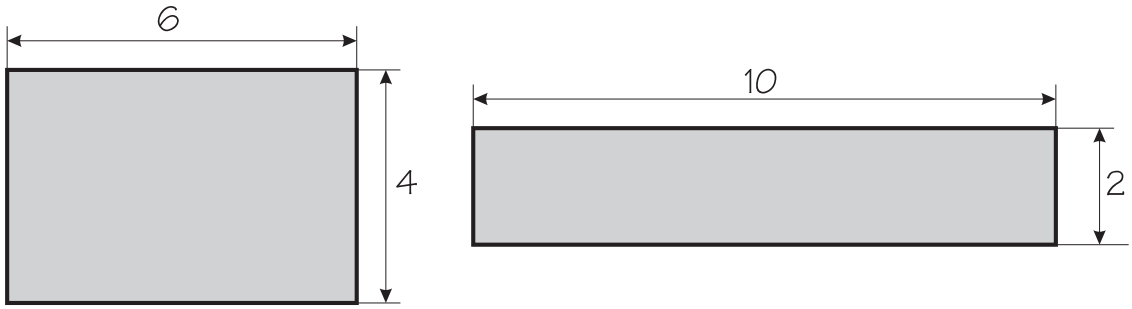
UPPGIFT 1 – VÄNSKAPLIGA REKTANGLAR



FIGUR 1. *Dessa två rektanglar är vänskapliga. Den ena har samma mätetal för arean som den andra har för omkretsen och tvärtom.*

Rektangeln till vänster har *omkretsen* $2 \cdot 4 + 2 \cdot 6 = 20$. Rektangeln till höger har *arean* $2 \cdot 10 = 20$. Rektangeln till höger har *omkretsen* $2 \cdot 2 + 2 \cdot 10 = 24$. Rektangeln till vänster har *arean* $6 \cdot 4 = 24$. Därför kallas detta par av rektanglar för *vänskapliga*.

Skriv ett program som listar *längd* och *bredd* för samtliga par av *vänskapliga rektanglar*. Par av rektanglar där den ena har samma mätetal för arean som den andra har för omkretsen och tvärt om. Observera att en rektangel kan vara vänskaplig med sig själv och att det finns ett ändligt antal par av rektanglar av denna sort, alla med största sidan < 100.

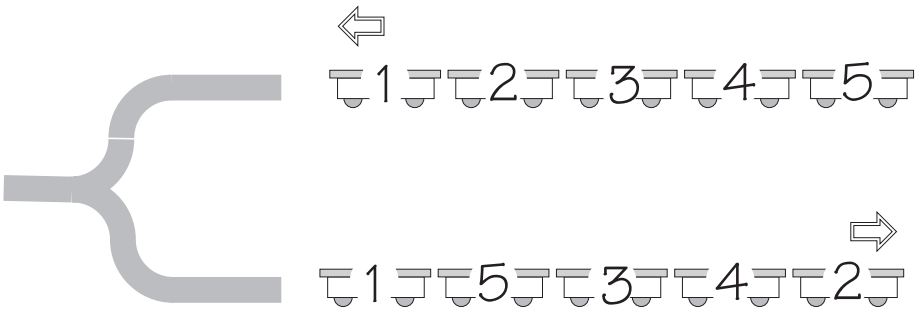
Indata: -

Utdata: En rad för varje funnet par där en av raderna kommer från exemplet:

```
6 x 4     10 x 2
```

Ordningen mellan rader och inom raden är oväsentlig.

UPPGIFT 2 – TÅGVÄXLING



FIGUR 2. *Det är möjligt att kasta om ordningen mellan vagnarna, från den övre ordningen till den undre, med hjälp av stickspåret.*

På översta spåret (kallat *inspåret*) står ett tåg med $n, n \leq 100$ vagnar, numrerade från 1 till n med början från den främsta vagnen. På det undre spåret (kallat *utspåret*), ser du samma vagnar i önskad ordning då tåget lämnar växlingsområdet.

Ordningen mellan vagnarna kan förändras genom att köra upp dem på stickspåret (som är tillräckligt långt för att rymma alla vagnarna på en gång). Endast två förflyttningar är tillåtna.

- Första vagnen på *inspåret* till sista på *stickspåret*
- Sista vagnen på *stickspåret* till sista på *utspåret*

Skriv ett program som tar emot önskad ordning på *utspåret* och som avgör om denna ordning *är möjlig* att erhålla. Ordningen på de n vagnarna på *inspåret* är alltid i stigande nummerordning från första till sista vagnen.

Indata: Indata läses från en fil med namnet UPPG2.DAT, som kan innehålla **flera** tester. Första raden innehåller antalet, n, vagnar för första testen och följande rad innehåller en permutation av heltalen $1 \ldots n$. Nästa rad åter innehåller ett nytt n följt av en rad med n heltal. Detta kan sedan upprepas ett obestämt antal gånger. Filen avslutas till sist med talet 0 på en egen rad.

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

Observera att vagnarna på utspåret presenteras i ordning från första till sista i färdriktningen räknat (se pilen i figuren).

Utdata: Endast ett **ja** eller ett **nej** för varje test. För vårt exempel

```
JA
NEJ
```

UPPGIFT 3 – KIMBERLINGS BLANDNING

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
2	**3**	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
4	2	**5**	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
6	2	7	**4**	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22
8	7	9	2	**10**	6	11	12	13	14	15	16	17	18	19	20	21	22	23
6	2	11	9	12	**7**	13	8	14	15	16	17	18	19	20	21	22	23	24
13	12	8	9	14	11	**15**	2	16	6	17	18	19	20	21	22	23	24	25
2	11	16	14	6	9	17	**8**	18	12	19	13	20	21	22	23	24	25	26
18	17	12	9	19	6	13	14	**20**	16	21	11	22	2	23	24	25	26	27

FIGUR 3. *De nitton första talen i de nio första raderna i Kimperlings blandning*

En man vid namn *Kimberling* har uppfunnit följande sätt att blanda de positiva heltalen.

Varje rad $i, i > 1$, i tabellen ovan bildas utifrån föregående rad, $i - 1$, genom att utgå från elementet j (tal med fet stil) i huvuddiagonalen i rad $i - 1$.

Första elementet i den nya raden i, blir första talet *efter* j i rad $i - 1$. *Andra* talet blir talet *före* j. Därefter kopieras i tur och ordning andra talet *efter* j, andra talet *före* j och så vidare tills första talet i rad $i - 1$ har kopierats. Därefter kopieras återstående tal, som fortfarande är i numerisk ordning till samma plats som de har i rad $i - 1$. Talen i huvuddiagonalen försvinner alltså successivt och bildar en talföljd

$$1, 3, 5, 4, 10, 7, 15, 8, 20 \ldots$$

Studera tabellen tills du är förvissad om hur en ny rad skapas och skriv sedan ett program som tar emot ett tal $n, 1 \leq n \leq 200$ och som bestämmer vilken plats n har i talföljden.

Indata: Programmet frågar efter ett tal n

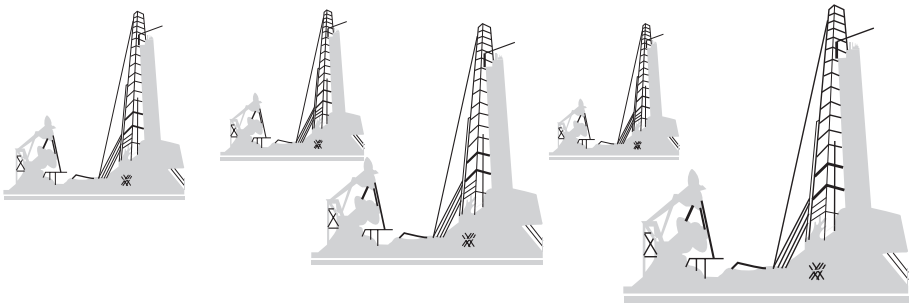
```
Vilket tal: 20
```

Observera att i testerna kommer inget tal n att ges, som har en plats som är > 5000. Här ytterligare några testdata: talet 6 har platsen 22, talet 12 har platsen 83 och talet 17 har platsen 169. Undvik att testa med talet 19 som har platsen 49595.

Utdata: Programmet skriver ut platsen på talet

```
Talet finns på plats 9
```

UPPGIFT 4 – OLJEKÄLLORNA



FIGUR 4.

I den här uppgiften gäller det att släcka bränder i ett antal oljekällor.

Nr	**1**	**2**	**3**	**4**	**5**
T_k	3	5	8	10	12
O_k	4	5	12	8	10

En grupp brandmän ska ensamt släcka $N, 1 \leq N \leq 10$ brinnande oljekällor, en i taget. Det tar T_k dygn att släcka källa k (tiden är känd från början för samtliga källor). För källa k går O_k liter olja per dygn upp i rök så länge den brinner. (givet i 10 000-tal liter)

Skriv ett program som bestämmer i vilken ordning källorna ska släckas för att mängden förlorad olja ska bli så liten som möjligt.

Indata: Programmet ska inleda med att fråga efter namnet på indatafilen.

```
Filnamn: uppg4a.dat
```

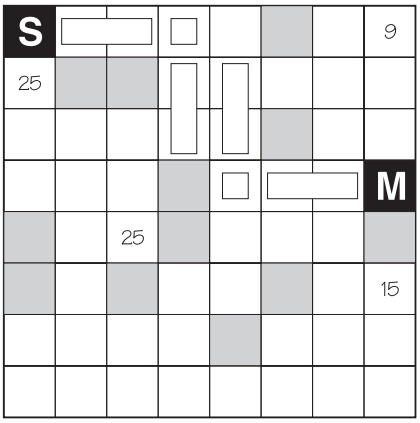
Filen inleds med ett heltal n, ensamt på första raden, som anger antalet brinnande oljekällor. Därefter kommer n rader var och en bestående av två heltal T_k och O_k för källa k.

Utdata: En rad bestående av talen $1 \ldots n$ i den ordning de ska släckas för att minimera mängden förlorad olja. Exemplet ger:

```
3 1 2 5 4
```

Om det finns flera ordningar som leder till samma lösning räcker det att ange en av dem.

UPPGIFT 5 – RULLANDE BLOCKET



FIGUR 5. *Fyra rutor anger hur många drag som behövs om dessa är målrutor.*

Spelplanen hos detta pussel består alltid av ett bräde med 8×8 rutor. I rutan högst upp i vänstra hörnet (startrutan, märkt S i figuren) är från början en klots med måtten $1 \times 1 \times 2$ placerad. Någon av de andra 63 rutorna är målruta (märkt M i figuren) och ett antal av de återstående innehåller ett *hinder* (grå rutor i figuren). Målet är att flytta den stående klotsen från *startrutan* högst upp i vänstra hörnet, stående till *målrutan*, i så få drag som möjligt.

Ett *drag* är att 'välta klotsen över kanten', i någon av de fyra riktningarna *uppåt, nedåt, åt höger* eller *nedåt*, utan att den kommer att hamna över en ruta med hinder. I figuren ser vi hur man kan lösa problemet i sju drag.

Skriv ett program som tar emot en utgångsställning och som bestämmer det minsta antal drag som behövs för att nå målet.

Indata: Programmet startar med att fråga efter indatafilens namn:

```
Filnamn: uppg5.dat
```

Filen inleds med en rad som innehåller två tal, *rad* och *kolumn* för målets placering. (Startrutan är alltid $(1,1)$) Därefter följer 8 rader med 8 tal, 0 eller 1 i varje rad. Där 1 står för *ruta med hinder* och 0 för *ruta utan hinder*. De tre första raderna i filen från exemplet:

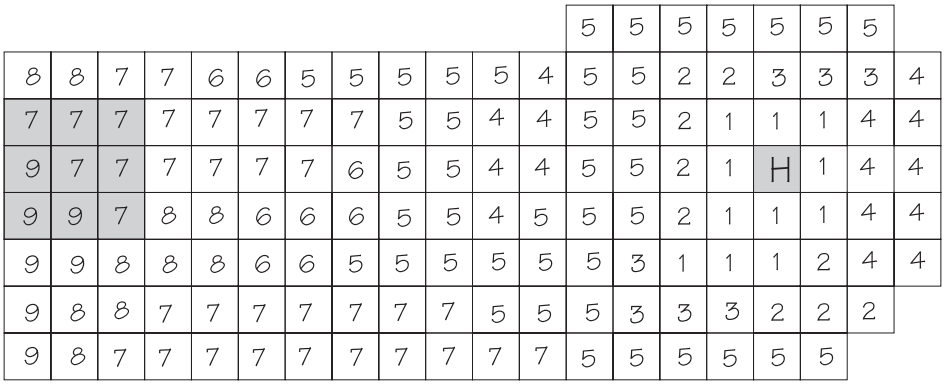
```
4 8
0 0 0 0 0 1 0 0
0 1 1 0 0 0 0 0
...
```

Utdata: Programmet ska beräkna och skriva det minsta antal drag som behövs för att nå målet. Från exemplet:

```
Det behövs minst 7 drag för att nå målet
```

Alla testexempel kommer att ha lösningar med ≤ 40 drag

UPPGIFT 6 – GOLFHÅLET



FIGUR 6.

Figuren visar en del av en golfbana! Utslaget sker från någon av de nio rutorna i den grå kvadraten till vänster (tee). Från den ruta man väljer kan bollen sedan slås i åtta olika riktningar (N, NÖ, Ö, SÖ, S, SV, V, NV), så många rutor som talet i utgångsrutan anger. Detta under förutsättning att bollen hamnar inom banans geometri.

Från den ruta i vilken bollen stannar går spelet sedan vidare i en av de åtta riktningarna med en slaglängd som motsvarar talet i den rutan. Målet är att med så få slag som möjligt nå *hålet* märkt H i figuren.

Skriv ett program som tar emot data om banans geometri och som enligt reglerna ovan bestämmer det minsta antal slag som krävs för att komma från *tee* till *hål*

Indata: Programmet startar med att fråga efter indatafilens namn:

```
Filnamn: uppg6.dat
```

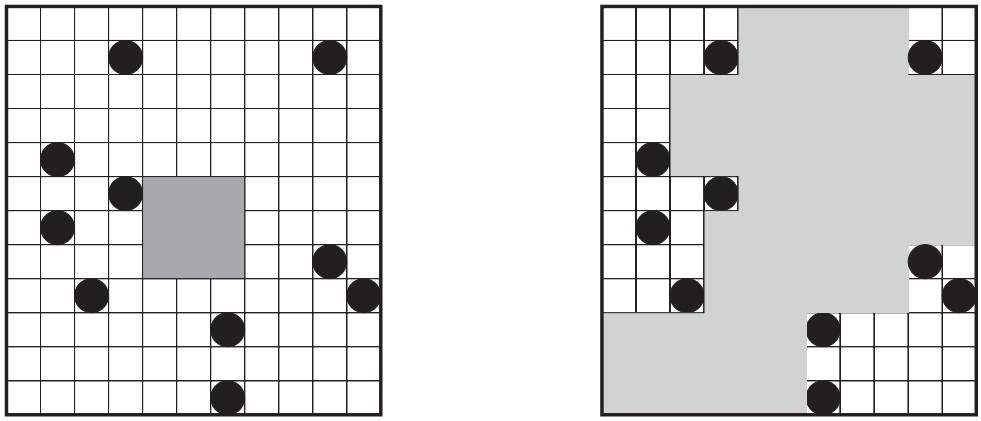
Filen inleds med en rad innehållande 6 tal: Banans *bredd* (antal rader $b, 3 \leq b \leq 40$), banans *längd* (antal kolumner $l, 4 \leq l \leq 60$), *rad* och *kolumn* övre vänstra hörnet hos tee (alltid 3×3) och *rad* och *kolumn* för hålets placering. Följande b rader innehåller l siffror $0 \ldots 9$ som anger slaglängden från denna ruta. För vårt exempel:

```
8 20 3 1 4 17
00000000000055555550
88776655555455223334
77777777554455211144
97777776554455210144
99788666554555211144
99888665555553111244
98877777775553332220
98777777777755555500
```

Utdata: Programmet skriver ut det minsta antalet slag som behövs för att nå hålet

```
Hålet kan klaras av på 3 slag
```

UPPGIFT 7 – GRÄSKLIPPARE



FIGUR 7.

I den vänstra delen av figuren ser vi en gräsmatta på vilken det finns 10 träd och en gräsklippare! Sidlängden hos en delkvadrat kan vi säga är 20 cm (spelar egentligen ingen roll). Gräsklipparen som är 60×60 cm tar följaktligen upp 9 kvadrater.

Du ska nu skriva ett program som bestämmer hur stor del (hur många rutor) av gräsmattan som kan klippas. Att klipparen inte kommer åt överallt beror på att det finns träd i vägen och att klipparen bara kan förflyttas i någon av fyra riktningar (uppåt, nedåt, åt höger eller åt vänster) och aldrig utanför gräsmattan. Till höger i figuren ser du den del av gräsmattan som kan klippas, markerad med grått, bestående av 82 rutor.

Indata: Programmet ska fråga efter namnet på indatafilen.

```
Filnamn: uppg7a.dat
```

Denna fil inleds med en rad som anger gräsmattans *bredd* $b, 3 \leq l \leq 40$. (antal rader i figuren). Nästa rad innehåller ett tal som anger mattans *längd*, $l, 3 \leq b \leq 40$. Filen fortsätter sedan med b rader var och en innehållande l tecken där:

- 0 står för gräs
- 1 står för gräsklippare
- 2 står för träd

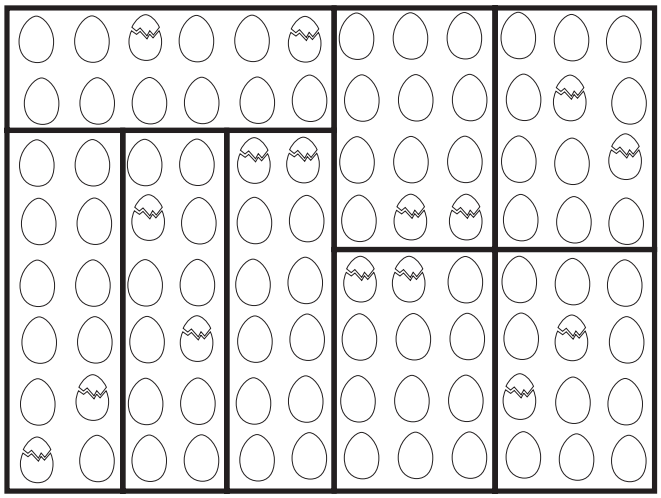
Indatafilen som hör till exemplet inleds:

```
12 11
00000000000
00020000020
00000000000
...
```

Utdata: Utdata består av en rad:

```
Det går att klippa 82 rutor.
```

UPPGIFT 8 – PÅSKÄGG



FIGUR 8. *Ett exempel på placering av de* 16 *trasiga äggen och med lösningen inritad.*

Snart är det påsk med ägg och annat. Här gäller det att skriva ett program som förpackar påskägg på ett rättvist sätt. Vi utgår alltid från en äggkartong med måtten $8 \times 12 = 96$ ägg som i figuren. Av dessa råkar alltid 16 vara trasiga. Kartongen ska nu delas in i 8 rektangulära delar med ett dussin ägg (12 stycken) i varje. Dessutom ska de trasiga äggen fördelas rättvist så att exakt två av dem hamnar i varje del.

Det finns alltså fem olika rektanglar som kan komma i fråga: $1 \times 12, 2 \times 6, 3 \times 4, 4 \times 3$ och 6×2.

Indata: Programmet ska fråga efter namnet på indatafilen.

```
Filnamn: uppg8a.dat
```

Denna fil innehåller 8 rader med 12 tecken i varje rad. Siffran 0 står för *ett helt ägg* och siffran 1 står för *ett trasigt ägg*.

Utdata: Programmet ska skriva ut 8 rader med 12 tecken i varje rad. Utskriften från exemplet förklarar:

```
111111222333
111111222333
445566222333
445566222333
445566777888
445566777888
445566777888
445566777888
```

Varje del beskrivs alltså med en av siffrorna $1 \ldots 8$. I vilken ordning du väljer att numrera bitarna spelar ingen roll. Om det finns fler än en lösning räcker det att presentera en av dem.