|  |  |
| --- | --- |
|  | **Informática II Semestre 2024 – 1**  **Desafío 1. Marzo 21 de 2024** |

# Consideraciones Iniciales

Suponga que existe una estructura de datos M como la que se muestra en la Figura 1a, M tiene la particularidad que puede girar y producir resultados como los que se muestran en las Figuras 1b - 1d. Se considera que la posición neutra M es como se muestra en la Figura 1a, al estado de las Figuras 1b - 1d se les considera rotación 1, 2 y 3 respectivamente

|  |  |  |
| --- | --- | --- |
| 1 | 2 | 3 |
| 4 |  | 5 |
| 6 | 7 | 8 |

|  |  |  |
| --- | --- | --- |
| 3 | 5 | 8 |
| 2 |  | 7 |
| 1 | 4 | 6 |

|  |  |  |
| --- | --- | --- |
| 8 | 7 | 6 |
| 5 |  | 4 |
| 3 | 2 | 1 |

|  |  |  |
| --- | --- | --- |
| 6 | 4 | 1 |
| 7 |  | 2 |
| 8 | 5 | 3 |

(a) (b) (c) (d) Figura 1

Además de rotar, M puede cambiar sus dimensiones como se muestra en las Figuras 2a y 2b. No existe límite para el tamaño, siempre y cuando la cantidad de filas sea impar e igual al número de columnas.

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

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| 8 | 9 | 10 | 11 | 12 | 13 | 14 |
| 15 | 16 | 17 | 18 | 19 | 20 | 12 |
| 22 | 23 | 24 |  | 25 | 26 | 27 |
| 28 | 29 | 30 | 31 | 32 | 33 | 34 |
| 35 | 36 | 37 | 38 | 39 | 40 | 41 |
| 42 | 43 | 44 | 45 | 46 | 47 | 48 |

(a) (b)

Figura 2

La empresa de seguridad Informa2, utiliza este concepto para diseñar cerraduras, que se denominarán X, y además agrega las siguientes características a su sistema:

* Se alinean varias M una tras otra, usando la celda del centro como referencia. No existe restricción para su cantidad y sus tamaños pueden ser diferentes. Por ejemplo,

puede haber una cerradura X(5, 7, 5, 9) con cuatro estructuras alineadas, de tamaño 5x5, 7x7, 5x5 y 9x9, respectivamente.

* El sistema de apertura de X funciona a partir de la validación de una regla K, que considera el valor de una celda específica, su posición y la ubicación dentro de las diferentes estructuras alineadas. Por ejemplo, una posible entrada K a X(5, 7, 5, 9) puede ser K(4,3,1,-1,1), lo que indica que para la celda ubicada en la fila 4, columna 3 de la primera estructura, su valor A debe ser mayor al de la celda (valor B) de la siguiente estructura con la que se alinea. A su vez el valor B debe ser menor al de la celda (valor C) de la siguiente estructura con la que se alinea. Por último, el valor C debe ser mayor al de la celda (valor D) de la siguiente estructura con la que se alinea.
* Para abrir la cerradura, se deben rotar cada una de las estructuras de forma independiente, con el fin de alinear las celdas de tal forma que la validación de K sea verdadera.

# Objetivos

* Desarrollar la capacidad de solución de problemas en los estudiantes, enfrentándolos a problemáticas de la vida cotidiana.
* Identificar si el estudiante adquirió las destrezas y conocimientos fundamentales de la programación con C++, en donde resaltamos estructuras de programación (iterativas, secuenciales y decisión), tipos de datos, apuntadores, arreglos, memoria dinámica y funciones.

El objetivo principal de esta actividad es poner a prueba sus destrezas en análisis de problemas y manejo del lenguaje C++. Si usted ha llevado un proceso disciplinado de aprendizaje a lo largo del semestre, esta es una oportunidad de demostrarlo. Podrá plantear una solución viable y su resultado será satisfactorio. En caso contrario, podrá identificar sus debilidades y deberá tomar medidas con el fin de poder ser apto para enfrentar situaciones como las que se plantean en la sección Contexto.

Trate en lo posible de valorar muy bien la verdadera complejidad del problema planteado, no se rinda antes de intentarlo o de plantear los posibles escenarios de solución. Se dará cuenta que si bien, al principio le puede parecer difícil, realmente ya ha tenido la oportunidad de enfrentarse a problemas similares. Si se toma el tiempo adecuado para analizar, el proceso de codificación no le tomará mucho tiempo.

Esperamos que disfrute del desafío propuesto. Le recomendamos que lea primero todo el documento antes de comenzar y asegúrese de entender muy bien las instrucciones antes de desarrollar esta actividad evaluativa.

Fue revisado por los profesores Aníbal Guerra y Augusto Salazar.

# Especificaciones

La empresa Informa2 tiene la necesidad de que dada una regla K, se genere una configuración de X que la satisfaga. La salida del programa deber ser la configuración X, tamaño de cada una de las dimensiones y las rotaciones que se deben hacer en cada una de sus estructuras para abrir la correspondiente cerradura.

De acuerdo con lo anterior, entre otras cosas, usted deberá:

1. **[10%]** Desarrollar un módulo que permita crear estructuras de datos de tamaño variable, consistentes con las características descritas en la Consideraciones Iniciales.
2. **[10%]** Implementar funciones que permitan realizar las rotaciones a las estructuras, tal como se muestra en la Figura 1.
3. **[10%]** Desarrollar un módulo para configurar cerraduras de la tal forma que la cantidad y el tamaño de las estructuras que la componen sea variable.
4. **[10%]** Implementar funciones para validar una regla de apertura sobre una cerradura.
5. **[60%]** Desarrollar un módulo para que, a partir de una regla, se genere al menos una configuración de cerradura que se pueda abrir con dicha regla.

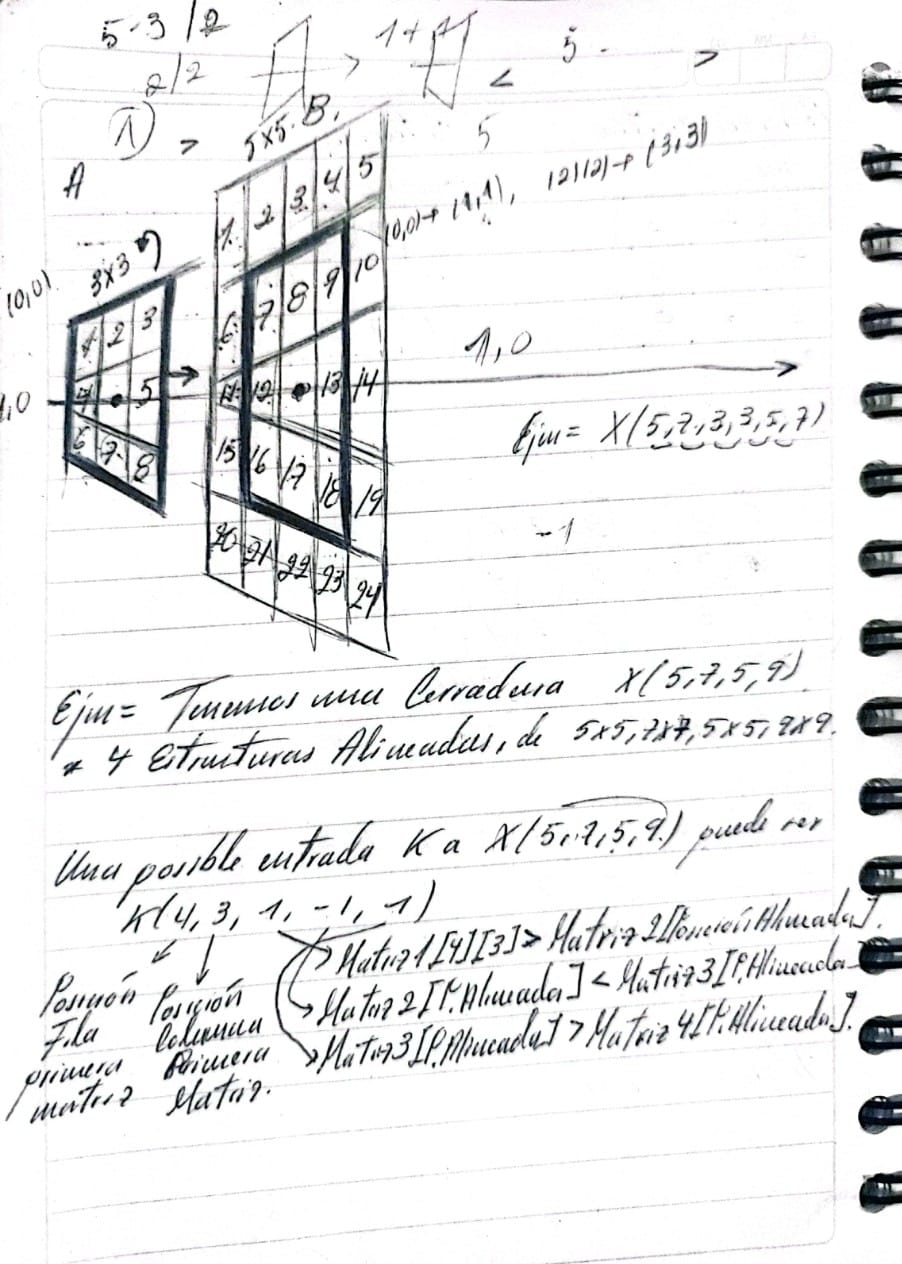
# Requisitos mínimos

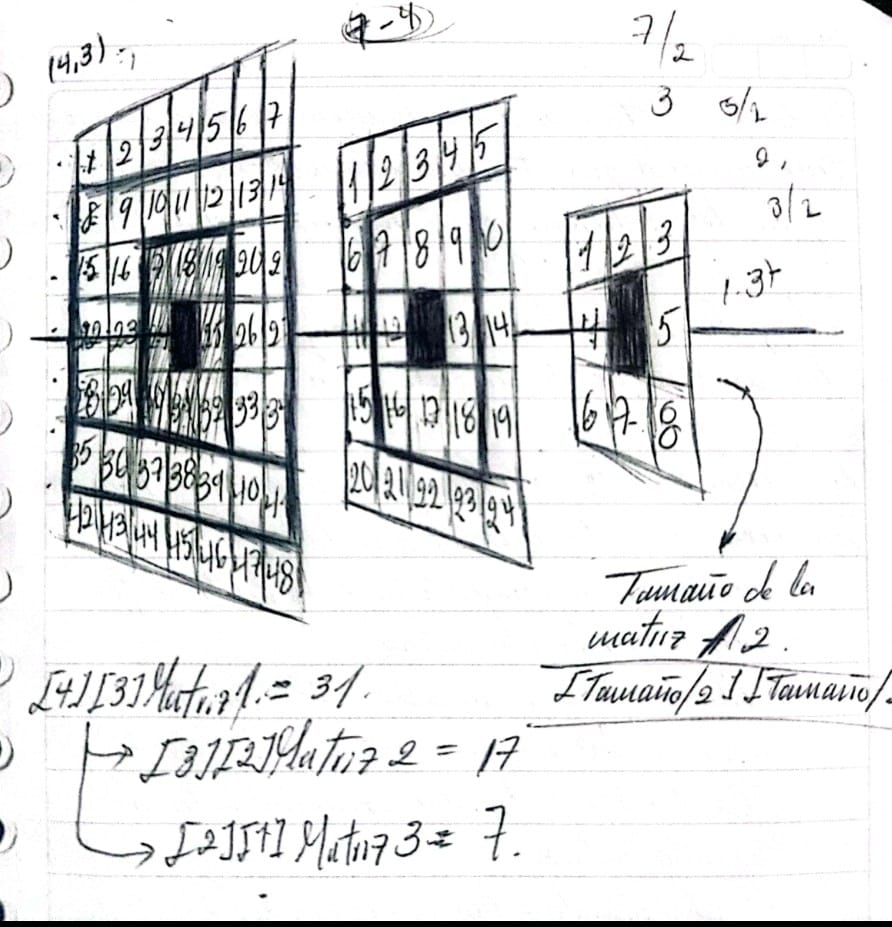
A continuación, se describen los requisitos que se deben cumplir. El incumplimiento de cualquiera de ellos implica que su nota sea cero.

* 1. Genere un informe en donde se detalle el desarrollo del proyecto, explique entre otras cosas:
     1. Análisis del problema y consideraciones para la alternativa de solución propuesta.
     2. Esquema donde describa las tareas que usted definió en el desarrollo de los algoritmos.
     3. Algoritmos implementados.
     4. Problemas de desarrollo que afrontó.
     5. Evolución de la solución y consideraciones para tener en cuenta en la implementación.
  2. La solución debe ser implementada en lenguaje C++.
  3. La implementación debe incluir el uso de punteros, arreglos y memoria dinámica.
  4. Se debe crear un repositorio público en el cual se van a poder cargar todos los archivos relacionados a la solución planteada por usted (informe, código fuente y otros anexos).
  5. Una vez cumplida la fecha de entrega no se podrá hacer modificación alguna al repositorio.
  6. Se deben hacer *commits* de forma regular (al menos dos al día) de tal forma que se evidencie la evolución de la propuesta de solución y su implementación.
  7. Se debe adjuntar un enlace de *youtube* a un video que debe incluir lo siguiente:
     1. Presentación de la solución planteada. Análisis realizado y explicación de la arquitectura del sistema (3 minutos máximo).
     2. Demostración de funcionamiento del sistema. Explicar cómo funciona: ejemplos demostrativos (3 minutos máximo).
     3. Explicación del código fuente. Tenga en cuenta que debe justificar la elección de las variables y estructuras de control usados. Por qué eligió uno u otro tipo de variable o estructura de control en cada caso particular y que ventaja ofrecen estos en comparación de otras que también podrían haber sido usados (5 minutos máximo).
     4. La duración total del video no debe exceder 11 minutos ni ser inferior a 5 minutos.
     5. Asegúrese que el video tenga buen sonido y que se puede visualizar con resolución suficiente para apreciar bien los componentes presentados.
  8. El plazo de entrega se divide en dos momentos:
     1. El día 29 de marzo para adjuntar la evidencia del proceso de análisis y diseño de la solución.
     2. El día 5 de abril para adjuntar la evidencia del proceso de implementación.
  9. Se deben adjuntar **dos enlaces**: uno al repositorio y otro al video, nada más.
  10. Para la evaluación del desafío se realizará una sustentación oral en un horario concertado con el profesor. La asistencia a la sustentación es obligatoria.

Implementación del problema  
  
Para este desafío planteado, primero, debemos tener una cerradura con varias estructuras, en la cual, si nos dan una “llave”, tenemos que tratar de buscar “el candado” que abra con esa “llave”.

Para saber cuántas estructuras o matrices debe tener el “candado”, tenemos que ver cuantas condiciones nos dan en la llave, es decir, por ejemplo, en la llave que nos dan:  
K(4,3,1,-1,1): Como nos dicen, las 2 primeras revelan la primera matriz, y las otras 3, nos dicen las condiciones que deben de tener para que este “candado” se abra.  
Por lo cual, el candado va a tener: #condiciones + 1, es decir:  
numero de estructuras = número de condiciones + 1.  
  
Ahora bien, para llenar estas matrices menos el de la posición de la mitad, necesitaremos 2 bucles for, en la cual rellenen estas posiciones empezando desde la 1 hasta terminar, menos, la posición de la mitad, la cual es:  
posición central = [tamaño de la matriz / 2][tamaño de la matriz / 2].   
  
Por ahora, no sabemos cuales matrices son las que utilizaremos, es decir, podrían ser de forma ordenada y ascendente o descendente, o podrían totalmente al azar, pero esto sería una pésima práctica.  
  
Ahora bien, tenemos otro problema, se nos dijo que las matrices están alineadas a través del centro, entonces como saber, dependiendo de la primera posición de la matriz, que posición de la siguiente matriz le corresponde.  
  
Para eso, estuvimos pensando y analizando a través de graficas:





primero, debíamos asegurarnos de verificar los límites de la segunda matriz para determinar si una posición dada en la primera matriz tiene una posición correspondiente válida en la segunda matriz.  
  
para esto implementamos una función de tipo buleano, el cual nos permitirá ver si los limites son correctos:  
  
bool esCompatible(int fila, int columna, int tamano\_primera, int tamano\_segunda) {

// Verificar que la posición dada esté dentro de los límites de la segunda matriz

if (fila >= 0 && columna >= 0 && fila < tamano\_segunda && columna < tamano\_segunda) {

return true; // La posición dada en la primera matriz es compatible con la segunda matriz

}

return false; // La posición dada no es compatible con la segunda matriz

}

Ahora bien, falta lo mas importante, saber en qué posición de la siguiente matriz va a quedar dependiendo de la posición dada en la primera matriz:

Para esto, decidí dividirlo en dos.  
Por un lado esta cuando la primera matriz evaluada es mas pequeña que la segunda.  
  
para esto tenemos las siguientes formulas:  
  
fila\_segunda= fila\_primera + (tamaño\_segunda – tamaño\_primera)/2.  
  
columna\_segunda= columna\_primera + (tamaño\_segunda – tamaño\_primera)/2.  
  
  
Para esto, primero, calculamos cuántas filas deben moverse hacia abajo la matriz pequeña para estar centrada en la matriz grande. Esto se hace tomando la diferencia de tamaños entre las dos matrices y dividiéndola entre 2 para centrarla. A eso le sumamos la fila de la posición dada en la matriz pequeña.

Por ejemplo, si la matriz grande es de tamaño 9x9 y la matriz pequeña es de tamaño 5x5, la diferencia de tamaño es 9- 5 = 4. Dividimos esto entre 2 para obtener 2, que es la cantidad de filas que la matriz pequeña debe moverse hacia abajo para estar centrada en la matriz grande. Si la posición dada en la matriz pequeña es la fila 2, entonces la fila correspondiente en la matriz grande sería 2 + 2 = 4. Si las filas son enteras, redondearíamos hacia abajo para obtener la fila 4 en la matriz grande.  
  
  
De manera similar, calculamos cuántas columnas debe moverse hacia la derecha la matriz pequeña para estar centrada en la matriz grande. Esto también se hace tomando la diferencia de tamaños entre las dos matrices y dividiéndola entre 2. A eso le sumamos la columna de la posición dada en la matriz pequeña.

Siguiendo el ejemplo anterior, si la posición dada en la matriz pequeña es la columna 3, entonces la columna correspondiente en la matriz grande sería 3 + 2 = 5. Nuevamente, redondearíamos en caso de hacerlo, hacia abajo para obtener la columna 5 en la matriz grande.

En resumen, estas fórmulas calculan la posición en la matriz grande que corresponde a una posición dada en la matriz pequeña, manteniendo ambas matrices centradas por el centro una respecto a la otra.  
Esto, solo cuando tenemos primero una matriz pequeña y la siguiente una mas grande que esta.

Ahora bien, necesitamos otras “formulas” para saber la posición respectiva en el caso contrario, es decir, cuando la primera matriz es más grande que la siguiente.

Para esto igualmente tenemos las siguientes formulas:  
  
fila\_segunda= (fila\_primera – (tamaño\_primera – tamaño\_segunda)/2)  
  
columna\_segunda= (columna\_primera – (tamaño\_primera – tamaño\_segunda)/2)  
  
  
Supongamos que tenemos una matriz grande y una matriz pequeña, ambas cuadradas, y queremos encontrar la posición en la matriz pequeña que corresponde a una posición dada en la matriz grande, usamos casi que el mismo razonamiento con las anteriores.

Primero, calculamos cuántas filas debe moverse hacia arriba la matriz grande para estar centrada en la matriz pequeña. Esto se hace tomando la diferencia de tamaños entre las dos matrices y dividiéndola entre 2. Luego, a eso le restamos la fila de la posición dada en la matriz grande.

Por ejemplo, si la matriz grande es de tamaño 9x9 y la matriz pequeña es de tamaño 5x5, la diferencia de tamaño es 9 - 5 = 4. Dividimos esto entre 2 para obtener 2, que es la cantidad de filas que la matriz grande debe moverse hacia arriba para estar centrada en la matriz pequeña. Si la posición dada en la matriz grande es la fila 8, entonces la fila correspondiente en la matriz pequeña sería 2 - (8 - 5) = 2 - 3 = -1. En este caso, al ser una posición negativa, significa que la posición correspondiente en la matriz pequeña no existe en términos enteros, lo que indica que la posición dada en la matriz grande no está alineada con la matriz pequeña en este contexto.

De manera similar, calculamos cuántas columnas debe moverse hacia la izquierda la matriz grande para estar centrada en la matriz pequeña. Esto también se hace tomando la diferencia de tamaños entre las dos matrices y dividiéndola entre 2. Luego, a eso le restamos la columna de la posición dada en la matriz grande.

Siguiendo el ejemplo anterior, si la posición dada en la matriz grande es la columna 9, entonces la columna correspondiente en la matriz pequeña sería 2 - (9 - 5) = 2 - 4 = -2. Al igual que en el caso de las filas, una posición negativa indica que la posición correspondiente en la matriz pequeña no existe en términos enteros.  
  
  
  
  
  
  
Ahora bien, pensando en esto, nos podemos dar cuenta de que, con esto, no es necesario esta función:  
  
bool esCompatible(int fila, int columna, int tamano\_primera, int tamano\_segunda) {

// Verificar que la posición dada esté dentro de los límites de la segunda matriz

if (fila >= 0 && columna >= 0 && fila < tamano\_segunda && columna < tamano\_segunda) {

return true; // La posición dada en la primera matriz es compatible con la segunda matriz

}

return false; // La posición dada no es compatible con la segunda matriz

}

Ya que solo con ponerle un condicional de que si la fila\_siguiente o columna\_siguiente es menor a 0   
((fila siguiente<0)||(columna\_siguiente<0)), esto quiere decir que se sale de los limites.  
  
Luego de esto, tenemos que hacer creo que lo mas importante, que seria rotar las diferentes estructuras o matrices en los 4 estados, el neutro, el estado 1, el 2 y el 3.   
  
para el neutro no tenemos que hacer nada, ya que es la misma matriz, ahora bien, para el estado neutro, analizamos la matriz y llegamos a la conclusión para poder rotarla 90 grados en sentido anti horario.

Ejm: tenemos una matriz cuadrada de tamaño 3x3.

|  |  |  |
| --- | --- | --- |
| 1 | 2 | 3 |
| 4 |  | 5 |
| 6 | 7 | 8 |

Ahora, le sacamos la transpuesta, esto, en términos de código:

Esta sería la original:  
\*(\*(puntero\_matriz+i)+j)  
  
y esta la transpuesta:  
\*(\*(puntero\_matriz+j)+i)  
  
es decir, se cambian las columnas por las filas y las filas por las columnas, quedando la matriz de la siguiente manera:

|  |  |  |
| --- | --- | --- |
| 1 | 4 | 6 |
| 2 |  | 7 |
| 3 | 5 | 8 |

Ahora bien, notemos que queremos llegar a esto:

|  |  |  |
| --- | --- | --- |
| 3 | 5 | 8 |
| 2 |  | 7 |
| 1 | 4 | 6 |

Como notamos, simplemente lo que nos falta, es cambiar la posición de las filas, la primera debería ser la última, la segunda la penúltima y así sucesivamente.   
En términos de código, la fila 0, debería ser la (n-1), y la (n-1) debería ser la fila 0, etc.  
  
Por lo que la matriz nos queda en el estado 1:

|  |  |  |
| --- | --- | --- |
| 3 | 5 | 8 |
| 2 |  | 7 |
| 1 | 4 | 6 |

Ahora, para llegar al estado2 y 3, donde hay que rotarla 180 grados y 270 grados, notamos que es hacer el mismo procedimiento, ya que se están rotando 90 grados siempre en sentido antihorario, por lo que para el estado 2 simplemente es hacer 2 veces el procedimiento del estado 1 con la matriz neutra. Lo mismo con el estado 3, simplemente, repetir 3 veces el procedimiento del estado 1 con la matriz neutra.