; j <= n+1-(1<<i); j++)
48         {
49             best[i][j] = min(best[i-1][j], best[i-1][j+(1<<(i-1))]);
50         }
51 }
52 int askRMQ(int a,int b)
53 {
54     int t = 1<<l2[b-a+1];
55     return min(best[t][a], best[t][b-t+1]);
56 }
57 //longest common subsequence
58 char s1[maxn], s2[maxn];
59 //multiply 3 if it's dc3
60 int r[maxn], sa[maxn];
61 int main() {
62     while(scanf("%s", s1) != EOF){
63         scanf("%s", s2);
64         int len1 = strlen(s1), len2 = strlen(s2);
65         for(int i = 0; i < len1; i++) r[i] = (int)s1[i];
66         r[len1] = 1;
67         for(int i = 0; i < len2; i++) r[i+len1+1] = (int)s2[i];
68         int n = len1 + len2 + 1;
69         r[n] = 0;
70         dc3(r, sa, n+1, 128);
71         ans = 0;
72         calheight(r, sa, n);

```

```

73     for(int i = 2; i <= n; i++){
74         if((sa[i-1] > len1 && sa[i] < len1) ||
75            (sa[i-1] < len1 && sa[i] > len1)){
76             ans = max(ans, h[i]);
77         }
78     }
79     printf("%d\n", ans);
80 }
81 return 0;
82 }

```

Aho-Corasick

```

1  class Trie {
2  public:
3      const static int st = 'A', en = 'z';
4      const static int m = en - st + 1;
5      const static int maxn = 10002;
6      int d[maxn][m];    //graph
7      int t[maxn];        //state
8      int p[maxn];        //suffix
9      int n, len;
10     void init() {
11         len = 1;
12         n = 0;
13         t[0] = 0;
14         memset(d[0], -1, sizeof(d[0]));
15     }
16     void insert(char *s, int id) {
17         int i;
18         for(i = 0; *s; ++s) {
19             int &k = d[i][*s - st];
20             if(k == -1) {
21                 k = len++;
22                 memset(d[k], -1, sizeof(d[k]));
23                 t[k] = 0;
24             }
25             i = k;
26         }
27         t[i] |= 1<<id;
28     }
29     void bfs() {

```

```

30     int i;
31     queue<int> q;
32     q.push(0);
33     p[0] = 0;
34     while(!q.empty()) {
35         int k = q.front();
36         q.pop();
37         for(i = 0; i < m; i++) {
38             int &j = d[k][i];
39             if(-1 == j) {
40                 j = d[p[k]][i];
41                 if(j == -1) j = 0;
42             }
43             else {
44                 if (k) p[j] = d[p[k]][i];
45                 else p[j] = 0;
46                 t[j] |= t[p[j]];
47                 q.push(j);
48             }
49         }
50     }
51 }
52 };

```

Computational Geometry

CircleTangent

```

1  vector<pair<P, P>> circle_tangent(const C &a, const C &b) {
2      vector<pair<P, P>> ans;
3      double len2 = dist2(a.mid, b.mid);
4      if (sgn(len2 - SQR(a.r + b.r)) > 0) {
5          double len = sqrt(len2);
6          double x = len * a.r / (b.r + a.r);
7          double y = len * b.r / (a.r + b.r);
8          double ta = sqrt(abs(sqr(x) - sqr(a.r)));
9          double ha = a.r * ta / x;
10         double pa = sqrt(abs(sqr(a.r) - sqr(ha)));
11         double tb = sqrt(abs(sqr(y) - sqr(b.r)));
12         double hb = b.r * tb / y;
13         double pb = sqrt(abs(sqr(b.r) - sqr(hb)));
14         P ab = b.mid - a.mid;
15         P ba = a.mid - b.mid;

```

```

16     ans.push_back(make_pair(a.mid + ab.trunc(pa) +
ab.turn_right().trunc(ha),
17         b.mid + ba.trunc(pb) + ba.turn_right().trunc(hb)));
18     ans.push_back(make_pair(a.mid + ab.trunc(pa) +
ab.turn_left().trunc(ha),
19         b.mid + ba.trunc(pb) + ba.turn_left().trunc(hb)));
20 }
21 if (!a.in(b) && !b.in(a) && a != b) {
22     double len = sqrt(len2);
23     double n = sqrt(abs(len2 - sqr(a.r - b.r)));
24     double ha = a.r * n / len;
25     double hb = b.r * n / len;
26     double pa = sqrt(abs(sqr(a.r) - sqr(ha)));
27     double pb = sqrt(abs(sqr(b.r) - sqr(hb)));
28     P p = (a.r > b.r? b.mid - a.mid: a.mid - b.mid);
29     ans.push_back(make_pair(a.mid + p.trunc(pa) +
p.turn_right().trunc(ha),
30         b.mid + p.trunc(pb) + p.turn_right().trunc(hb)));
31     ans.push_back(make_pair(a.mid + p.trunc(pa) +
p.turn_left().trunc(ha),
32         b.mid + p.trunc(pb) + p.turn_left().trunc(hb)));
33 }
34 return ans;
35 }

```

Hp Intersection

```

1 P line_intersection(const P &a, const P &b, const P &c, const P &d) {
2     double s1 = cross(a, b, c);
3     double s2 = cross(a, b, d);
4     return P((c.x * s2 - d.x * s1) / (s2 - s1), (c.y * s2 - d.y * s1) / (s2
- s1));
5 }
6 P line_intersection(const pair<P, P> &a, const pair<P, P> &b) {
7     return line_intersection(a.first, a.second, b.first, b.second);
8 }
9
10 bool cmp_cross(const pair<P, P> &a, const pair<P, P> &b) {
11     int k = dcmp(cross(a.second - a.first, b.second - b.first));
12     if (k == 0) {
13         return dcmp(cross(b.first - a.first, a.second - a.first)) > 0;
14     }
15     return k > 0;
16 }
17 bool cmp_cross_equal(const pair<P, P> &a, const pair<P, P> &b) {
18     return dcmp(cross(a.second - a.first, b.second - b.first)) == 0;

```

```

19 }
20
21 vector<P>& hp_intersection(const vector<pair<P, P> > &p) {
22     static vector<pair<P, P> > a, b;
23     a.clear(), b.clear();
24     for (int i = 0; i < SZ(p); ++i) {
25         if (dcmp(p[i].second.y - p[i].first.y) > 0
26             || dcmp(p[i].second.y - p[i].first.y) == 0 &&
dcmp(p[i].second.x - p[i].first.x) < 0) {
27             a.push_back(p[i]);
28         } else {
29             b.push_back(p[i]);
30         }
31     }
32     sort(a.begin(), a.end(), cmp_cross);
33     a.erase(unique(a.begin(), a.end(), cmp_cross_equal), a.end());
34     sort(b.begin(), b.end(), cmp_cross);
35     b.erase(unique(b.begin(), b.end(), cmp_cross_equal), b.end());
36     static vector<P> stp;
37     static vector<pair<P, P> > st;
38     static deque<P> pa, pb;
39     pa = pb = deque<P> (0);
40     if (SZ(a) == 1) {
41         pa.push_back(a[0].first);
42         pa.push_back(a[0].second);
43     } else {
44         stp.clear();
45         st.clear();
46         st.push_back(a[0]);
47         for (int i = 1; i < SZ(a); ++i) {
48             P p = line_intersection(st.back(), a[i]);
49             if (st.size() == 1 || dcmp(p.y - stp.back().y) > 0) {
50                 stp.push_back(a[i]);
51                 stp.push_back(p);
52             } else {
53                 stp.pop_back();
54                 st.pop_back();
55                 --i;
56             }
57         }
58         for (int i = 0; i < SZ(stp); ++i) {
59             pa.push_back(stp[i]);
60         }
61         pa.push_front(pa.front() + st.front().first - st.front().second);
62         pa.push_back(pa.back() + st.back().second - st.back().first);
63     }
64     if (SZ(b) == 1) {

```

```

65     pb.push_back(b[0].first);
66     pb.push_back(b[0].second);
67 } else {
68     stp.clear();
69     st.clear();
70     st.push_back(b[0]);
71     for (int i = 1; i < SZ(b); ++i) {
72         P p = line_intersection(st.back(), b[i]);
73         if (st.size() == 1 || dcmp(p.y - stp.back().y) < 0) {
74             st.push_back(b[i]);
75             stp.push_back(p);
76         } else {
77             stp.pop_back();
78             st.pop_back();
79             --i;
80         }
81     }
82     for (int i = 0; i < SZ(stp); ++i) {
83         pb.push_back(stp[i]);
84     }
85     pb.push_front(pb.front() + st.front().first - st.front().second);
86     pb.push_back(pb.back() + st.back().second - st.back().first);
87 }
88 if (dcmp(cross(*++pa.begin() - pa.front(), *++pb.rbegin() -
pb.back())) != 0) {
89     P p = line_intersection(*++pa.begin(), pa.front(),
*++pb.rbegin(), pb.back());
90     if (dcmp(cross(pa.front(), *++pa.begin(), pb.back())) > 0) {
91         pb.back() = p;
92     }
93     if (dcmp(cross(pb.back(), *++pb.rbegin(), pa.front())) < 0) {
94         pa.front() = p;
95     }
96 }
97 if (dcmp(cross(*++pb.begin() - pb.front(), *++pa.rbegin() -
pa.back())) != 0) {
98     P p = line_intersection(*++pb.begin(), pb.front(),
*++pa.rbegin(), pa.back());
99     if (dcmp(cross(pb.front(), *++pb.begin(), pa.back())) > 0) {
100         pa.back() = p;
101     }
102     if (dcmp(cross(pa.back(), *++pa.rbegin(), pb.front())) < 0) {
103         pb.front() = p;
104     }
105 }
106 while (SZ(pa) >= 2 && SZ(pb) >= 2) {
107     if (dcmp(cross(pb.back(), *++pb.rbegin(), *++pa.begin())) > 0))

```

```

{
108     pa.pop_front();
109     } else if (dcmp(cross(pa.front(), *++pa.begin(),
*++pb.rbegin())) < 0) {
110         pb.pop_back();
111     } else if (dcmp(cross(pa.back(), *++pa.rbegin(),
*++pb.begin())) > 0) {
112         pb.pop_front();
113     } else if (dcmp(cross(pb.front(), *++pb.begin(),
*++pa.rbegin())) < 0) {
114         pa.pop_back();
115     } else {
116         pa.front() = pb.back() = line_intersection(pa.front(),
*++pa.begin(), pb.back(), *++pb.rbegin());
117         pa.back() = pb.front() = line_intersection(pa.back(),
*++pa.rbegin(), pb.front(), *++pb.begin());
118         break;
119     }
120 }
121 static vector<P> ans;
122 ans.clear();
123 if (SZ(pa) < 2 || SZ(pb) < 2) {
124     return ans;
125 }
126 for (int i = 0; i + 1 < SZ(pa); ++i) {
127     ans.push_back(pa[i]);
128 }
129 for (int i = 0; i + 1 < SZ(pb); ++i) {
130     ans.push_back(pb[i]);
131 }
132 return ans;
133 }

```

Intersection of Two Triangles

```

1 bool intersection(const P &a, const P &b, const P &c, const P &d, P &p) {
2     double s1 = cross(a, b, c);
3     double s2 = cross(a, b, d);
4     if (dcmp(s2 - s1) == 0) {
5         return false;
6     }
7     p.x = (c.x * s2 - d.x * s1) / (s2 - s1);
8     p.y = (c.y * s2 - d.y * s1) / (s2 - s1);
9     if (dcmp(dmul(p, a, b)) <= 0 && dcmp(dmul(p, c, d)) <= 0) {
10         return true;
11     }

```



```

12     return false;
13 }
14
15 double uno(const P &a, const P &b, const P &c, const P &d) {
16     int flag = dcmp(cross(a, b)) * dcmp(cross(c, d));
17     if (flag == 0) {
18         return 0;
19     }
20     P p[5];
21     int len = 0;
22     int ab = dcmp(cross(a, b));
23     int ba = -ab;
24     int cd = dcmp(cross(c, d));
25     int dc = -cd;
26     int ac = dcmp(cross(a, c));
27     int ca = -ac;
28     int ad = dcmp(cross(a, d));
29     int da = -ad;
30     int bc = dcmp(cross(b, c));
31     int cb = -bc;
32     int bd = dcmp(cross(b, d));
33     int db = -bd;
34
35     if (cd == ca && dc == da || ac == 0 && dmul(a, c) > 0 || ad == 0 && dmul(a,
d) > 0) {
36         if (intersection(c, d, P(0, 0), a, p[len])) {
37             ++len;
38         } else {
39             p[len++] = a;
40         }
41     }
42     if (cd == cb && dc == db || bc == 0 && dmul(b, c) > 0 || bd == 0 && dmul(b,
d) > 0) {
43         if (intersection(c, d, P(0, 0), b, p[len])) {
44             ++len;
45         } else {
46             p[len++] = b;
47         }
48     }
49     if (ab == ac && ba == bc) {
50         if (intersection(a, b, P(0, 0), c, p[len])) {
51             ++len;
52         } else {
53             p[len++] = c;
54         }
55     }
56     if (ab == ad && ba == bd) {

```

```

57         if (intersection(a, b, P(0, 0), d, p[len])) {
58             ++len;
59         } else {
60             p[len++] = d;
61         }
62     }
63     if (intersection(a, b, c, d, p[len])) {
64         ++len;
65     }
66     double ans = 0;
67     if (len == 3) {
68         ans = abs(cross(p[0], p[2])) + abs(cross(p[1], p[2]));
69     } else if (len == 2) {
70         ans = abs(cross(p[0], p[1]));
71     }
72     return flag * abs(ans);
73 }

```

Intersection of Triangle and Circle

```

1 //三角形和圓的交
2 bool cut(const Point &pt1, const Point &pt2, double r, Point &p1, Point &p2)
{
3     Point c = pt2 - pt1;
4     double t = c.dist() * r * r - c.dist() * pt1.dist() + c.dot(pt1) * c.dot(pt1);
5     if (sgn(t) < 0) return false;
6     double t1 = (c.dot(pt1 * (-1)) - sqrt(t)) / c.lenth();
7     double t2 = (c.dot(pt1 * (-1)) + sqrt(t)) / c.lenth();
8     p1 = pt1 + (pt2 - pt1).set(t1);
9     p2 = pt1 + (pt2 - pt1).set(t2);
10    if (sgn((pt1 - p1).dot(pt2 - p1)) < 0) {
11        return true;
12    }
13    return false;
14 }
15 double in(double x) {
16     x = max(x, -1.);
17     x = min(x, 1.);
18     return x;
19 }
20 double shan(const Point &pt1, const Point &pt2, double r) {
21     double alpha = acos(in(pt1.dot(pt2) / (pt1.lenth() * pt2.lenth())));
22     double res = alpha * r * r * 0.5;

```

```

23     return res;
24 }
25 double area(Point pt1, Point pt2, double r) {
26     double s1 = pt1.dist(), s2 = pt2.dist();
27     double r2 = r * r, res;
28     int type = 1;
29     if (sgn(pt1 * pt2) < 0) type = -1;
30     Point p1, p2;
31     if (sgn(s1 - r2) > 0 && sgn(s2 - r2) > 0) {
32         if (!cut(pt1, pt2, r, p1, p2)) {
33             res = shan(pt1, pt2, r);
34         } else {
35             res = shan(pt1, p1, r) + shan(pt2, p2, r) + fabs(p1 * p2) * 0.5;
36         }
37     } else if (sgn(s1 - r2) <= 0 && sgn(s2 - r2) <= 0) {
38         res = fabs(pt1 * pt2) * 0.5;
39     } else {
40         if (sgn(s1 - r2) > 0) {
41             swap(s1, s2);
42             swap(pt1, pt2);
43         }
44         cut(pt2, pt1, r, p2, p1);
45         res = shan(pt2, p2, r) + fabs(pt1 * p2) * 0.5;
46     }
47     return res * type;
48 }

```

Point_in_poly

//点在多边形内，多边形上的点也作为多边形内

```

2 int find(Point t)
3 {
4     return t.x >= 0 ? (t.y >= 0 ? 0 : 3) : (t.y >= 0 ? 1 : 2);
5 }
6
7 bool in_play(Point t, Point *p, int size)
8 {
9     int i;
10    for(i = 0; i <= size; ++i) p[i].x -= t.x, p[i].y -= t.y;
11    int t1, t2;
12    t1 = find(p[0]);

```

```

13    int sum = 0;
14    for(i = 1; i <= size; ++i)
15    {
16        if(fabs(p[i].x) < eps && fabs(p[i].y) < eps) break;
17        double f = p[i - 1] * p[i];
18        if(fabs(f) < eps && p[i - 1].x * p[i].x <= 0 && p[i - 1].y * p[i].y
19        <= 0) break;
20        t2 = find(p[i]);
21        if(t2 == (t1 + 1) % 4) ++sum;
22        else if(t2 == (t1 + 3) % 4) --sum;
23        else if(t2 == (t1 + 2) % 4)
24        {
25            if(f > 0) sum += 2;
26            else sum -= 2;
27        }
28        t1 = t2;
29    }
30    int k = i;
31    for (i = 0; i <= size; ++i) p[i].x += t.x, p[i].y += t.y;
32    if (k <= size) return true;    //点在多边形上
33    return (sum);
34 }

```

Convex_poly_area_union

```

1 //凸多边形面积并
2 bool segcross(const Point &a, const Point &b, const Point &c, const Point &d,
Point &pt) {
3     double s1 = cross(a, b, c), s2 = cross(a, b, d);
4     if (sgn(s1) * sgn(s2) > 0 || sgn(s1 - s2) == 0) return false;
5     pt = (c * s2 - d * s1) / (s2 - s1);
6     if (sgn((a - pt).dot(b - pt)) <= 0) return true;
7     return false;
8 }
9 bool in_poly(const Point &pt, const vector<Point> &a) {
10    for (int i = 0; i < (int)a.size(); ++i) {
11        if (sgn(cross(pt, a[i], a[next(i + 1, a.size())])) <= 0) return false;
12    }
13    return true;
14 }
15 bool in_polygon(const Point &mid, const vector<vector<Point>> &a) {

```

```

16     for (int i = 0; i < (int)a.size(); ++i) {
17         if (in_poly(mid, a[i])) return true;
18     }
19     return false;
20 }
21 double poly_area(const vector<vector<Point> > &a) {
22     vector<vector<Point> > c(a.size());
23     for (int i = 0; i < (int)a.size(); ++i) {
24         for (int j = 0; j < (int)a[i].size(); ++j) {
25             c[i].push_back(a[i][j]);
26             for (int k = 0; k < i; ++k) {
27                 for (int p = 0; p < (int)a[k].size(); ++p) {
28                     Point pt;
29                     if (segcross(a[i][j], a[i][next(j + 1, a[i].size())],
a[k][p], a[k][next(p + 1, a[k].size())], pt)) {
30                         c[i].push_back(pt);
31                         c[k].push_back(pt);
32                     }
33                 }
34             }
35         }
36     }
37     for (int i = 0; i < (int)a.size(); ++i) {
38         ins = Point(0, 0);
39         for (int j = 0; j < (int)a[i].size(); ++j) {
40             ins = ins + a[i][j];
41         }
42         ins = ins / a[i].size();
43         stable_sort(c[i].begin(), c[i].end(), cmp);
44         c[i].erase(unique(c[i].begin(), c[i].end()), c[i].end());
45     }
46     set<pair<Point, Point> > re;
47     double area = 0;
48     for (int i = 0; i < (int)c.size(); ++i) {
49         for (int j = 0; j < (int)c[i].size(); ++j) {
50             Point &p1 = c[i][j], &p2 = c[i][next(j + 1, c[i].size())];
51             if (re.count(make_pair(p1, p2)) == false) {
52                 if (!in_polygon((p1 + p2) * 0.5, a)) {
53                     area += c[i][j] * c[i][next(j + 1, c[i].size())];
54                 }
55                 re.insert(make_pair(p1, p2));
56             }
57         }
58     }

```

```

58     }
59     return fabs(area * 0.5);
60 }
61

```

Circle_area_union

```

1 //圆的面积并
2 const double eps = 1e-9;
3 const double pi = acos(-1.0);
4
5 const int zx[] = {0, 1, 0, -1};
6 const int zy[] = {1, 0, -1, 0};
7 double dist2(const P &a, const P &b) {
8     return SQR(a.x - b.x) + SQR(a.y - b.y);
9 }
10 double dist(const P &a, const P &b) {
11     return sqrt(SQR(a.x - b.x) + SQR(a.y - b.y));
12 }
13 double cross(const P &a, const P &b, const P &c) {
14     return (b.x - a.x) * (c.y - a.y) - (b.y - a.y) * (c.x - a.x);
15 }
16 double dmul(const P &a, const P &b, const P &c) {
17     return (b.x - a.x) * (c.x - a.x) + (b.y - a.y) * (c.y - a.y);
18 }
19 struct C {
20     P mid;
21     double r;
22     C(const P &_mid, const double &_r)
23         :mid(_mid), r(_r) {}
24     C() {}
25     bool operator == (const C &a) const {
26         return mid == a.mid && sgn(r - a.r) == 0;
27     }
28     bool in(const C &a) const {
29         return sgn(r + dist(mid, a.mid) - a.r) <= 0;
30     }
31     const C &input() {
32         mid.input();
33         scanf("%lf", &r);
34         return *this;

```

```

35     }
36     const C &output() const {
37         printf("P: %.12lf %.12lf R: %.12lf\n", mid.x, mid.y, r);
38     }
39 };
40 double cal_angle(const C &c, const P &a, const P &b) {
41     double k = dmul(c.mid, a, b) / SQR(c.r);
42     get_min(k, 1.0);
43     get_max(k, -1.0);
44     return acos(k);
45 }
46 double cal_area(const C &c, const P &a, const P &b) {
47     return SQR(c.r) * cal_angle(c, a, b) / 2 - cross(c.mid, a, b) / 2;
48 }
49 struct cmp {
50     P mid;
51     cmp(const P &_mid)
52         :mid(_mid) {}
53     bool operator () (const P &a, const P &b) {
54         return atan2(a.y - mid.y, a.x - mid.x) < atan2(b.y - mid.y, b.x - mid.x);
55     }
56 };
57 bool circles_intersection(const C &a, const C &b, P &c1, P &c2) {
58     double dd = dist(a.mid, b.mid);
59     if (sgn(dd - (a.r + b.r)) >= 0) {
60         return false;
61     }
62     double l = (dd + (SQR(a.r) - SQR(b.r)) / dd) / 2;
63     double h = sqrt(SQR(a.r) - SQR(l));
64     c1 = a.mid + (b.mid - a.mid).trunc(l) + (b.mid -
a.mid).turn_left().trunc(h);
65     c2 = a.mid + (b.mid - a.mid).trunc(l) + (b.mid -
a.mid).turn_right().trunc(h);
66     return true;
67 }
68 bool cover(const C &c, const P &a, const P &b, const vector<C> &cir) {
69     P p = c.mid + ((a + b) / 2 - c.mid).trunc(c.r);
70     for (vector<C>::const_iterator it = cir.begin(); it != cir.end(); ++it)
71     {
72         if (sgn(dist2(p, it->mid) - SQR(it->r)) < 0) {
73             return true;
74         }
75     }

```

```

75     return false;
76 }
77 double cal_area(const vector<C> &in) {
78     vector<C> cir;
79     for (int i = 0; i < SZ(in); ++i) {
80         if (sgn(in[i].r) == 0) {
81             continue;
82         }
83         bool flag = false;
84         for (int j = i + 1; j < SZ(in); ++j) {
85             if (in[i] == in[j]) {
86                 flag = true;
87                 break;
88             }
89         }
90         if (flag) {
91             continue;
92         }
93         for (int j = 0; j < SZ(in); ++j) {
94             if (!in[i] == in[j] && in[i].in(in[j])) {
95                 flag = true;
96                 break;
97             }
98         }
99         if (flag) {
100             continue;
101         }
102         cir.push_back(in[i]);
103     }
104     vector<vector<P> > point_on_circle(SZ(cir));
105     for (int i = 0; i < SZ(cir); ++i) {
106         for (int z = 0; z < 4; ++z) {
107             point_on_circle[i].push_back(cir[i].mid + P(zx[z],
zy[z]).trunc(cir[i].r));
108         }
109     }
110     for (int i = 0; i < SZ(cir); ++i) {
111         for (int j = i + 1; j < SZ(cir); ++j) {
112             P a, b;
113             if (circles_intersection(cir[i], cir[j], a, b)) {
114                 point_on_circle[i].push_back(a);
115                 point_on_circle[i].push_back(b);
116                 point_on_circle[j].push_back(a);

```

```

117         point_on_circle[j].push_back(b);
118     }
119 }
120 }
121 for (int i = 0; i < SZ(cir); ++i) {
122     sort(point_on_circle[i].begin(), point_on_circle[i].end(),
cmp(cir[i].mid));
123     point_on_circle[i].erase(unique(point_on_circle[i].begin(),
point_on_circle[i].end()), point_on_circle[i].end());
124 }
125 double ans = 0;
126 for (int i = 0; i < SZ(cir); ++i) {
127     for (int j = 0; j < SZ(point_on_circle[i]); ++j) {
128         const P &a = point_on_circle[i][j];
129         const P &b = point_on_circle[i][NEXT(j) + 1,
SZ(point_on_circle[i])];
130         if (!cover(cir[i], a, b, cir)) {
131             ans += cross(P(0, 0), a, b) / 2;
132             ans += cal_area(cir[i], a, b);
133         }
134     }
135 }
136 return ans;
137 }
138

```

Plane_cross

```

1 //半平面交 nlgn
2 const double eps = 1e-8;
3 const int maxn = 20000 + 10;
4
5 double cross(const Point &a, const Point &b, const Point &c) {
6     return (b - a) * (c - a);
7 }
8
9 struct Line {
10     Point p1, p2;
11     double angle;
12     void get_angle() {
13         Point p = p2 - p1;

```

```

14         angle = atan2(p.y, p.x);
15         if (p.y < 0) angle += pi * 2;
16     }
17     Line() {}
18     Line(const Point &p1, const Point &p2): p1(p1), p2(p2) {
19         Point p = p2 - p1;
20         get_angle();
21     }
22     bool operator < (const Line &a) const {
23         return sgn(angle - a.angle) < 0 || sgn(angle - a.angle) == 0 &&
sgn(cross(a.p1, a.p2, p1)) > 0;
24     }
25 };
26
27 Line line[maxn], ln[maxn];
28 Point pt1[maxn], pt2[maxn];
29 int len1, len2, n;
30
31 Point lncross(const Point &a, const Point &b, const Point &c, const Point &d)
{
32     double s1 = cross(a, b, c);
33     double s2 = cross(a, b, d);
34     return (c * s2 - d * s1) / (s2 - s1);
35 }
36
37 int segcross(const Point &a, const Point &b, const Point &c, const Point &d,
Point &pt) {
38     pt = lncross(a, b, c, d);
39     int d1 = sgn((a - pt).dot(b - pt)), d2 = sgn((c - pt).dot(d - pt));
40     if (d1 < 0 && d2 < 0) return 0;
41     if (pt == a && d2 <= 0 || pt == c && d1 <= 0) return 0;
42     if (sgn(cross(c, d, a)) > 0 && sgn(cross(c, d, b)) >= 0) return 1;
43     return -1;
44 }
45
46 double cal(Point *pt1, Point *pt2, int len1, int len2) {
47     int p1 = 0, p2 = len2, tmp;
48     Point pp1, pp2;
49     while (p1 <= len1 && p2 >= 0 && (tmp = segcross(pt1[p1], pt1[p1 + 1], pt2[p2],
pt2[p2 - 1], pp1))) {
50         if (tmp > 0) ++p1;
51         else --p2;
52     }

```

```

53     int p3 = len1, p4 = 0;
54     while (p3 >= 0 && p4 <= len2 && (tmp = segcross(pt2[p4], pt2[p4 + 1], pt1[p3],
pt1[p3 - 1], pp2))) {
55         if (tmp > 0) ++p4;
56         else --p3;
57     }
58     int len = 0;
59     Point cv[maxn];
60     cv[len++] = pp1;
61     for (int i = p1 + 1; i < p3; ++i) cv[len++] = pt1[i];
62     cv[len++] = pp2;
63     for (int i = p4 + 1; i < p2; ++i) cv[len++] = pt2[i];
64     cv[len] = cv[0];
65     double sum = 0;
66     for (int i = 0; i < len; ++i) sum += cv[i] * cv[i + 1];
67     return fabs(sum * 0.5);
68 }
69
70 void init(Point *pt, int &len, Line *line, int n, bool flag) {
71     int i;
72     ln[len = 0] = line[0];
73     pt[0] = line[0].p1 + (line[0].p1 - line[0].p2) * 1e10;
74     for (i = 1; i < n; ++i) {
75         Point p;
76         while (len > 0) {
77             p = lncross(line[i].p1, line[i].p2, ln[len].p1, ln[len].p2);
78             if (flag && sgn(p.y - pt[len].y) <= 0 || !flag && sgn(p.y -
pt[len].y) >= 0) --len;
79             else break;
80         }
81         ln[++len] = line[i];
82         pt[len] = lncross(line[i].p1, line[i].p2, ln[len - 1].p1, ln[len -
1].p2);
83     }
84     pt[++len] = line[i - 1].p1 + (line[i - 1].p2 - line[i - 1].p1) * 1e10;
85 }
86
87 double solve() {
88     int cut;
89     for (int i = 1; i < n; ++i) {
90         if (sgn(line[i].angle - pi) >= 0) {
91             cut = i;
92             break;

```

```

93     }
94 }
95 init(pt1, len1, line, cut, 1);
96 init(pt2, len2, line + cut, n - cut, 0);
97 return cal(pt1, pt2, len1, len2);
98 }
99
100 bool get_input() {
101     if (scanf("%d", &n) == EOF) return false;
102     for (int i = 0; i < n; ++i) {
103         line[i].p1.input();
104         line[i].p2.input();
105         line[i].get_angle();
106     }
107     line[n++] = Line(Point(0, 0), Point(10000, 0));
108     line[n++] = Line(Point(10000, 0), Point(10000, 10000));
109     line[n++] = Line(Point(10000, 10000), Point(0, 10000));
110     line[n++] = Line(Point(0, 10000), Point(0, 0));
111     sort(line, line + n);
112     int nn = n;
113     n = 0;
114     for (int i = 0; i < nn; ++i) {
115         if (i && sgn(line[i].angle - line[i - 1].angle) == 0) continue;
116         line[n++] = line[i];
117     }
118     return true;
119 }
120

```

Others

Extended_GCD

```

1 int egcd(int a, int b, int &x, int &y) {
2     int t, d;
3     if (b == 0) return x = 1, y = 0, a;
4     d = egcd(b, a % b, x, y);
5     t = x; x = y; y = t - a / b * y;
6     return d;
7 }

```

```

1 long long mul(long long a, long long b, long long c) {
2     long long ans = 0;
3     for (int i = 62; i >= 0; --i) {
4         ans <=<= 1;
5         if (ans >= c) {
6             ans -= c;
7         }
8         if ((a >> i) & 1) {
9             ans += b;
10            if (ans >= c) {
11                ans -= c;
12            }
13        }
14    }
15    return ans;
16 }
17 bool check(long long a, long long b, long long c) {
18     long long ans = 1;
19     for (; b; b >>= 1) {
20         if (b & 1) {
21             ans = mul(ans, a, c);
22         }
23         long long t = mul(a, a, c);
24         if (t == 1 && a != 1 && a != c - 1) {
25             return false;
26         }
27         a = t;
28     }
29     return ans == 1;
30 }
31 bool isPrime(long long a) {
32     if (a % 2 == 0 || a % 3 == 0 || a % 7 == 0 || a % 11 == 0) {
33         return false;
34     }
35     for (int i = 0; i < 10; ++i) {
36         if (!check(rand() % (a - 3) + 2, a - 1, a)) {
37             return false;
38         }
39     }
40     return true;
41 }

```

```

1 const int base = 10000;
2 const int cap = 200;
3 struct xnum {
4     int len;
5     int data[cap];
6     xnum() : len(0) {}
7     xnum(const xnum& v) : len(v.len) { memcpy(data, v.data, len *
sizeof *data); }
8     xnum(int v) : len(0) { for (; v > 0; v /= base) data[len++] = v % base; }
9     xnum& operator=(const xnum& v) { len = v.len; memcpy(data, v.data, len
* sizeof *data); return *this; }
10    int& operator[](int index) { return data[index]; }
11    int operator[](int index) const { return data[index]; }
12 };
13 int compare(const xnum& a, const xnum& b) {
14     int i;
15     if (a.len != b.len) return a.len > b.len ? 1 : -1;
16     for (i = a.len - 1; i >= 0 && a[i] == b[i]; i--);
17     if (i < 0) return 0;
18     return a[i] > b[i] ? 1 : -1;
19 }
20 xnum operator+(const xnum& a, const xnum& b) {
21     xnum r;
22     int i, c = 0;
23     for (i = 0; i < a.len || i < b.len || c > 0; i++) {
24         if (i < a.len) c += a[i];
25         if (i < b.len) c += b[i];
26         r[i] = c % base;
27         c /= base;
28     }
29     r.len = i;
30     return r;
31 }
32 xnum operator-(const xnum& a, const xnum& b) {
33     xnum r;
34     int c = 0;
35     r.len = a.len;
36     for (int i = 0; i < r.len; i++) {
37         r[i] = a[i] - c;
38         if (i < b.len) r[i] -= b[i];
39         if (r[i] < 0) c = 1, r[i] += base;
40         else c = 0;
41     }
42     while (r.len > 0 && r[r.len - 1] == 0) r.len--;

```

```

43     return r;
44 }
45 xnum operator*(const xnum& a, const int b) {
46     int i;
47     if (b == 0) return 0;
48     xnum r;
49     int c = 0;
50     for (i = 0; i < a.len || c > 0; i++) {
51         if (i < a.len) c += a[i] * b;
52         r[i] = c % base;
53         c /= base;
54     }
55     r.len = i;
56     return r;
57 }
58 xnum operator*(const xnum& a, const xnum& b) {
59     if (b.len == 0) return 0;
60     xnum r;
61     for (int i = 0; i < a.len; i++) {
62         int c = 0;
63         for (int j = 0; j < b.len || c > 0; j++) {
64             if (j < b.len) c += a[i] * b[j];
65             if (i + j < r.len) c += r[i + j];
66             if (i + j >= r.len) r[r.len++] = c % base;
67             else r[i + j] = c % base;
68             c /= base;
69         }
70     }
71     return r;
72 }
73 xnum operator/(const xnum& a, const int b) {
74     xnum r;
75     int c = 0;
76     for (int i = a.len - 1; i >= 0; i--) {
77         c = c * base + a[i];
78         r[i] = c / b;
79         c %= b;
80     }
81     r.len = a.len;
82     while (r.len > 0 && r[r.len - 1] == 0) r.len--;
83     return r;
84 }
85 xnum operator/(const xnum& a, const xnum& b)
86 {
87     xnum r, c = 0;
88     int left, right, mid;
89     for (int i = a.len - 1; i >= 0; i--) {

```

```

90         c = c * base + a[i];
91         left = 0;
92         right = base - 1;
93         while (left < right) {
94             mid = (left + right + 1) / 2;
95             if (compare(b * mid, c) <= 0) left = mid;
96             else right = mid - 1;
97         }
98         r[i] = left;
99         c = c - b * left;
100     }
101     r.len = a.len;
102     while (r.len > 0 && r[r.len - 1] == 0) r.len--;
103     return r;
104 }
105 xnum operator%(const xnum& a, const xnum& b) {
106     xnum r, c = 0;
107     int left, right, mid;
108     for (int i = a.len - 1; i >= 0; i--) {
109         c = c * base + a[i];
110         left = 0;
111         right = base - 1;
112         while (left < right) {
113             mid = (left + right + 1) / 2;
114             if (compare(b * mid, c) <= 0) left = mid;
115             else right = mid - 1;
116         }
117         r[i] = left;
118         c = c - b * left;
119     }
120     r.len = a.len;
121     while (r.len > 0 && r[r.len - 1] == 0) r.len--;
122     return c;
123 }
124 istream& operator>>(istream& in, xnum& v) {
125     char ch;
126     for (v = 0; in >> ch;) {
127         v = v * 10 + (ch - '0');
128         if (cin.peek() <= ' ') break;
129     }
130     return in;
131 }
132 ostream& operator<<(ostream& out, const xnum& v) {
133     out << (v.len == 0 ? 0 : v[v.len - 1]);
134     for (int i = v.len - 2; i >= 0; i--) for (int j = base / 10; j > 0; j /=
135         10) out << v[i] / j % 10;
136     return out;

```


136 }

Vimrc

```
1 source $VIMRUNTIME/mswin.vim
2 behave mswin
3 imap <cr> <cr><left><right>
4 imap <c-j> {<cr>}<c-o>O<left><right>
5 imap <c-d> <c-o>dd
6 map <f6> =a{
7 map <c-t> :tabnew<cr>
8 syn on
9 colo torte
10 set gfn=Courier\ 10\ Pitch\ 12
11 set ru nu et sta nowrap ar acd ww=<,>,[,] sw=4 ts=4 cin noswf
12
13 map <f10> :call CR2()<cr><space>
14 func CR2()
15 exec "update"
16 exec "!xterm -fn 10*20 -e \"g++ %<.cpp -Wall -o %< && time ./%< ; read -n 1\"
17 endfunc
18 map <f9> :call CR()<cr><space>
19 func CR()
20 exec "update"
21 exec "!xterm -fn 10*20 -e \"g++ %<.cpp -Wall -o %< && time ./%< < %<.in ; read
-n 1\"
22 endfunc
23
24 map<f4> :call AddComment()<cr>
25 func AddComment()
26     if (getline('.')[0] == '/')
27         normal ^xx
28     else
29         normal Oi//
30     endif
31 endfunc
```