Java Notes for Competitive Programming

Juan Pablo Paulsen C.
ACM-ICPC 2015 South America South - Chile 1st place
ACM-ICPC 2016 World Finals - Phuket, Thailand - Contestant
ACM-ICPC 2016 South America South - Chile 1st place
ACM-ICPC 2017 World Finals - Rapid City, SD, USA - Contestant
juanpablo.paulsen@gmail.com
November 22nd 2016
Latest version here

Index

Segment Tree

```
class ST{
	ST l,r;
	int i,j;
	long v;
	private ST(int x, long m) {
		i=j=x;
		v=m;
	}
	private ST(ST st1, ST st2) {
		l=st1;
		r=st2;
		i=st1.i;
		j=st2.j;
		v=f(st1.v,st2.v);
	}
	long update(int x, long v2){
		if (i==x && j==x) return v=v2;
		if (i>x || j<x) return v;
		return v=f(l.update(x,v2),r.update(x,v2));
	}
	long query(int a,int b){
		if (i>=a && j<=b) return v;
		if (i>b || j<a) return neutral();
		return f(l.query(a,b),r.query(a,b));
	}
	static long neutral(){
		return 1; //Change for neutral value
	}
	static long f(long x,long y){
		return x*y; //Change for some binary function
	}
	static ST build(long[] v){
		int length=1;
		while(length<v.length) length<<=1;
		ST[] array=new ST[length];
		for (int i = 0; i < v.length; i++) array[i]=new ST(i,v[i]);
		for (int i = v.length; i < length; i++) array[i]=new ST(i,neutral());
		for(length>>=1;length>0;length>>=1)
			for (int i = 0; i < length; i++) array[i]=new ST(array[2*i],array[2*i+1]);
		return array[0];
	}
}
```

Segment Tree with Indexes

```
class ST{
	ST l,r;
	int i,j;
	Pair v;
	private ST(int x, Pair m) {
		i=j=x;
		v=m;
	}
	private ST(ST st1, ST st2) {
		l=st1;
		r=st2;
		i=st1.i;
		j=st2.j;
		v=f(st1.v,st2.v);
	}
	Pair update(int x, long v2){
		if (i==x && j==x) return v=new Pair(x,v2);
		if (i>x || j<x) return v;
		return v=f(l.update(x,v2),r.update(x,v2));
	}
	Pair query(int a,int b){
		if (i>=a && j<=b) return v;
		if (i>b || j<a) return neutral();
		return f(l.query(a,b),r.query(a,b));
	}
	static Pair neutral(){
		return new Pair(-1,Long.MIN_VALUE); //Change for neutral value
	}
	static Pair f(Pair x,Pair y){
		if (x.v>y.v) return x; //Change for some binary function
		else return y;
	}
	static ST build(long[] v){
		int length=1;
		while(length<v.length) length<<=1;
		ST[] array=new ST[length];
		for (int i = 0; i < v.length; i++) array[i]=new ST(i,new Pair(i,v[i]));
		for (int i = v.length; i < length; i++) array[i]=new ST(i,neutral());
		for(length>>=1;length>0;length>>=1)
			for (int i = 0; i < length; i++) array[i]=new ST(array[2*i],array[2*i+1]);
		return array[0];
	}
}
class Pair{int i; long v; public Pair(int i, long v){this.i=i;this.v=v;}}
```

Fenwick Tree (Range-Queries Point-Updates)

```
class FT {
	long[] ft;
	public FT(int n) {
		ft=new long[n+1];
	}
	void set(int i, long v){
		add(i,v-query(i,i));
	}
	long query(int i, int j) {
		return query(j)-query(i-1);
	}
	long query(int i) {
		long ans=0;
		while (i>0) {
			ans+=ft[i];
			i-=i&-i;
		}
		return ans;
	}
	void add(int i, long v){
		while (i<ft.length) {
			ft[i]+=v;
			i+=i&-i;
		}
	}
}
```

Fenwick Tree (Point-Queries Range-Updates)

```
class FT{
	long[] ft;
	public FT(int n) {
		ft=new long[n+1];
	}
	long query(int i) {
		long ans=0;
		while (i>0) {
			ans+=ft[i];
			i-=i&-i;
		}
		return ans;
	}
	void add(int i, int j, long v){
		 add(i,v);
		 add(j+1,-v);
	}
	private void add(int i, long v){
		while (i<ft.length) {
			ft[i]+=v;
			i+=i&-i;
		}
	}
}
```

Fenwick Tree (Range-Queries Range-Updates)

```
class FT{
	long[] A;
	long[] B;
	void add(int i, int j, long v){
		add(A,i,v);
		add(A,j+1,-v);
		add(B,i,v*(i-1));
		add(B,j+1,-v*j);
	}
	long query(int i){
		return query(A,i)*i-query(B,i);
	}
	long query(int i, int j){
		return query(j)-query(i-1);
	}
	private void add(long[] ft, int i, long v){
		while (i<ft.length) {
			ft[i]+=v;
			i+=i&-i;
		}
	}
	private long query(long[] ft, int i) {
		long ans=0;
		while (i>0) {
			ans+=ft[i];
			i-=i&-i;
		}
		return ans;
	}
}
```

Fenwick Tree 2D

```
class FT2D {
	long[][] ft;
	public FT2D(int r, int c) {
		ft=new long[r+1][c+1];
	}
	void add(int r, int c, long value){
		for (int i=r; i<ft.length;i|=i+1)
			for (int j=c;j<ft[0].length;j|=j+1)
				ft[i][j]+=value;
	}
	long sum(int r, int c){
		long ans=0;
		for (int i=r;i>= 0;i=(i&(i + 1))-1)
			for (int j=c;j>=0;j=(j&(j+1))-1)
				ans+=ft[i][j];
		return ans;
	}
	long sum(int r1, int c1, int r2, int c2){
		return sum(r2,c2)-sum(r1-1,c2)-sum(r2,c1-1)+sum(r1-1,c1-1);
	}
	long get(int r, int c){
		return sum(r,c,r,c);
	}
	void set(int r, int c, int value){
		add(r,c,value-get(r,c));
	}
}
```

Lowest Common Ancestor

```
class LCA{
	ArrayList<Integer>[] graph;
	ArrayList<Long>[] cost;
	int n,m;
	int P[][];
	int prof[];
	long dist[];
	public LCA(ArrayList<Integer>[] graph, ArrayList<Long>[] cost, int root){
		this.graph=graph;
		this.cost=cost;
		n=graph.length;
		m=log2(n)+1;
		P=new int[n][m+1];
		prof=new int[n];
		dist=new long[n];
		P[root][0]=root;
		dfs(root);
		for (int i=0;i<n;i++) {
			for (int j=1; j<m;j++) {
				P[i][j]=P[P[i][j-1]][j-1];
			}
		}
	}
	int lca(int a, int b) {
		if (prof[a] < prof[b]) {
			int aux=a;
			a=b;
			b=aux;
		}
		a=ancDist(a,prof[a]-prof[b]);
		if (a==b) return a;
		for (int i=m;i>=0;i--) {
			if (P[a][i]!=P[b][i]) {
				a=P[a][i];
				b=P[b][i];
			}
		}
		return P[a][0];
	}
	int edgesBetween(int i, int j){
		return prof[j]+prof[i]-2*prof[lca(i,j)];
	}
	long distance(int i, int j) {
		return dist[j]+dist[i]-2*dist[lca(i,j)];
	}
```

```
        // k starts in 0 to |path|-1
        int kth(int i,int j, int k){
                int lca=lca(i,j);
                int left=prof[i]-prof[lca]+1;
                if (k<left) return ancDist(i,k);
                else return ancDist(j,edgesBetween(i, j)-k);
        }
        private void dfs(int s) {
                ArrayDeque<Integer> q=new ArrayDeque<Integer>();
                q.add(s);
                while(!q.isEmpty()){
                        int u=q.pollLast();
                        for (int i = 0; i < graph[u].size(); i++) {
                                int v=graph[u].get(i);
                                if (v != P[u][0]) {
                                        P[v][0]=u;
                                        prof[v]=prof[u]+1;
                                        dist[v]=dist[u]+cost[u].get(i);
                                        q.add(v);
                                }
                        }
                }
        }
        private int ancDist(int a, int dist) {
                for (int i = 0; i < m; i++) {
                        if (((1<<i)&dist)!=0) a=P[a][i];
                }
                return a;
        }
        private static int log2(long n){
           return 63-Long.numberOfLeadingZeros(n);
        }
}
```

Graph - Out Degree=1

```
class Tree{
	int n;
	Node[] nodes;
	int[] d;
	public Tree(int[] graph){
		n=graph.length;
		nodes=new Node[n];
		for (int i = 0; i < n; i++) {
			nodes[i]=new Node(i);
		}
		for (int i = 0; i < n; i++) {
			nodes[i].add(nodes[graph[i]]);
		}
		d=new int[n];
		Arrays.fill(d, -1);
		int size=0;
		for (int i = 0; i < n; i++) {
			if (d[i]==-1){
				bfs(i,size);
				size++;
			}

		}
	}
	void bfs(int s, int comp){
		Node n=nodes[s];
		int pos=0;
		while(true){
			if (d[n.i]!=-1){
				cycle(n.i,comp,pos-d[n.i]);
				break;
			}
			else{
				d[n.i]=pos;
				n=n.next;
				pos++;
			}
		}
	}
	private void cycle(int s, int comp,int length) {
		Node first=nodes[s];
		first.root=first;
		Node aux=first.next;
		for (int pos=0;;pos++){
			aux.tree(pos,comp,length,d);
```

```
                        if(aux==first) break;
                        aux=aux.next;
                }
        }
        public int distance(int u,int v){
                Node n1=nodes[u];
                Node n2=nodes[v];
                if (n1.comp!=n2.comp) return -1;
                else if (n1.root.lca==n2.root.lca){
                        return n1.root.lca.edgesBetween(n1.tree_i,n2.tree_i);
                }
                else return n1.prof+n2.prof+n1.root.distCycle(n2.root);
        }
}
class Node{
        int i,size,prof,tree_i,comp,pos,length;
        Node next;
        Node root;
        ArrayList<Integer>[] graph;
        LCA lca;
        ArrayList<Node> neighs;
        public Node(int i){
                this.i=i;
                this.size=0;
                neighs=new ArrayList<Node>();
        }
        public int distCycle(Node o) {
                int pos1=pos;
                int pos2=o.pos;
                if (pos2<pos1){
                        int aux=pos1;
                        pos1=pos2;
                        pos2=aux;
                }
                int length=this.length;
                return Math.min(pos2-pos1, length-pos2+pos1);
        }
        public void tree(int pos, int comp, int length, int[] d) {
                update(this,0,comp,pos,length);
                ArrayDeque<Node> q=new ArrayDeque<Node>();
                q.add(this);
                while(!q.isEmpty()){
                        Node u=q.pollFirst();
                        d[u.i]=1;
                        for(Node v: u.neighs){
                                if (v.root!=null) continue;
                                v.update(this, u.prof+1, comp, pos, length);
                                q.add(v);
```

```
                        }
                }
                graph=new ArrayList[size];
                for (int i = 0; i < size; i++) {
                        graph[i]=new ArrayList<Integer>();
                }
                q.add(this);
                while(!q.isEmpty()){
                        Node u=q.pollFirst();
                        for(Node v: u.neighs){
                                if (v.root!=this) continue;
                                graph[u.tree_i].add(v.tree_i);
                                q.add(v);
                        }
                }
                lca=new LCA(graph,0);
        }
        public void add(Node o) {
                next=o;
                o.neighs.add(this);
        }
        public void update(Node root, int prof,int comp, int pos,int length){
                this.root=root;
                this.prof=prof;
                this.tree_i=root.size;
                root.size++;
                this.comp=comp;
                this.pos=pos;
                this.length=length;
        }
}
```

BFS/DFS/Dijkstra

```
class Graph{
	ArrayList<Integer>[] graph;
	ArrayList<Long>[] cost;
	long[] d;
	public Graph(ArrayList<Integer>[] graph, ArrayList<Long>[] cost){
		this.graph=graph;
		this.cost=cost;
		this.d=new long[graph.length];
	}
	void bfs(int s) {
		Arrays.fill(d, Long.MAX_VALUE);
		ArrayDeque<Par> q=new ArrayDeque<Par>();
		q.add(new Par(s,0));
		while(!q.isEmpty()){
			Par p=q.pollFirst(); //bfs
			//Par p=q.pollLast(); //dfs
			if(d[p.u]!=Long.MAX_VALUE) continue;
			d[p.u]=p.d;
			for (int u:graph[p.u]) {
				if(d[u]==Long.MAX_VALUE) q.add(new Par(u,p.d+1));
			}
		}
	}
	void dijkstra(int s) {
		Arrays.fill(d, Long.MAX_VALUE); //Remove if necessary
		PriorityQueue<Par> q=new PriorityQueue<Par>();
		q.add(new Par(s,0));
		while(!q.isEmpty()){
			Par p=q.poll();
			if (d[p.u]!=Long.MAX_VALUE) continue;
			d[p.u]=p.d;
			for (int i=0;i<graph[p.u].size();i++) {
				int u=graph[p.u].get(i);
				long c=cost[p.u].get(i);
				if(d[u]==Long.MAX_VALUE) q.add(new Par(u,p.d+c));
			}
		}
	}
}
class Par implements Comparable<Par>{
	int u; long d;
	public Par(int u, long d){this.u=u;this.d=d;}
	@Override
	public int compareTo(Par o){return Long.compare(d, o.d);}
}
```

BFS/DFS Graph in a 2D-Map

```
class Graph2d{
	int n;
	int m;
	int w;
	int[][] map;
	boolean[][] marked;
	public Graph2d(int[][] map, int n, int m, int w){
		this.map=map;
		this.n=n;
		this.m=m;
		this.w=w;
	}
	void bfs(Par s) {
		ArrayDeque<Par> q=new ArrayDeque<Par>();
		q.add(s);
		while(!q.isEmpty()){
			Par p=q.pollFirst(); //bfs
			//Par p=q.pollLast(); //dfs
			if (!check(p.x,p.y)) continue;
			marked[p.x][p.y]=true;
			add(q,p,new Par(p.x+2,p.y));
			add(q,p,new Par(p.x-2,p.y));
			add(q,p,new Par(p.x,p.y+2));
			add(q,p,new Par(p.x,p.y-2));
		}
	}
	void add(ArrayDeque<Par> q, Par p1, Par p2) {
		int x=(p1.x+p2.x)>>1;
		int y=(p1.y+p2.y)>>1;
		if (check(x,y)) q.add(p2);
	}
	boolean check(int x,int y){
		return !(x<0 || x>=n || y<0 || y>=m || map[x][y]>=w || marked[x][y]);
	}
}
class Par{
	int x,y;
	public Par(int x, int y){
		this.x=x;this.y=y;
	}
}
```

Floyd-Warshall

```
class Graph{
	ArrayList<Integer>[] graph;
	ArrayList<Integer>[] cost;
	long[][] d;
	public Graph(ArrayList<Integer>[] graph, ArrayList<Integer>[] cost){
		this.graph=graph;
		this.cost=cost;
	}
	void floydWarshall() {
		int n=graph.length;
		d=new long[n][n];
		for (int i = 0; i < n; i++) Arrays.fill(d[i],Long.MAX_VALUE);
		for (int i=0;i<n;i++) d[i][i]=0;
		for (int i=0;i<n;i++){
			for (int j=0;j<graph[i].size();j++){
				int v=graph[i].get(j);
				d[i][v]=cost[i].get(j);
			}
		}
		for (int k=0;k<n;k++){
			for (int i=0;i<n;i++){
				for (int j=0;j<n;j++){
					d[i][j]=Math.min(d[i][j],d[i][k]+d[k][j]);
				}
			}
		}
	}
}
```

Bellman-Ford

```
class Graph{
	ArrayList<Integer>[] graph;
	ArrayList<Integer>[] cost;
	long[] d;
	int[] pre;
	public Graph(ArrayList<Integer>[] graph, ArrayList<Integer>[] cost){
		this.graph=graph;
		this.cost=cost;
	}
	boolean bellmanFord(int s) {
		int n=graph.length;
		d=new long[n];
		pre=new int[n];
		Arrays.fill(d, Integer.MAX_VALUE);
		Arrays.fill(pre, -1);
		d[s]=0;
		for (int i = 1; i < n; i++) {
			for (int u = 0; u < n; u++) {
				for (int k=0;k<graph[u].size();k++) {
					int v=graph[u].get(k);
					long c=cost[u].get(k);
					if (d[v]>d[u]+c){
						d[v]=d[u]+c;
						pre[v]=u;
					}
				}
			}
		}
		for (int u = 0; u < n; u++) {
			for (int i=0;i<graph[u].size();i++) {
				int v=graph[u].get(i);
				long c=cost[u].get(i);
				if (d[v]>d[u]+c){
					return false;
				}
			}
		}
		return true;
	}
}
```

Dinic

```
class Dinic {
	int n, sink;
	int[] q, work;
	long[] dist;
	ArrayList<Edge>[] graph;
	public Dinic(int n) {
		this.n=n; graph=new ArrayList[n];
		for (int i=0;i<n;i++) graph[i]=new ArrayList<Edge>();
		dist=new long[n]; q=new int[n];
	}
	boolean bfs(int start, int finish) {
		dist=new long[n];
		Arrays.fill(dist, -1);
		dist[start]=0;
		int head=0,tail=0;
		q[tail++]=start;
		while (head<tail){
			int u=q[head++];
			for (Edge e: graph[u]){
				int v=e.to;
				if (dist[v]==-1 && e.f<e.cap){
					dist[v]=dist[u]+1;
					q[tail++]=v;
				}
			}
		}
		return dist[finish]!=-1;
	}
	long dfs(int u, long f) {
		if (u == sink)
			return f;
		for (int i = work[u]; i < graph[u].size(); i++) {
			Edge e = graph[u].get(i);
			int v = e.to;
			if (e.cap <= e.f || dist[v] != dist[u] + 1)
				continue;
			long df = dfs(v, Math.min(f, e.cap - e.f));
			if (df > 0) {
				e.f += df;
				graph[v].get(e.rev).f -= df;
				return df;
			}
		}
		return 0;
	}
```

```
        void add_edge(int u, int v, long cap) {
                Edge a = new Edge(v, graph[v].size(), 0, cap);
                Edge b = new Edge(u, graph[u].size(), 0, cap);
                graph[u].add(a);
                graph[v].add(b);
        }
        long max_flow(int source, int dest) {
                sink = dest;
                long ans = 0;
                while (bfs(source, dest)) {
                        work=new int[n];
                        while (true){
                                long delta = dfs(source, Long.MAX_VALUE);
                                if (delta==0) break;
                                ans += delta;
                        }
                }
                return ans;
        }
        static class Edge {
                int to, rev;
                long f, cap;
                public Edge(int to, int rev, long f, long cap) {
                        this.to=to;this.rev=rev;this.f=f;this.cap=cap;
                }
        }
}
```

Rolling Hashing

```
class RH{
	static long b=127, m1=1000000007, m2=1000000009;
	static int l=10000001;
	static long[] bPow1, bPow2;
	char[] w;
	long[] h1, h2;
	public RH(char[] word) {
		w=word;
		h1=new long[w.length+1]; h2=new long[w.length+1];
		for (int i=w.length-1;i>=0;i--) {
			h1[i]=(h1[i+1]*b%m1+w[i])%m1; h2[i]=(h2[i+1]*b%m2+w[i])%m2;
		}
	}
	static void fillPow() { //Remember to call this function once at the beginning
		bPow1=new long[l]; bPow2=new long[l];
		bPow1[0]=1; bPow2[0]=1;
		for (int i = 1; i < l; i++) {
			bPow1[i]=bPow1[i-1]*b%m1; bPow2[i]=bPow2[i-1]*b%m2;
		}
	}
	long hash(int i, int j) { return hash1(i,j)<<32 | hash2(i,j); }
	private long hash1(int i, int j) { return (h1[i]-(h1[j+1]*bPow1[j-i+1]%m1)+m1)%m1; }
	private long hash2(int i, int j) { return (h2[i]-(h2[j+1]*bPow2[j-i+1]%m2)+m2)%m2; }
	boolean equal(int i1, int j1,RH o, int i2, int j2) {
		return hash(i1,j1)==o.hash(i2,j2);
	}
	long concat(int i1, int j1,RH o, int i2, int j2) {
		long c1=(hash1(i1,j1)+o.hash1(i2,j2)*bPow1[j1-i1+1])%m1;
		long c2=(hash2(i1,j1)+o.hash2(i2,j2)*bPow2[j1-i1+1])%m2;
		return c1<<32 | c2;
	}
	int compare(int i1, int j1, RH o, int i2, int j2){
		int n1=j1-i1+1; int n2=j2-i2+1;
		int ans=compare(i1,o,i2,Math.min(n1, n2));
		if (ans==0) ans=Integer.compare(n1, n2);
		return ans;
	}
	int compare(int i1, RH o,int i2, int n){
		if (n==1) return Character.compare(w[i1], o.w[i2]);
		int mid=(n-1)/2;
		if (equal(i1,i1+mid,o,i2,i2+mid)) return compare(i1+mid+1,o,i2+mid+1,n-mid-1);
		else return compare(i1,o,i2,mid+1);
	}
}
```

Lines

```
class Lines{
        ArrayList<Integer>[] h;
        ArrayList<Integer>[] v;
        ArrayList<Integer>[] s;
        ArrayList<Integer>[] bs;
        public Lines(int[][] values, int r, int c){
                h=new ArrayList[r];  //horizontal lines ( - )
                v=new ArrayList[c];  //vertical lines ( | )
                s=new ArrayList[r+c-1]; //slash lines ( / )
                bs=new ArrayList[r+c-1]; //backslash lines ( \ )
                for (int i=0;i<r;i++) h[i]=new ArrayList<Integer>(c);
                for (int i=0;i<c;i++) v[i]=new ArrayList<Integer>(r);
                for (int i=0;i<r+c-1;i++) s[i]=new ArrayList<Integer>(Math.min(r,c));
                for (int i=0;i<r+c-1;i++) bs[i]=new ArrayList<Integer>(Math.min(r,c));
                for (int i=0;i<r;i++) {
                        for (int j = 0; j < c; j++) {
                                h[i].add(values[i][j]);
                                v[j].add(values[i][j]);
                                s[i+j].add(values[i][j]);
                                bs[i-j+c-1].add(values[i][j]);
                        }
                }
        }
}
```

Geometry

```
class Geometry {
	static double area(ArrayList<Point> polygon) {
		long ans=0; //Change to double if necessary
		int j=polygon.size()-1;
		for (int i=0; i<polygon.size(); i++){
			Point p1=polygon.get(i);
			Point p2=polygon.get(j);
			ans+=(p2.x+p1.x)*(p2.y-p1.y);
			j=i;
		}
		return Math.abs(ans/2.0);
	}
}
class Point{
	long x,y;
	public Point(long x, long y){
		this.x=x;
		this.y=y;
	}
}
```

2D Vector

```
class Segment{
	long x1,y1,x2,y2;
	public Segment(long x1, long y1, long x2, long y2){
		this.x1=x1;this.y1=y1;this.x2=x2;this.y2=y2;
		//m=(y2-y1)/(x2-x1);
		//n=-m*x1+y1;
	}
	long cross(long x, long y){return (x2-x1)*(y-y1)-(y2-y1)*(x-x1);}

	long dot(Segment s){return (x2-x1)*(s.x2-s.x1)+(y2-y1)*(s.y2-s.y1);}

	long sqNorm(){return (y2-y1)*(y2-y1)+(x2-x1)*(x2-x1);}

	double norm(){return Math.sqrt(sqNorm());}

	Segment perp(){return new Segment(x1,y1,x1+y2-y1,y1+x1-x2);}

	Segment antiPerp(){return new Segment(x2,y2,x2+y1-y2,y2+x2-x1);}

	Segment product(long v){return new Segment(x1,y1,(x2-x1)*v+x1,(y2-y1)*v+y1);}

	//Segment proyection(Segment s){return product(dot(s)/norm());} //Doubles

	//Segment unit(){return product(1/norm());} //Doubles

	double distanceTo(long x, long y){
		if (perp().cross(x,y)>0 && antiPerp().cross(x,y)>0){
			return Math.abs(perp().dot(new Segment(x1,y1,x,y)))/norm();
			//return perp().proyection(new Segment(x1,y1,x,y)).norm();
		}
		else{
			Segment s1=new Segment(x1,y1,x,y);
			Segment s2=new Segment(x2,y2,x,y);
			return Math.min(s1.norm(), s2.norm());
		}
	}
	//void intersect(Segment s, boolean b){
		if (cross(s.x1, s.y1)<=0 && cross(s.x2, s.y2)>0){
			double x=(s.n-n)/(m-s.m);
			double y=m*x+n;
			if (x>x1 && x<=x2 && x>=s.x1 && x<s.x2)
				ints.add(new Intersection(x, y, s));
		}
	}
}
```

Sweep Line

```
class Segment implements Comparable<Segment>{
	double x1,y1,x2,y2;
	public Segment(double x1, double y1, double x2, double y2){
		this.x1=x1;this.y1=y1;this.x2=x2;this.y2=y2;
	}
	double cross(double x, double y){return (x2-x1)*(y-y1)-(y2-y1)*(x-x1);}
	@Override
	public int compareTo(Segment s) {
		if(x1>s.x1) return -1*s.compareTo(this);
		return Double.compare(0,cross(s.x1,s.y1));
	}
}
```

Sweep Circle

```
class Segment implements Comparable<Segment>{
	static long xs,ys;
	long x1,y1,x2,y2;
	public Segment(long x1, long y1, long x2, long y2){
		this.x1=x1;this.y1=y1;this.x2=x2;this.y2=y2;
	}
	long cross(long x, long y){return (x2-x1)*(y-y1)-(y2-y1)*(x-x1);}
	@Override
	public int compareTo(Segment s) {
		Segment s1=new Segment(xs,ys,x1,y1);
		if (s1.cross(s.x1, s.y1)>0) return -1*s.compareTo(this); //Clockwise sense
		return Long.compare(0, cross(s.x1,s.y1));
	}
}
class Event implements Comparable<Event>{
	long x,y;
	Segment s;
	boolean open;
	public Event(long x, long y){
		this(x,y,null,true);
	}
	public Event(long x, long y, Segment s, boolean open){
		this.x=x;
		this.y=y;
		this.s=s;
		this.open=open;
	}
	public int getCuad() {
		long xs=Segment.xs;
		long ys=Segment.ys;
		Segment v=new Segment(xs,ys,xs,ys+1);
		if (v.cross(x,y)<=0) return 0;
		else return 1;
	}
	@Override
	public int compareTo(Event o) {
		int ans=Integer.compare(getCuad(),o.getCuad());
		if (ans==0){
			Segment s=new Segment(Segment.xs,Segment.ys,x,y);
			return Long.compare(s.cross(o.x, o.y),0);
		}
		return ans;
	}
}
```

3D Vector

```
class Segment{
	double x,y,z;
	public Segment(double x, double y, double z){
		this.x=x;
		this.y=y;
		this.z=z;
	}
	Segment cross(Segment s){
		double x=this.y*s.z-this.z*s.y;
		double y=this.z*s.x-this.x*s.z;
		double z=this.x*s.y-this.y*s.x;
		return new Segment(x,y,z);
	}
	double dot(Segment s){
		return x*s.x+y*s.y+z*s.z;
	}
	double sqNorm(){
		return x*x+y*y+z*z;
	}
	double norm(){
		return Math.sqrt(sqNorm());
	}
	Segment product(double v){
		return new Segment(x*v,y*v,z*v);
	}
	Segment proyection(Segment s){
		return product(dot(s)/sqNorm());
	}
	Segment unit(){
		return product(1/norm());
	}
	double distanceToPlane(Segment s){
		return s.proyection(this).norm();
		//return s.unit().product(s.unit().point(this)).norm();
		//return s.point(this)/s.sqNorm();
	}
	double distanceToPoint(Segment s){
		return new Segment(s.x-x,s.y-y,s.z-z).norm();
	}
}
```