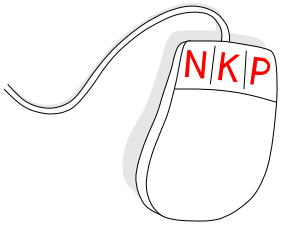


Opgavenset

voor de **finale** van het

Nederlands **K**ampioenschap **P**rogrammeren

1997

submitnaam	opgave	pagina's
RSA	RSA	1 — 2
KLOK	Klokkijken	3 — 4
MANHATTAN	Manhattan	5 — 8
MINKOWSKI	Minkowski-som	9 — 10
DNA	Jurassic Park	11 — 12
TURING	Turing-compiler	13 — 16
KLEUREN	Driekleuring	17 — 18
RIEMANN	Het vermoeden van Riemann	19 — 20
SIMULATIE	Simulatie van knopen in grafen	21 — 22

Succes!

RSA

Om geheime berichten te versturen moet je deze berichten coderen. Een methode om berichten te coderen en te decoderen heet een *cryptosysteem*. Een veelgebruikt cryptosysteem is **RSA**, dat staat voor de initialen van de bedenkers ervan: Rivest, Shamir en Adleman. Het ontleent zijn bijna-onkraakbaarheid aan het feit dat het erg moeilijk is grote getallen in factoren te ontbinden.

RSA werkt als volgt. De codeur C kiest twee grote priemgetallen p en q, en kiest verder twee getallen e en d zodanig dat ed modulo $\phi(pq)$ gelijk is aan 1. Hier is ϕ de Euler ϕ-functie: $\phi(pq) = (p-1)(q-1)$. Hij/zij maakt het produkt pq bekend, evenals het getal e. Het andere getal, d, houdt hij/zij angstvallig geheim. De decodeur D is de enige die d mag weten.

Als C nu een bericht wil versturen (laten we zeggen, een bepaald getal b), dan berekent hij/zij $c = b^e$ modulo pq. Dit gecodeerde bericht c verstuurt hij/zij naar D. Wat D moet doen om het bericht b te ontcijferen is bijna hetzelfde: hij/zij berekent c^d modulo pq. Dit is gelijk aan b^{ed} modulo pq, en omdat $ed = 1$ modulo $\phi(pq)$ is dit gelijk aan b^1 modulo pq = b (zie ook een boek/dictaat over elementaire getaltheorie). Feitelijk zijn de codeer- en decodeer-slagen gelijk.

Het vinden van het getal d, gegeven pq en e, blijkt neer te komen op het factoriseren van pq. Dit is erg moeilijk voor grote getallen. Om **RSA** veilig te laten zijn, moet pq dus groot zijn: 150 decimalen is (tot nu toe) veilig maar meestal wordt pq zo'n 300 decimalen groot gemaakt.

Probleem

Er wordt jou gevraagd een programma te schrijven dat een (de)codeer-slag uitvoert. Voor het gemak gebruiken we alleen getallen die zijn geschreven m.b.v. een 26-tallig stelsel, om zo een stuk tekst (weliswaar zonder spaties en leestekens) te kunnen versturen. De *digits* zijn de kleine letters `a`=0, `b`=1, enzovoorts, tot en met `z`=25. Zoals gebruikelijk staat het meest significante digit voorop. Op deze manier is het getal direct te gebruiken als bericht. Als spatie zou bijvoorbeeld de letter `q` of `x` gebruikt kunnen worden.

Invoer

De invoer begint met een regel met daarop het aantal runs. Daarna volgen per run drie regels. Op de eerste, tweede en derde regel staat een 26-tallig-geschreven getal a, b en c respectievelijk. Er is gegeven dat $0 < a, b, c < 26^{80}$ en dat $b < a$.

Z.O.Z.

Uitvoer

De uitvoer bestaat per run uit een enkele regel met daarop b^c modulo a, van de corresponderende groep van drie getallen in de invoer, genoteerd in 26-tallig stelsel. Per definitie geldt dat $0 \leq b^c$ modulo $a < a$. Bovendien mag het getal niet worden voorafgegaan door leidende nul-digits (`a`'s).

Voorbeeld

Bij de **invoer**

```
2
vjyklfr
uvpfbyr
d
wtnbgpgzgupezl
fhxjlhrjgqvatl
cpbyoosdqfuoov
```

hoort de **uitvoer**

```
kijkuit
grotespin
```

Klokkijken

Men neme een digitale klok. En zie, er staan cijfers op. Iedere seconde wordt het display van de klok bijgewerkt. Meestal hoeft er maar één cijfer veranderd te worden, eens in de tien seconden meer cijfers.

Probleem

Gegeven twee tijdstippen op een digitale vierentwintiguurs klok, hoeveel cijfers zijn er cumulatief veranderd om van het ene naar het andere tijdstip te komen?

Invoer

Een regel met het aantal runs, daarna per run een regel met het eerste tijdstip t_1 en nog een regel met het tweede tijdstip t_2. Een tijdstip bestaat uit acht tekens: twee cijfers die de uren aangeven, een dubbele punt, twee cijfers die de minuten aan geven, een dubbele punt en twee cijfers die de seconden aangeven.

Tijdstip t_2 valt niet voor t_1. Dit kan betekenen dat t_2 op de volgende dag valt (t_2 kan tot 24 uur later vallen dan t_1).

Uitvoer

Per run één regel met daarop het getal dat aangeeft hoeveel cijfers er cumulatief veranderd zijn om van t_1 naar t_2 te komen.

Voorbeeld

Bij de **invoer**

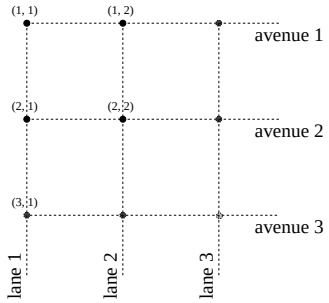
```
2
01:02:03
01:02:13
23:59:59
00:00:00
```

hoort de **uitvoer**

```
11
6
```

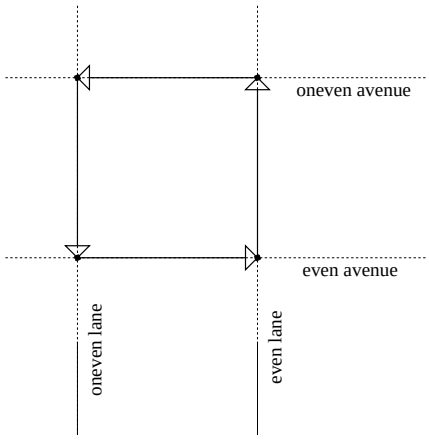
Manhattan

Amerika is een mooi land voor stedenplanners. Het enige wat ze hoeven te doen om een stad te ontwerpen, is een $n \times m$-rooster tekenen, de horizontale lijnen *avenues* en de verticale lijnen *lanes* te noemen en wat winkelcentra erin schetsen. Het is zelfs zo gemakkelijk dat ze niet eens straatnamen hoeven te bedenken: ze geven ze gewoon een nummer (zie figuur 1). Zo af en toe treft het dat een kruispunt afgesloten is voor verkeer — denk aan Central Park of de opnameset van Jurassic Park 3: "Killer-Velociraptors eat pedestrians" — maar dat hindert niet zoveel: je rijdt gewoon een blocky om en je komt er alsnog wel.



Figuur 1: Nummering van de *avenues* en *lanes*

Dan doet éne Mr. Adrian Nuisance z'n intrede in het politieke bestel. Deze meneer wil het verkeersbeeld drastisch gaan reorganiseren. Zijn plan is als volgt: maak de lanes en avenues één-richting, waarbij je de richting laat afhangen van de pariteit (d.w.z. even/oneven) van het nummer van de lane of avenue: zie onderstaande figuur voor een schematische weergave.



Figuur 2: Verkeersstromen na doorvoering van het plan

Probleem

Nu wil Mr. Nuisance weten in hoeverre kruispunten nog bereikbaar zijn na doorvoering van zijn plan. Omdat het oog ook wat wil, moet dit op een mooie grafische manier gerepresenteerd worden. Daarom heeft hij besloten gebruik te maken van een zogenaamd grafisch histogram: de klassen van het histogram worden gerepresenteerd d.m.v. *zwart* en de stoplichtkleuren *rood, oranje, groen* aan de hand van onderstaande tabel:

klasse	quantitatief	kleur
niet bereikbaar	$b_{(a,l)} = 0$	`Zwart`
matig bereikbaar	$0 < b_{(a,l)} < \frac{1}{3}B$	`Rood`
redelijk bereikbaar	$\frac{1}{3}B \leq b_{(a,l)} < \frac{2}{3}B$	`Oranje`
goed bereikbaar	$b_{(a,l)} \geq \frac{2}{3}B$	`Groen`

Tabel 1: De klassen van het grafisch histogram

Hierin is $b_{(a,l)}$ de **bereikbaarheid** van kruispunt (a, l). Deze is gedefinieerd als het aantal kruispunten van waaruit kruispunt (a, l) in niet meer dan s stappen te bereiken is, waarbij kruispunt (a, l) zelf niet meegerekend wordt. De waarde van s wordt vooraf gegeven. Verder is B de maximale bereikbaarheid, d.w.z. :

$$B = \max \left\{ b_{(a,l)} \mid 1 \leq a \leq n, 1 \leq l \leq m \right\}$$

Gevraagd wordt dus een programma te schrijven dat een dergelijk grafisch histogram levert.

Invoer

De invoer begint met één regel met daarop het aantal steden waarvan de interne bereikbaarheid getoetst moet worden. Per stad volgt dan een regel met het aantal avenues n, het aantal lanes m en het maximaal aantal te zetten stappen s, respectievelijk. Er geldt dat $1 \leq n, m \leq 100$ en dat $s \geq 1$. Vervolgens komt per avenue een regel met m karakters, die de kruispunten (in de volgorde lane 1 – m) als volgt beschrijven: is een kruispunt toegankelijk, dan staat er een plusje ('`+`'); anders staat er een kruisje ('`x`'), wat overeenkomt met een afgesloten kruispunt.

Uitvoer

De uitvoer bestaat per stad uit het grafisch histogram voor die stad. Waar in de invoer een kruisje staat, komt in de uitvoer ook een kruisje te staan. Waar in de invoer een plusje staat, komt in de uitvoer een karakter '`Z`', '`R`', '`O`' of '`G`' te staan, overeenkomend met de bereikbaarheid van dat kruispunt. De histogrammen dienen gescheiden te worden door één lege regel.

Voorbeeld

Bij de **invoer**

```
4
2 2 1
++
++
3 3 2
+++
+++
+++
4 5 4
+++++
+++++
+++++
+++++
4 5 4
+++++
++xx+
+xxx+
+++++
```

hoort de **uitvoer**

```
GG
GG

GGZ
OGG
GOG

GGOGZ
OGGGG
GGGGO
OOGGG

GGRRZ
GOxxR
GxxxR
OOOOG
```

Minkowski-som

Enkele definities:

- De Minkowski som C van twee sets van coördinaten A en B is gedefinieerd als $C = \{ a + b \mid a \in A \wedge b \in B \}$
- Een *convex* polygon is als volgt gedefinieerd: een lijn l snijdt ten hoogste 2 edges van het polygon of valt met ten hoogste één edge van het polygon samen.
- Een *strikt* polygon bevat niet drie (of meer) vertices die op één lijn liggen.

Probleem

Bepaal de Minkowski-som van twee strikte, convexe polygonen.

Invoer

De invoer begint met een regel met daarop het aantal runs. Daarna volgen per run twee strikte, convexe polygonen: ieder polygon bestaat uit één regel met het aantal vertices n en één regel met de vertices zelf, waarbij $1 \leq n \leq 10000$ (tienduizend).

Een vertex bestaat uit een x- en een y-coördinaat, gescheiden door een spatie. Vertices worden gescheiden door twee spaties. De eerste vertex is de vertex met de kleinste y-coördinaat van het polygon (zijn er verscheidene vertices met dezelfde kleinste y-coördinaat, dan is de eerste vertex degene uit die groep met de kleinste x-coördinaat). Daarna volgt de rest van de vertices tegen de klok in.

Uitvoer

Per run is de uitvoer de Minkowski-som van de twee betreffende polygonen als strikt polygon. Het uitvoerformaat en de ordening van het polygon moet gelijk zijn aan dat van een polygon bij de invoer.

Z.O.Z.

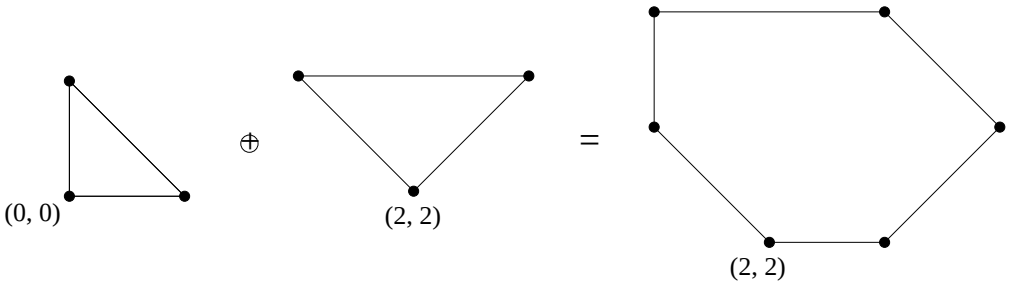
Voorbeeld

Bij de **invoer**

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

hoort de **uitvoer**

```
6
2 2   3 2   4 3   3 4   1 4   1 3
```



Figuur 3: Plaatje behorend bij de voorbeeldinvoer

Jurassic Park

Sinds enige tijd is het mogelijk om kleine stukjes dinosauriër-DNA (van een Tyrannosaurus Rex, om maar iets te noemen) te winnen. Een onderzoeker (we noemen de naam van Michael Crichton niet) wil met dit DNA de dinosauriër nieuw leven in blazen.

Hij wil dit doen door gevonden T-rex-DNA te combineren met het DNA van bestaande beesten die in de verte wel wat lijken op dino's, bijvoorbeeld varanen. Nu weet de onderzoeker echter niet welk beest daarvoor het meest geschikt is. Gelukkig heeft hij van een aantal dieren een aardig gedeelte van hun DNA *gescand*.

Van het gescande T-rex-DNA bepaalt hij een gedeelte dat karakteristiek is voor het hele DNA. Vervolgens kijkt hij welk hedendaags dier de beste *match* oplevert, d.w.z. van welk beestje het DNA het grootste aantal voorkomens van het T-rex-DNA heeft (waarbij overlap wordt meegeteld).

Zoals bekend is **D**eoxyribo**N**ucleic**A**cid een lange dubbele streng van aminozuren (zie figuur hieronder), waarvan er steeds twee aan elkaar liggen, de zogenaamde *base-paren*. Er zijn slechts twee base-paren, nl. `a -- t, c -- g`. Dit is handig, want om DNA op te slaan hoef je maar één helft van de streng op te slaan, waaruit je de andere weer kunt afleiden.

```
... — a — g — c — t — ...
      |   |   |   |
... — t — c — g — a — ...
```

Probleem

Nu is DNA over het algemeen heel erg lang. Zonder computer zal de onderzoeker dan ook niet echt ver komen. Het bedrijf waar de onderzoeker werkt, InGen™, schakelt jou in om een programma te schrijven voor het bepalen van de beste match.

Invoer

De invoer begint met een regel met daarop het aantal base-paren m van het T-rex-DNA, waarbij $1 \leq m \leq 50.000$ (vijftigduizend). Vervolgens komt er een regel ter lengte van m karakters met het gescande T-rex-DNA (elk karakter komt uit het base-alphabet '`a`', '`c`', '`g`', '`t`' en stelt het amonizuur uit steeds één helft van de DNA-streng voor).

Hierna volgt een regel met daarop het aantal dieren t, waarbij $2 \leq t \leq 100$ dat hij wil testen op geschiktheid. Hierop volgen t keer twee regels. De eerste regel bevat een getal n, waarbij $m \leq n \leq 50.000.000$ (vijftig miljoen), dat de lengte van het gescande DNA van het betreffende beest voorstelt. De tweede regel bevat n karakters met het gescande beesten-DNA.

Uitvoer

De uitvoer bestaat uit één regel met daarop, gescheiden door één spatie, het rangnummer van het dier wiens DNA het meeste aantal voorkomens van het gegeven T-rex-DNA heeft en dat aantal voorkomens zelf. Indien er twee beesten zijn wiens DNA evenveel voorkomens van het T-rex-DNA bevat, dient het rangnummer van het dier met het kortst bekende DNA gekozen te worden. **Opmerking:** De invoer bevat niet twee saurus-achtigen, die even geschikt zijn en van welke evenveel DNA gescand is.

Restrictie

Omdat InGen™ nog zo'n jong bedrijfje is en al het geld gaat zitten in de benodigde biochemische apparatuur, moet men het doen met wat tweedehands computers van een jaar of tien oud. Dit heeft zo z'n gevolgen voor het beschikbare geheugen...

De te gebruiken ruimte is gelimiteerd tot 250kB *in totaal*, dus inclusief op de **heap** gealloceerde ruimte.

Voorbeeld

Bij de **invoer**

```
5
accgt
4
10
accgtaccga
20
accgtggaccgtaaagggtt
30
accgtgagcagcaccgtgagcagcaccgtg
40
aaaaaaaaaaaaaaaaaccgtaaaaaaaaaaaaaaaaaa
```

hoort de **uitvoer**

```
3 3
```

Turing-compiler

Een Turingmachine is een eindige stappen-automaat, die werkt op een *band*. Een band is een, in beide richtingen oneindig uitgebreide, 1-dimensionale aaneenschakeling van cellen. In elke cel staat een symbool, dat uit een vaste, eindige verzameling komt: het bandalfabet Σ. Een Turingmachine heeft een lees/schrijf-kop, die zich boven één van de cellen bevindt en het symbool in die cel kan lezen of een symbool in die cel kan schrijven. De lees/schrijf-kop kan zich zowel naar links als naar rechts over de band bewegen; één cel per stap.

Formeel kan een Turingmachine worden gedefinieerd door een tupel (Σ, S, τ, q_0), waarbij:

Σ: een eindige verzameling van symbolen: het (band-)alfabet
S: een eindige verzameling van toestanden
τ: een functie $S \times \Sigma \to S \times \Sigma \times \{\texttt{L}, \texttt{R}, \texttt{h}, \texttt{.}\}$: de transitiefunctie
q_0: de begintoestand

We beschouwen Turingmachines waarvoor geldt dat:

- $\Sigma = \{\text{`}\texttt{a}\text{'}, \ldots, \text{`}\texttt{g}\text{'}, \text{`}\texttt{\~{}}\text{'}\}$ (`~` is het leeg-symbool)
- de verzameling toestanden S gelijk is aan $\{0, \ldots, n\}$ voor een zeker geheel getal $n \geq 0$
- $q_0 = 0$

Een Turingmachine die zich in toestand q bevindt, handelt als volgt:

1. lees het symbool c dat zich in de cel onder de kop (de actuele cel) bevindt
2. bereken $(q', c', d) = \tau(q, c)$
3. schrijf het symbool c' in de actuele cel
4. schuif de kop een positie naar links als $d = \texttt{L}$, naar rechts als $d = \texttt{R}$; stop als $d = \texttt{h}$; laat de kop staan als $d = \texttt{.}$
5. als de Turingmachine niet gestopt is (dus als $d \neq \texttt{h}$), maak dan q' de huidige toestand

Het ontwerpen van een Turingmachine die een gegeven probleem oplost, is nogal lastig. Het zou dan ook handig zijn als er een compiler was die een programma, geschreven in een wat praktischer programmeertaal, omzette naar een Turingmachine. Gelukkig hoeft dat taaltje niet al te krachtig te zijn, om (uiteindelijk) krachtige programma's te schrijven, zolang de taal maar een zekere vorm van structurering kent.

Z.O.Z.

Een (aanzet tot een) dergelijke taal is de taal `PANP` — dat staat voor *"PAnp is Not Pascal"* — die de volgende instructies kent:

instructie	actie
`L`	schuif een positie naar links
`R`	schuif een positie naar rechts
`h`	stop de Turingmachine
`.`	laat de kop staan
`w(<symb>,<dir>)`	schrijf symbool `<symb>` op de band en verplaats de kop volgens `<dir>`
`l(<symb>,<frag>)`	test of het symbool `<symb>` onder de kop staat; zo nee, voer dan de code `<frag>` uit en herhaal dit net zo lang tot `<symb>` onder de kop staat of de Turingmachine gestopt is bij de uitvoering van `<frag>`

Grammatica

Een `PANP`-programma `<prog>` voldoet aan de volgende grammatica:

```
<prog>  ::= <frag> !
<frag>  ::= <instr> | <instr>;<frag>
<instr> ::= <dir> | w(<symb>,<dir>)
                  | l(<symb>,<frag>)
<dir>   ::= L | R | h | .
<symb>  ::= a | b | c | d | e | f | g | ~
```

Spaties en regelovergangen dienen genegeerd te worden.

Probleem

Gevraagd wordt een vertaler te schrijven, die een gegeven `PANP`-programmatekst omzet in een bijbehorende transitiefunctie τ. De Turingmachine (Σ, S, τ, q_0) moet voldoen aan de eis dat **tijdens** executie de handelingen van de lees/schrijf-kop conform het `PANP`-programma zijn.

Invoer

De invoer bestaat uit het te vertalen programma, waarvan gegeven is dat de programmatekst

1. correct is m.b.t. boven gegeven grammatica
2. niet meer dan 5.000 (vijfduizend) voorkomens van instructies bevat

Uitvoer

De uitvoer bestaat uit een tabel die de transitiefunctie τ beschrijft. De tabel begint met een regel

␣q␣␣␣␣␣a␣␣␣␣␣␣b␣␣␣␣␣␣c␣␣␣␣␣␣d␣␣␣␣␣␣e␣␣␣␣␣␣f␣␣␣␣␣␣g␣␣␣␣␣␣~

die de tabellering weergeeft ('␣' staat voor een spatie). Dan volgen er $\#S = n + 1$ regels, waarbij $n + 1$ maximaal 10.000 (tienduizend) groot is. De toestandsbeschrijvingen dienen op volgorde van toestandsnummer te verschijnen. Elk van die regels bestaat uit:

1. het nummer q van de toestand die beschreven wordt, uitgedrukt in precies vier plaatsen (spaties i.p.v. leidende nullen)
2. voor elk symbool uit het alfabet in de volgorde 'a' – 'g', '~' een beschrijving van de transitiefunctie $\tau(q, c) = (q', c', d)$:
   (a) één scheidende spatie
   (b) q', uitgedrukt in precies vier plaatsen (spaties i.p.v. leidende nullen)
   (c) het symbool c'
   (d) de richting d

Na die $n + 1$ regels volgt een regel met een sterretje/asterisk ('*').

Voorbeeld

Bij de **invoer**

```
l(a, l(b, R) ; R) ;
w(b, L)
!
```

past de **uitvoer**

```
 q      a      b      c      d      e      f      g      ~
   0    4a.    1b.    1c.    1d.    1e.    1f.    1g.    1~.
   1    2a.    3b.    2c.    2d.    2e.    2f.    2g.    2~.
   2    1aR    1bR    1cR    1dR    1eR    1fR    1gR    1~R
   3    0aR    0bR    0cR    0dR    0eR    0fR    0gR    0~R
   4    5bL    5bL    5bL    5bL    5bL    5bL    5bL    5bL
   5    5ah    5bh    5ch    5dh    5eh    5fh    5gh    5~h
*
```

Driekleuring

Al millenia lang reizen mensen over de aardbol (al dachten ze lange tijd dat die plat was). Thor Heyerdahl beweert dat de Egyptenaren de wereldzeeën overstaken. Marco Polo trok over land naar China. Al snel werd enig overzicht verkregen door landkaarten te maken. Deze kaarten bevatten grote stukken wit aan de randen waar men nog nooit was geweest, een bonte schakering van gebieden waar men al wel was geweest.

Jullie kennen allemaal (hopelijk) de vierkleurenstelling: een landkaart kan met vier kleuren ingekleurd worden, zodanig dat voor twee landen die aan elkaar grenzen twee verschillende kleuren worden gebruikt. Dat helpt echter voor geen meter bij het kiezen van een kleurverdeling van een landkaart.

Probleem

Wij vragen nu van jullie een programma te schrijven dat het antwoord geeft op de vraag of een gegeven landkaart met *drie* kleuren ingekleurd kan worden, zodanig dat twee aan elkaar grenzende landen een verschillende kleur krijgen, waarbij de landkaart enige restricties kent:

- Er komen niet meer dan drie landen samen in één punt op de kaart (nog wel drie-landenpunten, maar geen hoger-dan-drie-landenpunten).
- Een land is een aaneengesloten gebied (geen enclaves, geen ambassades).

Invoer

Een landkaart is een vlakke graaf, waarbij landen gerepresenteerd worden door vertices en grenzen door edges. De invoer begint met een regel met daarop het aantal runs. Daarna volgen per run twee regels:

1. De eerste regel bevat het aantal vertices $\mathcal{V}$, waarbij $3 \leq \mathcal{V} \leq 10000$ (tienduizend) en het aantal edges $\mathcal{E}$.
2. De tweede regel bevat alle $\mathcal{E}$ edges. Een edge bestaat uit twee nummers, v_1 en v_2, verwijzend naar vertices. Er geldt $1 \leq v_1 < v_2 \leq \mathcal{V}$. De twee nummers zijn gescheiden door één spatie; twee edges door twee spaties.

Uitvoer

Per run moet worden aangegeven of een driekleuring van de landkaart wel of niet mogelijk is. Is dit mogelijk, lever dan een regel met de string "`mogelijk`"; zo nee, lever dan een regel met de string "`onmogelijk`".

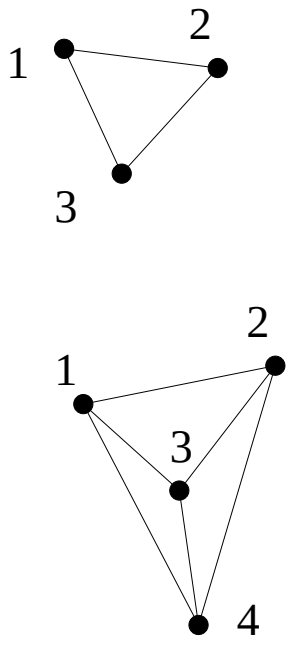
Voorbeeld

Bij de **invoer**

```
2
3 3
1 2  1 3  2 3
4 6
1 2  1 3  1 4  2 3  2 4  3 4
```

hoort de **uitvoer**

```
mogelijk
onmogelijk
```



Figuur 4: Twee grafen die de voorbeeldinvoer representeren

Het vermoeden van Riemann

Andrew Wiles heeft enkele jaren geleden (voortbordurend op het werk van een heleboel andere wiskundigen) het welbekende vermoeden van Fermat bewezen. Nu hij iedereen van de juistheid van zijn bewijs overtuigd heeft, heeft hij alle tijd om zich op een ander vermoeden te storten. Hij hoeft zich niet te vervelen, want er zijn er nog meer dan genoeg van.

Eén van die vermoedens is dat van Riemann, dat iets zegt over de nulpunten van de zogenaamde Riemann-Zeta-functie. Er is ook een herformulering van dit vermoeden in termen van priemgetallen. Daarvoor introduceren we een aantal begrippen.

Definities:

1. Een positief geheel getal heet *kwadraatvrij* als het niet deelbaar is door een geheel kwadraat ongelijk aan 1.
2. Een kwadraatvrij positief geheel getal heet *priem even/oneven* als het een product is van een even/oneven aantal verschillende priemgetallen. Het getal 1 beschouwen we als priem even.
3. We noteren het verschil van het aantal priem even en priem oneven positieve gehele getallen kleiner dan of gelijk aan n als $V(n)$.

Voorbeelden:

1. 12 is niet kwadraatvrij (want deelbaar door 2^2).
2. $15 = 3.5$ is priem even, terwijl $30 = 2.3.5$ priem oneven is.
3. $V(30) = 3$, want er zijn 11 kwadraatvrije priem oneven getallen ≤ 30 (nl. 2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 30) en 8 kwadraatvrije priem even getallen (nl. 1, 6, 10, 14, 15, 21, 22, 26).

Riemann-hypothese: Zij $\epsilon > 0$. Dan bestaat er een positieve gehele N zódanig dat voor alle $n > N$ geldt dat

$$V(n) \leq n^{\frac{1}{2}+\epsilon}$$

Probleem

Hoewel Wiles niet zoveel waarde aan computers schijnt te hechten, zou hij het toch wel handig vinden om al die $V(n)$ niet met de hand uit te hoeven rekenen. Gevraagd wordt dus om een programma te schrijven dat $V(n)$ uitrekent voor gegeven n.

Invoer

De **invoer** begint met een regel met het aantal testgevallen $m \geq 1$. Daarna volgen er m regels met op elke regel een geheel getal n waarvoor geldt dat $1 \leq n \leq 1.000.000$ (één miljoen).

Uitvoer

De **uitvoer** bestaat uit m regels met op elke regel de waarde van $V(n)$ voor de corresponderende n in de invoer.

Voorbeeld

Bij de **invoer**

```
3
1
30
1000
```

hoort de **uitvoer**

```
1
3
2
```

Hint

Beschouw de **Möbius-mu-functie**

$$\mu(n) = \begin{cases} 0, & n \text{ niet kwadraatvrij} \\ (-1)^{\#\{p \text{ priem} \mid p \text{ deelt } n\}}, & n \text{ kwadraatvrij} \end{cases}$$

zodat

$$V(n) = \left| \sum_{u=1}^{n} \mu(u) \right|$$

Simulatie van knopen in grafen

Een gelabelde gerichte graaf bestaat uit een verzameling knopen V en een verzameling gelabelde gerichte ribben $E \subseteq V \times \mathbb{N} \times V$. Een ribbe is dus een triple (x, r, y) waarbij x en y knopen zijn en r een natuurlijk getal, het label van de ribbe.

We definiëren nu het begrip *simulatie* tussen knopen: Een knoop x in graaf (V, E) wordt gesimuleerd door een knoop y in graaf (W, F) (notatie $(x, y) \in \mathit{Sim}$), als er een binaire relatie $R \subseteq V \times W$ is, zodanig dat $(x, y) \in R$ en dat voor elk paar $(p, q) \in R$ en elke ribbe $(p, r, v) \in E$ er een knoop $w \in W$ is met $(q, r, w) \in F$ en $(v, w) \in R$.

Probleem

De opdracht luidt nu: ontwerp een programma dat een rij van paren van dergelijke grafen inleest en dan voor elk paar van grafen, zeg (V, E) en (W, F), de simulatie-relatie $\mathit{Sim} \subseteq V \times W$ berekent en afdrukt.

Invoer

De invoer begint met een regel met daarop het aantal runs. Daarna volg per run twee grafen. Een graaf bestaat uit:

1. een regel met daarop het aantal knopen n en het aantal ribben r, $0 \leq n < 300$.
2. een regel met r ribben, elk bestaande uit een knoop k, een label l en een knoop m, gescheiden door een spatie. Ribben worden gescheiden door twee spaties.

Uitvoer

Per run dient de uitvoer te bestaan uit de complete lijst met paren van knopen uit de simulatie-relatie van de twee grafen van die run. Twee knopen van een paar worden gescheiden door één spatie, twee paren worden gescheiden door twee spaties. Per relatie dienen de paren lexicografisch geordend te worden afgedrukt: $p\ v$ voor $q\ w$ als $p < q$, of $p = q$ en $v < w$.

Z.O.Z.

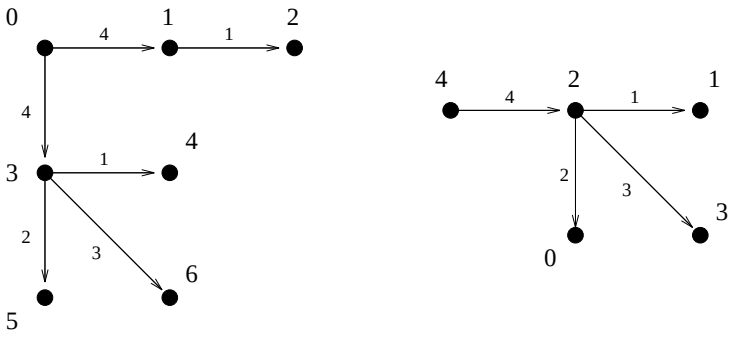
Voorbeeld

Bij de **invoer**

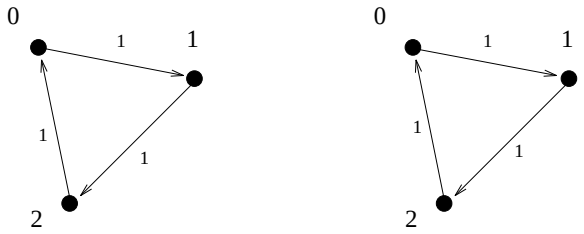
```
2
7 6
0 4 1  1 1 2  0 4 3  3 1 4  3 2 5  3 3 6
5 4
4 4 2  2 1 1  2 2 0  2 3 3
3 3
1 1 2  2 1 0  0 1 1
3 3
1 1 2  2 1 0  0 1 1
```

hoort de **uitvoer** (let wel: de eerste twee regels vormen samen één lange regel, ook al past-ie niet op dit papier!)

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



Figuur 5: De grafen van de eerste run van de invoer



Figuur 6: De grafen van de tweede run van de invoer