Problema 2

Implementar un algoritmo que verifique la siguiente especificación

```
predicate EstricCreciente (s : seq<int>)
ensures EstricCreciente(s) == forall u , w :: 0 <= u < w < |s| ==> s[u] < s[w]

method Problema02(v : array<int>)
      requires v != null
      requires forall k :: 0 <= k < v.Length ==> v[k] > 0
      requires EstricCreciente(v[..])

      ensures forall i::0<=i<old(v.Length) && old(v[i])%2 == 0 ==> old(v[i]) in v[..]
      ensures forall i::0<=i<old(v.Length) && old(v[i])%2 == 1 ==> !(old(v[i]) in v[..])
      ensures 0 <= v.Length <= old(v.Length)
      ensures forall i :: 0 <= i < v.Length ==> v[i] in old(v[..])
      ensures EstricCreciente(v[..])
      modifies v
```

Requisitos de implementación.

- Utilizar la plantilla `PlantillaJuezNumeroCasos.cpp` que se encuentra en el campus virtual en la pestaña `laboratorio`.
- Los datos de entrada deben almacenarse en un *vector* de la clase `vector`. La función `resuelveCaso` leerá los datos, llamará a la función `resolver` con el vector y escribirá el resultado cuando la función termine de ejecutarse. La función `resolver` recibe el vector y lo modifica.
- La función `resolver` debe tener coste lineal.
- La entrada la leéis como si fuera por teclado, utilizando el `std::cin`. Las instrucciones

  ```
  std::ifstream in("datos.txt");
  auto cinbuf = std::cin.rdbuf(in.rdbuf()); //save old buf and redirect std::cin to casos.
  ```

  del `main` redirigen la entrada de teclado a fichero. Por lo tanto los datos no se teclearán, sino que se darán en un fichero. Baja del campus virtual el fichero con los casos del ejemplo, y modifica el nombre del fichero que aparece en la instrucción `std::ifstream in("datos.txt");` al nombre que tenga tu fichero de prueba. Ejecuta el programa y las instrucciones `std::cin` leerán los datos del fichero automáticamente.
- El método `resize()` de la clase `vector` redimensiona un vector al tamaño que se le indique. Por ejemplo `v.resize(n)` modifica el tamaño (número de elementos significativos) del vector `v` a `n`.
- La profesora ha resuelto el problema (función `resolver`) con 6 asignaciones, una instrucción condicional y una instrucción de repetición.
- Dafny no verifica automáticamente el programa hay que ayudarle por medio de `asserts` y `lemas` por lo que no lo verificaremos.

Entrada

La entrada comienza con un valor entero que indica el número de casos de prueba. Cada caso de prueba consta de dos líneas. La primera indica el número de elementos del vector. La segunda contiene los valores del vector.

El número de elementos del vector puede ser muy grande, por lo que no puede suponerse un valor máximo, y el valor de los elementos se sabe que pueden almacenarse en una variable de tipo int.

Salida

Para cada caso de prueba se escribe en una línea el vector resultado.

Entrada de ejemplo

```
3
8
3 4 6 7 11 14 16 18
3
3 5 7
4
2 4 6 8
```

Salida de ejemplo

```
4 6 14 16 18

2 4 6 8
```

Autor: Isabel Pita.

Problema 3

Implementar un algoritmo que resuelva el siguiente problema:

```
method problema3 (a : array<int>, p : int) returns (b : bool)
requires a != null && a.Length > 0
requires 0 <= p < a.Length
ensures b == forall u, w :: 0 <= u <= p < w < a.Length ==> a[u] < a[w]
```

Requisitos de implementación.

- El orden de complejidad del algoritmo debe ser lineal respecto al número de elementos del vector.
- Debe utilizarse la clase `vector` para almacenar los datos y llamar a la función `resolver`

Entrada

La entrada comienza con un valor entero que indica el número de casos de prueba. Cada caso de prueba consta de dos líneas. La primera contiene el número de elementos del vector y el valor de p. La segunda línea contiene los valores del vector.

Salida

Para cada caso de prueba se escribe en una línea `SI` si el resultado de la función es cierto y `NO` si es falso.

Entrada de ejemplo

```
4
15 5
5 7 3 4 7 3 8 9 8 10 8 9 8 9 10
4 0
4 7 5 6
4 3
4 3 6 2
5 2
5 3 4 6 4
```

Salida de ejemplo

```
SI
SI
SI
NO
```

Autor: Isabel Pita.

Asalto al tren

Uno de los robos más famosos de todos los tiempos es sin duda el asalto al tren de Glasgow en 1963. Se trataba de un tren correo que viajaba de Glasgow a Londres. En el segundo vagón de la parte delantera llevaba paquetes de gran valor...El atraco se produjo entre las estaciones de .... El asalto estuvo bien planificado, los ladrones conocian los vagones que debían asaltar para obtener en mejor botín.

En este problema nos invitan a mejorar la estrategía utilizada por los ladrones. Nos dan la ganancia que obtendríamos en cada vagón del tren y el número de vagones que podemos desvalijar antes de llegar a la próxima estación. Para minimizar el riesgo los vagones que asaltaremos serán consecutivos. Tenemos que encontrar el vagón por el que debemos empezar el robo para maximizar nuestra ganancia. Ante la duda (dos posibilidades iguales) elegiremos los vagones más cerca a la cola del tren, para estar lo más alejados posible de los maquinistas.

Requisitos de implementación.

Indicar el coste de la solución obtenida.

La función que resuelve el problema debe recibir los datos en un vector y devolver el vagón que debemos asaltar en primer lugar, suponiendo que luego nos desplazaremos hacia la parte posterior del tren.

Entrada

La entrada comienza con el número de casos de prueba. Cada caso de prueba tiene dos líneas. En la primera se indica el número de vagones del tren, seguido del número de vagones que tendremos tiempo de asaltar. En la segunda se indica la ganacia que podemos sacar de cada uno de los vagones. El primer número indica el vagón más cercano a la cabecera del tren.

El número de vagones que nos dará tiempo a asaltar es menor o igual que los vagones totales del tren. El tren siempre tiene al menos un vagón.

Salida

Para cada caso de prueba se escriben en una línea el vagón en el que debemos empezar el asalto contado desde la cabecera del tren, seguido de la ganancia que esperamos obtener. El primer vagón es el cero. En caso de existir dos posibilidades que nos reporten el mismo beneficio, elegiremos la que se encuentre más lejos de la cabecera del tren.

Entrada de ejemplo

```
4
10 3
3 6 4 7 1 8 7 2 8 3
7 4
1 3 2 2 3 2 1
6 3
2 2 2 2 2 2
4 2
9 8 3 4
```

Salida de ejemplo

```
6 17
1 10
3 6
0 17
```

Autor: Isabel Pita.

Rescate aereo 1

Después de sufrir un devastador ataque alienígena, el alto mando ha recibido una llamada de rescate de la ciudad de Nueva York. El grupo de supervivientes se encuentra detras de la línea de rascacielos de la ciudad. Al otro lado de los edificios se encuentra vigilando una nave extraterrestre. Para evitar que el transporte aereo encargado del rescate sea detectado por la nave enemiga, hemos encontrado la secuencia más larga de edificios cuya altura es estrictamente mayor que la de nuestro transporte. El alto mando ya ha avisado a los supervivientes para que se dirigan a este punto de encuentro. Suponemos que todos los edificios tienen la misma anchura.

Requisitos de implementación.
Indicar el coste de la solución obtenida.
La función que resuelve el problema debe recibir los datos en un vector y devolver el comienzo y final del intervalo.

Entrada

La entrada comienza con el número de casos de prueba. Cada caso de prueba tiene dos líneas. En la primera se indica el número de edificios de la línea n, seguido de la altura del transporte t. En la segunda se indica la altura de cada edificio de la línea.

Se supone que la altura del transporte es menor que la altura de alguno de los edificios considerados y por lo menos existe un edificio.

Salida

Para cada caso de prueba se escriben en una línea el comienzo y el final del intervalo. En caso de existir dos intervalos iguales se eligirá el de la izquierda.

Entrada de ejemplo

```
2
10 5
3 6 4 8 9 8 7 2 8 9
7 5
8 8 8 2 3 9 9
```

Salida de ejemplo

```
3 6
0 2
```

Autor: Isabel Pita.

Todos con la selección

Si la selección nacional gana el próximo partido en Málaga, habrá ganado 6 partidos seguidos. Hacia bastante tiempo que no tenía una racha ganadora seguida tan larga. Nuestro periodista encargado de seguir el partido del próximo sábado quiere saber cual ha sido la racha ganadora más larga de la selección en todo la historia, para poder contarlo durante el partido.

Para ello recopila los datos y le pide a un amigo informático que le ayude a analizarlos con un programa. Deben obtener el máximo número de partidos seguidos que ha conseguido ganar la selección, si ha ocurrido varias veces que se ganasen este número de partidos, y hace cuantos partidos que finalizó la última racha.

Requisitos de implementación.

Implementar una función que reciba en un vector los datos, y devuelva la información pedida en el problema.

Entrada

La entrada consta de una serie de casos de prueba. Cada caso de prueba tiene dos líneas. En la primera se indica el número de partidos jugados por la selección. En la segunda se indica la diferencia de goles entre los dos equipos. Un valor positivo indica que la selección ganó el partido, un valor cero indica que empató y un valor negativo que perdió.

Salida

Para cada caso de prueba se escribe en una línea el número máximo de partidos seguidos ganados, el número de veces que se ha ganado este número de partidos seguidos y el número de partidos jugados desde que finalizó la última racha ganadora.

Entrada de ejemplo

```
10
2 0 -3 1 1 0 2 1 -1 2
9
-1 3 1 2 0 1 -2 4 3
10
1 1 3 0 1 -1 4 3 1 2
3
-1 0 -1
```

Salida de ejemplo

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

Autor: Isabel Pita.

Piedras preciosas

En la novela "Kim de la India", cuando Kim visita al "Curandero de perlas", éste le propone jugar junto a su otro discípulo al "Juego de las joyas". Sobre una bandeja se encuentran una serie de joyas que cada uno de los jugadores debe recordar, después de mirarlas durante un corto espacio de tiempo. Ganará aquel que consiga describir con mayor exactitud las piedras preciosas. La primera vez que juega, Kim no consigue recordar todas ellas y es vencido por el otro chico. Sin embargo, aprende pronto y las siguientes veces consigue hacer la relación exacta del contenido de la bandeja.

Dado que ahora los dos chicos son capaces de recordar las piedras de la bandeja con toda exactitud, el sahib Lurgan ha decidido modificar un poco el juego para hacerlo más complicado. Coloca las piedras preciosas formando una línea y les pregunta cuantas veces aparece una secuencia de longitud 7 que tenga al menos 3 zafiros y 2 rubís. Viendo que el juego capta su interés sigue realizando este tipo de preguntas, ¿Cuantas veces aparece una secuencia de longitud 5 con al menos 3 diamantes y 1 rubí?, o ¿Cuantas veces aparece una secuencia de tamaño 4 con al menos 2 esmeraldas y 2 jades?

Para no tener que comprobar visualmente que discípulo ha respondido de forma correcta, el sahib desarrolla un programa que dada la lista con las piedras calcula cuantas veces aparece la secuencia deseada. De esta forma no tiene miedo de equivocarse al dar el premio.

Requisitos de implementación.

Para representar las piedras preciosas se utilizará un tipo enumerado

```
enum piedrasPreciosas {diamante, rubi, esmeralda, zafiro, jade};
```

Para leer los valores se sobrecarga el operador extractor para el tipo enumerado anterior:

```
std::istream& operator>> (std::istream& entrada, piedrasPreciosas& p) {
   char num;
   entrada >>  num;
   switch (num) {
     case 'd': p = diamante; break;
     case 'r': p = rubi; break;
     case 'e': p = esmeralda; break;
     case 'z': p = zafiro; break;
     case 'j': p = jade; break;
   }
   return entrada;
}
```

Cuando se lee un valor del tipo enumerado se debe utilizar el operador de extracción.

```
int numpiedras; int numtipo1, numtipo2; piedrasPreciosas tipo1, tipo2;
std::cin  >> numpiedras  >> tipo1 >> numtipo1 >> tipo2 >> numtipo2;
```

El vector debe guardar valores del tipo `piedrasPreciosas`. Y la lectura de los valores del vector se hará con el for basado en iteradores.

```
std::vector<piedrasPrecisas> v(numElem);
for (piedrasPreciosas& i : v) std::cin >>  i;
```

Entrada

La entrada comienza con el número de casos de prueba. Cada caso tiene dos líneas. En la primera se indica el número de piedras, el tamaño de la secuencia que se busca, el primer tipo de piedra y el número de veces que debe aparecer y el segundo tipo de piedra y el número de veces que debe aparecer. En la segunda línea se indica la lista de piedras preciosas representadas por su inicial en minúsculas.

Las piedras preciosas que se consideran son diamante, rubí, esmeralda, zafiro y jade. Cada una se identifica por el primer carácter de su nombre. Se garantiza que el tamaño de la secuencia es menor o igual que el número de piedras y que la suma del número de veces que debe aparecer la primera y la segunda piedra es menor o igual que la longitud de la secuencia.

Salida

Para cada caso de prueba se escribe en una línea el número de secuencias que cumplen la propiedad pedida.

Entrada de ejemplo

```
4
6 3 d 1 z 1
r d z e d z
7 3 e 2 j 1
e j e r e e j
5 2 r 1 z 0
d z j r e
7 4 z 2 r 1
z r d z z r e
```

Salida de ejemplo

```
4
2
2
4
```

Autor: Isabel Pita.

Encuentra la última posición de equilibrio

Implementa una función que resuelva el siguiente problema:

```
function numUnos(s : seq<int>) : int
ensures s == [] ==> numUnos(s) == 0
ensures s != [] && s[0] == 1 ==> numUnos(s) == 1 + numUnos(s[1..])
ensures s != [] && s[0] != 1 ==> numUnos(s) == numUnos(s[1..])

function numCeros(s : seq<int>) : int
ensures s == [] ==> numCeros(s) == 0
ensures s != [] && s[0] == 0 ==> numCeros(s) == 1 + numCeros(s[1..])
ensures s != [] && s[0] != 0 ==> numCeros(s) == numCeros(s[1..])

method xxx (v : array<int>) returns (p : int)
requires v!= null
ensures -1 <= p < v.Length
ensures numUnos(v[..p+1]) == numCeros(v[..p+1])
ensures forall k :: p < k < v.Length ==> numUnos(v[..k+1]) != numCeros(v[..k+1])
```

Requisitos de implementación.
Indicar el coste de la solución obtenida.
La función que resuelve el problema debe recibir los datos en un vector y devolver el valor de la posición (observad que puede ser -1).

Entrada

La entrada comienza con el número de casos de prueba. Cada caso de prueba tiene dos líneas. En la primera línea se indica el tamaño del vector, en la segunda los valores separados por blancos.

Salida

Para cada caso de prueba se escribe en una línea el valor de la posición pedida. Si el tamaño del vector es cero la posición debe ser -1.

Entrada de ejemplo

```
5
8
2 1 1 0 2 3 0 2
6
0 1 0 1 1 1
5
1 1 1 1 1
4
2 3 2 3
1
0
```

Salida de ejemplo

```
7
3
-1
3
-1
```

Autor: Isabel Pita.

Suma de valores.

Implementa una función que resuelva el siguiente problema:

```
function posMaximo (s : seq<int>) : nat
ensures 0 <= posMaximo(s) < |s|
ensures forall k :: 0 <= k < |s| ==> s[k] <= s[posMaximo(s)]

function SumSinMaximo(s:seq<int>):int
      ensures s==[] ==> SumSinMaximo(s)==0
      ensures s!=[] && posMaximo(s) != 0 ==>
             SumSinMaximo(s)==s[0]+SumSinMaximo(s[1..])
      ensures s!=[] && posMaximo(s) == 0 ==>
             SumSinMaximo(s)==SumSinMaximo(s[1..])

method SumaValores (v : array<int>) returns (s : int)
requires v != null && v.Length > 0
ensures s == SumSinMaximo(v[..])
```

Requisitos de implementación.

Indicar el coste de la solución obtenida. Solo se puede recorrer el vector una vez.

La función que resuelve el problema debe recibir como mínimo los datos en un vector y devolver el valor de la suma.

Entrada

La entrada comienza con el número de casos de prueba. Cada caso de prueba tiene dos líneas. En la primera línea se indica el tamaño del vector, en la segunda los valores separados por blancos.

Salida

Para cada caso de prueba se escribe en una línea el valor de la suma pedida.

Entrada de ejemplo

```
3
6
4 3 6 4 6 6
4
9 9 9 1
8
6 6 6 3 3 7 7 4
```

Salida de ejemplo

```
11
1
28
```

Autor: Isabel Pita.

Heidi, una nueva cabaña

La cabaña del abuelo de Heidi está situada en los alpes suizos, sobre la aldea de Maienfeld. La cabaña está construida en un pequeño llano en la ladera de la montaña, rodeada en su parte trasera por tres enormes abetos, y con una preciosa vista sobre el valle desde su parte delantera.

Heidi está buscando un lugar donde construir otra cabaña en el camino hasta las cumbres. Para ello necesita localizar terrenos que tengan una superficie llana suficiente para construir, una vista tan buena como la de la cabaña de su abuelo y que estén cerca del camino.

Hoy ha ido con Pedro a recolectar los datos. Entre todos los valores que han recogido, ahora deben buscar aquella secuencia que tenga al menos $l > 1$ valores consecutivos iguales, y que sean mayores o iguales que todos los valores de su derecha, para que tenga buenas vistas.

Requisitos de implementación.

Implementar una función que reciba en un vector los datos, y devuelva en un vector diferente del de entrada los puntos donde empiezan los espacios apropiados para construir mirados desde la derecha del vector. La implementación de la función debe tratar cada dato una única vez.

La función debe devolver también la longitud de la secuencia mas larga de valores iguales y que cumpla las condiciones para construir la casa que se haya encontrado.

Entrada

La entrada consta de una serie de casos de prueba. Cada caso de prueba tiene dos líneas. En la primera se indica el número de datos tomados por Heidi y Pedro seguido de la cantidad de valores iguales que necesitan para poder construir ($l > 1$). En la segunda se indican los valores que han tomado.

Salida

Para cada caso de prueba se escribe en una línea la longitud de la secuencia más larga, seguido del número de llanos encontrados, seguido del comienzo de cada llano mirado desde la parte derecha del vector.

Tal como está dados los datos, los llanos que aparecen en el listado de salida están dados desde la derecha del vector, primero las posiciones más altas (valores mas bajos).

Entrada de ejemplo

```
10 3
3 8 8 8 5 5 6 6 6 1
3 3
4 4 4
11 2
9 9 9 2 1 2 2 4 4 1 2
6 3
4 4 8 6 6 2
```

Salida de ejemplo

```
3 2 8 3
3 1 2
3 2 8 2
0 0
```

Autor: Isabel Pita.

Cenando con los amigos

Cada año nos reunimos los amigos a cenar unos días antes de Navidad. El año pasado, como casi todos los anteriores, acabamos hablando de lo que comeríamos durante las fiestas y de lo difícil que es mantener el tipo en estos días. Sin embargo, algunos afirman que hay otras épocas peores en el año, como los cocidos en carnaval, o los helados en verano. Al final, hicimos una apuesta. Cada uno apuntaría durante este año los gramos que engordase o adelgazase cada día.

Esta noche nos vamos a reunir de nuevo para cenar y tenemos que analizar los datos para ver quien pierde la apuesta y paga la cena. Para cada secuencia de datos, buscaremos los días consecutivos en los que cada uno engordamos más, es decir, entre que dos días la suma de los pesos es máxima (mayor que entre otros dos días cualquiera). Si existen dos secuencias de igual suma, nos quedaremos con la que tenga menor longitud. Si existen dos secuencias de igual suma e igual longitud nos quedamos con la que ocurre antes en el año (la primera que aparece). De esta forma veremos en que época del año engordamos más cada uno. Como todos los amigos somos grandes comedores, nunca hacemos régimen durante muchos días seguidos, por lo que todos acabamos engordando algo por lo menos un día al año.

Requisitos de implementación.

Implementar una función que reciba en un vector los datos, encuentre la subsecuencia en la cual la suma de valores es máxima y devuelva la suma de esta subsecuencia, su punto de comienzo y su número de días. Cada valor del vector debe tratarse una única vez, y no debe utilizarse ningún vector auxiliar.

Los datos de la secuencia pueden ser positivos o negativos, pero siempre habrá al menos un valor positivo. Ten en cuenta que una subsecuencia deja de ser interesante cuando su suma es cero o negativa, ya que al haber números positivos en el vector estos serán siempre mejores que cualquier valor negativo. Al tratar cada dato, no importa si el dato es positivo o negativo, sino si la suma acumulada teniendo en cuenta ese dato es positiva a negativa. Si la suma acumulada es positiva la subsecuencia sigue siendo interesante, si es cero o negativa debemos cambiar de subsecuencia para no acumular algo negativo.

Entrada

La entrada consta de una serie de casos de prueba. Cada caso de prueba tiene dos líneas. En la primera se indica el número elementos del vector. En la segunda se indican los valores.

Los valores pueden almacenarse en una variable de tipo int. El tamaño del vector es mayor a igual que uno y se garantiza que siempre hay al menos un valor estrictamente positivo.

Salida

Para cada caso de prueba se escribe en una línea la suma de la subsecuencia, el dia en que comienza y la longitud de la subsecuencia.

Entrada de ejemplo

```
6
-3 2 1 -1 -3 2
6
2 -1 2 -3 1 -1
7
-1 1 -3 4 1 -1 2
1
4
3
-1 1 -1
3
0 2 3
8
2 3 -6 0 3 2 -1 1
```

Salida de ejemplo

```
3 1 2
3 0 3
6 3 4
4 0 1
1 1 1
5 1 2
5 0 2
```

Autor: Isabel Pita.

Número de elementos de una matriz que se encuentran situados en su fila

Teníamos una matriz cuyos valores coincidían con la fila en la que estaba el elemento. Es decir, todos los elementos de la primera fila tenían el valor 0, los de la segunda fila el valor uno etc. En algún momento los valores se han revuelto. ¿Podrías calcular cuántos de ellos siguen estando bien colocados?.

Requisitos de implementación.
Indicar el coste de la solución obtenida.
Se utilizará una matriz definida como un vector de vectores

```
std::vector< std::vector<int> > m(filas, std::vector<int> (columnas));
```

Con esta instrucción se reserva memoria al declarar la matriz, por lo que luego se pueden leer directamente los valores en la matriz.
No olvidéis pasar el parámetro constante por referencia.

Entrada

La entrada consta de una serie de casos de prueba. Cada caso consta de *n+1* líneas, en la primera se indica el número de filas (n) y el número de columnas (m) de la matriz. En las n líneas siguientes se muestran m números que se corresponden con cada fila de la matriz.
Se cumple que n y m son mayores que cero.

Salida

Para cada caso de prueba se escribe en una línea el número de elementos que siguen colocados en su fila.

Entrada de ejemplo

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

Salida de ejemplo

```
7
6
```

Autor: Isabel Pita.

Cuenta las franjas de k columnas con mas de x ceros en al menos y columnas de la franja

Nos dan una matriz de números enteros. Debemos calcular el número de franjas de k columnas que hay en la matriz, tales que en la franja hay al menos y columnas que tienen más de x ceros.

Requisitos de implementación.

Explicar como se resuelve el problema e indicar el coste de la solución obtenida justificandolo brevemente.

La función resolver recibirá la matriz que se proporciona en la entrada. En la función resolver se puede utilizar un vector auxiliar de tamaño el número de columnas de la matriz, donde se guarde información sobre cada columna. El vector debe utilizarse para mejorar la eficiencia del algoritmo, reduciendo el problema de matrices a un problema de vectores.

Entrada

La entrada consta de una serie de casos de prueba. Cada caso consta de *n+1* líneas, en la primera se indica el número de filas (n) de la matriz; el número de columnas (m) de la matriz; el ancho (k) de la franja que se considera; el número (y) de columnas de la franja que deben tener más de x ceros y por último el número (x) de ceros que deben tener como mínimo las columnas. En las n líneas siguientes se muestran m números que se corresponden con cada fila de la matriz.

Se cumple que n y m son mayores que cero.

Salida

Para cada caso de prueba se escribe en una línea el número de franjas que cumplen la condición.

Entrada de ejemplo

```
4 5 3 2 3
0 8 0 0 2
2 0 0 0 0
0 0 5 0 6
0 7 0 0 3
5 4 2 2 3
4 0 4 0
0 4 7 0
2 0 5 0
1 6 0 0
4 0 0 0
2 7 5 3 2
0 0 0 0 0 0 0
0 0 0 0 5 5 0
```

Salida de ejemplo

```
3
0
3
```

Autor: Isabel Pita.

Por una buena pesca

Al patrón de mi barco de pesca se le ha ocurrido una buena forma de incrementar nuestras capturas. Ha conseguido fotos por satélite de las áreas en que operamos, las ha dividido en cuadrantes y ha obtenido el número aproximado de peces que se encuentran en cada cuadrante en base al color del mar. Ahora tiene que buscar para cada área la zona que tiene más peces para ir alli con el barco. Además el barco permanece en cada área un número fijo de días pescando, lo que limita el número de cuadrantes en los que puede operar en cada área.

Requisitos de implementación.
Especificar la función que resuelve el problema
Indicar el coste de la solución obtenida.

Entrada

Cada caso de prueba comienza con una línea en que se indica el número de cuadrantes del área en que se está pescando, n, seguido del número de cuadrantes en los que el barco va a pescar p. En las n líneas siguientes se indica la cantidad de peces que hay en cada cuadrante.

Se supone que tanto el área en que se realiza la pesca, como los cuadrántes en que el barco va a pescar son superficies cuadradas, y se cumple: $0 < p <= n < 100$.

Salida

Para cada caso de prueba se escribe en una línea el númeo de peces que se espera encontrar en el área seleccionada.

Entrada de ejemplo

```
5 2
0 0 1 0 2
2 0 0 3 2
1 0 2 0 4
3 4 0 2 4
2 3 0 1 0
6 3
0 1 0 2 0 2
0 4 3 9 0 5
0 0 0 3 4 2
2 0 3 0 0 0
1 0 0 3 8 0
0 2 2 0 3 0
```

Salida de ejemplo

```
12
27
```

Autor: Isabel Pita.