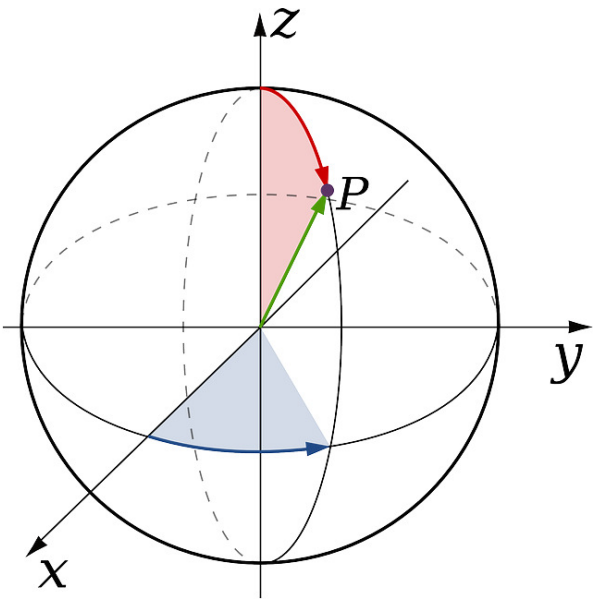
12 ARVUTUSGEOMEETRIA



Viimases peatükis tuleme mõnede teistega võrreldes väga vana valdkonna juurde, milleks on geomeetria. Geomeetrial on seoses maamõõtmise ja ehitusega väga praktilised väljundid ning sellele pandi alus juba vanas Kreekas, kuuendal sajandil enne meie aega. Kolmandal sajandil e.m.a lõi Eukleides oma teosega „Elemendid“ distsipliinile ka formaalsema raamistiku. Eukleidese töö seadis ka paljudeks järgnevateks sajanditeks standardi, millele kogu seonduv uurimistöö tugines.

Nagu ka teiste matemaatilise taustaga ülesannete puhul, annab arvuti kasutamine meile võimaluse käsitleda palju suuremaid andmehulki ja teha nendega rohkem arvutusi. Praeguses kontekstis tähendab see võimalust uurida suuremaid punktide, lõikude ja muude kujundite hulki kui see paberi ja pliiatsiga võimalik oleks.

Arvutiga lahendatavad geomeetriaülesanded võibki jagada laias laastus kaheks: ühed on sellised, mida on siiski realistlik enne paberi ja pliiatsiga lahendada ja edasi on ka küllaltki lihtne kirjutada programm, mis seejärel seda tüüpi ülesannet lahendada oskab. Teised on sellised ülesanded, mis nõuavad keerulisemaid algoritme – neid nimetataksegi arvutusgeomeetria ülesanneteks.

Arvutusgeomeetria ülesanded nõuavad enamasti nii head algoritmitundmist kui ka väga täpselt programmeerimist. Võrreldes paljude teiste ülesandeklassidega tuleb arvestada paljude erijuhtudega, mis teeb maksimumpunktidele lahendamise vaevalisemaks. Näiteks:

- Mis siis, kui jooned on paralleelsed?
- Mis siis, kui punktid kattuvad?
- Mis siis, kui hulknurk ei ole kumer?

Täiendavat keerukust lisab see, et tehteid tuleb enamasti teha ujukomaarvudega, mis toob kaasa vajaduse arvestada arvutuste täpsusest tulenevate probleemidega.

12.1 GEOMEETRILISED OBJEKTID

12.1.1 Punkt

Punkt on lihtsaim geomeetria objekt, sellel puuduvad mõõtmed ja seda iseloomustab ainult asukoht. Tasapinnal saab selle edasi anda kahe koordinaadiga, mida enamasti tähistatakse tähtedega x ja y. Seega on ka programmeerimises vaja, et säilitame iga punkti kohta tema asukoha ehk koordinaadid. Punkti tähistame $P(x, y)$.

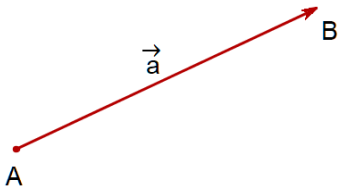
12.1.2 Lõik

Lõik (i.k. l*ine segment*) on kaht punkti ühendav sirgjoon. Neid punkte nimetatakse lõigu otspunktideks. Lõik on üheselt määratud oma otspunktidega. Lõigul on pikkus ja siht. Lõigud on näiteks kolmnurga küljed. Lõiku tähistame AB,kus A ja B on lõigu otspunktid. $AB = BA$.

12.1.3 Vektor

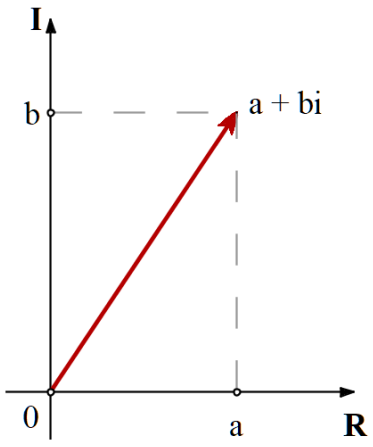
Vektor (i.k *vector*) on suunatud lõik, sellel on pikkus, siht ja suund. Kui kõik kolm parameetrit ühtivad, on vektorid võrdsed. Vektoreid, mis on samal sirgel, nimetatakse **kollineaarseteks**. Vektorit tähistame $\overrightarrow{AB}$. $\overrightarrow{AB} = -\overrightarrow{BA}$.

Vektori määrab üheselt selle algus ja lõpp-punkt. Abstraktsemalt saab aga kõik vektorid esitada ka ühtsest alguspunktist (enamasti siis koordinaatide alguspunktist). Sellisel juhul piisab vektori määramiseks ainult lõpp-punktist.



Vektorid on peamiselt huvitavad just seetõttu, et neil on suund ja nendega saab teha tehteid – neid saab liita, lahutada, korrutada arvuga ja teise vektoriga.

Punkti asukohta saab määrata ka tema kaugusega koordinaatide alguspunktist ja nurgaga x-telje suhtes. Sellisel moel on üles ehitatud kompleksarvud ja seega saab punktide hoidmiseks kasutada ka kompleksarvu klassi, kui selline on su kelle teegis defineeritud. Siin on väike ülevaade, mida saab teha keeles C++:



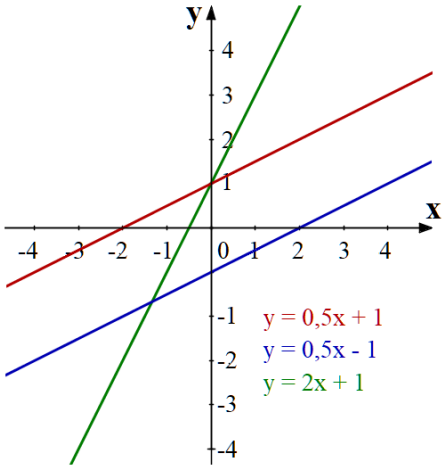
Kompleksarvu reaalosa a on punkti x koordinaadiks ja imaginaarosa kordaja b punkti y koordinaadiks. Kui meil on kaks punkti u ja v, siis $u + v$ on vektorite summa, mille alguspunkt on (0,0) ja lõpp-punktid vastavalt u ja v.
`abs(u)` on vektori u pikkus. Seda saab kasutada ka kahe punkti vahelise kauguse leidmiseks: punktide u ja v omavaheline kaugus on `abs(v-u)`.

`arg(u)` arvutab vektori u nurga x-telje suhtes radiaanides.

`polar(s, a)` loob vektori pikkusega s ja nurgaga a. Vektorit saab keerata a kraadi, korrutades selle vektoriga, mille pikkus on 1 ja nurk a: `u*polar(1,a)`, keerab vektorit u a kraadi.

12.1.4 Sirge

Sirge e sirgjoon on lihtsaim kahemõõtmeline objekt, see on kõverusteta joon, mis on lühim iga oma kahe punkti vahel. Sirge saab edasi anda sirge võrrandiga: $ax + by = c$. Iga punkt asukohaga (x, y), mis rahuldab seda võrrandit, asub antud sirgel ja ka vastupidi: iga sirgel asuv punkt rahuldab seda võrrandit.



Ühte punkti läbib lõpmata palju sirgeid, kuid kaks punkti määravad sirge üheselt. Kahe punkti järgi saab leida sirge võrrandi. Oletame, et meil on kaks punkti $P = (x_1, y_1)$ ja $Q = (x_2, y_2)$. Kuna need asuvad samal sirgel, siis $ax_1 + by_1 = c$ ja $ax_2 + by_2 = c$. Siit

$$ax_1 + by_1 = ax_2 + by_2 \Rightarrow a(x_1 - x_2) = b(y_2 - y_1).$$

See võrdus kehtib siis, kui valime a väärtuseks b kordaja ja vastupidi, st $a = (y_2 - y_1)$ ja $b = (x_1 - x_2)$.

Funktsioon, mis seda arvutab, on otsene:

```
void punktid_sirgeks(pair<double, double> P, pair<double, double> Q)
{
	double a = Q.second - P.second;
	double b = P.first - Q.first;
	double c = a*(P.first) + b*(P.second);

	if (b<0) {
		cout << a << "x " << b << "y = " << c << endl;
	}
	else{
		cout << a << "x + " << b << "y = " << c << endl;
	}
}
```

Sirget saab esitada ka ühe punkti ja **kaldenurga** kaudu.

12.1.5 Kiir

Kiir (i.k *ray*) on sirgjoon, millel on alguspunkt, kuid ei ole lõpp-punkti. Kiir tekib näiteks ühe lõigu pikendamisel lõpmatuseni üle ühe otspunkti. Programmeerimises tähendab see üldiselt lõiku, mille lõpp-punkt on piisavalt kaugel.

12.1.6 Murdjoon

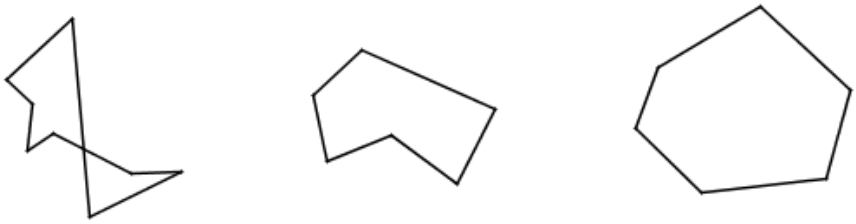
Murdjoon (i.k *polygonal chain*) on selline joon, mis koosneb järjestikustest, paarikaupa ühise otspunktiga lõikudest ehk lülidest $A_1A_2, A_2A_3, \dots, A_{n-1}A_n$. Murdjoone lülide otspunktid $A_1 \dots A_n$ on kõik erinevad. Kui $A_1 = A_n$, on tegemist kinnise murdjoonega. Murdjoont, mille ükski lüli ei lõiku, nimetatakse lihtsaks murdjooneks.

Murdjoont on mõistlik esitada punktiloendina.

12.1.7 Hulknurk

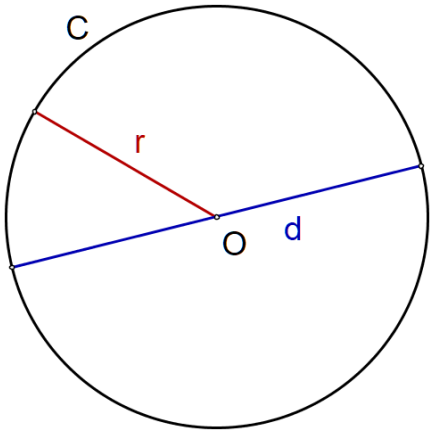
Hulknurk (i.k. *polygon*)on kinnine murdjoon. Lõike, millest murdjoon koosneb, nimetatakse külgedeks ja nende otspunkte tippudeks. Hulknurka, mille moodustab kinnine lihtne murdjoon, nimetatakse lihtsaks hulknurgaks. Kui hulknurga mistahes külje pikendamine mõlemas suunas lõpmatuseni ei lõika hulknurga ühtki külge, nimetatakse hulknurka **kumeraks**.

Järgmisel joonisel on ennast lõikav hulknurk, lihtne hulknurk ja kumer hulknurk:



12.1.8 Ringjoon

Ringjooneks (i.k. *circle*) nimetatakse sellist joont, mille iga punkt asub ühest punktist samal kaugusel. Seda punkti nimetatakse ringjoone **keskpunktiks** ja joone kaugust sellest nimetatakse ringjoone **raadiuseks**.



Keskpunkt ja raadius määravad ringjoone üheselt.

Ringjoone võrrand avaldub kujul

$$(x-a)^2+(y-b)^2=r^2,$$

Kus (a,b) on ringjoone keskpunkt ja r raadius.

Ringi ümbermõõdu ja pindala valemid on ehk kõigile tuttavad:

$$C=2\pi r$$

$$S=\pi r^2$$

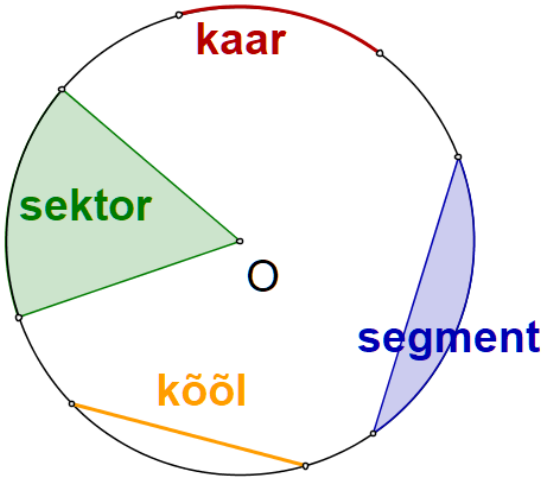
Programmeerimisülesannete lahendamisel on oluline mitte hakata π-d ise ümardama, vaid kasutada olemasolevat konstanti või määrata π trigonomeetria abil: $\pi=\mathrm{acos}(-1)$.

12.1.9 Kaar, kõõl, sektor ja segment

Kaareks (i.k. *arc*) nimetatakse üht pidevat osa ringjoonest. Kaarel on otspunktid, mis asuvad samuti ringjoonel. Kui tõmmata lõigud kaare kummastki otspunktist ringi keskpunkti, siis nende lõikude omavaheline nurk aitab leida kaare pikkuse:

$$P = \frac{\alpha}{360} \cdot C$$

kus α on eelpool kirjeldatud lõikude vaheline nurk ja C ringjoone pikkus.



Kõõluks (i.k *chord*) nimetatakse lõiku, mis ühendab kaks ringjoonel asuvat punkti omavahel.

Sektoriks nimetatakse ala ringist, mis on piiratud kaarega ja kaare otspunktidest tõmmatud raadiustega. Sektori pindala saab leida sarnaselt kaare pikkusele:

$$S = \frac{\alpha}{360} \cdot A$$

Kus A on ringi pindala.

Segment on osa ringist, mis on piiratud kõõluga ja kõõlu otspunktide vahel asuva kaarega. Segmendi pindala avaldub sektori pindala ja võrdhaarse kolmnurga, mille haaradeks on raadiused ning aluseks kõõl, vahena.

Ringjoone **puutujaks** (i.k *tangent*) nimetatakse sirget, mil on ringjoonega täpselt üks ühine punkt.

12.2 OPERATSIOONE PUNKTIDE JA SIRGETEGA

12.2.1 Punkti pööramine tasandil

Punti pööramiseks θ kraadi vastupäeva, saab kasutada seost:

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \times \begin{bmatrix} x \\ y \end{bmatrix}$$

12.2.2 Kahe punkti vaheline kaugus

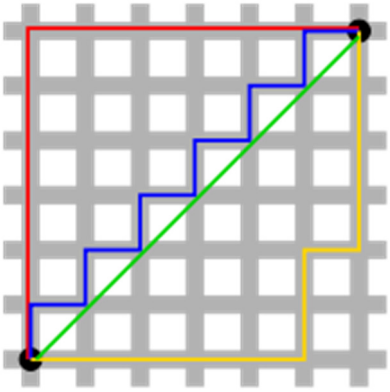
Kahe punkti vahelise kauguse tasandil saab leida Pythagorase teoreemi abil:

$$s = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

C++ on kõige mugavam kasutada funktsiooni hypot, mis just seda teebki:

```
int main()
{
	pair<double, double> p1(1,2), p2(3,4);
	double s = hypot(p2.first - p1.first, p2.second - p1.second);
	return 0;
}
```

Lisaks sellele kasutatakse ülesannetes vahetevahel nn **Manhattani kaugust**. Nimetus tuleb sellest, et Manhattanil on tänavad ilusti kas põhja-lõuna või ida-lääne suunalised ja ühest punktist teise saamiseks saab kasutada ainult neid tänavaid. Manhattani kaugus on vähim kaugus kahe punkti vahel ainult mööda horisontaalseid ja vertikaalseid lõike liikudes. Erinevalt tavalisest, Eukleidese kaugusest on lühimaks marsruudiks mitmeid võimalusi:



Kuigi võimalusi on palju, avaldub Manhattani kaugus väga lihtsalt, nimelt algus- ja lõpp-punkti koordinaatide vahede summaga:

$$MK = |x_2 - x_1| + |y_2 - y_1|$$

12.2.3 Paralleelsuse ja samasihilisuse kontroll

Sirget iseloomustab veel sirge **tõus** (i.k *slope*). Kui on antud kaks sirgel asuvat punkti $P = (x_1, y_1)$ ja $Q = (x_2, y_2)$, siis neid läbiva sirge tõus on $(x_1 - x_2)/(y_1 - y_2)$. Sirge võrrandist on tõus $-a/b$. Kaldenurga kaudu avaldub tõus kujul $tan(\alpha)$.

Horisontaalse sirge tõus on 0, vertikaalse sirge tõus on aga määramata (nulliga jagamine!). Programmeerides tuleb seda juhtu alati eraldi käsitleda.

Kaks sirget on paralleelsed siis ja ainult siis, kui neil on sama tõus. Kaks sirget ristuvad, kui nende tõusude korrutis on -1 või kui üks on horisontaalne (tõus 0) ja teine vertikaalne (tõus ∞).

12.2.4 Sirgete lõikepunkt

Sirgete lõikumist on kerge määrata, sest meil ongi tasandil 3 võimalust: sirged kas kattuvad, on paralleelsed või lõikuvad mingis punktis.

Kahe sirge lõikumispunkti saab määrata lihtsalt sirge võrrandist. Kui on antud sirged

$a_1x + b_1y = c_1$ ja $a_2x + b_2y = c_2$, siis nende lõikepunktiks on selline punkt, mis rahuldab samaaegselt mõlemat võrrandit. Korrutame esimest b_2 ja teist b_1-ga, saame

$$\begin{cases} a_1b_2x + b_1b_2y = b_2c_1 \\ a_2b_1x + b_1b_2y = b_1c_2 \end{cases}$$

Lahutades esimesest teise, saame

$$(a_1b_2 - a_2b_1)x = b_2c_1 - b_1c_2$$

millest

$$x = \frac{b_2c_1 - b_1c_2}{(a_1b_2 - a_2b_1)}$$

Analoogselt

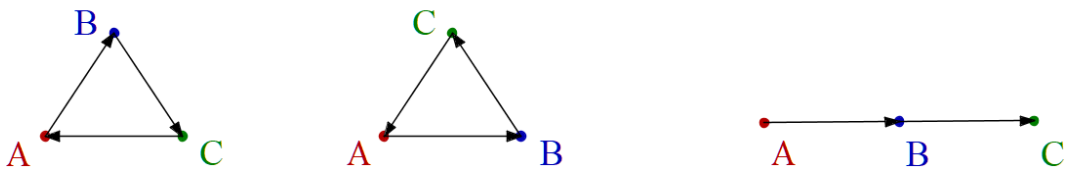
$$y = \frac{a_2c_1 - a_1c_2}{(a_1b_2 - a_2b_1)}$$

Funktsioon, mis leiab lõikepunktid:

```
pair<double,double> leia_loikepunkt(sirge s1, sirge s2)
{
	double determinant = s1.a*s2.b - s2.a*s1.b;

	if (determinant == 0) { //paralleelsed
		return make_pair(FLT_MAX, FLT_MAX);
	}
	else {
		double x = (s2.b*s1.c - s1.b*s2.c) / determinant;
		double y = (s1.a*s2.c - s2.a*s1.c) / determinant;
		return make_pair(x, y);
	}
}
```

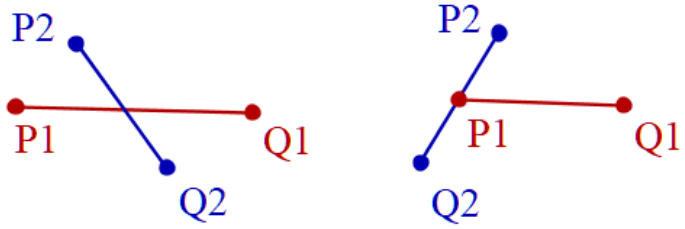
12.2.5 Lõikude lõikumine

Huvitavam ülesanne on aga määrata, kas kaks lõiku lõikuvad või kattuvad. Selleks vaatame kolme punkti tasandil ja nendevahelisi vektoreid. Eristame kolme võimalust:

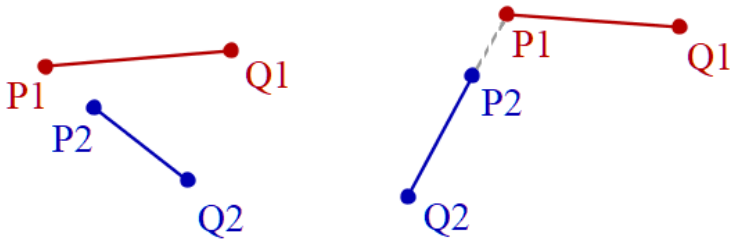


Esiteks, et punktid asuvad päripäeva, teiseks, et need asuvad vastupäeva ja kolmandaks, et need asuvad samal sirgel.

Vaatame nüüd, kuidas saavad tasandil asetseda omavahel kaks lõiku.

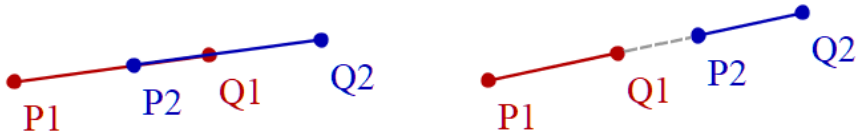


Kaks lõiku (p1,q1) ja (p2,q2) lõikuvad, kui p1,q1,p2 ja p1,q1,q2 on erineva suunaga ja p2,q2,p1 ja p2,q2, q1 on erineva suunaga.



Kui aga lõigud ei lõiku, siis p1,q1,p2 ja p1,q1,q2 on erineva suunaga, aga p2,q2,p1 ja p2,q2, q1 on sama suunaga.

Erijuhuks on osaliselt kattuvad lõigud:



Sellisel juhul vaatame projektsioone x ja y teljele. Kattumise korral kattuvad mõlemad projektsioonid. Kui üks projektsioonidest ei kattu, siis lõigud ei kattu.

Nende tähelepanekute põhjal saame defineerida järgmised funktsioonid:

```
bool on_loigul(Point p, Point q, Point r)
{
    if (q.x <= max(p.x, r.x) && q.x >= min(p.x, r.x) &&
        q.y <= max(p.y, r.y) && q.y >= min(p.y, r.y))
        return true;
    return false;
}

int suund(Point p, Point q, Point r)
{
    int val = (q.y - p.y) * (r.x - q.x) -
        (q.x - p.x) * (r.y - q.y);

    if (val == 0) return 0;  // kollineaarsed

    return (val > 0) ? 1 : 2; // suund: 1-päripäeva, 2-vastupäeva
}
```

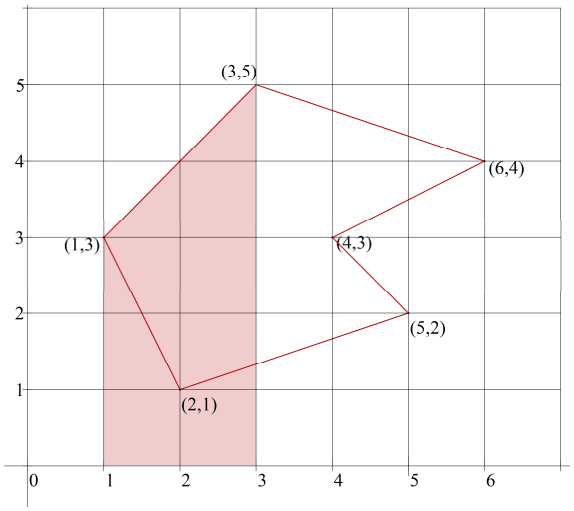
```
bool loikuvad(Point p1, Point q1, Point p2, Point q2)
{
    int o1 = suund(p1, q1, p2);
    int o2 = suund(p1, q1, q2);
    int o3 = suund(p2, q2, p1);
    int o4 = suund(p2, q2, q1);

    // lõikuvad
    if (o1 != o2 && o3 != o4)
        return true;

    // p1, q1 ja p2 on kollineaarsed ja p2  asub lõigul (p1,q1)
    if (o1 == 0 && on_loigul(p1, p2, q1)) return true;

    // p1, q1 ja q2 on kollineaarsed ja q2 asub lõigul (p1,q1)
    if (o2 == 0 && on_loigul(p1, q2, q1)) return true;

    // p2, q2 ja p1 on kollineaarsed ja p1 asub lõigul (p2,q2)
    if (o3 == 0 && on_loigul(p2, p1, q2)) return true;

    // p2, q2 ja q1 on kollineaarsed ja q1 asub lõigul (p2,q2)
    if (o4 == 0 && on_loigul(p2, q1, q2)) return true;

    return false; // ei kattu
}
```

12.3 HULKNURK

12.3.1 Hulknurga ümbermõõt ja pindala

Lihtsa hulknurga ümbermõõdu leidmine vast probleeme ei valmista – tuleb kokku liita hulknurka moodustavate lõikude ehk hulknurga servade pikkused. Lõigu pikkuseks on selle otspunktide vaheline kaugus.

Pindalaga on veidi keerulisem, selle leidmiseks jagatakse hulknurk tavaliselt osadeks. Vaatame ühe külje ja x-telje vahele jäävat trapetsit:

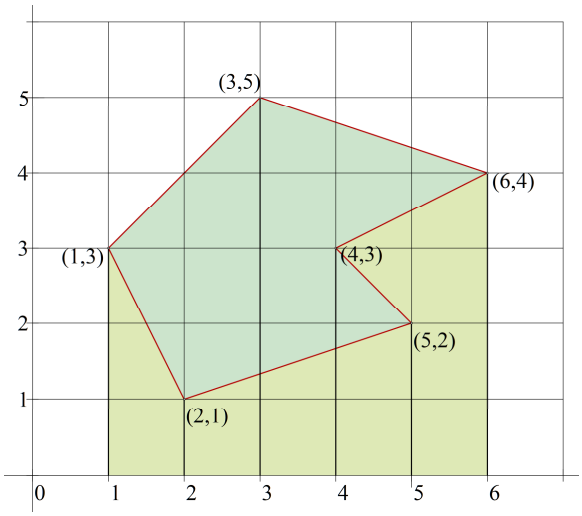


Selle trapetsi pindala on ristküliku ja täisnurkse kolmnurga pindalade summa. Üheks kolmnurga kaatetiks on külje projektsioon x teljele ehk serva otspunktide x koordinaatide vahe. Teiseks kaatetiks

on projektsioon y teljele, mis on võrdne otspunktide y koordinaatide vahega. Ristküliku küljed on samuti projektsioonid. Saame

$$\frac{(x_2 - x_1)(y_2 - y_1)}{2} + (x_2 - x_1) * y_1 = (x_2 - x_1)\left(\frac{y_2 - y_1 + 2y_1}{2}\right) = \frac{(x_2 - x_1)(y_2 + y_1)}{2}.$$

Kordame sama iga külje jaoks:



Märke hoolega silmas pidades tulevad mõned pindalad negatiivsed, täpsemalt nende külgede omad, kus $x_2 < x_1$, mis joonisele vaadates tähendab, et heleroheline ala lahutatakse ära ja alles jääb tumerohelise ala pindala ehk just see, mis vaja:

$$\frac{(3-1)(5+3)}{2} + \frac{(6-3)(4+5)}{2} + \frac{(4-6)(3+4)}{2} + \frac{(5-4)(2+3)}{2} + \frac{(2-5)(1+2)}{2} + \frac{(1-2)(3+1)}{2} =$$

$$= \frac{2*8 + 3*9 - 2*7 + 5 - 3*3 - 4}{2} = \frac{21}{2} = 10{,}5\ (ruutühikut)$$

Funktsioon, mis sellel meetodil hulknurga pindala leiab:

```
double pindala(double X[], double Y[], int n)
{
	double S = 0.0;

	int j = n - 1;
	for (int i = 0; i < n; i++) {
		S += (X[j] + X[i]) * (Y[j] - Y[i]);
		j = i;  // j on i-le eelneva tipu nr
	}

	// absoluutväärtus!
	return abs(S / 2.0);
}
```

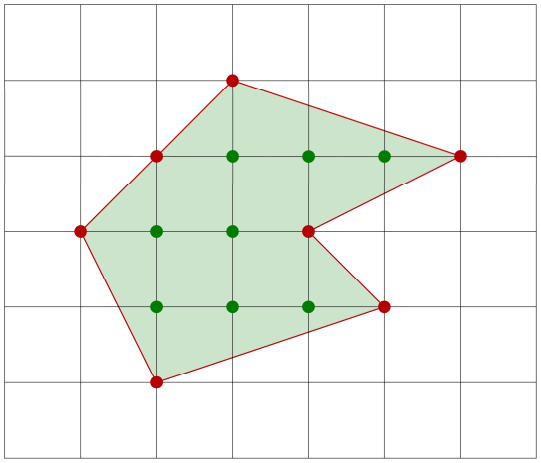
NB! See lähenemine eeldab, et tipud on sorteeritud järjekorras!

12.3.2 Picki teoreem

Kui hulknurga tippude koordinaadid on täisarvud, saab pindala leidmiseks kasutada ka **Picki teoreemi**. Nimelt avaldub hulknurga pindala valemiga

$$S = a + \frac{b}{2} - 1,$$

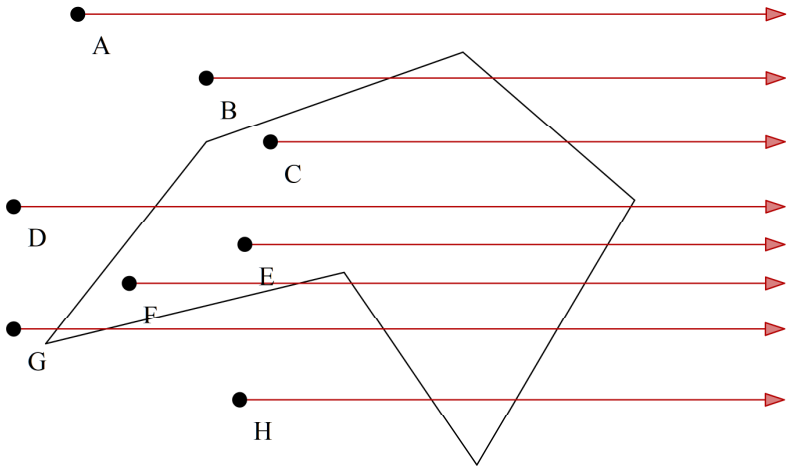
kus a on hulknurga sees asuvate täisarvuliste koordinaatidega punktide arv ja b on hulknurga servadel asuvate täisarvuliste koordinaatidega punktide arv.



Joonisel on $a = 8$ punkti hulknurga sees (rohelised) ja $b = 7$ punkti hulknurga servadel (punased), seega on selle hulknurga pindala $8 + 7/2 - 1 = 10{,}5$ ruutühikut.

12.3.3 Punkti sisaldumine hulknurgas

Uurime, millal sisaldub antud punkt $P(x, y)$ hulknurgas. Kui tõmmata kiir punktist P paralleelselt horisontaalse teljega, saab loendada, mitu korda selline kiir läbib mõnd hulknurga külge. Kui läbimisi on paaritu arv, asub punkt hulknurga sees, kui aga paarisarv, siis hulknurgast väljapool. Hulknurga servadel asuvad punktid loetakse hulknurga sees olevateks.



Tekitame punktist lõigu, mille alguspunktiks on punkt ise ja lõpp-punktiks punkt, mille y-koordinaat võrdub punkti y-koordinaadiga ja x-koordinaat on võimalikult kaugel (lõpmatuses). Tegelikult piisab, kui see on kaugemal kui kõige suurema x-koordinaadiga hulknurga tipu x-koordinaat. Edasi kontrollime iga hulknurga küljega, kas see lõikub punktist tõmmatud lõiguga ning loendame, mitme küljega lõikumine toimus.

```
bool on_sees(Point polygon[], int n, Point p)
{
	if (n < 3)  return false; //hulknurgal on vähemalt 3 tippu

	// p-st kiire lõpp-punkt(lõpmatuses)
	Point extreme = { INF, p.y };

	// loendame lõikumised külgedega
	int count = 0, i = 0;
	do {
		int next = (i + 1) % n;

		// kas kiir puntist p lõikab mõnd külge
		// 'polygon[i]' st kuni 'polygon[next]'-ini
		if (doIntersect(polygon[i], polygon[next], p, extreme)) {
			// Kui on kollineaarne, siis kas on serval?
			if (orientation(polygon[i], p, polygon[next]) == 0)
				return onSegment(polygon[i], p, polygon[next]);

			count++;
		}
		i = next;
	} while (i != 0);

	return count & 1;  // =(count%2 == 1)
}
```

12.4 KOORDINAATIDE PAKKIMINE

On antud NxN ruudustik ($1 \le N \le 10^9$). Ruudustikus on M ($1 \le M \le 100$) takistust, mille laius on 1 ruut ja pikkus 1 kuni K ruutu. Kõik takistused on asetatud kas vertikaalselt või horisontaalselt. Ruudustikus töötab robottolmuimeja, mis asub alguses ruudul S koordinaatidega x ja y. Leia, mitu ruutu saab robot puhastada. Robot takistusi ei suuda ületada.
Sisendi esimesel real on kaks arvu N ja M. Teisel real on kaks täisarvu: roboti stardipositsiooni koordinaadid. Järgmisel M real on neli täisarvu: iga takistuse alguse ja lõpu koordinaadid.
Väljastada üks täisarv: ruutude arv, mida robot saab puhastada (k.a ruut, kus ta alguses on).
NÄIDE:

```
4 2
0 0
1 2 1 3
1 2 3 2
```

Vastus:

```
10
```

Näite puhul piisab muidugi, kui käsitleda ruudustikku graafina ja läbida see laiuti. Kuid mida peale hakata, kui ruudustik on maksimaalse lubatud suurusega?

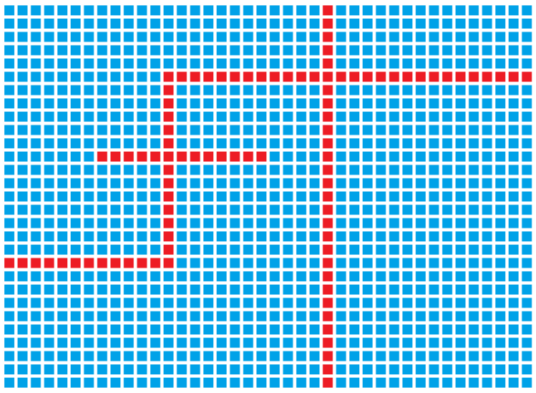
Paneme tähele, et kuigi ruudustik on suur, on takistusi seal hõredalt. Enamik read on täpselt samasugused, seega ei pea iga rida üksikult töötlema. Selle asemel saab kasutada meetodit, mida nimetatakse **koordinaatide pakkimiseks** (i.k. *coordinate compression*)

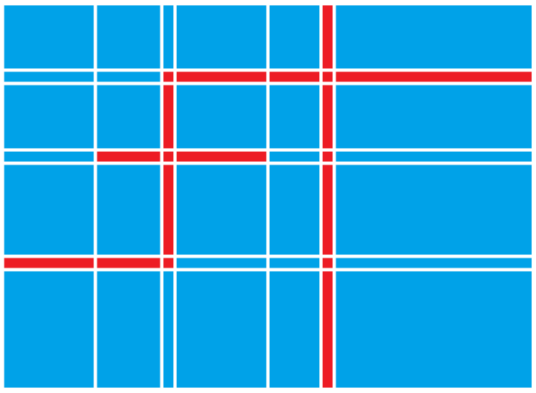
Selle meetodi peamine idee seisnebki selles, et enamik read (või veerud) on tegelikult samasugused, aga ülesande seisukohalt olulist infot sisaldavad ainult need read, mis on eelmisest erinevad.

Antud ülesandes on järgmine rida eelmisest erinev ainult siis, kui takistus algab või lõpeb vaadeldaval või sellele eelneval real. Kuna takistusi on vaid 100, on võimalikke algus ja lõpp-punkte vaid paarsada. Sama kehtib ka veergude jaoks.

Hoiamegi alles meid huvitavad read ja veerud, st esimese ja viimase ning iga takistuse otspunktile vastava rea ning samuti sellele eelneva ja järgneva rea. See teeb kokku maksimaalselt 602 rida. Iga rea kohta hoiame meeles ka, mitut rida see esindab (mitu samasugust rida sellele järgneb). Sama teeme ka veergudega ning saame 602x602 ruudustiku, mida on juba võimalik sügavuti või laiuti läbida.

Vastuse leidmiseks peame arvestama, et iga ruut (R, V) vastab nii mitmele ruudule, kui mitut rida R ja veergu V see ruut esindab: kui rida R esindab N_r ruutu ja veerg V N_v ruutu, siis (R, V) esindab $N_r \times N_v$ ruutu.





12.4.1 Ülesanne: Tikuruudud

See ülesanne on Eesti informaatikaolümpiaadi lõppvoorust aastast 2014 (autoriks Avo Muromägi).

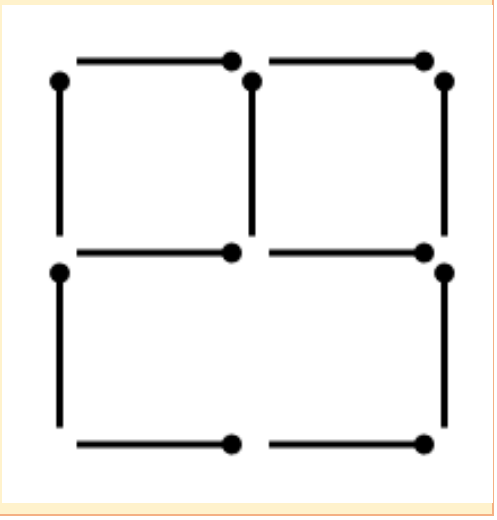
Kirjutada programm, mis saab tikkude paigutuse laual ja loendab, mitu ruutu need tikud kokku moodustavad. Näiteks alloleval joonisel kujutatud tikud moodustavad kokku 3 ruutu (kaks ruutu suurusega 1×1 ja ühe ruudu suurusega 2×2).
Sisendi esimesel real on tikkude arv `N` (`1≤N≤1000`). Järgmisel `N` real on igaühel ühe tiku asend kujul `X Y S`, kus `X` ja `Y` on tiku väävlita otsa täisarvulised koordinaadid (`0≤X≤1000, 0≤Y≤1000`) ja `S` selle suund. Suund võib olla kas `H`, mis tähendab, et tiku väävliga ots on suunatud paremale, või `V`, mis tähendab, et väävliga ots on suunatud üles. Võib eeldada, et joonisel pole kuskil mitu tikku üksteise peal.
Väljundi ainsale reale väljastada moodustuvate ruutude arv.

NÄIDE:

```
11
0 0 H
1 0 H
0 0 V
2 0 V
0 1 H
1 1 H
0 1 V
1 1 V
2 1 V
0 2 H
1 2 H
```

Vastus:

```
3
```



Kuna ruudustik on küllaltki suur, siis tasub tähele panna, et ruute ei tasu otsida koordinaatidelt, kus ei asugi tikke. Veelgi enam - iga ruudu vasakust alumises nurgas peab olema nii horisontaalse kui ka vertikaalse tiku väävlita ots.

Kaval ongi juba andmete sisse lugemisel jagada tikud kaheks - horisontaalsed ja vertikaalsed - ning hoida neid eraldi loendites. Sorteerime horisontaalsed tikud esmalt y-koordinaadi järgi, võrdsete y-koordinaatide korral x-koordinaadi järgi. Vertikaalsete tikkude puhul toimime vastupidi: sorteerime esmalt x-koordinaadi, teiseks y-koordinaadi järgi.

Nüüd saab alustada ruutude otsimisega. Selleks käime üle horisontaalsete tikkude loendi. Iga horisontaalse tiku korral vaatame, kas leidub vertikaalne tikk, mis asub samadel koordinaatidel. Kuna tikud on järjestatud, saab seda teha kahendotsinguga. Kui sellist vertikaalset tikku ei ole, siis võtame vaatluse alla järgmise horisontaalse tiku. Kui aga on olemas vastavate koordinaatidega vertikaalne tikk, siis võivad need kaks tikku olla ühe või mitme ruudu alumiseks vasakuks nurgaks. Sellisel juhul asume loendama tekkivaid ruute.

```
int loenda_ruudud() {
     int vastus = 0;
     for (int i = 0; i < hor_tikud.size(); i++)
          vastus += loenda_ruudud_tikust(i);
     return vastus;
}
```

Kui on olemas ka teised kaks tikku, nii et moodustub ruut küljepikkusega 1, siis suurendame vastust ühe võrra. Sõltumata eelmisest sammust kontrollime, kas horisontaalse tiku väävliga otsas asub horisontaalne tikk ning teeme sama ka vertikaalse tikuga. Mõlemal juhul peab see tikk olema vastavas loendis järgmine, see tähendab, et kui me hetkel vaatasime horisontaalset tikku indeksiga i, siis kontrollime, kas horisontaalne tikk indeksiga $i + 1$ on sama y-koordinaadi ja ühe võrra suurema x-koordinaadiga. Kui on olemas nii vertikaalne kui ka horisontaalne tikk, mis sobib külgi pikendama, siis kontrollime, kas moodustub ruut küljepikkusega 2. Sama protseduuri jätkame seni, kuni jõuame olukorda, kus horisontaalse või vertikaalse tiku väävliga otsa juurde pole sama suunaga tikku asetatud. Sellisel juhul võib loendamise katkestada, sest suuremaid ruute enam samast alguskohast moodustuda ei saa, ja edasi liikuda järgmise horisontaalse tiku juurde.

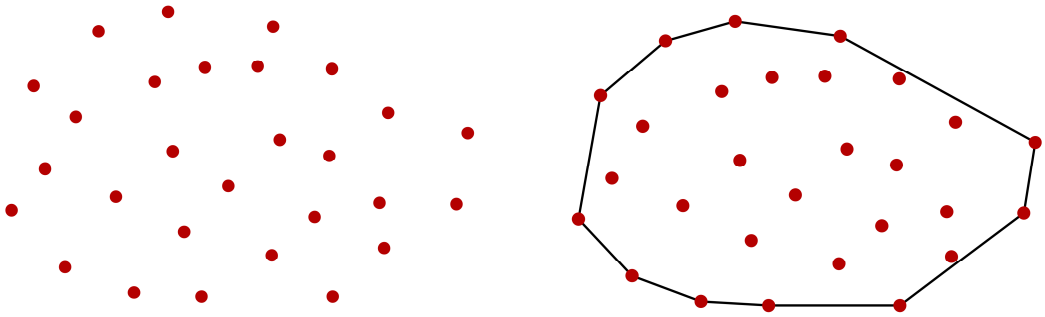
```
int loenda_ruudud_tikust(int tiku_nr) {
	int  kylje_pikkus;
	int vasak_indeks, alumine_indeks;
	pair<int, int> vasak_tikk, alumine_tikk;
	int vastus = 0;
	pair<int, int> tikk = hor_tikud[tiku_nr];
	kylje_pikkus = 1;
	alumine_indeks = tiku_nr;
	vasak_indeks = otsi_tikk(make_pair(tikk.second, tikk.first), ver_tikud);
	if (vasak_indeks == -1) return vastus; //Selle algkoordinaadiga vertikaalset
// tikku pole
	while ((alumine_indeks < hor_tikud.size()) && (vasak_indeks < ver_tikud.size()))
{
		alumine_tikk = hor_tikud[alumine_indeks];
		if ((alumine_tikk.second - kylje_pikkus + 1 != tikk.second) ||
(alumine_tikk.first != tikk.first)) break;
		vasak_tikk = ver_tikud[vasak_indeks];
		if ((vasak_tikk.first != tikk.second) || (vasak_tikk.second - kylje_pikkus
+ 1 != tikk.first)) break;

		if (on_hor_yhendus(tikk.second, tikk.second + kylje_pikkus - 1, tikk.first
+ kylje_pikkus) && // ülemine külg
			on_ver_yhendus(tikk.second + kylje_pikkus, tikk.first, tikk.first +
kylje_pikkus - 1)) // parem külg
			vastus++; //ruut leitud!
		kylje_pikkus++; //proovime suuremat
		alumine_indeks++;
		vasak_indeks++;
	}
	return vastus;
}
```

Praegu on veel kirjeldamata, kuidas teha kindlaks, et moodustub ruut küljepikkusega p? Tänu eelpool kirjeldatud viisile me teame, et (potentsiaalse) ruudu vasakus ja alumises küljes on kummaski p tikku olemas. Jääb veel ainult kontrollida, kas paremas ja ülemises küljes on kõik tikud olemas. Siin on oluline märgata, et iga tiku olemasolu kontroll polegi vajalik. Veendumaks, et eksisteerib p tikust moodustatud külg, piisab tänu järjestatud tikkude loendile kontrollist, kas on olemas külje esimene ja viimane tikk ning kui need on olemas, siis nende tikkude indeksite vahe peab olema $p - 1$. Indeksid saab leida taas kahendotsingut kasutades.

```
bool on_hor_yhendus(int X1, int X2, int Y) {
	int  indeks1, indeks2;
	indeks1 = otsi_tikk(make_pair(Y, X1), hor_tikud);
	indeks2 = otsi_tikk(make_pair(Y, X2), hor_tikud);
	return (indeks1 >= 0) && (indeks2 >= 0) && (indeks2 - indeks1 == X2 - X1);
}
```

12.5 KUMER KATE

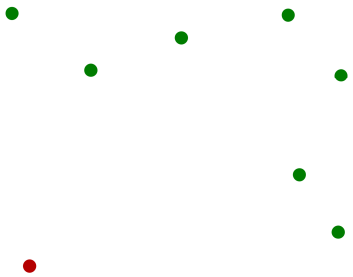
Kumer kate (i.k *convex hull*) on väikseim selline kumer hulknurk, mis sisaldab kõik etteantud punktid. Sisaldumine tähendab, et iga punkt asub kas hulknurga sees või mõnel selle küljel.



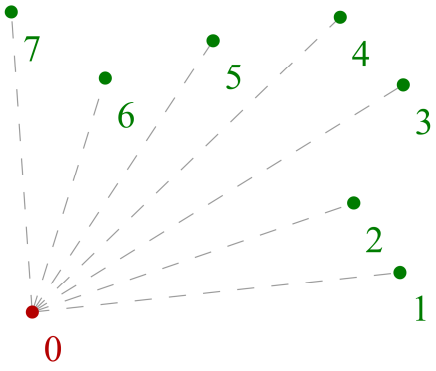
Jõumeetodil võiks käia selline otsimine nii, et kontrollime iga punktipaari korral, kas selle vahele tõmmatud lõik saab olla hulknurga küljel. Kui saab, siis peavad kõik ülejäänud punktid asuma sirgest, mis läbib vaadeldavat kahte punkti, ühel pool. n punktist saab moodustada n^2 paari ja iga paari jaoks tuleb kontrollida veel $n-2$ punkti asukohta, on sellise naiivse lähenemise keerukuseks $O(n^3)$.

Kumera katte leidmiseks on olemas mitmeid paremaid algoritme. Üks neist on **Grahami seire**. Selle idee on järgmine:

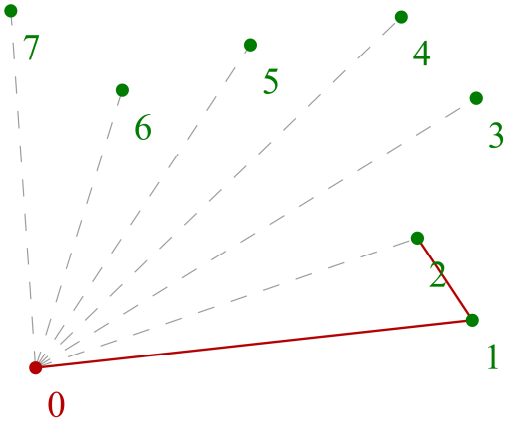
1. Leia kõige väiksema y-koordinaadiga punkt. Kui sama y-koordinaadiga on mitu punkti, vali neist see, mille x-koordinaat on kõige väiksem. Nimetame seda punkti vaatlus- ehk seirepunktiks (joonisel punane):

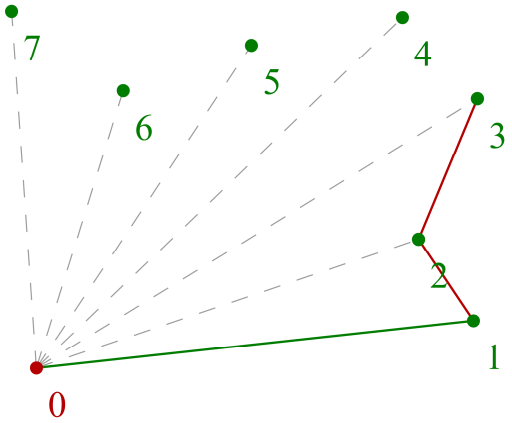


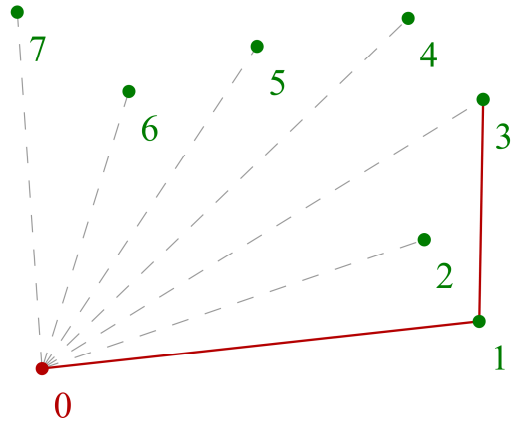
   Sea see punkt punktiloendis esimeseks ja kuulub ise kindlasti kumerasse kattesse.
2. Ülejäänud $n-1$ punkti jaoks leia seirepunkti ja vaadeldavat punkti läbiva sirge tõusunurk. Sorteeri punktid leitud tõusunurga järgi vastupäeva.

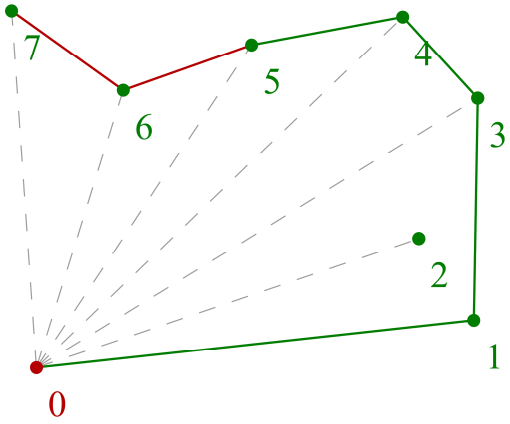


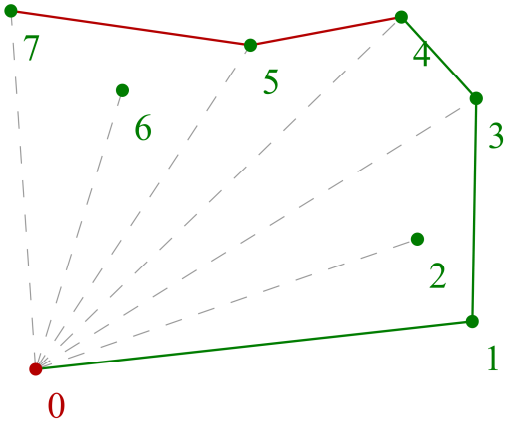
3. Kõigist sama tõusunurgaga punktidest jäta alles ainult see, mis on valitud punktist kõige kaugemal, sest vaid see saab kuuluda kumerasse kattesse.
4. Kui järele jääb vähem kui kolm punkti, siis kumerat katet ei eksisteeri (punktid asuvad samal sirgel).
5. Edasi tuleb iga punkti p_i jaoks otsustada, kas see kuulub kumerasse kattesse või mitte. Kumer hulknurk saab moodustuda ainult sellistest punktidest, mille vahele tõmmatud servad keeravad eelmise serva suhtes rangelt vastupäeva. Seega
    a. kui toimub pööre vastupäeva, jäta punkt p_i kumerasse kattesse ja korda 4. sammu punkti p_{i+1} jaoks.
    b. Vastasel juhul eemalda kattest **eelmine** punkt p_{i-1} ja korda 4. sammu.

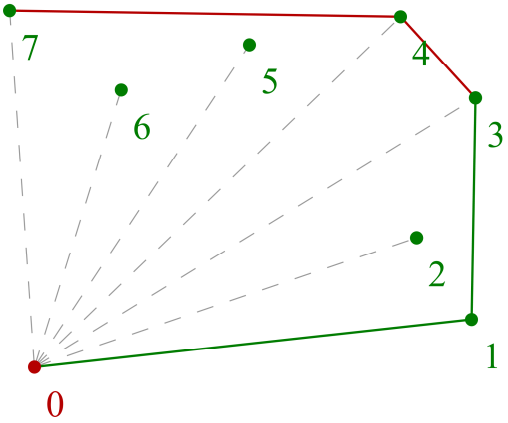




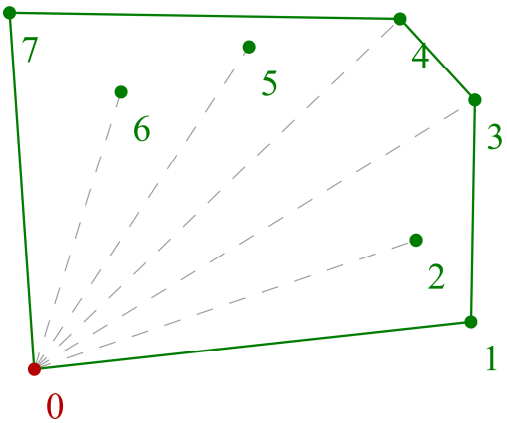








Joonisel esiteks vaatame pööret punktikolmikul (0,1,2). pööre on vastupäeva ja punktid jäävad kattesse. Edasi vaatame pööret (1,2,3), mis on aga päripäeva. Seega punkt 2 kattesse siiski ei kuulu, eemaldame selle kattest. Nüüd tuleb kontrollida kolmikut (0, 1, 3), mis on ilusti vastupäeva. Edasi on kenasti vastupäeva pöörded, kuni jõuame punktideni (5,6,7). Kuna seal on pööre päripäeva, eemaldame punkti 6 kattest. Aga ka pööre (4, 5, 7) on vales suunas, seega tuleb kattest eemaldada ka punkt 5. Kumera katte moodustavad punktid 0, 1, 3, 4 ja 7.



Siin on ka protseduur, mis leiab kumera katte:

```
void leia_kumer_kate(Point punktid[], int n)
{
    // vasak alumine punkt
    int ymin = punktid[0].y, min = 0;
    for (int i = 1; i < n; i++) {
        int y = punktid[i].y;
        if ((y < ymin) || (ymin == y && punktid[i].x < punktid[min].x))
            ymin = punktid[i].y, min = i;
    }

    //vaheta vasak alumine punkt esimeseks
    swap(punktid[0], punktid[min]);

    // sorteeri punktid tõusunurga järgi
    p0 = punktid[0];
    qsort(&punktid[1], n - 1, sizeof(Point), compare);

    // hoiame sobivaid punkte pinus. Alustame esimese kolme punktiga
    stack<Point> S;
    S.push(punktid[0]);
    S.push(punktid[1]);
    S.push(punktid[2]);

    for (int i = 3; i < n; i++) {
        // eemalda pealmine punkt,kuni õige suunani
        while (leia_suund(eelmine(S), S.top(), punktid[i]) != 2)
            S.pop();
        S.push(punktid[i]); //suund õige, lisa punkt
    }

    // nüüd on pinus kumera katte moodustavad punktid
    while (!S.empty()) {
        Point p = S.top();
        cout << "(" << p.x << ", " << p.y << ")" << endl;
        S.pop();
    }
}
```

Grahami seiremeetodi keerukuseks on $O(n\ log\ n)$.

12.6 Ülesanne: ring ja ruut

See ülesanne oli 1997. aastal Balti olümpiaadil. Sõnastuse poolest on ülesanne ülilihtne, seda saab kirjeldada kõigest paari lausega:

On antud ring ja ruut oma koordinaatidega. Leida nende kattuva osa pindala.
Sisendi esimesel real on kolm täisarvu: ringi keskpunkti koordinaadid ja ringi raadius. Teisel real on 8 täisarvu: ruudu tippude koordinaadid.
Väljundi ainsale reale kirjutada üks reaalarv: ringi ja ruudu kattuva osa pindala.
***NÄIDE 1*:**

```
0 0 1
0 0 0 2 2 0 2 2
```

Vastus:

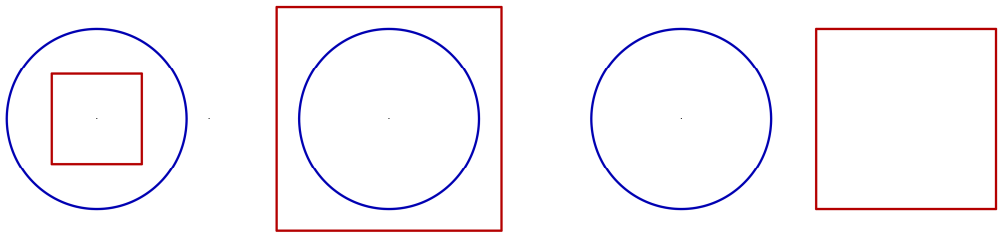
***NÄIDE 2*:**

```
0 0 1
0 -2 0 2 4 -2 4 2
```

Vastus:

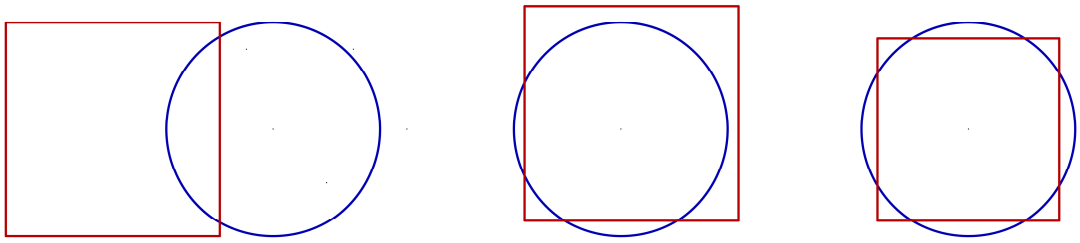
Algselt arvas ka ülesande väljamõtleja Indrek Jentson, et kui raske see ülesanne ikka olla saab, kuid erinevad testjuhud lähevad väga kiiresti käest ära. Uurime neid korraks, tegu on ka kasuliku testimisalase harjutusega.

Kõige lihtsamad ringi ja ruudu omavahelise paiknemise võimalused on järgmised kolm:



- Ruut paikneb tervenisti ringi sees. Kattuva osa pindala on ruudu pindala.
- Ring paikneb tervenisti ruudu sees. Kattuva osa pindala on ringi pindala.
- Ruudul ja ringil ühine osa puudub. Kattuva osa pindala on null.

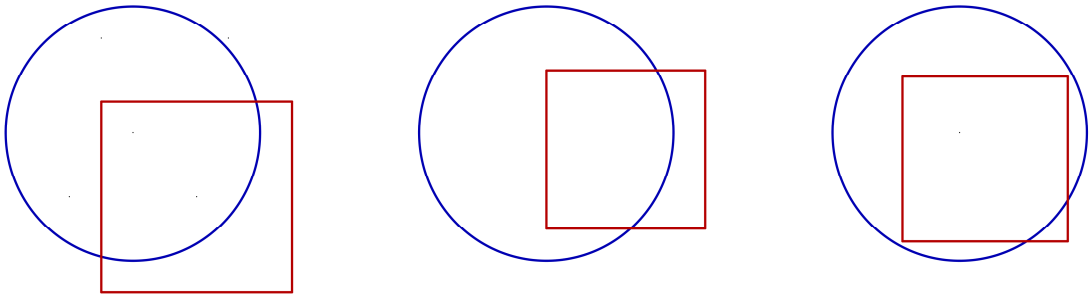
Lisaks on muidugi ka keerulisemaid võimalusi, näiteks asetused, kus ükski nurk ei ole ringi sees ja:



- Ring lõikab ruudu ühte külge
- Ring lõikab ruudu kaht külge
- Ring lõikab ruudu kõiki külgi

Neil juhtudel tuleb leida segmendi pindala ja otsustada, kas segment on ringi sees või sellest väljas.

Ruudu nurgad aga võivad asetseda ka ringi sees:



- Üks nurk ringi sees
- Kaks nurka ringi sees
- Kolm nurka ringi sees

Täiendavalt tekivad erijuhud, kui ring lihtsalt puudutab ruudu külge või tippu. Kiiresti saab selgeks, et **geomeetriline lähenemine** – proovida iga erijuhu jaoks leida vastavad geomeetrilised valemid – läheb õige pea käest ära: lahendamine on võimalik, aga lahenduse kirjapanek võtaks tõenäoliselt sadu ja sadu ridu koodi ning mitu päeva aega.

Kui jooniste üle natuke mõtiskleda, võib järgmiseks pähe tulla **analüütiline lähenemine**. Paneme tähele, et:

1. Geomeetrilise kõvera alla jääva ala pindala saab leida integraaliga.
2. Ruudu küljed annavad meile enamasti integreerimiseks sobilikud rajad.
3. Pöörates joonise kõigepealt sobivat pidi ning seejärel ringjoone valemit $y = \sqrt{r^2 - x^2}$ vastavate radadega integreerides saamegi vastuse. Integreerimiseks vajalikud vihjed võib leida näiteks siit: https://www.quora.com/How-can-I-integrate-to-find-the-area-of-a-circle

See lahendus on juba parem, kuid ka selle realisatsioon võttis mul endal pea terve päeva.

Selgub aga, et selliste ülesannete puhul on sageli võimalik ka **programmeerija lähenemine**, kus me paneme tähele, et üldjuhul pole vaja leida absoluutse täpsusega vastust ning piisab sellest, kui me leiame vastuse **lähendamise** teel.

Algoritmi idee on järgmine:

1. On väga lihtne kontrollida, kas ruut on **täielikult ringi sees** (sel juhul liidame vastusele ruudu pindala) või **täielikult ringist väljas** (sel juhul ei liida vastusele midagi).
2. Kui ruut kattub ringiga osaliselt, jagame selle **rekursiivselt neljaks osaks** ja kordame arutlust iga osa jaoks, kuni oleme saavutanud sobiva täpsuse.

Realisatsiooni juurde minnes on kõigepealt kasulik koordinaadid mugavamaks teisendada, siis on edasises arvutused lihtsamad. Esiteks võib nihutada koordinaate nii, et ringi keskpunkt oleks

koordinaatidega (0, 0). Selleks ei tule teha midagi muud, kui sisse lugedes lahutada igast ruudu x-koordinaadist ringi x-koordinaat ja igast ruudu y-koordinaadist ringi y-koordinaat.

Teiseks on kasulik pöörata sisendit nii, et ruudu küljed on paralleelsed koordinaattelgedega. See on lihtne teisendus ja teeb edaspidi programmeerija elu palju kergemaks:

```
void keera(pair<double, double> ruut[4]) {
	if (ruut[0].second - ruut[2].second == 0) return; //on juba õige
	double tous = (ruut[0].first - ruut[2].first)/(ruut[0].second - ruut[2].second);
	tous = atan(tous);
	for (int i = 0; i < 4; i++) {
		double x = ruut[i].first;
		double y = ruut[i].second;
		ruut[i].first = x*cos(tous) - y*sin(tous);
		ruut[i].second = x*sin(tous) + y*cos(tous);
	}
	return;
}
```

Edasi vaatame läbi esimesed kolm triviaalset juhtu. Ring on ruudu sees, kui kaugus ringi keskpunktist iga servani on suurem või võrdne ringi raadiusega. Seda on lihtne kontrollida:

```
bool ring_ruudus(pair<double, double> ruut[4]) {
	if (r*(-1) >= ruut[0].first &&
		r <= ruut[3].first &&
		r*(-1) >= ruut[0].second &&
		r <= ruut[3].second)
		return true;
	else return false;
}
```

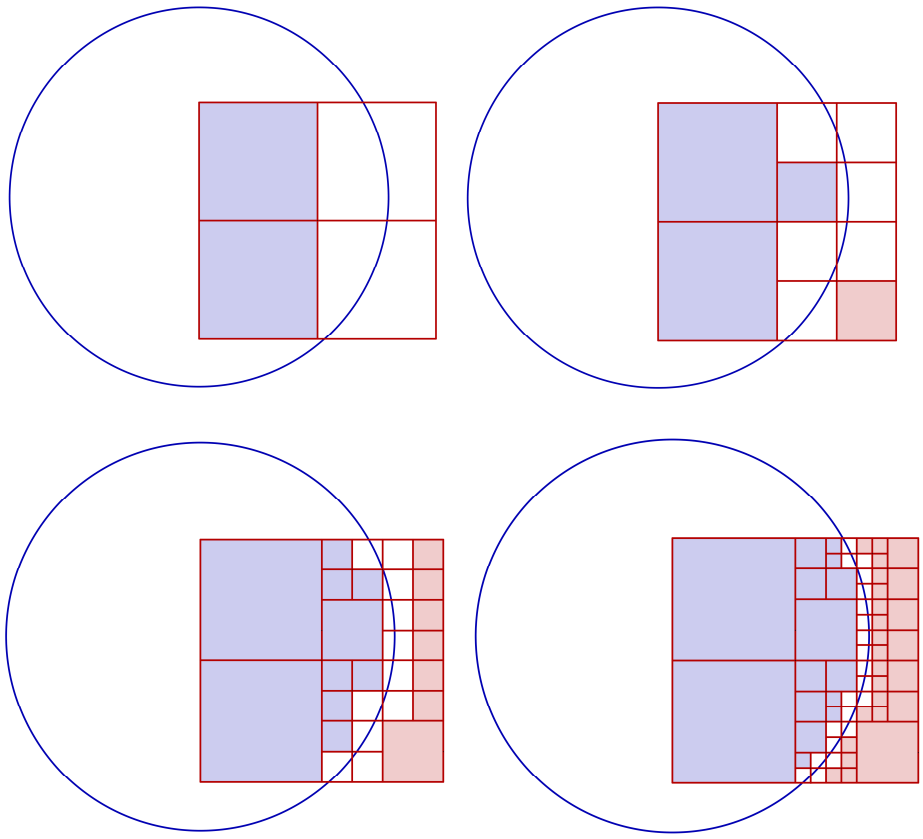
Ruut on ringi sees, kui ükski ruudu tipp ei ole ringi keskpunktist kaugemal kui ringi raadius. Selleks on kõige kindlam lugeda üle, mitu tippu on ringis:

```
int tipp_ringis(pair<double, double> tipp) {
	double s = hypot(tipp.first, tipp.second);
	if (s <= r) return 1;
	else return 0;
}

int tippe_ringis(pair<double, double> ruut[4]) {
	int summa = 0;
	for (int i = 0; i < 4; i++) {
		summa += tipp_ringis(ruut[i]);
	}
	return summa;
}
```

Kahjuks see, kui kõik tipud on ruudust väljas, ei garanteeri veel, et ringi ja ruudul kattuvat osa ei ole. Seega otsustamaks, et ruut on tervenisti väljaspool ringi, tuleb seda veel eraldi kontrollida.

```
bool on_valjas(pair<double, double> ruut[4])
{
	if (ruut[0].first >= r || ruut[3].first <= 0 - r ||
		ruut[0].second >= r || ruut[3].second <= 0 - r)
		return true;
	double sumx=0, sumy = 0, abssumx=0, abssumy = 0;
	for (int i = 0; i < 4; i++) {
		sumx += ruut[i].first;
		sumy += ruut[i].second;
		abssumx += abs(ruut[i].first);
		abssumy += abs(ruut[i].second);
	}
	if (abs(sumx) == abssumx && abs(sumy) == abssumy) return true;

	return false;
}
```

Nüüd järgneb rekursiivne osa: kui on teada, et osa ruutu on ringi sees, saame ruudu jagada neljaks väiksemaks ruuduks ja kontrollida samamoodi, kas need on täielikult ringi sees või täielikult ringist väljas või osaliselt ringis. Viimasel juhul saame jälle jagada need osaliselt sees olevad ruudud neljaks jne, kuni oleme saavutanud piisava täpsuse.



Lillad ruudud on täielikult ringi sees, roosad täielikult ringist väljas, valged need, mida vajadusel tuleb edasi jagada.

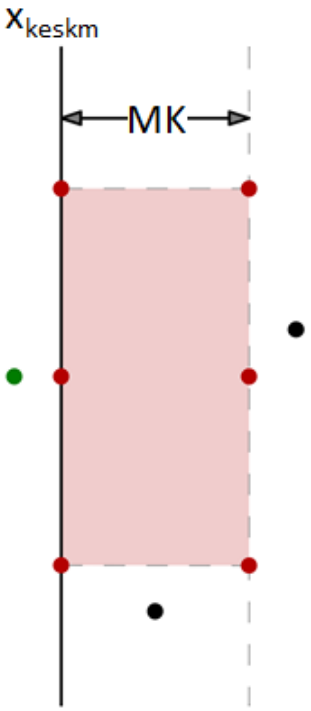
```
double leia_kattuv_pindala(pair<double, double>ruut[4])
{
	double serva_pikkus = abs(ruut[3].first - ruut[0].first);
	if (serva_pikkus < eps) return 0;
	double ruudu_pindala = serva_pikkus*serva_pikkus;
	int tipud_sees = tippe_ringis(ruut);
	if (tipud_sees == 4) return ruudu_pindala;
	if (tipud_sees == 0 && on_valjas(ruut)) return 0;
	double keskm_x = ruut[0].first + (serva_pikkus / 2);
	double keskm_y = ruut[0].second + (serva_pikkus / 2);
	pair<double, double> ruut1[4], ruut2[4], ruut3[4], ruut4[4];
	tee_ruut(ruut[0].first, ruut[0].second, keskm_x, keskm_y, ruut1);
	tee_ruut(ruut[0].first, keskm_y, keskm_x, ruut[3].second, ruut2);
	tee_ruut(keskm_x, ruut[0].second, ruut[3].first, keskm_y, ruut3);
	tee_ruut(keskm_x, keskm_y, ruut[3].first, ruut[3].second, ruut4);
	return leia_kattuv_pindala(ruut1) +
		leia_kattuv_pindala(ruut2) +
		leia_kattuv_pindala(ruut3) +
		leia_kattuv_pindala(ruut4);
}
```

12.7 LÄHIM PUNKTIPAAR

Üks klassikaline arvutusgeomeetria ülesanne on veel punktihulgast omavahel lähima punktipaari leidmine. Naiivne lähenemine oleks muidugi leida kõik kaugused kahe punki vahel. Selle keerukus aga on $O(n^2)$.

Kasutades aga "jaga ja valitse" tüüpi lähenemist, saame algoritmi, mis töötab $O(n\ log\ n)$ ajaga. Selleks

1. sorteerime punktid x-koordinaadi järgi
2. jagame punktid kahte võrdse suurusega alamhulka vertikaalse joone $x = x_{keskm}$
3. leia lähim paar kummaski alamhulgas kasutades sama võtet (rekursiivselt). Selle tulemusena saame vasaku ja parema poole minimaalse kauguse (MK_v ja MK_p).
4. leia vähim kaugus sellise punktipaari vahel, millest üks asub jaotusjoonest vasakul ja teine paremal
5. Vastuseks on minimaalne kaugus neist kolmest.



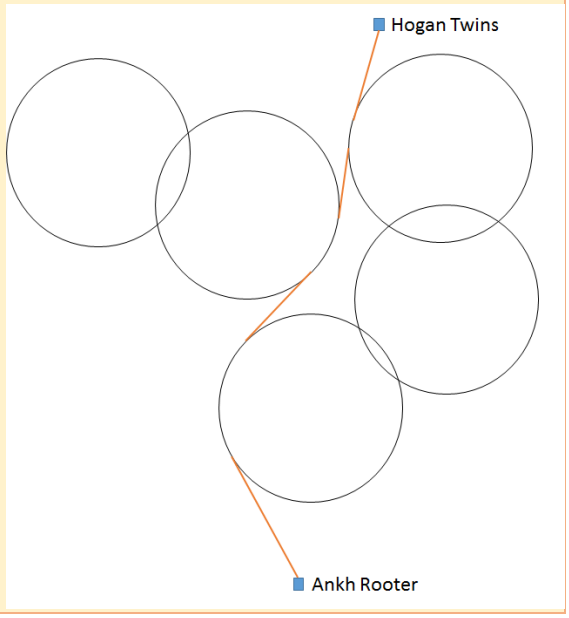
Tundub ilus, aga kas punkt 4 juures ei ole me sarnases lõksus, mis alguseski? Siin aga saab ära kasutada asjaolu, et kuna meid huvitab lõpuks vaid minimaalne kaugus, siis pole mõtet punkte kaugemalt vaatama hakata, kui seni leitud vähim kaugus ehk $\min(MK_v, MK_p) = MK$. Nii on vaja vaadelda vaid punkte, mis asuvad ristkülikus mõõtmetega $MK \times 2MK$. Lisaks ei saa see ristkülik sisaldada rohkem kui 6 punkti, sest muidu ei oleks MK meil vähim seni leitud kaugus! Endiselt tuleb kogu protseduur teha läbi iga punkti jaoks, aga maksimaalselt teeme n^2 võrdlemise asemel vaid $6n$ võrdlemist.

```
// Px on punktid sorteerituna x koordinaadi järgi ja
//Py punktid sorteerituna y koordinaadi järgi
float leia_lahim_paar(Point Px[], Point Py[], int n)
{
	// kuni 3 punktiga lihtsalt võrrelda:
	if (n <= 3)
		return leia_jouga(Px, n);

	// keskmine punkt x järgi
	int mid = n / 2;
	Point midPoint = Px[mid];

	Point Pyl[mid + 1];
	Point Pyr[n - mid - 1];
	int li = 0, ri = 0;
	for (int i = 0; i < n; i++)
	{
		if (Py[i].x <= midPoint.x)
			Pyl[li++] = Py[i];
		else
			Pyr[ri++] = Py[i];
	}

	float dl = leia_lahim_paar(Px, Pyl, mid);
	float dr = leia_lahim_paar(Px + mid, Pyr, n - mid);

	float d = min(dl, dr);

	Point strip[n];
	int j = 0;
	for (int i = 0; i < n; i++)
		if (abs(Py[i].x - midPoint.x) < d)
			strip[j] = Py[i], j++;

	return min(d, stripClosest(strip, j, d));
}
```

12.8 MAGIC MISSILE (ÜLESANNE)

Järgmise ülesande saatsime Balti olümpiaadile aastal 2016, kuid see jäi lõppvalikust välja kui liiga raske, s.t arvati, et enamik võistlejaid ei suudaks seda ettenähtud aja jooksul lahendada. Ülesande idee autor oli Oliver-Matis Lill, boonus läheb neile lugejaile, kes siit ajastukohase poliitilise olukorra viited üles leiavad.

Kettamaailm on tuntud kahe asja poolest: see asub lamedal tasapinnal, mitte kera pinnal, nagu meie maailm ja see jaguneb mitmeks võistlevaks riigiks.
Üks neist riikidest on Ankh Rooter, mida valitseb sõjakas diktaator Gin Junk Om, kes on ehitanud maagilise raketi, mida saab kasutada teiste hävitamiseks. Kõige rohkem vihkab Gin Junk Om võlur Kabob Maracat, kes elab linnas nimega Hogan Twins.
Kabob Maracal on mitmeid liitlasi Kettamaailmas ja ta on neile saatnud maagilise raketi vastase kilbi generaatorid. Kui maagiline rakett jõuab generaatorile lähemale kui R miili, kukub rakett ohutult alla. Gin Junk Om aga teab kilbigeneraatorite asukohti ning tahab, et sa arvutaks, kui mitu thaumi (võlujõu mõõtühik) võlujõudu peab raketti laadima. Selleks pead leidma lühima tee Ankh Rooter-ist Hogan Twinsi, mis väldib kõiki kilbigeneraatoreid.
Sisendi esimesel real on neli täisarvu: Gin Junk Omi võlutorni koordinaadid ja Kabob Maraca võlutorni koordinaadid. Teisel real on kaks täisarvu: generaatori ulatus R ja generaatorite arv N (1 <=N<=1000). Järgneval N real on igaühel kaks täisarvu: ühe generaatori koordinaadid.
Väljundisse kirjuta üks arv: vajalik võlujõud. Kui raketti lennutada ei saa, väljastada -1.
NÄIDE:

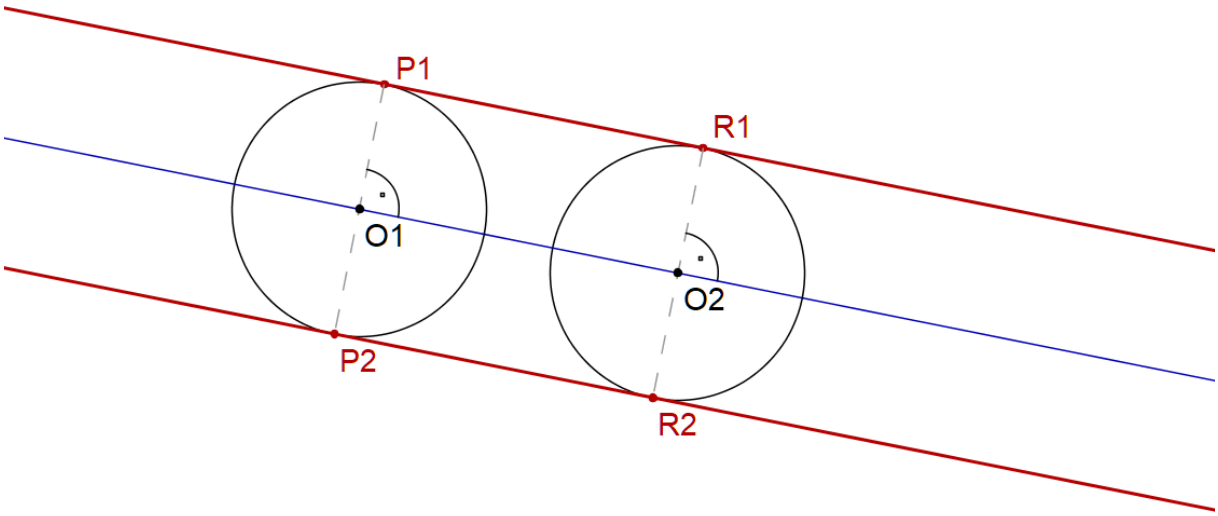


Selleks, et võimalikult otse lennata, peab rakett liikuma kas mööda kahe ringi ühist puutujat või mööda ringjoont. Algus ja lõppmarsruut on mööda puutujat, mis algab/lõpeb tornist.

Lühima tee leidmiseks tulebki leida kõik mõistlikud puutujad ja kaared ning moodustada nendest graaf, millest lühima tee leidmiseks saab kasutada Dijkstra algoritmi.

Kõlab väga lihtsalt, kuid tegelikult on sobivate puutujate ja kaarte leidmine omajagu nikerdamist vajav ettevõtmine.

Kahe ringi vahel võib olla kuni neli puutujat. Kui ringid puutuvad või lõikuvad, siis on puutujaid vaid kaks. Iga leitud puutuja juures tuleb kontrollida, ega see ei lõika mõnd muud ringi. Samuti ei tohi ära unustada algus ja lõpp-punkti ning võimalust, et need on omavahel otse ühendatavad.


P1
R1
O1
O2
P2
R2

```
void leia_puutujad()
{
    for (int i = 0; i < n; i++) {
        for (int j = i + 1; j < n; j++) {
            double rdx = x[j] - x[i];
            double rdy = y[i] - y[j];
            double vn = atan2(rdy, rdx);
            double mp = sin(vn) * r;
            double mu = cos(vn) * r;
            lisa_nihked(0, mp, mu, i, j);
            lisa_nihked(1, mp, mu, i, j);

            double kk = hypot(rdx, rdy) / 2;
            double un = vn - asin(r / kk);
            double ma = cos(un) * r;
            double mv = sin(un) * r;
            lisa_nihked(3, mv, ma, i, j);
            lisa_nihked(4, mv, ma, i, j);

            bool on_sisemised = (r / kk) <= 1;
            bool sobib[4] = { 1, 1, on_sisemised, on_sisemised };

            //käib kõik ringid läbi ja kontrollib, ega ei lõika ühtki ringi
            for (int k = 0; k < n; k++) {             {
                for (int m = 0; m < 4; m++) {
                    if (sobib[m] && liglah(ex[m], tx[m], ey[m], ty[m], x[k], y[k])) {
                        sobib[m] = 0;
                    }
                }
            }
            for (int m = 0; m < 4; m++) {
                if (sobib[m]) {
                    pv.push_back(make_pair(make_pair(i, j), make_pair(make_pair(ex[m],
ey[m]), make_pair(tx[m], ty[m]))));
                    pv.push_back(make_pair(make_pair(j, i), make_pair(make_pair(tx[m],
ty[m]), make_pair(ex[m], ey[m]))));
                }
            }
        }
    }

    //algus ja lõpppunkti jaoks puutujad
    for (int i = 0; i < n; i++)
    {
        leia_punkt(gx, gy, 0, i);
        leia_punkt(kx, ky, 2, i);
        bool sobib[4] = { 1, 1, 1, 1 };
        for (int k = 0; k < n; k++)
        {
            for (int m = 0; m < 4; m++) {
                if (sobib[m] && liglah(ex[m], tx[m], ey[m], ty[m], x[k], y[k])) {
                    sobib[m] = 0;
                }
            }
        }
        double px[4] = { gx, gx, kx, kx };
        double py[4] = { gy, gy, ky, ky };
        int inda[4] = { -1, -1, n, n };
        for (int m = 0; m < 4; m++) {
            if (sobib[m]) {
```

```
                pv.push_back(make_pair(make_pair(i, inda[m]),
make_pair(make_pair(xp[m], yp[m]), make_pair(px[m], py[m]))));
                pv.push_back(make_pair(make_pair(inda[m], i),
make_pair(make_pair(px[m], py[m]), make_pair(xp[m], yp[m]))));
            }
        }
    }
    // omavaheline sirge iga ringjoone jaoks
    bool o = 1;
    for (int k = 0; k < n; k++) {
        if (o && liglah(kx, gx, ky, gy, x[k], y[k])) {
            o = 0;
            break;
        }
    }
    if (o) {
        pv.push_back(make_pair(make_pair(-1, n), make_pair(make_pair(gx, gy),
make_pair(kx, ky))));
        pv.push_back(make_pair(make_pair(n, -1), make_pair(make_pair(kx, ky),
make_pair(gx, gy))));
    }
    sort(pv.begin(), pv.end());
}
```

Kaarte leidmiseks sorteerime ringjoonel asuvad puutepunktid esinemise järjekorras. Nii on vaja vaadata ainult võimalikke kaari järjestikuste punktide vahel, sealjuures ei tohi unustada võimalikku kaart esimese ja viimase punkti vahel. Ka kaarte puhul on oluline, et uuritav kaar ei lõikaks mõnd teist ringi.

```
void yhenda_kaared()
{
	int pr = -1;
	int pi = 0;
	for (int i = 1; i < np; i++) {
		int cr = pl[i].first.first.first;
		if (i == 1) {
			pr = cr;
			pi = 1;
			continue;
		}
		if (cr > pr) {
			double n1 = pl[pi].first.first.second;
			double n2 = pl[i - 1].first.first.second;
			double nv = (4 * k90) - (n2 - n1);
			double tp = nv * r;
			bool ok = 1;
			for (int j = 0; j < n; j++) {
				if (j == cr || hypot(x[j] - x[cr], y[j] - y[cr]) >= 2 * r) {
					continue;
				}
				int xv = x[j] - x[cr];
				int yv = y[j] - y[cr];
				double vn = kraad(xv, yv);
				if (vn > n2 || vn < n1) {
					ok = 0;
					break;
				}
			}
			if (!ok) continue;
			dt[pi].push_back(make_pair(i - 1, tp));
			dt[i - 1].push_back(make_pair(pi, tp));
			pr = cr;
			pi = i;
			continue;
		}
		double n1 = pl[i].first.first.second;
		double n2 = pl[i - 1].first.first.second;
		double nv = n1 - n2;
		double tp = nv * r;
		bool ok = 1;
		for (int j = 0; j < n; j++) {
			if (j == cr || hypot(x[j] - x[cr], y[j] - y[cr]) >= r) {
				continue;
			}
			int xv = x[j] - x[cr];
			int yv = y[j] - y[cr];
			double vn = kraad(xv, yv);
			if (vn > n2 && vn < n1) {
				ok = 0;
				break;
			}
		}
		if (!ok) continue;
		dt[i].push_back(make_pair(i - 1, tp));
		dt[i - 1].push_back(make_pair(i, tp));
	}
}
```

Kui leida iga ringipaari ühine puutuja, on algoritmi keerukuseks $O(N^3)$, kui aga ringid eelnevalt kauguse järgi sorteerida, saab kahandada seda $O(N^2 \log N)$-ile.

12.9 KONTROLLÜLESANDED

12.9.1 Liitumine

Juku ehitas endale uue maja. Nagu ikka, tahab Juku oma majja ka elektrit saada. Selle jaoks on vaja aga luua uus elektriliin Juku maja ning pealiini vahele. Pealiin on N lõigust koosnev murdjoon, mis läbib metsa Juku maja läheduses. Nagu ikka, maksab iga meeter ning seega on vaja leida lühima sirglõigu, mis ühendab maja ning pealiini, pikkust.
Sisendi esimesel real on kaks täisarvu, Juku maja koordinaadid. Järgmisel real on täisarv N, pealiini lõikude arv. Järgmisel N+1 real on järjest iga liini lõigu otspunktid. Enne on x-koordinaat, seejärel on y-koordinaat.
Väljundisse kirjutada kaks arvu: lähima punkti x ja y koordinaat.

NÄIDE 1:

```
6 -3
3
0 1
5 5
9 -5
15 3
```

Vastus:

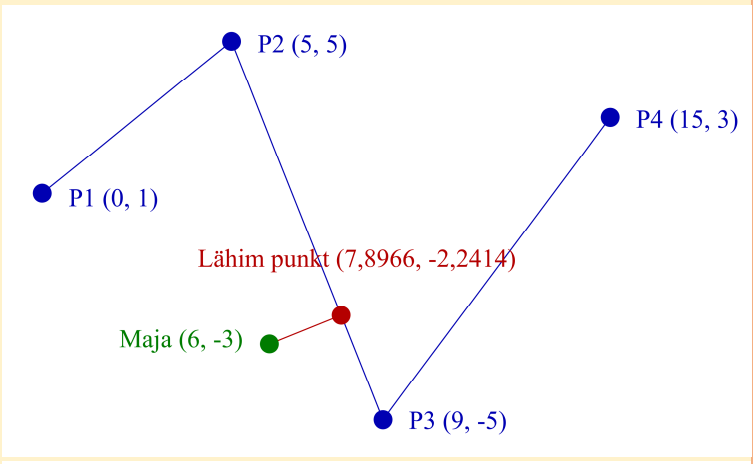
```
7,8966 -2,2414
```

NÄIDE 2:

```
0 0
1
1 0
2 0
```

Vastus:

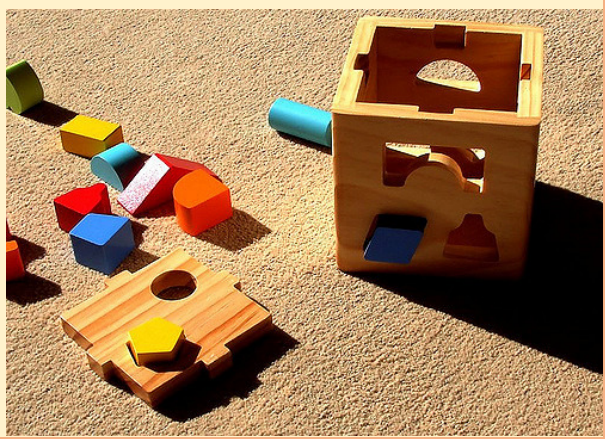
```
1.0000 0.0000
```



Esimesele näitele vastav joonis

12.9.2 Augumäng

Väike Kadi üritab kolmnurki toppida läbi ringikujuliste aukude. Kolmnurk on antud mõistes perfektne kolmnurkne püstprisma, mille pikkus on suurem kui mistahes ringi diameeter. Väikesel Kadil on N (1 ≤ N ≤ 26) kolmnurka ning M (1 ≤ M ≤ 100) ringi. Aita teda ning ütle, missuguseid kolmnurki saab läbi missuguste ringide toppida.
Sisendi esimesel real on täisarv M. Järgmisel real on M tühikutega eraldatud arvu, ringide diameetrid. Järgmisel real on täisarv N. Järgmisel N real on kolm tühikutega eraldatud arvu, vastava kolmnurga küljepikkused.
Väljundisse kirjutada N rida. Iga rea esimene arv on k, mitmesse ringi vastav kolmnurk mahub. Edasi on k tühikutega eraldatud täisarvu, ringide indeksid kasvavas järjekorras, kuhu kolmnurk mahub.



NÄIDE:

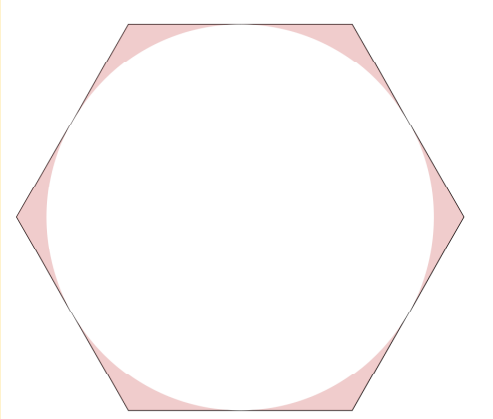
```
3
4.0 5.0 6.0
3
3.0 4.0 5.0
5.0 5.0 5.0
4.5 4.5 8.0
```

Vastus:

```
2 2 3
1 3
0
```

12.9.3 Pallimäng

Kettamaailmas tuntud mängu, N-nurk palli, mängitakse korrapärase N-nurga (3 ≤ N ≤ 50) siseringjoone sees, kohtunikud seisavad N-nurga alal, mis jääb väljapoole siseringjoont ning pealtvaatajad kogunevad kohtadesse, mis asuvad küll N-nurga ümberringjoone piirides, kuid mitte N-nurgas endas. Sulle on ette antud N ning N-nurga pindala S (0 ≤ S ≤ 30000). Leida kohtunike ning pealtvaatajate alade pindalad.
Sisendis on ette antud kaks tühikuga eraldatud arvu, N ja S.
Väljundisse kirjutada kaks tühikuga eraldatud arvu, pealtvaatajate ala pindala ning kohtunike ala pindala.



NÄIDE 1:

```
3 0.43301
```

Vastus:

```
0.61418 0.17121
```

NÄIDE 2:

```
6 2.59808
```

Vastus:

```
0.54352 0.24188
```

NÄIDE 3:

```
9 6.18182
```

Vastus:

```
0.53226 0.25314
```

12.9.4 Nelinurgad

Sulle antakse ette neli punkti, millest ükski kolm ei asu ühel sirgel. Leida, kas need neli punkti moodustavad, ruudu, ristküliku, rombi, rööpküliku, trapetsi või nelinurga. Antud juhul tuleb nimetada esimene punktinelikule vastav nimetus. Kui on ruut, siis tuleb väljastada „Ruut“, kui on nelinurk, mis ei vasta ühelegi teisele tingimusele, tuleb väljastada „Nelinurk“.
Sisendi esimesel real on täisarv N (1 ≤ N ≤ 50000), punktinelikute arv. Järgmisel 4N real on kaks tühikutega eraldatud täisarvu x ja y (-10000 ≤ x,y ≤ 10000), punkti x ja y koordinaat.
Väljundisse kirjutada N rida. N-nda punktineliku liik.

NÄIDE:

```
6
0 0
2 0
2 2
0 2
0 0
3 0
3 2
0 2
0 0
8 4
5 0
3 4
0 0
2 0
3 2
1 2
0 0
5 0
4 3
1 3
0 0
5 0
4 3
1 4
```

Vastus:

```
Ruut
Ristkylik
Romb
Roopkylik
Trapets
Nelinurk
```

12.9.5 Üleujutus

Suur üleujutus on tabanud Muumiorgu ning sinu ülesanne on nüüd aru saada, kui sügav on vesi ning kui suur osa Muumiorust on veega kaetud. Muumiorg koosneb 10*10 meetristest ruutudest, millel on teatud kõrgus. Samuti teame, et Muumiorus on kanalisatsioon nii hea, et vesi saab alati voolata ükskõik kust kõige madalamasse punkti, kus vett ei ole.
Sisendi esimesel real on kaks täisarvu N ja M (1 ≤ N,M ≤ 30), Muumioru mõõtmed ruutudes. Järgmisel N real on M tühikutega eraldatud täisarvu, ruudu kõrgus merepinnast. Järgmisel real on täisarv V, Muumiorgu voolanud vee hulk kuupmeetrites.
Väljundisse kirjutada eraldi ridadele kaks arvu, Muumioru veetaseme kõrgus merepinnast ning protsent, kui suur osa Muumioru ruutudest jääb täielikult vee alla.

NÄIDE:

```
3 3
25 37 45
51 12 34
94 83 27
10000
```

Vastus:

```
46,67
66,67
```

12.9.6 Park

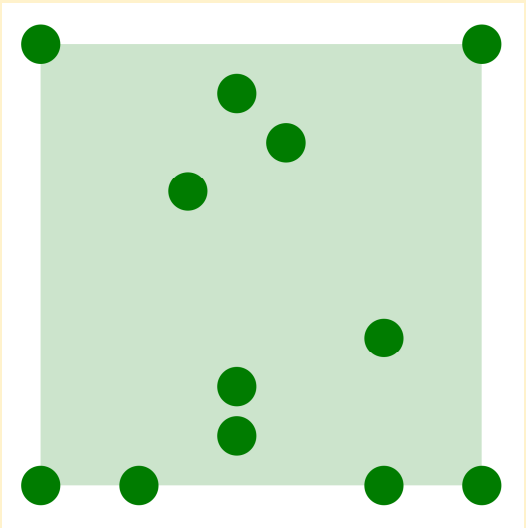
Linnavalitsus otsustas uut parki ehitama hakata. Selleks aga otsustati võtta mingisugune ala, kus kasvasid mõned üksikud puud ning võtta minimaalse ümbermõõduga hulknurk, mis sisaldab kõiki neid puid, ning muuta see ala pargiks. Leida tekkiva pargi pindala.
Sisendi esimesel real on täisarv N (1 ≤ N ≤ 100), pargis olevate puude arv. Järgmisel N real on kaks arvu, puu koordinaadid.
Väljundisse kirjutada üks arv, tekkiva pargi pindala.

NÄIDE:

```
12
1 1
5 3
4 7
1 10
6 8
3 1
10 1
8 1
5 9
10 10
5 2
8 4
```

Vastus:

```
81
```



12.9.7 Müür

Väike külake Süürias otsustas, et hea mõte oleks ümber küla müür ehitada. Selleks, et kulusid säästa, otsustati, et müür tuleb minimaalse ümbermõõduga, nii et kõik külas olevad majad oleksid müüriga piiratud. Leida see minimaalne ümbermõõt.
Sisendi esimesel real on täisarv N (1 ≤ N ≤ 100), külas olevate majade arv. Järgmisel N real on kaks arvu, maja koordinaadid.
Väljundisse kirjutada üks arv, ehitatava müüri ümbermõõt.

NÄIDE 1:

```
3
1 2
4 10
5 12,3
```

Vastus:

```
22,1
```

NÄIDE 2:

```
6
0 0
1 1
3,1 1,3
3 4,5
6 2,1
2 -3,2
```

Vastus:

```
19,66
```

NÄIDE 3:

```
7
1 0,5
5 0
4 1,5
3 -0,2
2,5 -1,5
0 0
2 2
```

Vastus:

```
12,52
```

12.9.8 Spioonid

Selleks, et tungida ülisuure valve all olevasse ettevõttesse ning sealt kohe signaale staapi saata, peab spioonifirma oma kommunikatsioonivarustust tükkidena hoidma, saates lisaks põhispioonidele ka tugispioone, kes kannavad endaga spetsialiseeritud tehnikat. Selleks, et välismaailmaga rääkida, peab põhispioon seisma tugispioonidest koosnevas kolmnurgas ning ta ongi ühenduses staabiga. Paraku on salatseval ettevõttel olemas *jammer*'id, mis blokeerivad põhispiooni signaali. Need *jammer*'id on peidetud pealtnäha tavaliste inimeste riietesse ning spioon, kes asub *jammer*'ite moodustatud kolmnurgas, ei saa staabiga suhelda hoolimata sellest, kas ta on sidekolmnurgas või mitte. Praegu loeme spiooni blokeerituks, kui tema signaali aktiivselt blokeeritakse, ühenduses, kui tal on ühendus staabiga ning levist väljas, kui teda aktiivselt ei blokeerita, kuid ta lihtsalt ei ole ühenduses. On antud *jammer*'ite, tugispioonide ning põhispioonide asukohad. Leida iga spiooni kohta, kas ta on blokeeritud, ühenduses või levist väljas.

Sisendi esimesel real on kolm tühikutega eraldatud täisarvu, J, T, P ($0 \le J,T,P \le 200$), *jammer*'ite, tugispioonide ning põhispioonide arv. Järgmisel J real on kaks tühikuga eraldatud täisarvu vahemikus -500 kuni 500, *jammer*'i koordinaadid. Järgmisel T real on kaks tühikuga eraldatud täisarvu vahemikus -500 kuni 500, tugispiooni koordinaadid. Järgmisel P real on kaks tühikuga eraldatud täisarvu vahemikus -500 kuni 500, põhispiooni koordinaadid.

Väljundisse kirjutada P stringi, millest igaüks on eraldi real. Kui vastav põhispioon on blokeeritud, kirjutada „Blokeeritud". Kui spioon on aga ühenduses, väljastada „Yhenduses". Kui ta on aga levist väljas, tagastada „Levist v2ljas".

NÄIDE 1:

```
3 3 2
0 0
10 0
0 10
20 20
20 0
0 20
5 5
15 15
```

Vastus:

```
Blokeeritud
Yhenduses
```

NÄIDE 2:

```
3 3 1
0 0
10 0
0 10
20 20
20 0
0 20
40 40
```

Vastus:

```
Levist v2ljas
```

12.9.9 Märklaud

Märklaud koosneb N (1 ≤ N ≤ 10) alast, mis on nummerdatud arvudega 0 kuni N-1. Iga ala märklaual on kas ristkülik, ring või kolmnurk ning annab 2^i punkti, kus i on ala number. Mängija viskab M (1 ≤ M ≤ 10) noolt vastu märklauda. Nool saab ala poolt antavaid punkte parajasti siis, kui see asub selle ala sees. Leida iga noole kohta, mitu punkti see saab.
Sisendi esimesel real on kaks täisarvu, N ja M. Järgmisel M real on mingi täht c, mis võib olla kas r, c või t. Kui see on r, on tegemist ristkülikuga ning samal real järgneb 4 tühikutega eraldatud arvu, ristküliku ülemise vasaku ning alumise parema nurga koordinaadid. Kui see on c, on tegemist ringiga ning järgmised 3 tühikutega eraldatud arvu on selle keskpunkti koordinaadid ning raadius. Kui see on t, on tegemist kolmnurgaga ning järgmised 6 arvu on selle iga tipu koordinaadid. Järgmisel M real on kaks arvu, mis näitavad noole asukohta. Kõik koordinaadid on vahemikus -1000 kuni 1000.
Väljundisse kirjutada M täisarvu, vastava noole poolt saadud punktisumma.
NÄIDE:

```
9 6
r 8.5 17.0 25.5 -8.5
c 20.2 7.3 5.8
t -1.0 -1.0 10.1 2.2 .4 1.4
r 0.0 10.3 5.5 0.0
c -5.0 -5.0 3.7
t 20.3 9.8 10.0 -3.2 17.5 -7.7
r 2.5 12.5 12.5 2.5
c 5.0 15.0 7.2
t -10.0 -10.0 10.0 25.0 30.0 -10.0
2.0 2.0
4.7 5.3
6.9 11.2
20.0 20.0
17.6 3.2
-5.2 -7.8
```

Vastus:

```
264
328
448
0
291
272
```

12.9.10 Viirus

Internet on ristkülik mõõtmetega W*H (1 ≤ W,H ≤ 1000), mida läbib N (0 ≤ N ≤ 100) sirgjoonelist tulemüüri. Iga tulemüür ulatub täpselt servast serva. Viirus sattus internetis ühte punkti koordinaatidega x,y (0 ≤ x ≤ W; 0 ≤ y ≤ H) ning nakatas kogu selle ala, mis ei olnud tema eest tulemüüriga kaitstud. Leida nakatunud interneti pindala.
Sisendi esimesel real on viis tühikutega eraldatud täisarvu, N, W, H, x ja y. Järgmisel N real on neli tühikutega eraldatud täisarvu, tulemüüri alg- ja lõpppunkt. Võib eeldada, et tulemüür algab ning lõpeb interneti servas.
Väljundisse kirjutada üks arv, nakatunud interneti pindala.

NÄIDE:

```
3 100 100 20 30
0 0 100 100
100 0 0 100
0 40 40 100
```

Vastus:

```
1780,000
```