```cpp
#include <bits/stdc++.h>
#define EPS 1e-10
#define PI 3.141592653589793238
#define equals(a,b) (fabs((a)-(b)) < EPS)
#define rep(i,n) for(int i=0;i<n;++i)
typedef long long ll;
using namespace std;

// p1とp2のp0を挟んだ位置関係が反時計回り
static const int COUNTER_CLOCKWISE = 1;
// 時計回り
static const int CLOCKWISE = -1;
// p1とp2がp0を挟んで直線状にある
static const int ONLINE_BACK = 2;
// p0とp2の間にp1がある
static const int ONLINE_FRONT = -2;
// p0とp1の間にp2がある
static const int ON_SEGMENT = 0;

//Intercsect Circle & Circle 共通接線の数
static const int ICC_SEPERATE = 4;
static const int ICC_CIRCUMSCRIBE = 3;
static const int ICC_INTERSECT = 2;
static const int ICC_INSCRIBE = 1;
static const int ICC_CONTAIN = 0;

// 点
struct Point {
    double x,y;
    Point(){}
    Point(double x, double y) : x(x),y(y){}
    Point operator+(Point p) {return Point(x+p.x, y+p.y);}
    Point operator-(Point p) {return Point(x-p.x, y-p.y);}
    Point operator*(double k){return Point(x*k,y*k);}
    Point operator/(double k){return Point(x/k,y/k);}
    double norm(){return x*x+y*y;}
    double abs(){sqrt(norm());}
    bool operator == (const Point &p) const {return equals(x,p.x)&&equals(y,p.y);}
    bool operator < (const Point &p) const {
        return x!=p.x?x<p.x:y<p.y;
        //grid-point only
        //return !equals(x,p.x)?x<p.x:!equals(y,p.y)?y<p.y:0;
    }
};

typedef Point P;
typedef vector<Point> Polygon;

double norm(P a){return a.x*a.x+a.y*a.y;}
double abs(P a){return sqrt(norm(a));}

// 線分
struct Segment {
    Point p1,p2;
    Segment(){}
    Segment(Point p1, Point p2):p1(p1),p2(p2){}
};

typedef Segment Line;

```

```cpp
struct Circle {
    Point c;
    double r;
    Circle(){}
    Circle(Point c, double r):c(c),r(r){}
};

// 法線ベクトル
Point orth(Point p){return Point(-p.y,p.x);}

// 内積
double dot(Point a, Point b) {return a.x*b.x + a.y*b.y;}

// 外積
double cross(Point a, Point b) {return a.x*b.y-a.y*b.x;}
// 2直線の直行判定
bool is_orthogonal(Point a1, Point a2, Point b1, Point b2) {
    return equals(dot(a1-a2, b1-b2), 0.0);
}
// 2直線の平行判定
bool is_parallel(Point a1, Point a2, Point b1, Point b2) {
    return equals(cross(a1-a2, b1-b2), 0.0);
}
// 点cが直線ab上にあるかないか
bool is_point_on_INF_line(Point a, Point b, Point c) {
    return equals(cross(b-a,c-a), 0.0);
}
// 点cが線分ab上にあるかないか
bool is_point_on_LIMITED_line(Point a, Point b, Point c) {
    return (Point(a-c).abs()+Point(c-b).abs() < Point(a-b).abs() + EPS);
}
// 直線と点の距離
double distance_l_p(Point a, Point b, Point c) {return abs(cross(b-a, c-a)) / (b-a).abs();}

// 点a,bを端点とする線分と点cとの距離
double distance_ls_p(Point a, Point b, Point c) {
    if (dot(b-a, c-a) < EPS) return (c-a).abs();
    if (dot(a-b, c-b) < EPS) return (c-b).abs();
    return abs(cross(b-a, c-a)) / (b-a).abs();
}

// 射影
Point project(Segment s, Point p) {
    Point base = s.p2-s.p1;
    double r = dot(p-s.p1,base)/norm(base);
    return s.p1+base*r;
}

// 点が線分のどちら側にあるかを計算
int ccw(Point p0,Point p1,Point p2) {
    P a = p1-p0;
    P b = p2-p0;
    if(cross(a,b) > EPS) return COUNTER_CLOCKWISE;
    if(cross(a,b) < -EPS) return CLOCKWISE;
    if(dot(a,b) < -EPS) return ONLINE_BACK;
    if(a.norm()<b.norm()) return ONLINE_FRONT;
    return ON_SEGMENT;
}

// a1,a2を端点とする線分とb1,b2を端点とする線分の交差判定
```

```
121 bool intersectSS(Point p1, Point p2, Point p3, Point p4) {
122     return (ccw(p1,p2,p3)*ccw(p1,p2,p4) <= 0&&ccw(p3,p4,p1)*ccw(p3,p4,p2) <= 0 );
123 }
124
125 // a1,a2を端点とする線分とb1,b2を端点とする線分の交差判定
126 bool intersectSS(Segment s1, Segment s2) {
127     return intersectSS(s1.p1,s1.p2,s2.p1,s2.p2);
128 }
129
130 // 多角形と線分の交差判定
131 bool intersectPS(Polygon p,Segment l){
132     int n=p.size();
133     for(int i=0;i<n;i++)
134     if(intersectSS(Segment(p[i],p[(i+1)%n]),l)) return 1;
135     return 0;
136 }
137
138 // 円と円の交差判定
139 int intersectCC(Circle c1,Circle c2){
140     if(c1.r<c2.r) swap(c1,c2);
141     double d=abs(c1.c-c2.c);
142     double r=c1.r+c2.r;
143     if(equals(d,r)) return ICC_CIRCUMSCRIBE;
144     if(d>r) return ICC_SEPERATE;
145     if(equals(d+c2.r,c1.r)) return ICC_INSCRIBE;
146     if(d+c2.r<c1.r) return ICC_CONTAIN;
147     return ICC_INTERSECT;
148 }
149 // 直線と点の距離
150 double getDistanceLP(Line l, Point p) {
151     return abs(cross(l.p2-l.p1,p-l.p1)/abs(l.p2-l.p1));
152 }
153
154 // 線分と点の距離
155 double getDistanceSP(Segment s,Point p){
156     if(dot(s.p2-s.p1,p-s.p1) < 0.0 ) return abs(p-s.p1);
157     if(dot(s.p1-s.p2,p-s.p2) < 0.0 ) return abs(p-s.p2);
158     return getDistanceLP(s,p);
159 }
160
161 // 線分と円の交差判定
162 bool intersectSC(Segment s,Circle c){
163     return getDistanceSP(s,c.c)<=c.r;
164 }
165
166 //
167 int intersectCS(Circle c,Segment s){
168     if(norm(project(s,c.c)-c.c)-c.r*c.r>EPS) return 0;
169     double d1=abs(c.c-s.p1),d2=abs(c.c-s.p2);
170     if(d1<c.r+EPS&&d2<c.r+EPS) return 0;
171     if((d1<c.r-EPS&&d2>c.r+EPS)||(d1>c.r+EPS&&d2<c.r-EPS)) return 1;
172     Point h=project(s,c.c);
173     if(dot(s.p1-h,s.p2-h)<0) return 2;
174     return 0;
175 }
176
177
178
179
180 // 線分と線分の距離
```

```cpp
double getDistanceSS(Segment s1,Segment s2){
    if(intersectSS(s1,s2)) return 0.0;
    return min(min(getDistanceSP(s1,s2.p1),getDistanceSP(s1,s2.p2)),
    min(getDistanceSP(s2,s1.p1),getDistanceSP(s2,s1.p2)));
}

// a1,a2を端点とする線分とb1,b2を端点とする線分の交点計算
// 前提として交差していることが必要
P getCrossPointSS(P a1, P a2, P b1, P b2) {
    P b = b2-b1;
    double d1 = abs(cross(b, a1-b1));
    double d2 = abs(cross(b, a2-b1));
    double t = d1/(d1+d2);
    return a1+(a2-a1)*t;
}

// a1,a2を通る直線とb1,b2を通る直線の交点計算
// 前提として平行でないことが必要
Point getCrossPointLL(Line l1,Line l2){
    double a=cross(l1.p2-l1.p1,l2.p2-l2.p1);
    double b=cross(l1.p2-l1.p1,l1.p2-l2.p1);
    if(abs(a)<EPS&&abs(b)<EPS) return l2.p1;
    return l2.p1+(l2.p2-l2.p1)*(b/a);
}

// getCrossPointCCに必要
double arg(Point p){
    return atan2(p.y,p.x);
}

// getCrossPointCCに必要
Point polar(double a,double r) {
    return Point(cos(r)*a,sin(r)*a);
}

// 円と直線の交点
Polygon getCrossPointCL(Circle c,Line l){
    Polygon ps;
    Point pr=project(l,c.c);
    Point e=(l.p2-l.p1)/abs(l.p2-l.p1);
    if(equals(getDistanceLP(l,c.c),c.r)){
        ps.emplace_back(pr);
        return ps;
    }
    double base=sqrt(c.r*c.r-norm(pr-c.c));
    ps.emplace_back(pr+e*base);
    ps.emplace_back(pr-e*base);
    return ps;
}

// 円と線分の交点
Polygon getCrossPointCS(Circle c,Segment s){
    Line l(s);
    Polygon res=getCrossPointCL(c,l);
    if(intersectCS(c,s)==2) return res;
    if(res.size()>1u){
        if(dot(l.p1-res[0],l.p2-res[0])>0) swap(res[0],res[1]);
        res.pop_back();
    }
    return res;
```

```
}

// 円と円の交点
Polygon getCrossPointCC(Circle c1,Circle c2){
    Polygon p(2);
    double d=abs(c1.c-c2.c);
    double a=acos((c1.r*c1.r+d*d-c2.r*c2.r)/(2*c1.r*d));
    double t=arg(c2.c-c1.c);
    p[0]=c1.c+polar(c1.r,t+a);
    p[1]=c1.c+polar(c1.r,t-a);
    return p;
}

// 多角形の面積
double area(Polygon s){
    double res=0;
    for(int i=0;i<(int)s.size();i++){
        res+=cross(s[i],s[(i+1)%s.size()])/2.0;
    }
    return res;
}

// 凸多角形ですか？
bool isConvex(Polygon p){
    bool f=1;
    int n=p.size();
    for(int i=0;i<n;i++){
        int t=ccw(p[(i+n-1)%n],p[i],p[(i+1)%n]);
        f&=t!=CLOCKWISE;
    }
    return f;
}

// 多角形の中に点は含まれている？
// IN:2 ON:1 OUT:0
int contains(Polygon g,Point p){
    int n=g.size();
    bool x=false;
    for(int i=0;i<n;i++){
        Point a=g[i]-p,b=g[(i+1)%n]-p;
        if(fabs(cross(a,b)) < EPS && dot(a,b) < EPS) return 1;
        if(a.y>b.y) swap(a,b);
        if(a.y < EPS && EPS < b.y && cross(a,b) > EPS ) x = !x;
    }
    return (x?2:0);
}

// 凸包に使う
bool sort_x(Point a,Point b){
    return a.x!=b.x?a.x<b.x:a.y<b.y;
}

// 凸包に使う
bool sort_y(Point a,Point b){
    return a.y!=b.y?a.y<b.y:a.x<b.x;
}

// 凸包
Polygon convex_hull(Polygon ps){
    int n=ps.size();
```

```
301     sort(ps.begin(),ps.end(),sort_y);
302     int k=0;
303     Polygon qs(n*2);
304     for(int i=0;i<n;i++){
305         while(k>1&&cross(qs[k-1]-qs[k-2],ps[i]-qs[k-1])<0) k--;
306         qs[k++]=ps[i];
307     }
308     for(int i=n-2,t=k;i>=0;i--){
309         while(k>t&&cross(qs[k-1]-qs[k-2],ps[i]-qs[k-1])<0) k--;
310         qs[k++]=ps[i];
311     }
312     qs.resize(k-1);
313     return qs;
314 }
315
316 // 多角形の直径
317 double diameter(Polygon s){
318     Polygon p=s;
319     int n=p.size();
320     if(n==2) return abs(p[0]-p[1]);
321     int i=0,j=0;
322     for(int k=0;k<n;k++){
323         if(p[i]<p[k]) i=k;
324         if(!(p[j]<p[k])) j=k;
325     }
326     double res=0;
327     int si=i,sj=j;
328     while(i!=sj||j!=si){
329         res=max(res,abs(p[i]-p[j]));
330         if(cross(p[(i+1)%n]-p[i],p[(j+1)%n]-p[j])<0.0){
331         i=(i+1)%n;
332         }else{
333         j=(j+1)%n;
334         }
335     }
336     return res;
337 }
338
339 // 凸多角形を直線で切った時の左側
340 Polygon convexCut(Polygon p,Line l){
341     Polygon q;
342     for(int i=0;i<(int)p.size();i++){
343         Point a=p[i],b=p[(i+1)%p.size()];
344         if(ccw(l.p1,l.p2,a)!=-1) q.push_back(a);
345         if(ccw(l.p1,l.p2,a)*ccw(l.p1,l.p2,b)<0)
346         q.push_back(getCrossPointLL(Line(a,b),l));
347     }
348     return q;
349 }
350
351 // 円と点の接線
352 Polygon tangent(Circle c1,Point p2){
353     Circle c2=Circle(p2,sqrt(norm(c1.c-p2)-c1.r*c1.r));
354     Polygon p=getCrossPointCC(c1,c2);
355     sort(p.begin(),p.end());
356     return p;
357 }
358
359 // 円と円の接線
360 vector<Line> tangent(Circle c1,Circle c2){
```

```
    vector<Line> ls;
    if(c1.r<c2.r) swap(c1,c2);
    double g=norm(c1.c-c2.c);
    if(equals(g,0)) return ls;
    Point u=(c2.c-c1.c)/sqrt(g);
    Point v=orth(u);

    for(int s=1;s>=-1;s-=2) {
        double h=(c1.r+s*c2.r)/sqrt(g);
        if(equals(1-h*h,0)) {
            ls.emplace_back(c1.c+u*c1.r,c1.c+(u+v)*c1.r);
        }
        else if(1-h*h>0) {
            Point uu=u*h,vv=v*sqrt(1-h*h);
            ls.emplace_back(c1.c+(uu+vv)*c1.r,c2.c-(uu+vv)*c2.r*s);
            ls.emplace_back(c1.c+(uu-vv)*c1.r,c2.c-(uu-vv)*c2.r*s);
        }
    }

    return ls;
}

int main(int argc, char const *argv[]) {
    double x0,y0,r0,x1,y1,r1;
    cin>>x0>>y0>>r0>>x1>>y1>>r1;
    Circle c0(P(x0,y0),r0), c1(P(x1,y1),r1);
    vector<Line> ls = tangent(c0,c1);
    Polygon ans;

    for(auto&&e:ls) ans.emplace_back(getCrossPointCL(c0,e).front());
    sort(ans.begin(), ans.end());
    cout<<fixed<<setprecision(10);
    rep(i,ans.size()) {
        cout<<ans[i].x<<" "<<ans[i].y<<endl;
    }
    return 0;
}
```