UPPGIFT 1 – LYCKOTAL

Lyckotal är en serie heltal, som hittas på följande sätt. Starta med de naturliga talen:

$$1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13 \ldots$$

Sök upp det första talet i serien , som är större än 1 och som inte har använts tidigare. Det måste vara talet 2. Därför stryker vi nu vartannat tal i serien. Kvar är de udda talen:

$$1, 3, 5, 7, 9, 11, 13, 15, 17, 19, 21, 23 \ldots$$

Det första oanvända talet större än 1, är nu 3. Därför stryker vi nu vart tredje tal i serien. Kvar blir:

$$1, 3, 7, 9, 13, 15, 19, 21, 25 \ldots$$

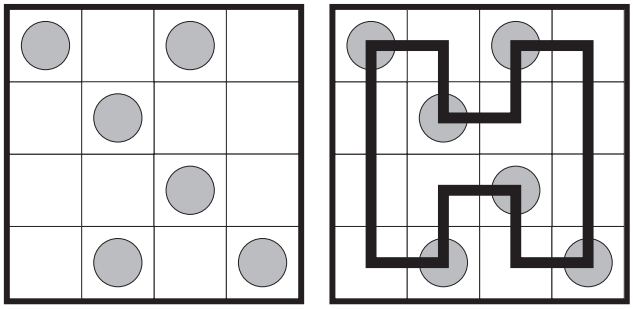
Nästa gång stryker vi vart sjunde tal. Proceduren att söka upp det första oanvända talet n och sedan stryka vart n:te tal i serien kan vi fortsätta i all oändlighet. De tal som överlever kallar vi för *lyckotal*. Serien med lyckotal börjar så här:

$$1, 3, 7, 9, 13, 15, 21, 25, 31, 33, 37, 43, 49, 51 \ldots$$

Skriv ett program som frågar efter ett tal $n, 1 \leq n \leq 1000$ och skriver ut hur många lyckotal det finns som är mindre än eller lika med n. Körningsexempel:

```
N ? 20
Det finns 6 lyckotal.
```

UPPGIFT 2 – KVADRATVANDRING



FIGUR 1.

Målet med den här uppgiften är att ta sig från en godtycklig ruta, runt till samtliga kvadratens rutor och återvända ”hem” (rutan där man startade). Varje ruta ska besökas precis en gång. Ett steg är att förflytta sig till en ruta som har en hel sida gemensam med den tidigare. Dessutom måste man byta riktning, göra en 90°-sväng, i alla rutor som innehåller en grå cirkel.

I filen `corner.dat` finns information om kvadratens storlek och var de grå cirklarna finns. Filen inleds med ett jämnt heltal $n, 2 \leq n \leq 6$ som anger kvadratens storlek i rutor räknat. Därefter följer n rader med n tal på varje. Talet 0 står för en tom ruta och 1 för en ruta med grå cirkel, till exempel:

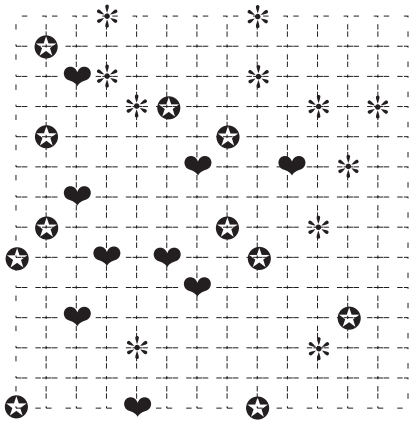
```
4
1 0 1 0
0 1 0 0
0 0 1 0
0 1 0 1
```

Resultatet skrivs som en tabell med $n \times n$ tal. Talet anger det nummer i ordningen steget har. Exemplet kan ge svaret:

```
 1  2  5  6
16  3  4  7
15 12 11  8
14 13 10  9
```

Det finns förstås flera lösningar till varje problem. Dels kan man välja den ruta där vandringen ska starta på flera sätt och dessutom kan vandringen ske i två olika riktningar. Sedan kan det finnas lösningar med helt andra vandringar. Vi vill att ditt program ska skriva ut en lösning vilken som helst.

UPPGIFT 3 – TJUV OCH POLIS



FIGUR 2. *Objektens placering i exemplet. Origo ligger i nedre vänstra hörnet*

I figur 2 ser vi ett antal *tjuvar* (✻), ett antal *poliser* (✪) och ett antal *guldhjärtan* (❤). Tjuvarna försöker förstås sno åt sig guldet, som poliserna försöker vakta. Följande regler gäller:

- Ett guldhjärta är *skyddat* av poliser om det finns *inuti* en rektangel eller kvadrat med sidorna parallella med rutnätet och med en polis *i varje hörn*.
- Ett guldhjärta är *hotat* av tjuvar om det dels finns *inuti* en rektangel eller kvadrat med sidorna parallella med rutnätet och med en tjuv *i varje hörn* och dessutom *inte är skyddat av poliser*.
- Ett guldhjärta, som varken är skyddat av poliser eller hotat av tjuvar, kallar vi för *gömt*.

Skriv ett program som hämtar data från filen `tjuv.dat`. Filen inleds med ett tal som anger hur många guldhjärtan $g, 1 \leq g \leq 100$, det innehåller. Därefter följer g rader med två heltal på varje, x- och y-koordinaterna, $(0 \leq x, y \leq 20000)$, för guldets placering. Därefter följer ett tal som anger hur många poliser $p, 0 \leq p \leq 1000$ det innehåller, på samma sätt följt av p rader med koordinater. Till sist behandlas de $t, 0 \leq t \leq 1000$ tjuvarna. Alla koordinater är unika. Filen till körningsexemplet uppdelad i fem spalter:

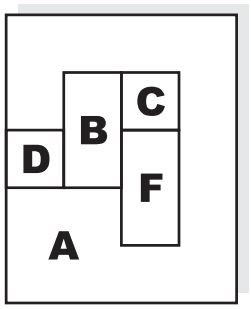
```
9          2 11       0  5        8  5      10  6
4 0        6  8       1  6       11         11  8
2 3        9  8       1  9        4  2      10 10
3 5       11          5 10        3 11      12 10
5 5        0  0       1 12        3 13       8 11
6 4        8  0       7  9        4 10       8 13
2 7       11  3       7  6       10  2
```

Programmet ska ange hur många hjärtan det finns av varje typ. Alltså för exemplet:

```
Skyddade hjärtan : 4
Hotade hjärtan   : 2
Gömda hjärtan    : 3
```

Observera att ett guldhjärta som befinner sig precis på gränsen av en rektangel inte räknas som att det befinner sig inuti.

UPPGIFT 4 – TAPETSERING



FIGUR 3.

I den här uppgiften ska ditt program tapetsera rummen i ett hus, så att två angränsande rum **inte** får samma sorts tapeter. Enligt det berömda *fyrfärgsteoremet* räcker det alltid med *fyra* sorters tapeter, oavsett hur huset ser ut. Tapetsorterna kan dock ha olika pris per meter. Ditt program ska utföra tapetseringen så att totala kostnaden minimeras.

Huset, som programmet får ritningen till i filen `tapet.dat`, är rektangulärt med måtten $l \times b$ meter. För enkelhets skull har rektangeln delats in i $l \times b$ kvadratiska rutor, som är vardera 1×1 meter. Eftersom husets innerväggar råkar sammanfalla med rutornas kanter kan hela ritningen beskrivas genom att ange vilket rum en ruta tillhör.

Varje rum är sammanhängande, det vill säga rutorna som tillhör rummet måste sitta ihop kantvis. Ett rum kan inte heller innesluta andra rum fullständigt. Två rum anses som angränsande om de har minst en meter vägg gemensam. Observera att ytterväggar tapetseras endast på insidan, medan alla innerväggar tapetseras från två håll (med olika tapeter).

Första raden i filen `tapet.dat` innehåller två heltal `l` och `b` ($1 \leq b \leq l \leq 50$), husets längd och bredd i meter. Sedan följer ritningen i form av `l` rader med vardera `b` tecken. Varje tecken är en bokstav i intervallet `A...M` och markerar ”namnet” på det rum som motsvarande ruta tillhör. Antalet rum är alltså högst 13. Slutligen följer fyra heltal som anger priset i kronor per meter vägg för de fyra tapetsorterna. Till exempel:

```
5 4
AAAA
ABCA
DBFA
AAFA
AAAA
23 59 36 45
```

Programmet ska skriva ut den minimala tapetkostnaden i kronor så att villkoren ovan uppfylls.

```
Minimal tapetkostnad: 1510
```

Kommentar: Rum A (28 meter) tapetseras med den första tapeten, C (4 meter) med den andra, D och F (4 respektive 6 meter) med den tredje, medan B (6 meter) tapetseras med den fjärde tapeten.

Observera att alla väggar i ett rum ska ha samma tapet och att alla sorter av tapeter inte nödvändigtvis behöver användas.

UPPGIFT 5 – FEMKORT

Femkort är ett enkelt kortspel för två spelare som går till på följande sätt:

- En vanlig kortlek används
- Båda spelarna har från början fem kort var
- Den *första* spelaren lägger ut ett av sina kort
- Den *andra* spelaren måste nu lägga ett kort i samma färg, om han har något sådant. Annars får han lägga valfritt kort
- De båda korten bildar ett ”stick”, och är inte mer med i spelet. Den som lagt det högsta kortet, om dessa är i samma färg, vinner sticket. Om korten har olika färg, vinns sticket av den som la ut det första.
- Den som vinner sticket lägger ut första kortet till nästa stick
- Spelet fortsätter tills alla kort har spelats
- Till skillnad från många andra stickspel är *antalet* vunna stick betydelselöst i femkort. Det är den spelare som vinner *sista* (femte) sticket som vinner hela spelet

Normalt sett är motståndarens kort okända för dig, men i den här uppgiften antar vi att både du och motståndaren har visat varandra sina kort.

Du ska skriva ett program som, givet dina och motståndarens fem kort, beräknar vem som kommer att vinna om båda spelar optimalt. Eller mer precist uttryckt: om du, oavsett hur motståndaren spelar, kan vinna sista sticket, vinner du spelet. I övriga fall förlorar du, eftersom motståndaren känner till dina kort och därmed kommer att spela på just ett sådant sätt att han vinner sista sticket.

Programmet ska fråga efter dina respektive motståndarens kort. Handen ges som fem strängar åtskilda av blanksteg. Varje sträng beskriver ett kort och består av en bokstav följd av ett tal. Bokstaven kan vara `R, S, H` eller `K`, och anger kortets färg. Talet är i intervallet `1..13` och anger kortets valör. Observera att `13` är högst, därefter följer `12` och så vidare. *Du* börjar alltid lägga ut.

Körningsexempel 1

```
Dina kort: S7 S6 R3 S9 K7
Motståndarens kort: R6 K5 R12 H10 H8
Du vinner!
```

Kommentar: Om du börjar med `R3` måste motståndaren ta sticket med `R6` eller `R12`. Nu kan han ta hem de flesta sticken men förr eller senare måste han lägga `K5` och då tar du sticket med `K7`, som du naturligtvis har sparat. Därefter kan du ta hem resten eftersom motståndaren inte har några `S`. Observera att om du börjar med något annat kort än `R3` så vinner motståndaren eftersom han då kan kasta bort sin `K5`.

Körningsexempel 2

```
Dina kort: S10 R8 S7 H12 R11
Motståndarens kort: K3 S5 S11 R12 R6
Du förlorar!
```

Poängberäkning Detta problem testas med `20` indata. Endast `16 – 20` rätt ger poäng (1-5).

UPPGIFT 6 – LAGERARBETE

På ett lager ska ett antal boxar flyttas. Lagret kan beskrivas som ett rutnät, där varje ruta motsvarar storleken på en box. Betrakta modellen:

```
BBBB....
.###...X
.XX#...X
...#....
........
```

`B` Box som ska flyttas

`.` Tom plats

`X` Position som en box ska inneha efter flytten

`#` Upptaget utrymme

En box kan flyttas till någon av de fyra angränsande rutorna, förutsatt att denna ruta är tom (det vill säga inte ockuperas av en annan box, eller är ett upptaget utrymme). Att flytta en box en ruta tar en tidsenhet, och endast en box kan flyttas per tidsenhet. Skriv ett program som läser in ett lagerutrymme enligt ovan och beräknar det snabbaste sättet att flytta samtliga boxar till de angivna destinationsrutorna (det spelare ingen roll vilken box som hamnar på vilken destinationsruta). Du kan utgå ifrån att arbetet kommer att vara möjligt att genomföra.

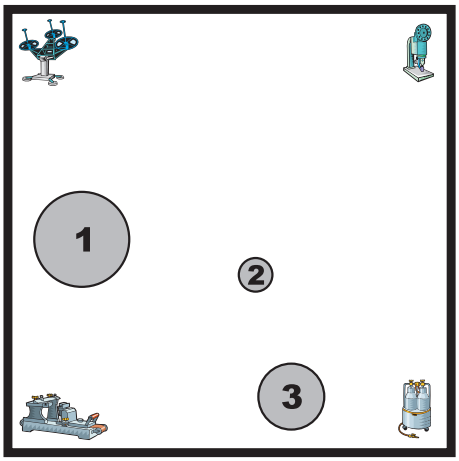
Första raden i filen `lager.dat` innehåller två heltal `l` och `b` ($1 \leq b, l \leq 40$), bredden och längden på lagret. Därefter följer `l` rader med vardera `b` tecken som beskriver lagret med de angivna tecknen ovan. Antalet `B` (mellan 1 och 15) kommer att vara detsamma som antalet `X`. Exempel på `lager.dat`

```
5 8
BBBB....
.###...X
.XX#...X
...#....
........
```

Körningsexemplet ska ge:

```
Minsta flyttid: 20
```

UPPGIFT 7 – FABRIKSROBOT



FIGUR 4.

På ett stort fabriksgolv (storlek 1000×1000 meter) finns ett antal cirkulära pelare med varierande radie. Pelarna tangerar inte varandra eller väggarna. Företaget, som äger fabriken, planerar att köpa in en vaktrobot som ska röra sig i lokalen. I lokalens fyra hörn finns maskiner placerade till vilka roboten måste kunna ta sig genom att sick-sacka fram mellan pelare och väggar. De vaktrobotar som finns på marknaden är alla, liksom pelarna, helt cirkulära. Innan företaget köper in en robot vill de dock veta vad den maximala radien på roboten får vara för att deras krav fortfarande ska kunna uppfyllas.

Ingen del av roboten får sticka ut utanför lokalen. Om roboten har radien r så ska den kunna ta sig till punkterna (r, r), $(1000 - r, r)$, $(r, 1000 - r)$ respektive $(1000 - r, 1000 - r)$ och därifrån betjäna de fyra maskinerna. Maskinerna utgör aldrig något hinder för robotens framfart.

Första raden i filen `robot.dat` kommer att innehålla ett heltal n, $(1 \leq n \leq 50)$, antalet pelare i lokalen. Därefter följer n rader innehållande vardera 3 heltal, x, y och r. Dessa tal beskriver koordinaterna för en pelares centrum samt dess radie. Körningsexempel:

```
3
165 520 110
560 430 30
590 115 75
```

Robotens maximala radie ska skrivas ut med minst 2 decimalers noggrannhet.

```
Maximal radie: 132.56
```

Figur 4 motsvarar körningsexemplet. Det kritiska avståndet är den mellan pelare 1 och 2 i indata. Avståndet mellan deras centrumpunkter är cirka 405.123, och det verkliga avståndet däremellan är $405.12 - 110 - 30 = 265.12$, vilket gör att den maximala radien på roboten är $265.12/2 = 132.56$.