ACM/ICPC at Wuhan University

FreeRadiant STL

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

目录

Graph

Maxflow

```
struct Graph {
    struct Adj {
        int v, c, b;
        Adj(int _v, int _c, int _b):
            v(_v), c(_c), b(_b) {}
        Adj(){}
    };
    int n, S, T, h[maxn], cnt[maxn];
    vector<Adj> adj[maxn];
    void clear() {
        for (int i = 0; i < n; ++i) {
            adj[i].clear();
        }
        n = 0;
    }
    void insert(int u, int v, int c, int d = 0) {
        get_max(n, max(u, v) + 1);
        adj[u].push_back(Adj(v, c, adj[v].size()));
        adj[v].push_back(Adj(u, c * d, adj[u].size() - 1));
    }
    int maxflow(int _S, int _T) {
        S = _S, T = _T;
        fill(h, h + n, 0);
        fill(cnt, cnt + n, 0);
        int flow = 0;
        while (h[S] < n) {
            flow += dfs(S, maxint);
        }
        return flow;
    }
    int dfs(int u, int flow) {
        if (u == T) {
            return flow;
        }
        int minh = n - 1, ct = 0;
        for (vector<Adj>::iterator it = adj[u].begin(); flow && it !=
adj[u].end(); ++it) {
            if (it->c) {
                if (h[it->v] + 1 == h[u]) {
                    int k = dfs(it->v, min(it->c, flow));
                    if (k) {
                        it->c -= k;
```

```
                        adj[it->v][it->b].c += k;
                        flow -= k;
                        ct += k;
                    }
                    if (h[S] >= n) {
                        return ct;
                    }
                }
                get_min(minh, h[it->v]);
            }
        }
        if (ct) {
            return ct;
        }
        if (--cnt[h[u]] == 0) {
            h[S] = n;
        }
        h[u] = minh + 1;
        ++cnt[h[u]];
        return 0;
    }
};
```

MinCostMaxFlow

```
struct Graph{
    struct Adj {
        int v, c, w, b;
        Adj(int _v, int _c, int _w, int _b):v(_v), c(_c), w(_w), b(_b) {};
    }*st[maxn];
    vector<Adj> adj[maxn];
    int n;
    void clear() {
        for (int i = 0; i < n; ++i) {
            adj[i].clear();
        }
        n = 0;
    }
    void insert(int u, int v, int c, int w, int d = 0) {
        get_max(n, max(u, v) + 1);
        adj[u].push_back(Adj(v, c, w, adj[v].size()));
        adj[v].push_back(Adj(u, 0, -w, adj[u].size() - 1));
        if (d) {
            adj[v].push_back(Adj(u, c, w, adj[u].size()));
            adj[u].push_back(Adj(v, 0, -w, adj[v].size() - 1));
        }
    }
    pair<int, int> mcmf(int S, int T) {
        int d;
        int flow = 0, cost = 0;
        while ((d = bell(S, T))) {
            flow += d;
            for (int v = T; v != S; v = adj[st[v]->v][st[v]->b].v) {
                cost += st[v]->w * d;
                st[v]->c -= d;
                adj[st[v]->v][st[v]->b].c += d;
            }
        }
        return make_pair(flow, cost);
    }
    int bell(int S, int T) {
        int d[maxn], bfs[maxn], hash[maxn];
        fill(hash, hash + n, 0);
        fill(d, d + n, maxint);
        hash[S] = 1; d[S] = 0; bfs[0] = S;
        for (int s = 0, t = 1; s != t; hash[bfs[s]] = 0, s = NEXT(s + 1, n))
{
            int v = bfs[s];
            for (vector<Adj>::iterator it = adj[v].begin(); it != adj[v].end();
++it) {
                if (it->c > 0 && d[v] + it->w < d[it->v]) {
                    d[it->v] = d[v] + it->w;
                    st[it->v] = &(*it);
                    if (hash[it->v] == 0) {
                        hash[it->v] = 1;
                        bfs[t] = it->v;
                        t = NEXT(t + 1, n);
                    }
                }
            }
        }
        if(d[T] == maxint) {
            return 0;
        }
        int ans = maxint;
        for (int v = T; v != S; v = adj[st[v]->v][st[v]->b].v) {
            get_min(ans, st[v]->c);
        }
        return ans;
    }
};
```

NonbipartiteMaxMatch

```
struct Graph {
    int n, match[maxn];
    bool adj[maxn][maxn];
    void clear() {
        memset(adj, 0, sizeof(adj));
        n = 0;
    }
    void insert(const int &u, const int &v) {
        get_max(n, max(u, v) + 1);
        adj[u][v] = adj[v][u] = 1;
    }
    int max_match() {
        memset(match, -1, sizeof(match));
        int ans = 0;
        for (int i = 0; i < n; ++i) {
            if (match[i] == -1) {
                ans += bfs(i);
            }
        }
        return ans;
    }
    int Q[maxn], pre[maxn], base[maxn];
    bool hash[maxn];
    bool in_blossom[maxn];
    int bfs(int p) {
        memset(pre, -1, sizeof(pre));
        memset(hash, 0, sizeof(hash));
        for (int i = 0; i < n; ++i) {
            base[i] = i;
        }
        Q[0] = p;
        hash[p] = 1;
        for (int s = 0, t = 1; s < t; ++s) {
            int u = Q[s];
            for (int v = 0; v < n; ++v) {
                if (adj[u][v] && base[u] != base[v] && v != match[u]) {
                    if (v == p || (match[v] != -1 && pre[match[v]] != -1))
{
                        int b = contract(u, v);
                        for (int i = 0; i < n; ++i) {
                            if (in_blossom[base[i]]) {
                                base[i] = b;
                                if (hash[i] == 0) {
                                    hash[i] = 1;
                                    Q[t++] = i;
                                }
                            }
                        }
                    } else if (pre[v] == -1) {
                        pre[v] = u;
                        if (match[v] == -1) {
                            argument(v);
                            return 1;
                        } else {
                            Q[t++] = match[v];
                            hash[match[v]] = 1;
                        }
                    }
                }
            }
        }
        return 0;
    }
    void argument(int u) {
        while (u != -1) {
            int v = pre[u];
            int k = match[v];
            match[u] = v;
            match[v] = u;
            u = k;
        }
    }
    void change_blossom(int b, int u) {
        while (base[u] != b) {
            int v = match[u];
            in_blossom[base[v]] = in_blossom[base[u]] = true;
            u = pre[v];
            if (base[u] != b) {
                pre[u] = v;
            }
        }
    }
    int contract(int u, int v) {
        memset(in_blossom, 0, sizeof(in_blossom));
        int b = find_base(base[u], base[v]);
        change_blossom(b, u);
        change_blossom(b, v);
        if (base[u] != b) {
            pre[u] = v;
        }
        if (base[v] != b) {
```

```
            pre[v] = u;
        }
        return b;
    }
    int find_base(int u, int v) {
        bool in_path[maxn] = {};
        while (true) {
            in_path[u] = true;
            if (match[u] == -1) {
                break;
            }
            u = base[pre[match[u]]];
        }
        while (!in_path[v]) {
            v = base[pre[match[v]]];
        }
        return v;
    }
};
```

KM

```
struct Graph {
    int w[maxn][maxn], lx[maxn], ly[maxn], matx[maxn], maty[maxn], n;
    bool fx[maxn], fy[maxn];
    void clear() {
        memset(w, 0, sizeof(w));
        n = 0;
    }
    void insert(int u, int v, int c) {
        get_max(n, max(u + 1, v + 1));
        w[u][v] = c;
    }
    int match() {
        memset(ly, 0, sizeof(ly));
        for (int i = 0; i < n; ++i) {
            lx[i] = -maxint;
            for (int j = 0; j < n; ++j) {
                get_max(lx[i], w[i][j]);
            }
        }
        memset(matx, -1, sizeof(matx));
        memset(maty, -1, sizeof(maty));
        for (int i = 0; i < n; ++i) {
            memset(fx, false, sizeof(fx));
            memset(fy, false, sizeof(fy));
            if (!dfs(i)) {
                --i;
                int p = maxint;
                for (int k = 0; k < n; ++k) {
                    if (fx[k] == true) {
                        for (int j = 0; j < n; ++j) {
                            if ((fy[j] == false)) {
                                get_min(p, lx[k] + ly[j] - w[k][j]);
                            }
                        }
                    }
                }
                for (int j = 0; j < n; ++j) {
                    ly[j] += fy[j] * p;
                }
                for (int k = 0; k < n; ++k) {
                    lx[k] -= fx[k] * p;
                }
            }
        }
        int ans = 0;
        for (int i = 0; i < n; ++i) {
            ans += w[maty[i]][i];
        }
        return ans;
    }
    bool dfs(int u) {
        fx[u] = 1;
        for (int v = 0; v < n; ++v) {
            if (lx[u] + ly[v] == w[u][v] && fy[v] == false) {
                fy[v] = true;
                if (maty[v] == -1 || dfs(maty[v])) {
                    matx[u] = v;
                    maty[v] = u;
                    return true;
                }
            }
        }
        return false;
    }
};
```

无源无汇有上下界网络流判定可行流存在

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

```
    que[0] = tt;
    level[tt] = n;
    for (int h = 0, t = 0; h <= t; h ++) {
        int u = que[h];
        for (int i = 0; i < edge[u].size(); i ++)
            if (level[edge[u][i].v] == -1 &&
edge[edge[u][i].v][edge[u][i].b].f > 0) {
                level[edge[u][i].v] = level[u] - 1;
                t ++;
                que[t] = edge[u][i].v;
            }
    }
    return (level[ss] != -1);
}
void            solve()
{
    if (dinic(0, n - 1) != s) {
        printf("NO\n");
        return ;
    }
    for (int i = 1; i < n - 1; i ++)
        for (int j = 0; j < edge[i].size(); j ++)
            if (edge[i][j].index > -1)
                ans[edge[i][j].index] += cap[i][j] - edge[i][j].f;
}
```

N^3 MinCut

```
int n, m;
int adj[210][210];
int f[210], bestf[210];
int find(int v) {
    if (v != bestf[v]) {
        return bestf[v] = find(bestf[v]);
    }
    return v;
}
int bestff;
int v[210], w[210];
bool a[210];
vector<pair<int, int> > res;
int mincut() {
    int ans = maxint;
    for (int i = 0; i < n; ++i) {
        for (int j = 0; j < n; ++j) {
            adj[i][j] = -adj[i][j];
        }
    }
    for (int i = 0; i < n; ++i) {
        f[i] = v[i] = i;
    }
    for (int t = n; t > 1; --t) {
        a[0] = true;
        for (int i = 1; i < t; ++i) {
            a[i] = false;
            w[i] = adj[v[0]][v[i]];
        }
        int prev = v[0];
        for (int i = 1; i < t; ++i) {
            int zj = -1;
            for (int j = 1; j < t; ++j) {
                if (a[j] == false && (zj == -1 || w[j] > w[zj])) {
                    zj = j;
                }
            }
            a[zj] = true;
            if (i == t - 1) {
                if (w[zj] < ans) {
                    ans = w[zj];
                    copy(f, f + n, bestf);
                    bestff = v[zj];
                }
                for (int k = 0; k < t; ++k) {
                    adj[v[k]][prev] = adj[prev][v[k]] += adj[v[zj]][v[k]];
                }
                f[v[zj]] = prev;
                v[zj] = v[t - 1];
                break;
            }
            prev = v[zj];
            for (int j = 1; j < t; ++j) {
                if (a[j] == false) {
                    w[j] += adj[v[zj]][v[j]];
                }
            }
        }
    }
    bool flag[210] = {};
    for (int i = 0; i < n; ++i) {
        if (find(i) == bestff) {
            flag[i] = true;
        }
```

```
    }
    res.clear();
    for (int i = 0; i < n; ++i) {
        for (int j = i + 1; j < n; ++j) {
            if (flag[i] == flag[j] && save[i][j]) {
                res.push_back(make_pair(i, j));
            }
        }
    }
    return ans;
}
```

最小树形图

```
//最小树形图 这个版本求的是欧几里德最小树形图,根为 1
#include <stdio.h>
#include <string.h>
#include <math.h>
const int MAXV=101,MAXE=10001;
const double MAXDOUBLE=1e20;
double gh[MAXV][MAXV],//原始有向图,顶点从 1 开始编号
    mincost[MAXV],ans;
int prev[MAXV],stack[MAXV],id[MAXV],mark[MAXV],numv,nume,cnt,top,sta,
    x[MAXV],y[MAXV];//顶点的 x,y 坐标
bool scaned[MAXV];
void Combine() {
    int i,j,now,x;
    double t;
    now=stack[sta];
    for (i=sta;i<top;i++) {
        x=stack[i];
        ans+=mincost[x];
        id[x]=now;
        for (j=1;j<=numv;j++) {
            if (gh[j][x]!=-1) {
                t=gh[j][x]-mincost[x];
                if (t<gh[j][now] || gh[j][now]==-1) gh[j][now]=t;
            }
            if ((gh[x][j]!=-1 && gh[x][j]<gh[now][j]) || gh[now][j]==-1)
gh[now][j]=gh[x][j];
        }
    }
    for (i=2;i<=numv;i++) if (id[i]==i) prev[i]=id[prev[i]];
    mincost[now]=MAXDOUBLE;
    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;
}
bool Find_Circle() {
    int i,temp,mark[MAXV];
    memset(scaned,0,sizeof(scaned));
    for (i=numv;i>1;i--) {
        if (id[i]!=i || scaned[i]) continue;
        memset(mark,0,sizeof(mark));
        top=1;
        temp=i;
        while (temp!=1 && !mark[temp] && !scaned[temp]) {
            scaned[temp]=1;
            mark[temp]=top;
            stack[top++]=temp;
            temp=prev[temp];
        }
        if (mark[temp]) {
            sta=mark[temp];
            return 1;
        }
    }
    return 0;
}
void DFS(int v) {
    cnt++;
    scaned[v]=1;
    for (int i=1;i<=numv;i++) if (!scaned[i] && gh[v][i]!=-1) DFS(i);
}
void Minimum_Arborescence () {
    while (Find_Circle()) Combine();
    for (int i=2;i<=numv;i++) if (i==id[i]) ans+=mincost[i];
}
int main () {
    int i,j,v1,v2,val;
    while (scanf("%d%d",&numv,&nume)!=EOF) {
        for (i=1;i<=numv;i++) for (j=1;j<=numv;j++) gh[i][j]=-1;
        memset(scaned,0,sizeof(scaned));
        for (i=1;i<=numv;i++) scanf("%d%d",&x[i],&y[i]);
        for (i=0;i<nume;i++) {
            scanf("%d%d",&v1,&v2);
            gh[v1][v2]=sqrt((x[v1]-x[v2])*(x[v1]-x[v2])+(y[v1]-y[v2])*(y[v
1]-y[v2]));
        }
        cnt=0;
        DFS(1);
        if (cnt<numv) printf("poor snoopy\n");//最小树形图不存在
        else {
            id[1]=1;
```

```
            for (i=2;i<=numv;i++) {
                id[i]=i;
                mincost[i]=MAXDOUBLE;
                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;
            }
            ans=0;
            Minimum_Arborescence();
            printf("%.2lf\n",ans);//答案保留两位小数
        }
    }
    return 0;
}
```

Gabow

```
int sc[maxn],pre[maxn],path[maxn],s[maxn];
int cnt0,cnt1,lens,lenp;
void dfs(int w)
{
    pre[w]=cnt0++;
    s[lens++]=w;    path[lenp++]=w;
    for(_adj*pt=adj[w];pt;pt=pt->next)
        if(pre[pt->v]==-1)dfs(pt->v);
        else
        if(sc[pt->v]==-1)
            while(pre[path[lenp-1]]>pre[pt->v])lenp--;
    if(path[lenp-1]!=w)return ;
    else lenp--;
    do sc[s[--lens]]=cnt1; while(s[lens]!=w);
    cnt1++;
}
void memset()
{
    memset(adj,0,sizeof(adj));
    memset(sc,-1,sizeof(sc));
    memset(pre,-1,sizeof(pre));
    lens=lenp=0;
    cnt0=0;
    cnt1=0;
}
```

Control

```
class Graph {
public:
    static const int maxn = 2600 / 2;
    static const int maxm = 51;
    static const int mmax = maxn * maxm + maxn + maxm + 100;
    int n, m;
    int adj[maxn][maxm];
    void init() {
        n = m = 0;
        memset(adj, 0, sizeof(adj));
    }
    void insert(int u, int v) {
        get_max(n, u + 1);
        get_max(m, v + 1);
        adj[u][v] = 1;
    }
    int find_ans() {
        build_dlx();
        for (int ans = 0; ans <= n; ++ans) {
            if (dfs(0, ans)) {
                return ans;
            }
        }
        return -1;
    }
private:
    int head;
    int U[mmax], D[mmax], L[mmax], R[mmax],
        CN[mmax], RN[mmax];
    void addUD(const int &a, const int &h) {
        U[a] = h;
        D[a] = D[h];
        U[D[h]] = a;
        D[h] = a;
        CN[a] = h;
    }
    void addLR(const int &a, const int &h) {
        L[a] = h;
        R[a] = R[h];
        L[R[h]] = a;
        R[h] = a;
        RN[a] = h;
    }
    void add(const int &k, const int &r, const int &c) {
        addUD(k, c);
        addLR(k, r);
    }
    void remove(const int &k) {
```

```
            for (int j = R[k]; j != k; j = R[j]) {
                for (int i = D[j]; i != j; i = D[i]) {
                    L[R[i]] = L[i];
                    R[L[i]] = R[i];
                }
                D[U[j]] = D[j];
                U[D[j]] = U[j];
            }
        }
        void unremove(const int &k) {
            for (int j = L[k]; j != k; j = L[j]) {
                D[U[j]] = j;
                U[D[j]] = j;
                for (int i = U[j]; i != j; i = U[i]) {
                    L[R[i]] = i;
                    R[L[i]] = i;
                }
            }
        }
        void build_dlx() {
            head = mmax - 1;
            U[head] = D[head] = L[head] = R[head] = head;
            int cnt = 0;
            for (int i = 0; i < m; ++i) {
                U[cnt] = D[cnt] = cnt;
                addLR(cnt++, head);
            }
            for (int i = 0; i < n; ++i) {
                L[cnt] = R[cnt] = cnt;
                addUD(cnt++, head);
            }
            for (int i = 0; i < n; ++i) {
                for (int j = 0; j < m; ++j) {
                    if (adj[i][j]) {
                        addLR(cnt, m + i);
                        addUD(cnt, j);
                        cnt++;
                    }
                }
            }

        }
        int h() {
            int hash[maxm] = {};
            int ans = 0;
            for (int c = R[head]; c != head; c = R[c]) {
                if (hash[c] == 0) {
                    hash[c] = 1;
                    ++ans;
                    for (int j = D[c]; j != c; j = D[j]) {
                        for (int i = R[j]; i != j; i = R[i]) {
                            if (CN[i] != head) {
                                hash[CN[i]] = 1;
                            }
                        }
                    }
                }
            }
            return ans;
        }
        bool dfs(int k, int lim) {
            if (k + h() > lim) {
                return false;
            }
            if (R[head] == head) {
                return true;
            }
            int c;
            c = R[head];
            L[R[c]] = L[c];
            R[L[c]] = R[c];
            for (int i = D[c]; i != c; i = D[i]) {
                L[R[i]] = L[i];
                R[L[i]] = R[i];
            }
            for (int i = D[c]; i != c; i = D[i]) {
                remove(RN[i]);
                if (dfs(k + 1, lim)) {
                    return true;
                }
                unremove(RN[i]);
            }
            for (int i = U[c]; i != c; i = U[i]) {
                L[R[i]] = i;
                R[L[i]] = i;
            }
            L[R[c]] = c;
            R[L[c]] = c;
            return false;
        }
};
```

Splay

```cpp
const int MaxN = 200000 + 1;

struct treeNode {
    int l, r, p, s;
    int Key, Min, Add;
    bool Reverse;
};

treeNode tree[MaxN];
int N, M, Root;

inline void Pass(int t) {
    int l = tree[t].l, r = tree[t].r;
    if (tree[t].Add) {
        tree[l].Add += tree[t].Add;
        tree[r].Add += tree[t].Add;
        tree[t].Key += tree[t].Add;
        tree[t].Min += tree[t].Add;
        tree[t].Add = 0;
    }
    if (tree[t].Reverse) {
        swap(tree[t].l, tree[t].r);
        tree[l].Reverse ^= 1;
        tree[r].Reverse ^= 1;
        tree[t].Reverse = false;
    }
}

inline void Update(int t) {
    tree[t].Min = tree[t].Key;
    int l = tree[t].l, r = tree[t].r;
    if (l) tree[t].Min = min(tree[t].Min, tree[l].Min + tree[l].Add);
    if (r) tree[t].Min = min(tree[t].Min, tree[r].Min + tree[r].Add);
    tree[t].Min += tree[t].Add;
    tree[t].s = tree[l].s + tree[r].s + 1;
}

inline void Left(int x) {
    int y = tree[x].p;
    int z = tree[y].p;
    if (tree[z].l == y) tree[z].l = x;
    else tree[z].r = x;
    tree[x].p = z;
    tree[y].p = x;
    tree[y].r = tree[x].l;
    tree[x].l = y;
    tree[tree[y].r].p = y;
    Update(y);
}

inline void Right(int x) {
    int y = tree[x].p;
    int z = tree[y].p;
    if (tree[z].l == y) tree[z].l = x;
    else tree[z].r = x;
    tree[x].p = z;
    tree[y].p = x;
    tree[y].l = tree[x].r;
    tree[x].r = y;
    tree[tree[y].l].p = y;
    Update(y);
}

int stack[MaxN];

inline void Splay(int x) {
    int top = 0;
    for (int t = x; t; t = tree[t].p)
        stack[++top] = t;
    for (int i = top; i >= 1; --i) Pass(stack[i]);
    while (tree[x].p) {
        int y = tree[x].p;
        int z = tree[y].p;
        if (z) {
            if (tree[z].l == y) {
                if (tree[y].l == x) {
                    Right(y); Right(x);
                }
                else {
                    Left(x); Right(x);
                }
```

```
            }
            else {
                if (tree[y].l == x) {
                    Right(x); Left(x);
                }
                else {
                    Left(y); Left(x);
                }
            }
        }
        else {
            if (tree[y].l == x) Right(x);
            else Left(x);
        }
    }
    Update(x);
    Root = x;
}

inline int Select(int K) {
    int t = Root;
    while (t) {
        Pass(t);
        if (tree[tree[t].l].s + 1 >= K) {
            if (tree[tree[t].l].s < K) break;
            t = tree[t].l;
        } else {
            K -= tree[tree[t].l].s + 1;
            t = tree[t].r;
        }
    }
    Splay(t);
    return t;
}

inline void ADD() {
    int P, Q, D;
    scanf("%d %d %d", &P, &Q, &D);
    P = Select(P); Q = Select(Q);
    int L = tree[P].l, R = tree[Q].r;
    tree[Q].Add += D;
    tree[L].Add -= D;
    tree[R].Add -= D;
}

inline void REVERSE() {
    int P, Q;
    scanf("%d %d", &P, &Q);
    P = Select(P); Q = Select(Q);
    int L = tree[P].l, R = tree[Q].r;
    tree[P].l = tree[Q].r = 0;
    tree[Q].Reverse ^= 1;
    Splay(P);
    tree[Q].l = L; tree[L].p = Q;
    tree[P].r = R; tree[R].p = P;
    Splay(Q);
}

inline void REVOLVE() {
    int P, Q, T;
    scanf("%d %d %d", &P, &Q, &T);
    T %= (Q - P + 1);
    if (!T) return;
    int I = Select(Q - T);
    P = Select(P); Q = Select(Q);
    int L = tree[P].l, R = tree[Q].r;
    tree[P].l = tree[Q].r = 0;
    Splay(P); Splay(I);
    int J = tree[I].r;
    tree[I].r = 0; tree[J].p = 0;
    tree[I].p = Q; tree[Q].r = I;
    while (Pass(J), tree[J].l) J = tree[J].l;
    tree[J].l = L; tree[L].p = J;
    tree[I].r = R; tree[R].p = I;
    Splay(J);
}

inline void INSERT() {
    int x, P;
    scanf("%d %d", &P, &x);
    tree[++N].Key = x;
    if (P) {
        P = Select(P);
        int Q = tree[P].r;
        tree[P].r = N; tree[N].p = P;
        tree[N].r = Q; tree[Q].p = N;
```

```
    }
    else {
        P = Select(1);
        tree[N].r = P; tree[P].p = N;
    }
    Splay(N);
}

inline void DELETE() {
    int P;
    scanf("%d", &P);
    P = Select(P);
    int Q = tree[P].l;
    tree[Q].p = 0;
    if (!Q) {
        Root = tree[P].r;
        tree[tree[P].r].p = 0;
        return;
    }
    if (!tree[P].r) return;
    while (Pass(Q), tree[Q].r) Q = tree[Q].r;
    tree[Q].r = tree[P].r;
    tree[tree[P].r].p = Q;
    Splay(Q);
}

inline void MIN() {
    int P, Q;
    scanf("%d %d", &P, &Q);
    P = Select(P); Q = Select(Q);
    int L = tree[P].l, R = tree[Q].r;
    tree[P].l = tree[Q].r = 0;
    Splay(P);
    printf("%d\n", tree[P].Min);
    tree[P].l = L; tree[Q].r = R;
    Splay(Q);
}

int main() {
    for (int i = 1; i <= N; ++i) {
        tree[i].p = i - 1; tree[i].r = i + 1;
    }
    Root = 1;
    tree[N].r = 0;
    for (int i = N; i >= 1; --i)
        Update(i);
}
```

LeftistTree

```
struct Tree {
    static const int maxn = 1000000;
    typedef int type_name;
    static int d[maxn], l[maxn], r[maxn], len;
    static type_name key[maxn];
    int root;
    void init() {
        root = -1;
    }
    void insert(const type_name &k) {
        root = merge(root, new_node(k));
    }
    const type_name& min() {
        return key[root];
    }
    void erase_min() {
        root = merge(l[root], r[root]);
    }
    void merge(const Tree& t) {
        root = merge(root, t.root);
    }
    int new_node(const type_name &k) {
        key[len] = k;
        l[len] = r[len] = d[len] = -1;
        return len++;
    }
    int merge(int a, int b) {
        if (a == -1) {
            return b;
        }
        if (b == -1) {
            return a;
        }
        if (key[b] < key[a]) {
            swap(a, b);
        }
        r[a] = merge(r[a], b);
        if (d[r[a]] > d[l[a]]) {
            swap(l[a], r[a]);
```

```
        }
        if (r[a] == -1) {
            d[a] = 0;
        } else {
            d[a] = d[r[a]] + 1;
        }
        return a;
    }
};
int Tree::d[maxn], Tree::l[maxn], Tree::r[maxn], Tree::len;
Tree::type_name Tree::key[maxn];
```

SuffixArray

```
//suffix array (da)
#define maxn 1000001
int wa[maxn],wb[maxn],wv[maxn],ws[maxn];
int cmp(int *r,int a,int b,int l)
{
    return r[a]==r[b] && r[a+l]==r[b+l];
}
void da(int *r,int *sa,int n,int m)
{
     int i, j, p, *x=wa, *y=wb, *t;
     for(i = 0; i < m; i++) ws[i] = 0;
     for(i = 0; i < n; i++) ws[x[i] = r[i]]++;
     for(i = 1; i < m; i++) ws[i] += ws[i-1];
     for(i = n-1; i >= 0; i--) sa[--ws[x[i]]] = i;
     for(j = 1, p = 1; p < n; j *= 2, m = p)
     {
       for(p = 0, i = n-j; i < n; i++) y[p++]=i;
       for(i = 0; i < n; i++) if(sa[i] >= j) y[p++] = sa[i]-j;
       for(i = 0; i < n; i++) wv[i] = x[y[i]];
       for(i = 0; i < m; i++) ws[i] = 0;
       for(i = 0; i < n; i++) ws[wv[i]] ++;
       for(i = 1; i < m; i++) ws[i] += ws[i-1];
       for(i = n-1; i >= 0; i--) sa[--ws[wv[i]]] = y[i];
       for(t=x, x=y, y=t, p=1, x[sa[0]]=0, i=1; i < n; i++){
           x[sa[i]] = cmp(y,sa[i-1],sa[i],j) ? p-1 : p++;
       }
     }
}
int rank[maxn], h[maxn];
void calheight(int *r,int *sa,int n)
{
     int i, j, k = 0;
     for(i = 0; i < n; i++) rank[sa[i]] = i;
     for(i = 0; i < n; h[rank[i++]] = k){
         for(k ? k-- : 0, j = sa[rank[i]-1]; r[i+k] == r[j+k]; k++);
     }
}
int l2[maxn], best[20][maxn];
void initRMQ(int n)
{
     int i, j;
     for(l2[0] = -1, i = 1; i <= n; i++){
         l2[i] = ((i&(i-1))==0) ? l2[i-1]+1 : l2[i-1];
     }
     for(i = 1; i <= n; i++) best[0][i] = h[i];
     for(i = 1; i <= l2[n]; i++)
     for(j = 1; j <= n+1-(1<<i); j++)
     {
         best[i][j] = min(best[i-1][j],  best[i-1][j+(1<<(i-1))]);
     }
}
int askRMQ(int a,int b)
{
    int t = 1<<l2[b-a+1];
    return min(best[t][a], best[t][b-t+1]);
}
//longest common subsequence
char s1[maxn], s2[maxn];
//multiply 3 if it's dc3
int r[maxn], sa[maxn];
int main() {
    while(scanf("%s", s1) != EOF){
        scanf("%s", s2);
        int len1 = strlen(s1), len2 = strlen(s2);
        for(int i = 0; i < len1; i ++) r[i] = (int)s1[i];
        r[len1] = 1;
        for(int i = 0; i < len2; i ++) r[i+len1+1] = (int)s2[i];
        int n = len1 + len2 + 1;
        r[n] = 0;
        dc3(r, sa, n+1, 128);
        ans = 0;
        calheight(r, sa, n);
```

```
        for(int i = 2; i <= n; i ++){
            if((sa[i-1] > len1 && sa[i] < len1)  ||
                    (sa[i-1] < len1 && sa[i] > len1)){
                ans = max(ans, h[i]);
            }
        }
        printf("%d\n", ans);
    }
    return 0;
}
```

Aho-Corasick

```
class Trie {
public:
    const static int st = 'A', en = 'z';
    const static int m = en - st + 1;
    const static int maxn = 10002;
    int d[maxn][m];     //graph
    int t[maxn];        //state
    int p[maxn];        //suffix
    int n, len;
    void init() {
        len = 1;
        n = 0;
        t[0] = 0;
        memset(d[0], -1, sizeof(d[0]));
    }
    void insert(char *s, int id) {
        int i;
        for(i = 0; *s; ++s) {
            int &k = d[i][*s - st];
            if(k == -1) {
                k = len++;
                memset(d[k], -1, sizeof(d[k]));
                t[k] = 0;
            }
            i = k;
        }
        t[i] |= 1<<id;
    }
    void bfs() {
        int i;
        queue<int> q;
        q.push(0);
        p[0] = 0;
        while(!q.empty()) {
            int k = q.front();
            q.pop();
            for(i = 0; i < m; i++) {
                int &j = d[k][i];
                if(-1 == j) {
                    j = d[p[k]][i];
                    if(j == -1)j=0;
                }
                else {
                    if (k) p[j] = d[p[k]][i];
                    else  p[j] = 0;
                    t[j] |= t[p[j]];
                    q.push(j);
                }
            }
        }
    }
};
```

Computational Geometry

CircleTangent

```
vector<pair<P, P> > circle_tangent(const C &a, const C &b) {
    vector<pair<P, P> > ans;
    double len2 = dist2(a.mid, b.mid);
    if (sgn(len2 - SQR(a.r + b.r)) > 0) {
        double len = sqrt(len2);
        double x = len * a.r / (b.r + a.r);
        double y = len * b.r / (a.r + b.r);
        double ta = sqrt(abs(sqr(x) - sqr(a.r)));
        double ha = a.r * ta / x;
        double pa = sqrt(abs(sqr(a.r) - sqr(ha)));
        double tb = sqrt(abs(sqr(y) - sqr(b.r)));
        double hb = b.r * tb / y;
        double pb = sqrt(abs(sqr(b.r) - sqr(hb)));
        P ab = b.mid - a.mid;
        P ba = a.mid - b.mid;
```

```
        ans.push_back(make_pair(a.mid + ab.trunc(pa) +
ab.turn_right().trunc(ha),
                    b.mid + ba.trunc(pb) + ba.turn_right().trunc(hb)));
        ans.push_back(make_pair(a.mid + ab.trunc(pa) +
ab.turn_left().trunc(ha),
                    b.mid + ba.trunc(pb) + ba.turn_left().trunc(hb)));
    }
    if (!a.in(b) && !b.in(a) && a != b) {
        double len = sqrt(len2);
        double n = sqrt(abs(len2 - sqr(a.r - b.r)));
        double ha = a.r * n / len;
        double hb = b.r * n / len;
        double pa = sqrt(abs(sqr(a.r) - sqr(ha)));
        double pb = sqrt(abs(sqr(b.r) - sqr(hb)));
        P p = (a.r > b.r? b.mid - a.mid: a.mid - b.mid);
        ans.push_back(make_pair(a.mid + p.trunc(pa) +
p.turn_right().trunc(ha),
                    b.mid + p.trunc(pb) + p.turn_right().trunc(hb)));
        ans.push_back(make_pair(a.mid + p.trunc(pa) +
p.turn_left().trunc(ha),
                    b.mid + p.trunc(pb) + p.turn_left().trunc(hb)));
    }
    return ans;
}
```

Hp Intersection

```
P line_intersection(const P &a, const P &b, const P &c, const P &d) {
    double s1 = cross(a, b, c);
    double s2 = cross(a, b, d);
    return P((c.x * s2 - d.x * s1) / (s2 - s1), (c.y * s2 - d.y * s1) / (s2
- s1));
}
P line_intersection(const pair<P, P> &a, const pair<P, P> &b) {
    return line_intersection(a.first, a.second, b.first, b.second);
}

bool cmp_cross(const pair<P, P> &a, const pair<P, P> &b) {
    int k = dcmp(cross(a.second - a.first, b.second - b.first));
    if (k == 0) {
        return dcmp(cross(b.first - a.first, a.second - a.first)) > 0;
    }
    return k > 0;
}
bool cmp_cross_equal(const pair<P, P> &a, const pair<P, P> &b) {
    return dcmp(cross(a.second - a.first, b.second - b.first)) == 0;
}

vector<P>& hp_intersection(const vector<pair<P, P> > &p) {
    static vector<pair<P, P> > a, b;
    a.clear(), b.clear();
    for (int i = 0; i < SZ(p); ++i) {
        if (dcmp(p[i].second.y - p[i].first.y) > 0
                || dcmp(p[i].second.y - p[i].first.y) == 0 &&
dcmp(p[i].second.x - p[i].first.x) < 0) {
            a.push_back(p[i]);
        } else {
            b.push_back(p[i]);
        }
    }
    sort(a.begin(), a.end(), cmp_cross);
    a.erase(unique(a.begin(), a.end(), cmp_cross_equal), a.end());
    sort(b.begin(), b.end(), cmp_cross);
    b.erase(unique(b.begin(), b.end(), cmp_cross_equal), b.end());
    static vector<P> stp;
    static vector<pair<P, P> > st;
    static deque<P> pa, pb;
    pa = pb = deque<P> (0);
    if (SZ(a) == 1) {
        pa.push_back(a[0].first);
        pa.push_back(a[0].second);
    } else {
        stp.clear();
        st.clear();
        st.push_back(a[0]);
        for (int i = 1; i < SZ(a); ++i) {
            P p = line_intersection(st.back(), a[i]);
            if (st.size() == 1 || dcmp(p.y - stp.back().y) > 0) {
                st.push_back(a[i]);
                stp.push_back(p);
            } else {
                stp.pop_back();
                st.pop_back();
                --i;
            }
        }
        for (int i = 0; i < SZ(stp); ++i) {
            pa.push_back(stp[i]);
        }
        pa.push_front(pa.front() + st.front().first - st.front().second);
        pa.push_back(pa.back() + st.back().second - st.back().first);
    }
    if (SZ(b) == 1) {
```

```
        pb.push_back(b[0].first);
        pb.push_back(b[0].second);
    } else {
        stp.clear();
        st.clear();
        st.push_back(b[0]);
        for (int i = 1; i < SZ(b); ++i) {
            P p = line_intersection(st.back(), b[i]);
            if (st.size() == 1 || dcmp(p.y - stp.back().y) < 0) {
                st.push_back(b[i]);
                stp.push_back(p);
            } else {
                stp.pop_back();
                st.pop_back();
                --i;
            }
        }
        for (int i = 0; i < SZ(stp); ++i) {
            pb.push_back(stp[i]);
        }
        pb.push_front(pb.front() + st.front().first - st.front().second);
        pb.push_back(pb.back() + st.back().second - st.back().first);
    }
    if (dcmp(cross(*(++pa.begin()) - pa.front(), *(++pb.rbegin()) -
pb.back())) != 0) {
        P p = line_intersection(*(++pa.begin()) , pa.front(),
*(++pb.rbegin()) , pb.back());
        if (dcmp(cross(pa.front(), *(++pa.begin()), pb.back())) > 0) {
            pb.back() = p;
        }
        if (dcmp(cross(pb.back(), *(++pb.rbegin()), pa.front())) < 0) {
            pa.front() = p;
        }
    }
    if (dcmp(cross(*(++pb.begin()) - pb.front(), *(++pa.rbegin()) -
pa.back())) != 0) {
        P p = line_intersection(*(++pb.begin()) , pb.front(),
*(++pa.rbegin()) , pa.back());
        if (dcmp(cross(pb.front(), *(++pb.begin()), pa.back())) > 0) {
            pa.back() = p;
        }
        if (dcmp(cross(pa.back(), *(++pa.rbegin()), pb.front())) < 0) {
            pb.front() = p;
        }
    }
    while (SZ(pa) >= 2 && SZ(pb) >= 2) {
        if (dcmp(cross(pb.back(), *(++pb.rbegin()), *(++pa.begin())) > 0))
{
            pa.pop_front();
        } else if (dcmp(cross(pa.front(), *(++pa.begin()),
*(++pb.rbegin()))) < 0) {
            pb.pop_back();
        } else if (dcmp(cross(pa.back(), *(++pa.rbegin()),
*(++pb.begin()))) > 0) {
            pb.pop_front();
        } else if (dcmp(cross(pb.front(), *(++pb.begin()),
*(++pa.rbegin()))) < 0) {
            pa.pop_back();
        } else {
            pa.front() = pb.back() = line_intersection(pa.front(),
*(++pa.begin()), pb.back(), *(++pb.rbegin()));
            pa.back() = pb.front() = line_intersection(pa.back(),
*(++pa.rbegin()), pb.front(),  *(++pb.begin()));
            break;
        }
    }
    static vector<P> ans;
    ans.clear();
    if (SZ(pa) < 2 || SZ(pb) < 2) {
        return ans;
    }
    for (int i = 0; i + 1 < SZ(pa); ++i) {
        ans.push_back(pa[i]);
    }
    for (int i = 0; i + 1 < SZ(pb); ++i) {
        ans.push_back(pb[i]);
    }
    return ans;
}
```

Intersection of Two Triangles

```
bool intersection(const P &a, const P &b, const P &c, const P &d, P &p) {
    double s1 = cross(a, b, c);
    double s2 = cross(a, b, d);
    if (dcmp(s2 - s1) == 0) {
        return false;
    }
    p.x = (c.x * s2 - d.x * s1) / (s2 - s1);
    p.y = (c.y * s2 - d.y * s1) / (s2 - s1);
    if (dcmp(dmul(p, a, b)) <= 0 && dcmp(dmul(p, c, d)) <= 0) {
        return true;
    }
```

```
    return false;
}

double uno(const P &a, const P &b, const P &c, const P &d) {
    int flag = dcmp(cross(a, b)) * dcmp(cross(c, d));
    if (flag == 0) {
        return 0;
    }
    P p[5];
    int len = 0;
    int ab = dcmp(cross(a, b));
    int ba = -ab;
    int cd = dcmp(cross(c, d));
    int dc = -cd;
    int ac = dcmp(cross(a, c));
    int ca = -ac;
    int ad = dcmp(cross(a, d));
    int da = -ad;
    int bc = dcmp(cross(b, c));
    int cb = -bc;
    int bd = dcmp(cross(b, d));
    int db = -bd;

    if (cd == ca && dc == da || ac == 0 && dmul(a, c) > 0 || ad == 0 && dmul(a,
d) > 0) {
        if (intersection(c, d, P(0, 0), a, p[len])) {
            ++len;
        } else {
            p[len++] = a;
        }
    }
    if (cd == cb && dc == db || bc == 0 && dmul(b, c) > 0 || bd == 0 && dmul(b,
d) > 0) {
        if (intersection(c, d, P(0, 0), b, p[len])) {
            ++len;
        } else {
            p[len++] = b;
        }
    }
    if (ab == ac && ba == bc) {
        if (intersection(a, b, P(0, 0), c, p[len])) {
            ++len;
        } else {
            p[len++] = c;
        }
    }
    if (ab == ad && ba == bd) {
        if (intersection(a, b, P(0, 0), d, p[len])) {
            ++len;
        } else {
            p[len++] = d;
        }
    }
    if (intersection(a, b, c, d, p[len])) {
        ++len;
    }
    double ans = 0;
    if (len == 3) {
        ans = abs(cross(p[0], p[2])) + abs(cross(p[1], p[2]));
    } else if (len == 2) {
        ans = abs(cross(p[0], p[1]));
    }
    return flag * abs(ans);
}
```

Intersection of Triangle and Circle

```
//三角形和圆的交
bool cut(const Point &pt1, const Point &pt2, double r, Point &p1, Point &p2)
{
    Point c = pt2 - pt1;
    double t = c.dist() * r * r - c.dist() * pt1.dist() + c.dot(pt1) * c.dot(pt1);
    if (sgn(t) < 0) return false;
    double t1 = (c.dot(pt1 * (-1)) - sqrt(t)) / c.lenth();
    double t2 = (c.dot(pt1 * (-1)) + sqrt(t)) / c.lenth();
    p1 = pt1 + (pt2 - pt1).set(t1);
    p2 = pt1 + (pt2 - pt1).set(t2);
    if (sgn((pt1 - p1).dot(pt2 - p1)) < 0) {
        return true;
    }
    return false;
}
double in(double x) {
    x = max(x, -1.);
    x = min(x, 1.);
    return x;
}
double shan(const Point &pt1, const Point &pt2, double r) {
    double alpha = acos(in(pt1.dot(pt2) / (pt1.lenth() * pt2.lenth())));
    double res = alpha * r * r * 0.5;
```

```
    return res;
}
double area(Point pt1, Point pt2, double r) {
    double s1 = pt1.dist(), s2 = pt2.dist();
    double r2 = r * r, res;
    int type = 1;
    if (sgn(pt1 * pt2) < 0) type = -1;
    Point p1, p2;
    if (sgn(s1 - r2) > 0 && sgn(s2 - r2) > 0) {
        if (!cut(pt1, pt2, r, p1, p2)) {
            res = shan(pt1, pt2, r);
        } else {
            res = shan(pt1, p1, r) + shan(pt2, p2, r) + fabs(p1 * p2) * 0.5;
        }
    } else if (sgn(s1 - r2) <= 0 && sgn(s2 - r2) <= 0) {
        res = fabs(pt1 * pt2) * 0.5;
    } else {
        if (sgn(s1 - r2) > 0) {
            swap(s1, s2);
            swap(pt1, pt2);
        }
        cut(pt2, pt1, r, p2, p1);
        res = shan(pt2, p2, r) + fabs(pt1 * p2) * 0.5;
    }
    return res * type;
}
```

Point_in_poly

```
//点在多边形内，多边形上的点也作为多边形内
int find(Point t)
{
    return t.x >= 0 ? (t.y >= 0 ? 0 : 3) : (t.y >= 0 ? 1 : 2);
}

bool in_ploy(Point t, Point *p, int size)
{
    int i;
    for(i = 0; i <= size; ++i) p[i].x -= t.x, p[i].y -= t.y;
    int t1, t2;
    t1 = find(p[0]);
    int sum = 0;
    for(i = 1; i <= size; ++i)
    {
        if(fabs(p[i].x) < eps && fabs(p[i].y) < eps) break;
        double f = p[i - 1] * p[i];
        if(fabs(f) < eps && p[i - 1].x * p[i].x <= 0 && p[i - 1].y * p[i].y
<= 0) break;
        t2 = find(p[i]);
        if(t2 == (t1 + 1) % 4) ++sum;
        else if(t2 == (t1 + 3) % 4) --sum;
        else if(t2 == (t1 + 2) % 4)
        {
            if(f > 0) sum += 2;
            else sum -= 2;
        }
        t1 = t2;
    }
    int k = i;
    for (i = 0; i <= size; ++i) p[i].x += t.x, p[i].y += t.y;
    if (k <= size) return true;     //点在多边形上
    return (sum);
}
```

Convex_poly_area_union

```
//凸多边形面积并
bool segcross(const Point &a, const Point &b, const Point &c, const Point &d,
Point &pt) {
    double s1 = cross(a, b, c), s2 = cross(a, b, d);
    if (sgn(s1) * sgn(s2) > 0 || sgn(s1 - s2) == 0) return false;
    pt = (c * s2 - d * s1) / (s2 - s1);
    if (sgn((a - pt).dot(b - pt)) <= 0) return true;
    return false;
}
bool in_poly(const Point &pt, const vector<Point> &a) {
    for (int i = 0; i < (int)a.size(); ++i) {
        if (sgn(cross(pt, a[i], a[next(i + 1, a.size())])) <= 0) return false;
    }
    return true;
}
bool in_polygon(const Point &mid, const vector<vector<Point> > &a) {
```

```
    for (int i = 0; i < (int)a.size(); ++i) {
        if (in_poly(mid, a[i])) return true;
    }
    return false;
}
double poly_area(const vector<vector<Point> > &a) {
    vector<vector<Point> > c(a.size());
    for (int i = 0; i < (int)a.size(); ++i) {
        for (int j = 0; j < (int)a[i].size(); ++j) {
            c[i].push_back(a[i][j]);
            for (int k = 0; k < i; ++k) {
                for (int p = 0; p < (int)a[k].size(); ++p) {
                    Point pt;
                    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)) {
                        c[i].push_back(pt);
                        c[k].push_back(pt);
                    }
                }
            }
        }
    }
    for (int i = 0; i < (int)a.size(); ++i) {
        ins = Point(0, 0);
        for (int j = 0; j < (int)a[i].size(); ++j) {
            ins = ins + a[i][j];
        }
        ins = ins / a[i].size();
        stable_sort(c[i].begin(), c[i].end(), cmp);
        c[i].erase(unique(c[i].begin(), c[i].end()), c[i].end());
    }
    set<pair<Point, Point> > re;
    double area = 0;
    for (int i = 0; i < (int)c.size(); ++i) {
        for (int j = 0; j < (int)c[i].size(); ++j) {
            Point &p1 = c[i][j], &p2 = c[i][next(j + 1, c[i].size())];
            if (re.count(make_pair(p1, p2)) == false) {
                if (!in_polygon((p1 + p2) * 0.5, a)) {
                    area += c[i][j] * c[i][next(j + 1, c[i].size())];
                }
                re.insert(make_pair(p1, p2));
            }
        }
    }
    return fabs(area * 0.5);
}
```

Circle_area_union

```
//圆的面积并
const double eps = 1e-9;
const double pi = acos(-1.0);

const int zx[] = {0, 1, 0, -1};
const int zy[] = {1, 0, -1, 0};
double dist2(const P &a, const P &b) {
    return SQR(a.x - b.x) + SQR(a.y - b.y);
}
double dist(const P &a, const P &b) {
    return sqrt(SQR(a.x - b.x) + SQR(a.y - b.y));
}
double cross(const P &a, const P &b, const P &c) {
    return (b.x - a.x) * (c.y - a.y) - (b.y - a.y) * (c.x - a.x);
}
double dmul(const P &a, const P &b, const P &c) {
    return (b.x - a.x) * (c.x - a.x) + (b.y - a.y) * (c.y - a.y);
}
struct C {
    P mid;
    double r;
    C(const P &_mid, const double &_r)
        :mid(_mid), r(_r) {}
    C() {}
    bool operator == (const C &a) const {
        return mid == a.mid && sgn(r - a.r) == 0;
    }
    bool in(const C &a) const {
        return sgn(r + dist(mid, a.mid) - a.r) <= 0;
    }
    const C &input() {
        mid.input();
        scanf("%lf", &r);
        return *this;
```

```
    }
    const C &output() const {
        printf("P: %.12lf %.12lf R: %.12lf\n", mid.x, mid.y, r);
    }
};
double cal_angle(const C &c, const P &a, const P &b) {
    double k = dmul(c.mid, a, b) / SQR(c.r);
    get_min(k, 1.0);
    get_max(k, -1.0);
    return acos(k);
}
double cal_area(const C &c, const P &a, const P &b) {
    return SQR(c.r) * cal_angle(c, a, b) / 2 - cross(c.mid, a, b) / 2;
}
struct cmp {
    P mid;
    cmp(const P &_mid)
        :mid(_mid) {}
    bool operator () (const P &a, const P &b) {
        return atan2(a.y - mid.y, a.x - mid.x) < atan2(b.y - mid.y, b.x - mid.x);
    }
};
bool circles_intersection(const C &a, const C &b, P &c1, P &c2) {
    double dd = dist(a.mid, b.mid);
    if (sgn(dd - (a.r + b.r)) >= 0) {
        return false;
    }
    double l = (dd + (SQR(a.r) - SQR(b.r)) / dd) / 2;
    double h = sqrt(SQR(a.r) - SQR(l));
      c1   =   a.mid   +   (b.mid   -   a.mid).trunc(l)   +   (b.mid   -
a.mid).turn_left().trunc(h);
      c2   =   a.mid   +   (b.mid   -   a.mid).trunc(l)   +   (b.mid   -
a.mid).turn_right().trunc(h);
    return true;
}
bool cover(const C &c, const P &a, const P &b, const vector<C> &cir) {
    P p = c.mid + ((a + b) / 2 - c.mid).trunc(c.r);
    for (vector<C>::const_iterator it = cir.begin(); it != cir.end(); ++it)
{
        if (sgn(dist2(p, it->mid) - SQR(it->r)) < 0) {
            return true;
        }
    }
    return false;
}
double cal_area(const vector<C> &in) {
    vector<C> cir;
    for (int i = 0; i < SZ(in); ++i) {
        if (sgn(in[i].r) == 0) {
            continue;
        }
        bool flag = false;
        for (int j = i + 1; j < SZ(in); ++j) {
            if (in[i] == in[j]) {
                flag = true;
                break;
            }
        }
        if (flag) {
            continue;
        }
        for (int j = 0; j < SZ(in); ++j) {
            if (!(in[i] == in[j]) && in[i].in(in[j])) {
                flag = true;
                break;
            }
        }
        if (flag) {
            continue;
        }
        cir.push_back(in[i]);
    }
    vector<vector<P> > point_on_circle(SZ(cir));
    for (int i = 0; i < SZ(cir); ++i) {
        for (int z = 0; z < 4; ++z) {
                point_on_circle[i].push_back(cir[i].mid     +     P(zx[z],
zy[z]).trunc(cir[i].r));
        }
    }
    for (int i = 0; i < SZ(cir); ++i) {
        for (int j = i + 1; j < SZ(cir); ++j) {
            P a, b;
            if (circles_intersection(cir[i], cir[j], a, b)) {
                point_on_circle[i].push_back(a);
                point_on_circle[i].push_back(b);
                point_on_circle[j].push_back(a);
```

```
                point_on_circle[j].push_back(b);
            }
        }
    }
    for (int i = 0; i < SZ(cir); ++i) {
            sort(point_on_circle[i].begin(),     point_on_circle[i].end(),
cmp(cir[i].mid));
              point_on_circle[i].erase(unique(point_on_circle[i].begin(),
point_on_circle[i].end()), point_on_circle[i].end());
    }
    double ans = 0;
    for (int i = 0; i < SZ(cir); ++i) {
        for (int j = 0; j < SZ(point_on_circle[i]); ++j) {
            const P &a = point_on_circle[i][j];
               const P     &b     =     point_on_circle[i][NEXT(j     +     1,
SZ(point_on_circle[i]))];
            if (!cover(cir[i], a, b, cir)) {
                ans += cross(P(0, 0), a, b) / 2;
                ans += cal_area(cir[i], a, b);
            }
        }
    }
    return ans;
}
```

Plane_cross

```
//半平面交 nlgn
const double eps = 1e-8;
const int maxn = 20000 + 10;

double cross(const Point &a, const Point &b, const Point &c) {
    return (b - a) * (c - a);
}

struct Line {
    Point p1, p2;
    double angle;
    void get_angle() {
        Point p = p2 - p1;
        angle = atan2(p.y, p.x);
        if (p.y < 0) angle += pi * 2;
    }
    Line() {}
    Line(const Point &_p1, const Point &_p2): p1(_p1), p2(_p2) {
        Point p = p2 - p1;
        get_angle();
    }
    bool operator < (const Line &a) const {
        return sgn(angle - a.angle) < 0 || sgn(angle - a.angle) == 0 &&
sgn(cross(a.p1, a.p2, p1)) > 0;
    }
};

Line line[maxn], ln[maxn];
Point pt1[maxn], pt2[maxn];
int len1, len2, n;

Point lncross(const Point &a, const Point &b, const Point &c, const Point &d)
{
    double s1 = cross(a, b, c);
    double s2 = cross(a, b, d);
    return (c * s2 - d * s1) / (s2 - s1);
}

int segcross(const Point &a, const Point &b, const Point &c, const Point &d,
Point &pt) {
    pt = lncross(a, b, c, d);
    int d1 = sgn((a - pt).dot(b - pt)), d2 = sgn((c - pt).dot(d - pt));
    if (d1 < 0 && d2 < 0) return 0;
    if (pt == a && d2 <= 0 || pt == c && d1 <= 0) return 0;
    if (sgn(cross(c, d, a)) > 0 && sgn(cross(c, d, b)) >= 0) return 1;
    return -1;
}

double cal(Point *pt1, Point *pt2, int len1, int len2) {
    int p1 = 0, p2 = len2, tmp;
    Point pp1, pp2;
    while (p1 <= len1 && p2 >= 0 && (tmp = segcross(pt1[p1], pt1[p1 + 1], pt2[p2],
pt2[p2 - 1], pp1))) {
        if (tmp > 0) ++p1;
        else --p2;
    }
```

```
    int p3 = len1, p4 = 0;
    while (p3 >= 0 && p4 <= len2 && (tmp = segcross(pt2[p4], pt2[p4 + 1], pt1[p3], pt1[p3 - 1], pp2))) {
        if (tmp > 0) ++p4;
        else --p3;
    }
    int len = 0;
    Point cv[maxn];
    cv[len++] = pp1;
    for (int i = p1 + 1; i < p3; ++i) cv[len++] = pt1[i];
    cv[len++] = pp2;
    for (int i = p4 + 1; i < p2; ++i) cv[len++] = pt2[i];
    cv[len] = cv[0];
    double sum = 0;
    for (int i = 0; i < len; ++i) sum += cv[i] * cv[i + 1];
    return fabs(sum * 0.5);
}

void init(Point *pt, int &len, Line *line, int n, bool flag) {
    int i;
    ln[len = 0] = line[0];
    pt[0] = line[0].p1 + (line[0].p1 - line[0].p2) * 1e10;
    for (i = 1; i < n; ++i) {
        Point p;
        while (len > 0) {
            p = lncross(line[i].p1, line[i].p2, ln[len].p1, ln[len].p2);
            if (flag && sgn(p.y - pt[len].y) <= 0 || !flag && sgn(p.y - pt[len].y) >= 0) --len;
            else break;
        }
        ln[++len] = line[i];
        pt[len] = lncross(line[i].p1, line[i].p2, ln[len - 1].p1, ln[len - 1].p2);
    }
    pt[++len] = line[i - 1].p1 + (line[i - 1].p2 - line[i - 1].p1) * 1e10;
}

double solve() {
    int cut;
    for (int i = 1; i < n; ++i) {
        if (sgn(line[i].angle - pi) >= 0) {
            cut = i;
            break;
        }
    }
    init(pt1, len1, line, cut, 1);
    init(pt2, len2, line + cut, n - cut, 0);
    return cal(pt1, pt2, len1, len2);
}

bool get_input() {
    if (scanf("%d", &n) == EOF) return false;
    for (int i = 0; i < n; ++i) {
        line[i].p1.input();
        line[i].p2.input();
        line[i].get_angle();
    }
    line[n++] = Line(Point(0, 0), Point(10000, 0));
    line[n++] = Line(Point(10000, 0), Point(10000, 10000));
    line[n++] = Line(Point(10000, 10000), Point(0, 10000));
    line[n++] = Line(Point(0, 10000), Point(0, 0));
    sort(line, line + n);
    int nn = n;
    n = 0;
    for (int i = 0; i < nn; ++i) {
        if (i && sgn(line[i].angle - line[i - 1].angle) == 0) continue;
        line[n++] = line[i];
    }
    return true;
}
```

Others

Extended_GCD

```
int egcd(int a,int b,int &x,int &y) {
    int t,d;
    if (b==0) return x=1,y=0,a;
    d=egcd(b,a%b,x,y);
    t=x; x=y; y=t-a/b*y;
    return d;
}
```

PrimeChecker

```cpp
long long mul(long long a, long long b, long long c) {
    long long ans = 0;
    for (int i = 62; i >= 0; --i) {
        ans <<= 1;
        if (ans >= c) {
            ans -= c;
        }
        if ((a >> i) & 1) {
            ans += b;
            if (ans >= c) {
                ans -= c;
            }
        }
    }
    return ans;
}
bool check(long long a, long long b, long long c) {
    long long ans = 1;
    for (; b; b >>= 1) {
        if (b & 1) {
            ans = mul(ans, a, c);
        }
        long long t = mul(a, a, c);
        if (t == 1 && a != 1 && a != c - 1) {
            return false;
        }
        a = t;
    }
    return ans == 1;
}
bool isPrime(long long a) {
    if (a % 2 == 0 || a % 3 == 0 || a % 7 == 0 || a % 11 == 0) {
        return false;
    }
    for (int i = 0; i < 10; ++i) {
        if (!check(rand() % (a - 3) + 2, a - 1, a)) {
            return false;
        }
    }
    return true;
}
```

BigInteger

```cpp
const int base = 10000;
const int cap = 200;
struct xnum {
    int len;
    int data[cap];
    xnum() : len(0) {}
    xnum(const xnum& v) : len(v.len) { memcpy(data, v.data, len *
sizeof *data); }
    xnum(int v) : len(0) { for (; v > 0; v /= base) data[len++] = v % base; }
    xnum& operator=(const xnum& v) { len = v.len; memcpy(data, v.data, len
* sizeof *data); return *this; }
    int& operator[](int index) { return data[index]; }
    int operator[](int index) const { return data[index]; }
};
int compare(const xnum& a, const xnum& b) {
    int i;
    if (a.len != b.len) return a.len > b.len ? 1 : -1;
    for (i = a.len - 1; i >= 0 && a[i] == b[i]; i--);
    if (i < 0) return 0;
    return a[i] > b[i] ? 1 : -1;
}
xnum operator+(const xnum& a, const xnum& b) {
    xnum r;
    int i, c = 0;
    for (i = 0; i < a.len || i < b.len || c > 0; i++) {
        if (i < a.len) c += a[i];
        if (i < b.len) c += b[i];
        r[i] = c % base;
        c /= base;
    }
    r.len = i;
    return r;
}
xnum operator-(const xnum& a, const xnum& b) {
    xnum r;
    int c = 0;
    r.len = a.len;
    for (int i = 0; i < r.len; i++) {
        r[i] = a[i] - c;
        if (i < b.len) r[i] -= b[i];
        if (r[i] < 0) c = 1, r[i] += base;
        else c = 0;
    }
    while (r.len > 0 && r[r.len - 1] == 0) r.len--;
```

```
    return r;
}
xnum operator*(const xnum& a, const int b) {
    int i;
    if (b == 0) return 0;
    xnum r;
    int c = 0;
    for (i = 0; i < a.len || c > 0; i++) {
        if (i < a.len) c += a[i] * b;
        r[i] = c % base;
        c /= base;
    }
    r.len = i;
    return r;
}
xnum operator*(const xnum& a, const xnum& b) {
    if (b.len == 0) return 0;
    xnum r;
    for (int i = 0; i < a.len; i++) {
        int c = 0;
        for (int j = 0; j < b.len || c > 0; j++) {
            if (j < b.len) c += a[i] * b[j];
            if (i + j < r.len) c += r[i + j];
            if (i + j >= r.len) r[r.len++] = c % base;
            else r[i + j] = c % base;
            c /= base;
        }
    }
    return r;
}
xnum operator/(const xnum& a, const int b) {
    xnum r;
    int c = 0;
    for (int i = a.len - 1; i >= 0; i--) {
        c = c * base + a[i];
        r[i] = c / b;
        c %= b;
    }
    r.len = a.len;
    while (r.len > 0 && r[r.len - 1] == 0) r.len--;
    return r;
}
xnum operator/(const xnum& a, const xnum& b)
{
    xnum r, c = 0;
    int left, right, mid;
    for (int i = a.len - 1; i >= 0; i--) {
        c = c * base + a[i];
        left = 0;
        right = base - 1;
        while (left < right) {
            mid = (left + right + 1) / 2;
            if (compare(b * mid, c) <= 0) left = mid;
            else right = mid - 1;
        }
        r[i] = left;
        c = c - b * left;
    }
    r.len = a.len;
    while (r.len > 0 && r[r.len - 1] == 0) r.len--;
    return r;
}
xnum operator%(const xnum& a, const xnum& b) {
    xnum r, c = 0;
    int left, right, mid;
    for (int i = a.len - 1; i >= 0; i--) {
        c = c * base + a[i];
        left = 0;
        right = base - 1;
        while (left < right) {
            mid = (left + right + 1) / 2;
            if (compare(b * mid, c) <= 0) left = mid;
            else right = mid - 1;
        }
        r[i] = left;
        c = c - b * left;
    }
    r.len = a.len;
    while (r.len > 0 && r[r.len - 1] == 0) r.len--;
    return c;
}
istream& operator>>(istream& in, xnum& v) {
    char ch;
    for (v = 0; in >> ch;) {
        v = v * 10 + (ch - '0');
        if (cin.peek() <= ' ') break;
    }
    return in;
}
ostream& operator<<(ostream& out, const xnum& v) {
    out << (v.len == 0 ? 0 : v[v.len - 1]);
    for (int i = v.len - 2; i >= 0; i--) for (int j = base / 10; j > 0; j /=
10) out << v[i] / j % 10;
    return out;
```

```
136 }
```

Vimrc

```
source $VIMRUNTIME/mswin.vim
behave mswin
imap <cr> <cr><left><right>
imap <c-]> {<cr>}<c-o>O<left><right>
imap <c-d> <c-o>dd
map <f6> =a{
map <c-t> :tabnew<cr>
syn on
colo torte
set gfn=Courier\ 10\ Pitch\ 12
set ru nu et sta nowrap ar acd ww=<,>,[,] sw=4 ts=4 cin noswf

map <f10> :call CR2()<cr><space>
func CR2()
exec "update"
exec "!xterm -fn 10*20 -e \"g++ %<.cpp -Wall -o %< && time ./%< ; read -n 1\""
endfunc
map <f9> :call CR()<cr><space>
func CR()
exec "update"
exec "!xterm -fn 10*20 -e \"g++ %<.cpp -Wall -o %< && time ./%< < %<.in ; read
-n 1\""
endfunc

map<f4> :call AddComment()<cr>
func AddComment()
    if (getline('.')[0] == '/')
        normal ^xx
    else
        normal 0i//
    endif
endfunc
```