"First, solve the problem. Then, write the code" John Johnson.

Programa-Me 2015
Final Nacional
Problemas

Patrocinado por

Universidad Complutense Madrid

Ejercicios realizados por

Universidad Complutense de Madrid

I.E.S. Dolores Ibárruri (Fuenlabrada)

Realizado en la **Facultad de Informática (U.C.M.)**
5 de junio de 2015

5 de junio de 2015
http://www.programa-me.com

Listado de problemas

Autores de los problemas:

- Marco Antonio Gómez Martín (Universidad Complutense de Madrid)
- Pedro Pablo Gómez Martín (Universidad Complutense de Madrid)
- Patricia Díaz García (I.E.S. Dolores Ibárruri - Fuenlabrada)

Revisores:

- Alberto Verdejo (Universidad Complutense de Madrid)

A
¿Cuántos números capicúa?

Los *números capicúa* son aquellos que, al ser escritos, resultan ser "simétricos", pues se leen igual de izquierda a derecha que de derecha a izquierda. Por ejemplo, los números 33, 121, 5.665 o incluso el 0 son capicúa. Por el contrario, no lo son el 47, el 251 o el 9.815.

Debido a su relativa poca frecuencia, los números capicúa son buscados por muchos coleccionistas, entre ellos los de boletos de lotería. Pero, ¿cuántos números capicúa hay? Todos los números de un sólo dígito son capicúa (0, 1, ..., 9), por lo que hay 10. De dos dígitos, los números capicúa son el 11, 22, 33, ..., 99, por lo que hay 9. ¿Y con más dígitos?

Entrada

El programa deberá leer, de la entrada estándar, múltiples casos de prueba, cada uno en una línea.

Un caso de prueba estará compuesto por un único número positivo entre 1 y 100.000. La entrada termina con el número 0, que no debe procesarse.

Salida

Para cada caso de prueba se escribirá en una línea la cantidad de números capicúa distintos que tienen el número de dígitos indicado en el caso de prueba.

Entrada de ejemplo

```
1
2
0
```

Salida de ejemplo

```
10
9
```

B
Alan Smithee

Muchos directores de cine que han tenido *diferencias artísticas* con los productores o distribuidores de sus películas terminan firmándolas con el seudónimo *Alan Smithee* para manifestar su insatisfacción con el resultado.

El nombre no es casualidad; fue acuñado por el sindicato de directores de Estados Unidos en 1968 precisamente para ese uso, y es un *anagrama* de *The Alias Men* ("los hombres con un alias").

Un anagrama es una palabra o frase que se construye con las mismas letras que otra. Algunos ejemplos son *esponja* y *japonés*, *letras* y *lastre* o *frase* y *fresa*.

En los libros también hay anagramas famosos. Por ejemplo el antagonista de *Harry Potter* se llama *Tom Marvolo Riddle* del que surge el anagrama *I am Lord Voldemort*. Por su parte, en la novela *El código Da Vinci* aparecen varios anagramas que resultan ser pistas ocultas para el protagonista.

Entrada

El programa recibirá, por la entrada estándar, un número indicando cuántos casos de prueba vendrán a continuación.

Cada caso de prueba estará compuesto de dos líneas, con dos palabras o frases a analizar. Ninguna de ellas tendrá más de 1.000 letras, y todas tendrán al menos una letra diferente de espacio.

Salida

Para cada caso de prueba el programa escribirá "`SI`" cuando la primera línea sea un anagrama de la segunda y "`NO`" en caso contrario. Para dar la respuesta, no se deberá tener en cuenta el uso de mayúsculas ni los espacios. Además, por simplicidad, sólo se utilizarán letras del alfabeto inglés y cualquier palabra se considerará anagrama de sí misma.

Entrada de ejemplo

```
5
Alan Smithee
The Alias Men
frase
fresas
esponja
japones
sol
luna
Tom Marvolo Riddle
I am Lord Voldemort
```

Salida de ejemplo

```
SI
NO
SI
NO
SI
```

C
Voltea el dado

Los dados son objetos habituales en infinidad de juegos. Los clásicos son cubos con sus caras numeradas del 1 al 6, pero la cantidad de variaciones es enorme, haciendo uso de muchos otros poliedros con números de caras diferentes.

La utilidad principal de los dados es la obtención de números aleatorios, pero hay usos más imaginativos, como el del juego para dos personas que nos ocupa, llamado *Voltea el dado*. En él, el primer jugador coloca el dado (de 6 caras) encima de la mesa, mostrando el número que desee. En cada turno, de manera alterna, los jugadores van *haciendo rotar el dado* con una ligera presión sobre alguna de las aristas, de modo que la nueva cara superior pase a ser una de las que antes eran caras laterales. El valor de la nueva cara superior se suma al acumulado a lo largo de la partida, empezando con el inicial elegido por el jugador que empieza. El jugador que hace que la suma llegue (o supere) al número 35 *pierde*.

Entrada

La entrada estará compuesta por un primer número indicando la cantidad de casos de prueba que vendrán a continuación.

Cada caso de prueba, en una línea, contendrá dos números. El primero $1 \leq n \leq 35$ especifica la suma acumulada hasta este momento, y el segundo $1 \leq v \leq 6$ el número que es visible en la cara superior del dado, que ya habrá sido sumado.

Salida

Para cada caso de prueba, se deberá analizar si el jugador que tiene el turno en la situación descrita ganará la partida si juega bien. Se escribirá "`GANA`" si ganará, y "`PIERDE`" en otro caso.

Recuerda que en los dados clásicos, la suma del valor de dos caras opuestas suma siempre 7. Por ejemplo, si la cara superior visible de un dado contiene el 6, la cara oculta, apoyada en la mesa, contendrá un 1. Eso significa que el jugador que tiene el turno únicamente podrá conseguir hacer visibles los valores 2, 3, 4 o 5.

Entrada de ejemplo

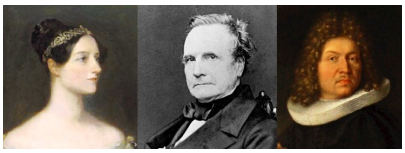
```
3
34 1
33 3
6 1
```

Salida de ejemplo

```
PIERDE
GANA
GANA
```

D

Ada, Babbage y Bernouilli



En 1815 nacía Ada Byron, conocida después como Ada Lovelace, quién estaba destinada a convertirse en la primera programadora de la historia. Amiga de Charles Babbage, siguió con interés los trabajos de éste relacionados con su *máquina analítica*, que hoy se considera un hito en la historia de la computación.

Con la ayuda de Babbage, estudió y mejoró algunas de sus ideas, centrándose en lo que hoy llamaríamos *software*, mientras que Babbage se preocupaba principalmente por el *hardware*. Fue Ada quien describió el primer *algoritmo* pensado específicamente para ser ejecutado por un "ordenador", por lo que se la reconoce como la primera programadora.

El algoritmo estaba dedicado a calcular los números de Bernouilli y la propia Ada fue consciente de la necesidad de *bifurcación* de la máquina analítica. Expresado en términos modernos, el primer algoritmo necesitaba condiciones y bucles.

Los números de Bernouilli son una secuencia de números racionales que tienen conexiones muy interesantes con teoría de números. Su cálculo es complejo, por lo que a pesar de que han pasado muchos años desde que Ada "programó" cómo obtenerlos, nos conformaremos con uno de sus usos, el cálculo de la fórmula de Faulhaber, o suma de los n primeros números elevados a un valor constante p:

$$1^p + 2^p + 3^p + \ldots + n^p$$

Entrada

El programa leerá de la entrada estándar múltiples casos de prueba, cada uno en una línea. Un caso de prueba se compondrá de dos números, n y p, entre 1 y 100 ambos incluídos.

La entrada terminará cuando $n = p = 0$.

Salida

Por cada caso de prueba, el programa escribirá el resultado de la expresión anterior, es decir la suma de los n primeros números naturales elevados a p. Dado que el resultado puede ser muy alto, se dará módulo 46.337[1].

Entrada de ejemplo

```
1 1
2 2
4 3
100 100
0 0
```

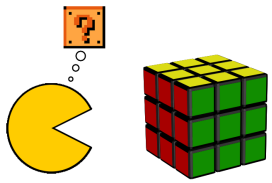
Salida de ejemplo

```
1
5
100
17171
```

[1] Ten cuidado con los desbordamientos en los cálculos intermedios; si a o b son muy grandes, la suma o la multiplicación podría ocasionar desbordamiento, y es preferible calcular el módulo primero para reducirlos. Recuerda que el módulo de la suma es igual al módulo de la suma de los módulos, y lo mismo ocurre con el producto:

$$(a + b) \,\%\, k = ((a \,\%\, k) + (b \,\%\, k)) \,\%\, k$$
$$(a \times b) \,\%\, k = ((a \,\%\, k) \times (b \,\%\, k)) \,\%\, k$$

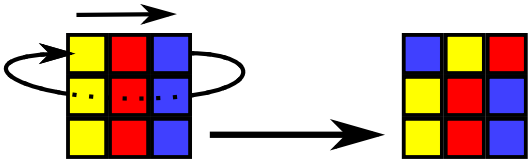
E
El cuadrado de Rubik



El *cubo de Rubik* es el juego de puzzle más vendido de la historia a nivel mundial. Como todo el mundo sabe, consiste en un cubo en el que cada una de sus seis caras está dividida en 9 cuadrados de un mismo color. Todo el cubo es articulado, de manera que los cuadrados pueden desplazarse de una cara a otra haciéndolas girar. El objetivo del juego es conseguir llegar a la configuración de partida, con cada cara de un único color, tras haberla descompuesto con movimientos aleatorios.

Los propietarios de la patente, tras cientos de millones de unidades vendidas, consideran que el mundo se les ha quedado pequeño y están intentando llegar a otros mercados. Se han dado cuenta de que algunas criaturas, como Mario, Sonic o Pacman, no son capaces de jugar al cubo de Rubik porque, por mucho que se les explica, no conciben la existencia de una tercera dimensión. De modo que están diseñando una versión simplificada de su juego: el *cuadrado* de Rubik.

En esencia, es similar al cubo, pero con sólo una cara. Los desplazamientos de los cuadrados se pueden realizar por filas o por columnas. Así, por ejemplo, si se desplaza una fila hacia la derecha, todos sus cuadrados avanzan una posición, y el cuadrado que estaba situado más a la derecha reaparece en el lado izquierdo.



Ejemplo de rotación hacia la derecha

Durante la primera fase del desarrollo de este nuevo producto, están planteándose sus dimensiones y su dificultad, por lo que quieren hacer un simulador del funcionamiento antes de comenzar el proceso de fabricación.

Entrada

El programa deberá leer varios casos de prueba de la entrada estándar. Cada caso de prueba estará compuesto de múltiples líneas. La primera indica el número n de cuadrados de ancho y alto del cuadrado de Rubik simulado, $1 \leq n \leq 50$. A continuación vendrá la situación inicial del cuadrado de Rubik a través de n líneas, conteniendo exactamente n letras del alfabeto inglés. Cada letra representa el color de un cuadro interior del cuadrado de Rubik, y puede ser tanto en mayúsculas como minúsculas, considerándose diferentes.

Tras la configuración del cuadrado, vendrá una línea con las operaciones realizadas sobre él, separadas por espacios. Una operación está compuesta de dos partes: una letra indicando si el movimiento es sobre una fila ("`f`") o una columna ("`c`"), y un número v, cuyo valor absoluto $1 \leq |v| \leq n$ indica la fila o la columna sobre la que se realiza la operación. La fila 1 se corresponde con la primera línea de la descripción; la columna 1 se corresponde con la primera letra de cada línea. Si el número v es positivo, el desplazamiento será hacia la derecha (filas) o hacia abajo (columnas). Si es negativo será hacia la izquierda o hacia arriba.

La lista de operaciones acaba con una "`x`".

La entrada acaba con un cuadrado de tamaño 0.

Salida

Para cada caso de prueba el programa escribirá, en la salida estándar, la configuración final del cuadrado de Rubik en el mismo formato que en la entrada.

Además, deberá escribir una línea con tres guiones ("`---`") después de cada caso de prueba.

Entrada de ejemplo

```
3
abc
abc
abc
f 1 f -2 x
3
aaa
bbb
ccc
c 1 c -2 x
0
```

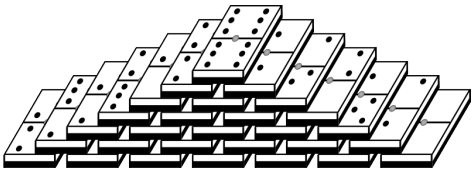
Salida de ejemplo

```
cab
bca
abc
---
cba
acb
bac
---
```

F
Pirámide de fichas de dominó

Blanca aún es demasiado pequeña para saber jugar al dominó, de modo que no entiende por qué sus abuelos se pasan las tardes dando golpes con las fichas sobre la mesa a la vez que gritan cosas como “me doblo”, “a doses” o “¡cerrado!”.

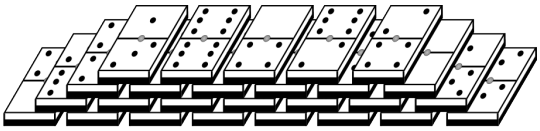
Pero cuando terminan y se marchan a la cocina a preparar la cena, se siente hipnotizada por las fichas blancas y negras, y se dedica a hacer construcciones con ellas. Una de sus preferidas es una pirámide. Cuando tiene las 28 fichas, hace una fila de 7 fichas, sobre esa coloca otra fila de 6, y así sucesivamente hasta construir una pirámide de 7 pisos, donde en la cúspide hay una única ficha.



Pirámide con 28 fichas de dominó

El fin de semana pasado empezó a montar su pirámide pero le faltaban dos fichas, que se habían caído al suelo. Al terminar, vio con pesar que su último piso tenía una única ficha, pero el anterior tenía tres, en lugar de dos.

No le gustan las pirámides que no son perfectamente escalonadas (al subir, quiere que cada piso pierda exactamente una ficha), de modo que prefiere construir una *pirámide truncada*, en la que el último piso tiene más de una ficha. Tras probar un buen rato, consiguió recolocar las 26 fichas en una pirámide de 4 pisos, en el que el piso inferior tenía 8 fichas y el superior 5.



Pirámide truncada con 26 fichas

Entrada

La entrada está compuesta por múltiples casos de prueba. Cada uno aparece en una línea independiente que contiene un número positivo hasta 10.000.000 indicando el número de fichas de dominó disponibles para construir la pirámide.

Salida

Para cada caso de prueba, el programa escribirá, en una línea independiente, la altura de la pirámide (quizá truncada) más alta que se puede construir usando *todas* las fichas disponibles.

Entrada de ejemplo

```
28
26
35
8
```

Salida de ejemplo

```
7
4
7
1
```

G
Suma descendente

Llamamos *suma descendente* de un número cualquiera a la suma de los números resultantes de ir quitando de forma reiterada el dígito más significativo de dicho número. Por ejemplo, si tenemos un número de 6 dígitos, $ABCDEF$, entonces la suma descendente será el resultado de la suma:

$$ABCDEF + BCDEF + CDEF + DEF + EF + F$$

Por ejemplo, la suma descendente del número 4.578 es igual a $4.578 + 578 + 78 + 8 = 5.242$.
Realiza un programa que calcule la suma descendente de un número dado.

Entrada

La entrada estará formada por múltiples casos de prueba y finalizará con el valor 0. Cada caso de prueba estará formado por un número entre 1 y 1.000.000.000.

Salida

Para cada caso de prueba el programa escribirá su suma descendente.

Entrada de ejemplo

```
4578
5083
999
0
```

Salida de ejemplo

```
5242
5252
1107
```

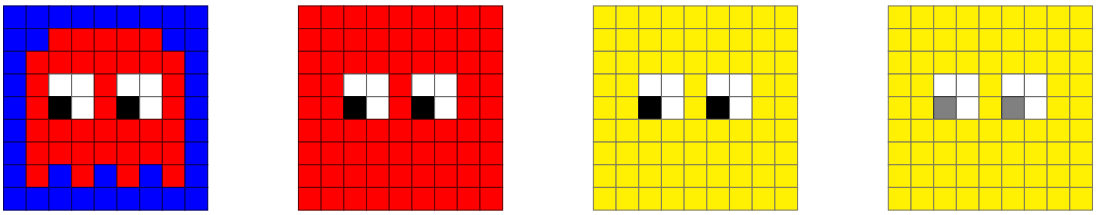
H
Copistas daltónicos

El daltonismo es un transtorno hereditario que ocasiona dificultad para distinguir ciertos colores. Hay distintos tipos de daltonismo que hacen que los colores que esas personas no distinguen varíen. Lo habitual, no obstante, es no distinguir algunos matices de verde y rojo.

Aunque el defecto genético no suele suponer ningún problema en la vida diaria de los afectados, la realidad es que les imposibilita para realizar algunos trabajos puntuales, como militares de carrera o pilotos.

Otro de los trabajos difíciles de realizar por los daltónicos es el de duplicador de obras de arte. El problema al hacer la copia de la obra es que esos dos colores que cualquier otro vería distintos terminan siendo el mismo en la copia. Si esa copia cae en las manos de un segundo duplicador daltónico que hace una nueva copia, y luego otro, y luego otro, el resultado final puede no parecerse en nada al original, sobre todo si el tipo de daltonismo de cada uno difiere[1].

Como ejemplo, en la figura aparece la transformación que puede sufrir un cuadro de *pixel art* con uno de los personajes del Pacman tras el paso por varios copistas daltónicos. Los colores originales son azul, rojo, blanco y negro. En la primera reproducción el copista sufría un tipo de daltonismo que le hacía ver igual los colores azul y rojo lo que hace que todos los azules terminen siendo rojos[2]. En la segunda reproducción el copista convirtió todos los rojos en amarillo. El último de la serie veía todos los negros grises.



Lo que haremos será, precisamente, simular esa transformación de la obra original tras pasar por las manos de numerosos copistas daltónicos.

Entrada

La entrada estará compuesta por distintos casos de prueba. Cada caso de prueba comienza con la descripción de un cuadro que será copiado en serie por distintos daltónicos.

Los cuadros se representarán mediante letras mayúsculas, cada uno representando un color. Para eso una primera línea contendrá el tamaño del cuadro: dos números entre 1 y 500 indicando el número de filas y el número de columnas respectivamente, a lo que seguirá el cuadro. Acto seguido aparecerá una línea con el número de daltónicos que copiarán el cuadro. Por último por cada uno de los copistas aparecerá una línea con dos caracteres; el primero indica el código del color que no es capaz de distinguir y que es sustituido por el código del color marcado por el segundo carácter.

La salida terminará con un cuadro de tamaño 0×0 que no debe procesarse.

Salida

Para cada caso de prueba se escribirá el cuadro tal y como lo dibuja el último copista utilizando la misma representación usada en la entrada.

[1] Aunque sorprendente, eso no resulta tan inverosímil. Hay estudiosos que afirman que Vincent Van Gogh, el famoso pintor postimpresionsta, era daltónico.

[2] Los colores del ejemplo están elegidos para minimizar las posibilidades de que un lector daltónico sufra el daltonismo indicado y cualquiera pueda entender el ejemplo sin confusión.

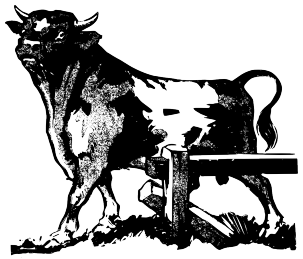
Entrada de ejemplo

```
1 4
ABCD
1
C D
9 9
AAAAAAAAA
AARRRRRAA
ARRRRRRRA
ARBBRBBRA
ARNBRNBRA
ARRRRRRRA
ARRRRRRRA
ARARARARA
AAAAAAAAA
3
A R
R Y
N G
0 0
```

Salida de ejemplo

```
ABDD
YYYYYYYYY
YYYYYYYYY
YYYYYYYYY
YYBBYBBYY
YYGBYGBYY
YYYYYYYYY
YYYYYYYYY
YYYYYYYYY
YYYYYYYYY
```

Postes para un cercado



Amanda ha heredado un terreno, de forma rectangular, en el pueblo donde se criaron sus abuelos. Siempre ha sido utilizado para cultivo, pero ella está decidida a utilizarlo para pastoreo de ganado por lo que se ve en la obligación de cercarlo para evitar que las reses se escapen.

Ha preguntado por el coste de una cerca electrificada, y la han dicho que éste depende únicamente del número de postes que sea necesario colocar. Como es lógico, es obligatorio poner un poste en cada una de las esquinas del terreno. Luego habrá que poner en cada lado postes situados a una distancia máxima los unos de los otros, para evitar que el viento, o los animales, vuelquen la cerca.

Amanda se ha dado cuenta de que averiguar el mínimo número de postes necesarios no es tan fácil. ¿La ayudas?

Entrada

La entrada consta de un conjunto de casos de prueba. Cada uno ocupa una línea, y contiene tres números mayores que 0 con las distancias de cada uno de los dos lados del rectángulo, así como la máxima distancia a la que se pueden colocar los postes. Todos los valores serán menores que 2^{31}.

La entrada acaba con una línea con tres ceros.

Salida

El programa escribirá, para cada caso de prueba, una línea con el número *mínimo* de postes necesarios para cercar el terreno. Se garantiza que el resultado será menor que 2^{31}.

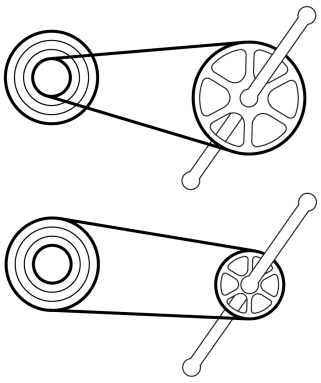
Entrada de ejemplo

```
10 10 10
6 6 10
10 15 10
0 0 0
```

Salida de ejemplo

```
4
4
6
```

J
Desarrollos en las bicicletas



En ciclismo, el *desarrollo* es la distancia que una bicicleta particular recorre por cada pedalada completa dada. El valor depende del diámetro de la rueda trasera, y de la *relación de marchas* (o *razón de cambio*), es decir, de los tamaños relativos entre el plato (lado de los pedales) y la corona o piñón (lado de la rueda).

Por ejemplo, si se da una pedalada completa en una bicicleta con un plato de 52 dientes, la rueda dará cuatro vueltas completas si se tiene una corona de 13 dientes, existiendo una relación 4:1 (parte superior de la imagen). Si, por el contrario, el plato tiene 24 y la corona 34, la rueda dará menos de una vuelta por pedalada (parte inferior).

La configuración 52-13 es alta (la rueda da muchas vueltas), por lo que proporciona velocidad a cambio de necesitar mucho esfuerzo. Por su parte, la configuración 24-34 es baja, y proporciona *fuerza* pues dar una pedalada cuesta menos trabajo, a costa de recorrer menos espacio. Para poder mantener una cadencia de pedaleo más o menos constante, las relaciones altas se utilizan en llanos o cuestas abajo y las bajas cuando se realizan ascensos.

Las bicicletas tienen varios platos y varias coronas, y el ciclista debe decidir en cada momento qué combinación utiliza. Cuando se comienza una cuesta arriba, por ejemplo, normalmente se va haciendo descender la relación, lo que da la sensación de que la bicicleta "pesa menos". El problema es que el cambio de plato y de corona es independiente, y no siempre es fácil conseguir una disminución secuencial de la relación. Por ejemplo, si una hipotética (e inexistente) bicicleta tiene dos platos, uno con 34 dientes y otro con 48, y tres coronas de tamaños 20, 26 y 34 dientes, el orden de menor a mayor desarrollo es:

Plato	34	34	48	34	48	48
Corona	34	26	34	20	26	20
Relación	1:1	17:13	24:17	17:10	24:13	12:5

Fíjate que para recorrerlos por orden, es necesario entrelazar diferentes tamaños en el plato. Conocer este orden es importante para minimizar el esfuerzo montando en bici. ¿Eres capaz de calcularlo?

Entrada

La entrada consiste en un conjunto de casos de prueba, cada uno compuesto de tres líneas. La primera contiene dos números indicando respectivamente el número de platos (P) y de coronas (C) de la bici, ambos menores que 20. La segunda línea contiene P números positivos con el número de dientes de cada plato. La tercera línea lista la cantidad de dientes de cada una de las C coronas. Todos los platos y coronas tendrán al menos 1 diente y nunca tendrán más de 1.000. Además, no habrá dos desarrollos iguales.

El último caso de prueba es seguido por una línea con dos ceros.

Salida

Por cada caso de prueba se escribirán, de menor a mayor desarrollo, todas las combinaciones posibles de platos y coronas. Para cada una se indicará primero el número de dientes del plato y luego de la corona, separándolos por un guión. Dos combinaciones consecutivas se separarán por un espacio.

Entrada de ejemplo

```
2 3
34 48
20 26 34
1 4
44
28 24 20 16
0 0
```

Salida de ejemplo

```
34-34 34-26 48-34 34-20 48-26 48-20
44-28 44-24 44-20 44-16
```