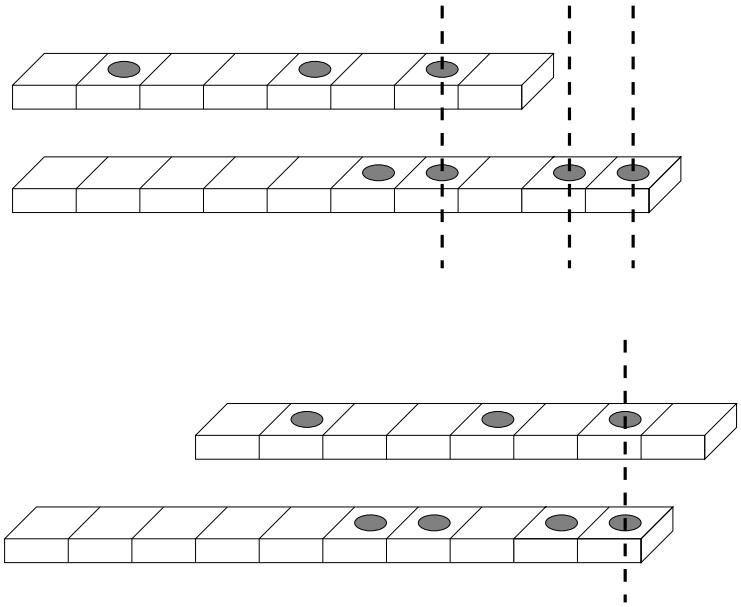
UPPGIFT 1 – LAMELLER



FIGUR 1. *Överst de två lamellerna som de ges till programmet. Underst samma två lameller när den ena förskjutits så att "bästa läge" uppkommit.*

I figur 1, övre delen, ser du två *lameller*, stänger med hål i. Varje lamell är indelad i kvadratiska områden (10 respektive 8 i exemplet) som vart och ett kan ha ett hål. Målet är nu att förskjuta lamellerna relativt varandra (men inte vända någon av dem) så att man får så litet antal genomgående hål som möjligt när man lägger dem intill varandra. I den nedre delen av figur 1 har en sådan förskjutning gjorts, vilket har minskat antalet genomgående hål från tre till ett. Observera att hål i en del av en lamell som skjuter ut utanför den andra lamellen också räknas som genomgående hål.

Skriv ett program som frågar efter lamellernas utseende och bestämmer det minsta antalet genomgående hål som kan erhållas genom en förskjutning av lamellerna. Varje lamell beskrivs med en sträng där varje tecken kan vara 0 för ett hål eller 1 för ett solitt område. Antalet tecken anger alltså lamellens längd, som är högst 10.

Två körningsexempel:

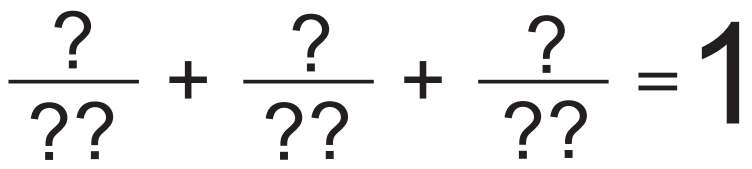
```
Första lamellen ? 10110101
Andra lamellen  ? 1111100100

Minsta antalet hål: 1


Första lamellen ? 100000
Andra lamellen  ? 10001

Minsta antalet hål: 3
```

UPPGIFT 2 – SUMMAN BLIR 1



FIGUR 2.

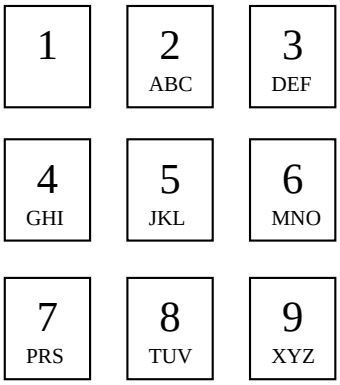
I den här uppgiften får du 9 siffror, som ska kombineras så att uttrycket i figur 2 blir sant. Talen som ska stå i nämnarna består av två siffror och i täljarna av en siffra. Siffrorna i indata kan ges i vilken ordning som helst. Samma siffra kan förekomma flera gånger, men siffran 0 förekommer inte. Om det finns flera lösningar, räcker det att ditt program skriver ut en av dessa.

Två körningsexempel:

```
Vilka siffror ? 123456789
9/12 + 5/34 + 7/68 = 1

Vilka siffror ? 274588812
8/14 + 5/28 + 7/28 = 1
```

UPPGIFT 3 – FÖRVIRRANDE SMS



FIGUR 3.

Du har fått ett förvirrande SMS. Tydligen har avsändaren haft på siffermod istället för bokstäver och skickat iväg det utan att titta på displayen. Lyckligtvis vet du hur inskrivningen fungerar. Varje bokstav (utom q, w, å, ä och ö som vi bortser från helt i denna uppgift) motsvaras av en siffertangent mellan 2 och 9 enligt figuren ovan. Samma tangent används alltså för att skriva tre olika bokstäver: trycker man en gång blir det den första, trycker man två gånger i snabb följd blir det den andra och trycker man tre gånger i snabb följd blir det den tredje. Vi förutsätter att man aldrig trycker fler än tre gånger.

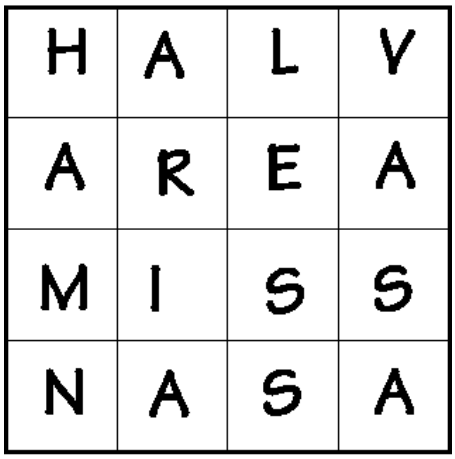
Det finns bara ett problem. Att trycka två gånger på tangent 2 kan betyda antingen ett B eller två A, beroende på hur lång paus man gör mellan tryckningarna och det finns inget sätt att avgöra var i siffersekvensen det har varit paus. Det enda sättet att få reda på vad meddelandet betyder är att titta på varje ord och avgöra vilken tolkning som verkar mest logisk. Du ska skriva ett program som, givet en siffersekvens (högst 10 siffror) som motsvarar ett ord, skriver ut alla tänkbara bokstavskombinationer som avsändaren kan ha avsett att skriva. Vi förutsätter naturligtvis att avsändaren inte har gjort något misstag. Ordningen på de tänkbara orden som programmet skriver ut spelar ingen roll.

Två körningsexempel:

```
Siffersekvens ? 552662
KAMMA
KANA
JJAMMA
JJANA


Siffersekvens ? 8888
TTTT
TTU
TUT
TV
UTT
UU
VT
```

UPPGIFT 4 – ORDFLÄTAN



FIGUR 4.

I figur 4 ser du en ordfläta med fyra vågräta och fyra lodräta ord. Du ska skriva ett program som, givet en lista med fyrbokstaviga ord, hittar alla möjliga sådana ordflätor. Ett ord får endast användas en gång i varje ordfläta.

Programmet ska läsa indata från filen `ord.dat`. Första raden innehåller ett tal n, där $8 \leq n \leq 800$, som anger hur många ord det finns i ordlistan. Sedan följer n rader med ett fyrbokstavigt ord på varje rad. Endast bokstäverna A-Z (versaler) förekommer. Alla ord i filen är olika och de är sorterade i alfabetisk ordning. När du designar din algoritm kan du räkna med att bokstäverna i orden är ungefär jämnt fördelade, så att inte exempelvis nästan alla orden börjar på samma bokstav.

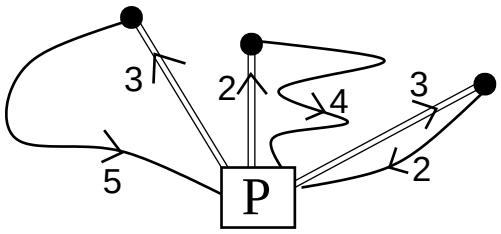
Programmet ska skriva ut alla möjliga ordflätor (i godtycklig ordning), separerade med en blank rad. Till givna indata finns alltid minst en lösning. Observera att det till varje ordfläta finns en spegellösning där vågrätt blivit lodrätt och tvärtom. Du avgör själv om du vill ta med båda dessa lösningar eller bara den ena.

Körningsexempel:

För testfilen blir svaret

```
HALV
AREA
MISS
NASA
```

UPPGIFT 5 – SKIDÅKNING



FIGUR 5.

När Olle har en dag i slalombacken vill han naturligtvis spendera så mycket tid som möjligt på skidåkningen och så lite tid som möjligt i liftarna och väntande. Hans optimeringsvilja har dock gått till viss överdrift; han får ägna så mycket tid åt att planera sin åkning att det inte blir särskilt mycket åkt överhuvudtaget. Därför ska du hjälpa honom genom att skriva ett program som beräknar den maximala åktiden.

Skidbacken består av ett antal (`n`) nedfarter och en lift till varje nedfart. Ett exempel visas i figuren ovan. Alla nedfarterna slutar på samma plats, markerad med ett P i figuren. Det är även här alla liftarna startar och det är samtidigt platsen där Olle anländer och avreser med buss. Varje nedfart har en viss *skidåkningstid*, angiven i hela minuter, samt en *lifttid*, också i hela minuter, vilken anger hur lång tid det tar att åka upp i den tillhörande liften. Vi förutsätter att inga köer bildas så summan av dessa två tider anger direkt hur lång tid det tar innan man är tillbaka vid P om man väljer denna nedfart. Olle har också en maximal totaltid `t`, angiven i minuter. Efter senast `t` minuter måste han vara tillbaka vid P för att inte missa bussen hem.

Vilken backe som är roligast att åka i är helt oviktigt för Olle och han har heller inget emot att åka samma nedfart gång på gång. Det enda som spelar roll är att den sammanlagda tiden han ägnar åt skidåkningen blir så stor som möjligt.

Programmet ska läsa indata från filen `skidor.dat`. På första raden står heltalen `n` (antal nedfarter) och `t` (maximal totaltid), där $1 \leq n \leq 100$ och $2 \leq t \leq 1000$. Därefter följer `n` rader med två heltal på varje rad: skidåkningstiden respektive lifttiden för varje nedfart. Dessa tal ligger alla i intervallet `1..t`. Totaltiden räcker alltid till att åka minst en nedfart. Programmet ska skriva ut ett enda heltal, den maximala sammanlagda skidåkningstiden.

Körningsexempel: För filen *skidor.dat* med innehållet

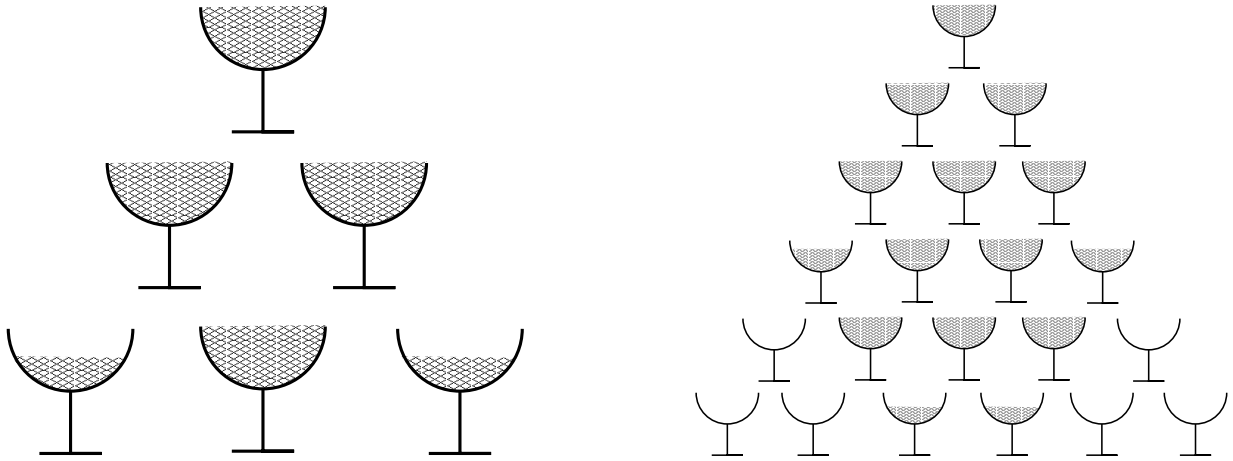
```
3 13
5 3
4 2
2 3
```

ska programmet ge svaret

```
8
```

Kommentar: I det här fallet är det faktiskt bättre att ta den andra nedfarten två gånger och vara tillbaka efter 12 minuter än att ta den första och tredje varsin gång och vara tillbaka efter 13 minuter, eftersom det senare alternativet endast ger 7 minuter skidåkning.

UPPGIFT 6 – RINNANDE VATTEN



FIGUR 6. *Till vänster: Situationen efter* 50 *sekunder då glas* 2 *på tredje raden börjar rinna över. Till höger: Motsvarande situation efter* 136.66666... *sekunder då glas* 2 *på femte raden börjar rinna över. Kantglasen på fjärde raden är fyllda till exakt* 5/6 *och mittglasen på sjätte raden är exakt halvfyllda.*

En konstutställning har uppställt en pyramid av identiska glas, där den översta raden har ett glas, raden under har två glas, raden därunder tre glas osv. Vatten rinner ner i det översta glaset i jämn hastighet. Det tar exakt 10 sekunder att fylla det glaset. När glaset är fullt och svämmar över kommer det tillrinnande vattnet istället att rinna ner i de två glasen på rad två, lika snabbt i varje. Efter ytterligare 20 sekunder kommer således de två glasen på rad två att vara fyllda. Glasen på rad tre kommer dock inte fyllas lika snabbt, då vatten rinner ner i det mittersta glaset dubbelt så snabbt som i de andra två glasen på samma rad.

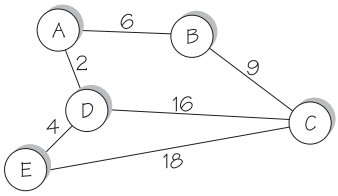
Skriv ett program som bestämmer efter hur många sekunder ett givet glas i pyramiden börjar rinna över. Programmet ska fråga efter en rad `r` ($2 \leq r \leq 50$) och vilket glas `g` på raden räknat från vänster ($1 \leq g \leq r$). Svaret ska vara korrekt med minst 3 decimalers noggrannhet, utom för de allra högsta tiderna där hänsyn tas till begränsad numerisk precision (men se till att använda double).

Körningsexempel:

```
Rad ? 5
Glas ? 2

Det tar 136.667 sekunder.
```

UPPGIFT 7 – AVSTÅNDSTABELL



	A	B	C	D	E
A	-	6	15	2	6
B		-	9	8	12
C			-	16	18
D				-	4
E					-

FIGUR 7. *Avståndstabellen i exemplet och det kortaste vägnätet som uppfyller den*

Som supplement till vägkartor finns ofta en avståndstabell. Den innehåller samtliga kortaste avstånd mellan ett antal (n) utvalda städer och har formen av en $n \times n$ tabell. Ofta är endast halva tabellen ifylld eftersom avståndet mellan A och B är samma som mellan B och A.

Vi antar nu att vägar inte korsar varandra utanför städerna. Det betyder att om kortaste avståndet mellan A och B är x km finns det två möjligheter. Antingen går det en väg direkt mellan A och B, som har längden x km, eller också måste man passera ett antal andra städer på vägen och om man gör det på det sätt som ger kortast totala vägsträcka är denna x km. Du ska skriva ett program som, givet en avståndstabell, räknar ut den kortast möjliga totala längden på hela vägnätet.

Programmet ska läsa indata från filen `tabell.dat`. På första raden står ett heltal $n, (3 \leq n \leq 100)$, antalet städer. Sedan följer avståndstabellen given som $n(n-1)/2$ rader med vardera ett heltal i intervallet $1 \ldots 10000$, det kortaste avståndet (i kilometer) mellan varje par av städer, ordnat på följande sätt (se även figuren ovan). De första $n-1$ talen är avstånden mellan stad 1 och, i tur och ordning, städerna $2, 3, 4 \ldots n$. De följande $n-2$ talen är avstånden mellan stad 2 och, i tur och ordning, städerna $3, 4, 5 \ldots n$ och så vidare. Det sista talet är sålunda avståndet mellan stad $n-1$ och stad n.

Programmet ska skriva ut den minsta totala längden på vägnätet för att avståndstabellen ska vara uppfylld. I samtliga testfall finns det alltid minst ett möjligt vägnät. Exempelvis, om $n = 3$, så kan inte avstånden i indata vara $(1, 2, 4)$ eftersom det kortaste avståndet mellan stad 2 och stad 3 inte kan vara större än $1+2$. Däremot kan man tänka sig indata $(1, 2, 3)$, som ger svaret 3 eftersom det kortast möjliga vägnätet inte har någon väg mellan stad 2 och stad 3. Om indata däremot hade varit $(1, 2, 2)$ hade det behövts vägar mellan alla städer och svaret hade blivit 5.

Körningsexempel: För filen `tabell.dat` med innehållet

```
5
6
15
2
6
9
8
12
16
18
4
```

ska programmet ge svaret `55`