Problema A
Comida

archivo: `Comida{.c,.cpp,.pas}`

Usted ha salido a comer a un restaurant muy fino en donde personas de la farándula y famosos como *Knuth*, *Dijsktra* y *Tarjan* han sido vistos frecuentemente disfrutando de su cena. El restaurante tiene una variedad de 12 opciones y su menú se ve de la siguiente forma:

Nombre del Plato	**Precio**
Bit de Agua	1
Bit Frito	2
Sopa de Bits	4
Huevo de Bits	8
Byte de Arroz Pequeño	16
Byte de Arroz Mediano	32
Byte de Arroz Grande	64
Byte de Arroz con Curry	128
Lomo de Bits y Bytes	256
Compilado de Assembler	512
Compilado de ANSI-C	1024
Compilado de C++	2048

Note que el i-ésimo plato del menú tiene precio 2^{i-1} (para $1 \leq i \leq 12$). Por ejemplo, el "Byte de Arroz Pequeño", que es el 5to plato, tiene precio $2^{5-1} = 2^4 = 16$. No todos los días se ofrecen todos los platos, por lo que al llegar al restaurant se le indica a los comensales hasta qué plato del menú se puede elegir ese día.

La única regla de este fino y famoso restaurant es que dependiendo de cuanto dinero usted piensa gastar, siempre se debe pedir la mínima cantidad de platos tal que el costo total sea exactamente la cantidad de dinero que se piensa gastar. Por ejemplo, si todos los platos están disponibles y se piensa gastar 10 pesos en comida, en un restauran normal usted podría elegir, entre otras, alguna de las siguiente opciones:

1) 10 Bits de agua (1+1+1+1+1+1+1+1+1+1=10) (10 platos)
2) 5 Bits Fritos (2+2+2+2+2=10) (5 platos)
3) 2 Sopas de Bits con 1 Bit frito (4+4+2 = 10) (3 platos)
4) 1 Huevo de bits acompanado con 1 Bits Fritos (8+2 = 10) (2 platos)

Para este restaurant, usted sólo puede elegir la última opción ya que es la combinación de platos mínima (2 platos) para que el costo total sea 10. Ahora, si por ejemplo ese día se está sirviendo sólo hasta el tercer plato ("Sopa de Bits"), entonces sólo puede elegir la opción (3) de arriba que minimiza la cantidad de platos (3 platos).

Entrada

La primera línea contiene un número entero N que corresponde a la máxima cantidad de platos que el restaurant estará ofreciendo (por ejemplo, si $N = 3$, sólo los platos que tienen costo 1,2 y 4 se servirán). En la siguiente línea encontrará un valor P que corresponde al monto que se piensa gastar.

Salida

Debe imprimir la mínima cantidad de platos de entre los N que se ofrecen ese día, cuyo precio total sea exactamente P.

Subtareas y Puntuación

20 puntos Se probará un conjunto de casos donde el valor de N será siempre 2, mientras que P podrá tomar cualquier valor que cumpla con la condición $1 \leq P \leq 10^8$.

20 puntos Se probará un conjunto de casos donde el valor de N estará entre $2 < N \leq 12$ y el valor de P será siempre una potencia de 2 con la condición que $1 \leq P \leq 2^{N-1}$.

30 puntos Se probará un conjunto de casos donde el valor de N estará entre $2 < N \leq 12$ y el valor de P puede ser un número arbitrario que cumpla con la condición que $P \leq 2^{N-1}$.

30 puntos Se probará un conjunto de casos donde el valor de N estará entre $2 < N \leq 12$ y el valor de P puede ser un número cualquiera tal que $1 \leq P \leq 10^8$.

Entrada de ejemplo	**Salida para la entrada de ejemplo**
2 11	5

Entrada de ejemplo	**Salida para la entrada de ejemplo**
10 128	1

Entrada de ejemplo	**Salida para la entrada de ejemplo**
3 10	3

Entrada de ejemplo	**Salida para la entrada de ejemplo**
12 111111	58

Problema D
Git

archivo: `git{.c,.cpp,.pas}`

Git es un software de control de versiones diseñado por Linus Torvalds, pensando en la eficiencia y la confiabilidad del mantenimiento de versiones de aplicaciones cuando estas tienen un gran número de archivos de código fuente. Al principio, Git se pensó como un motor de bajo nivel sobre el cual otros pudieran escribir la interfaz de usuario.

Sin embargo, Git se ha convertido desde entonces en un sistema de control de versiones con funcionalidad plena. Hay algunos proyectos de mucha relevancia que ya usan Git, en particular, el grupo de programación del núcleo Linux.

Uno de los tantos comandos que pueden ser ejecutados en un repositorio de Git es *merge*. Este comando permite mezclar dos *branches* o *ramas*. La sintaxis del comando merge es:

```
git merge master devel
```

El ejemplo anterior mezcla la rama *devel* en la rama *master*. Si no hay conflictos o problemas durante el proceso una posible salida de la ejecución sería:

```
Merge made by the 'recursive' strategy.
 app/async_tasks/send_alerts_to_pms_async_task.php                 |  2 +-
 app/libs/payment/paylink_payments/ar_profile_paylink_payment.php | 18 ++++++++++++++------
 app/libs/payment/paylink_payments/co_profile_paylink_payment.php | 17 ++++++++++++++-------
 app/libs/payment/paylink_payments/mx_profile_paylink_payment.php |  2 +-
 app/models/invoices_sent.php                                      | 10 +++++++----
 app/models/s3file.php                                             | 17 +++++++++++++----
```

La idea de un grupo de estudiantes de una prestigiosa universidad es hacer su propia versión simplificada de Git y tu debes ayudarlos a lograr su objetivo. Una funcionalidad básica es contar el número de inserciones y eliminaciones hechas durante el proceso de mezcla, representadas en el output de Git por los símbolos '`+`' y '`-`', para poder mostrar el resumen al final del proceso. En el ejemplo anterior ese resumen sería:

```
6 files changed, 40 insertions(+), 26 deletions(-)
```

En la versión de Git de los estudiantes, el proceso funciona a la inversa. Dadas la cantidad de inserciones y eliminaciones, tu tarea es imprimir la representación textual del resultado de la operación de mezcla usando sólo los símbolos '`+`' y '`-`'.

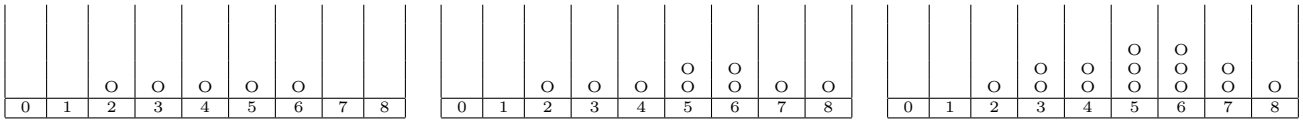
Problema M

Maíz

archivo: `maiz{.c,.cpp,.pas}`

En la granja de Don Pepe las gallinas se alimentan con maíz por una serie de tubos, un tubo por gallina. Cuando Don Pepe quiere alimentar a las gallinas, lanza maíz sobre los tubos y estos empiezan a apilar el maíz, exactamente un grano sobre otro. Suponga que los tubos están numerados de 0 hasta $N-1$. Cada vez que Don Pepe lanza maíz, su lanzamiento cubre una secuencia contigua de tubos. Por ejemplo si lanza entre el tubo 5 y el tubo 8, cubre los tubos $5, 6, 7$, y 8. El efecto del lanzamiento es que cada uno de los tubos alcanzados apila exactamente un grano de maíz adicional al que tenía previamente.

Por ejemplo, suponga que hay 9 tubos, todos inicialmente vacíos, y Don Pepe en su primer lanzamiento cubre entre el tubo 2 y el tubo 6, en su segundo lanzamiento cubre entre 5 y el 8, y en su tercer lanzamiento entre el 3 y el 7. El estado de los tubos después de cada lanzamiento se muestra en la siguiente figura.



Note que, después de los tres lanzamientos, la cantidad acumulada de granos entre los tubos 3 y 8 es 13, y la cantidad acumulada entre los tubos 0 y 4 es 5. Su tarea consiste en, dada una secuencia de lanzamientos de granos de maíz, responder varias preguntas acerca de la cantidad de granos acumulados entre pares de tubos.

Entrada

La primera línea del input contiene un entero positivo N correspondiente a la cantidad de tubos, y la segunda línea un entero positivo L correspondiente a la cantidad de lanzamientos realizados por Don Pepe. Después de esto encontrará L líneas, cada una con un par de enteros X e Y (tales que $0 \leq X \leq Y < N$) correspondiente a los tubos alcanzados por cada uno de los lanzamientos de maíz de Don Pepe. Posteriormente encontrará otro entero positivo M que corresponde a la cantidad de preguntas que debe responder. Después de esto encontrará M líneas, cada una con un par de enteros I y J (tales que $0 \leq I \leq J < N$), correspondiente a los tubos de la pregunta.

Salida

Para cada una de las M preguntas, usted debe responder la cantidad acumulada de maíz entre los tubos I y J consultados. Las respuestas deben seguir el orden de las preguntas y estar cada una en una línea diferente de la salida.

Puntuación

20 puntos Se probará un conjunto de casos donde $N \leq 100$ y $L = M = 1$, es decir habrá sólo un lanzamiento y sólo una consulta, y la consulta será tal que $I = J$ (es decir, se pedirá el valor acumulado en un único tubo).

20 puntos Se probará un conjunto de casos donde $N \leq 100$ y $L = M = 1$, es decir habrá solo un lanzamiento y sólo una consulta (pero la consulta podrá cumplir con $I \neq J$).

20 puntos Se probará un conjunto de casos donde $N, L, M \leq 10.000$.

20 puntos Se probará un conjunto de casos donde $N, M \leq 100.000$ y $L \leq 10.000$.

20 puntos Se probará un conjunto de casos donde $N, L, M \leq 100.000$.

NOTA: para los últimos casos la cantidad de maíz acumulada puede ser mucha por lo que se recomienda usar `long long` (`LongInt` en Pascal) para las variables de respuesta.

Entrada de ejemplo	**Salida para la entrada de ejemplo**
100 1 21 59 1 35 35	1

Entrada de ejemplo	**Salida para la entrada de ejemplo**
100 1 20 50 1 10 30	11

Entrada de ejemplo	**Salida para la entrada de ejemplo**
9 3 2 6 5 8 3 7 2 3 8 0 4	13 5

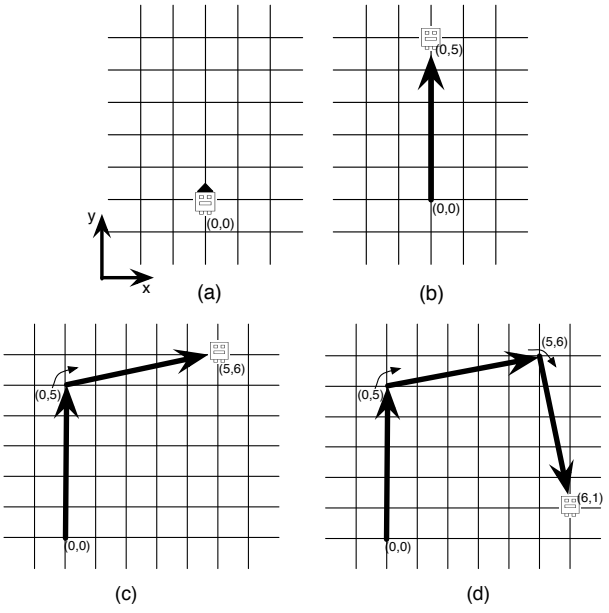
Problema R
Robot

archivo: `robot{.c,.cpp,.pas}`

Arturito es el último robot diseñado por la OCI y ha sido entrenado para seguir un plan de navegación formado por puntos en el plano. Arturito es un robot muy curioso, y no quiere obedecer respecto a su plan de navegación hasta que alguien le diga cuántas veces en su camino debe girar a la izquierda y cuántas veces debe girar a la derecha.

La OCI, desesperada por poder convencer a Arturito le pide ayuda a usted. Dado una lista de puntos en el plano que describen el plan de navegación, Arturito comienza en el primer punto de su plan y mirando hacia el segundo punto. Desde ahí en adelante, siempre va de cada punto al siguiente en línea recta. Antes de cada desplazamiento, Arturito debe girar de manera de estar mirando siempre en la dirección en que debe avanzar. Para este comportamiento, usted debe determinar cuántas veces Arturito tendrá que girar a la izquierda y cuántas veces tendrá que girar a la derecha.

Por ejemplo, suponga que Arturito tiene un plan de navegación compuesto por los puntos $(0,0)$, $(0,5)$, $(5,6)$, y $(6,1)$. De acuerdo a esto, Arturito empieza en el punto $(0,0)$ y mirando hacia el punto $(0,5)$. La siguiente secuencia muestra los desplazamientos de Arturito.



En la secuencia se puede observar que Arturito realiza dos giros a la derecha y ninguno a la izquierda. Note que en cada ocasión que Arturito debe girar, escoge el ángulo de giro más pequeño posible. En el último punto de su recorrido Arturito no necesita girar, simplemente se detiene. Note además que Arturito no necesariamente gira en cada punto. Por ejemplo, si en su trayecto debe ir del punto $(0,0)$ al $(0,1)$ y luego al $(0,2)$, dado que estos puntos son *colineales*, Arturito no necesita hacer un giro en el punto $(0,1)$. Puede suponer que el plan de navegación no contiene puntos repetidos, y es tal que Arturito no debe efectuar giros en 180 grados.