

Universidad De Antioquia

Facultad de ingeniería

Materia:

Informática II

Trabajo presentado por:

Carlos Manuel Gutiérrez Peñafiel

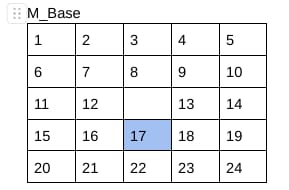
Fabio Andrés García Pérez

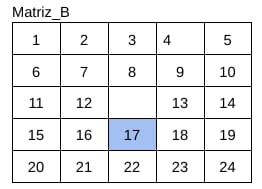
Presentado a:

Augusto Enrique Salazar Jimenez

Medellín, Colombia

**Análisis del problema:**

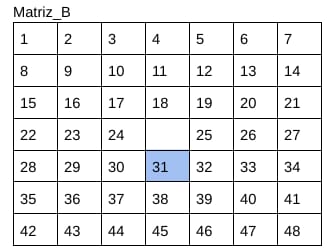
Para este programa se necesitará de una regla K que ingrese el usuario para armar los arreglos bidimensionales a comparar. Se deberá trabajar con dos arreglos al mismo tiempo. Con el primer arreglo se buscará obtener el numero a comparar y luego se creará el segundo arreglo que cumpla con la condición dada anteriormente.



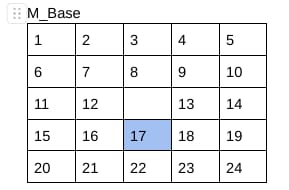


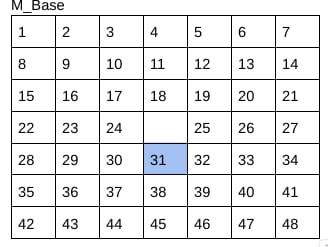
Después de crear el segundo arreglo, se compararán los números que están alineados al colocar un arreglo sobre otro. Si la condición no se llega a cumplir, entonces se comenzará a rotar el segundo arreglo en contra de las manecillas del reloj hasta un máximo de 3 veces (cada vez que rota vuelve a comparar), si al final de esto sigue sin cumplirse la condición, entonces se le aumentara +5 a la dimensión del segundo arreglo y se compararan de nuevo los dos números que estén alineados en los dos arreglos y este proceso se repetirá hasta que se cumpla la condición.

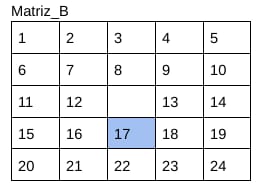






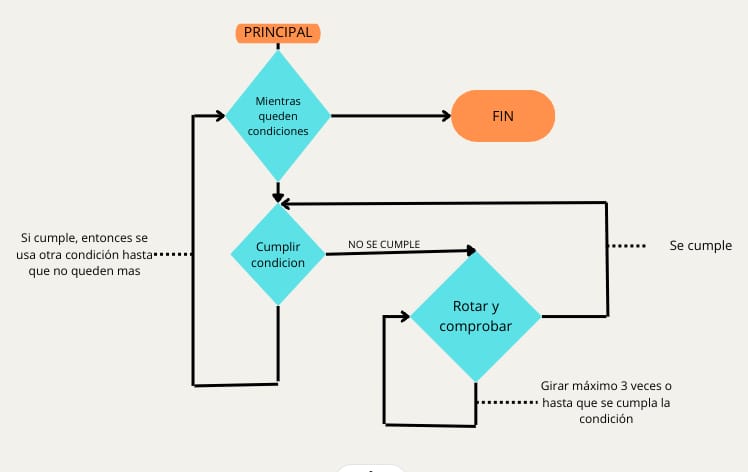


Cuando se cumpla la condición entonces M\_Base va a tener el mismo arreglo que Matriz\_B y luego Matriz\_B va a comenzar de nuevo con el arreglo base y se hará de nuevo el proceso de comparación siempre y cuando aún queden condiciones por cumplir.



Existen 3 condiciones diferentes y se podrían separar por casos:

* Caso1: cuando la condición es -1 quiere decir que el número que vamos a comparar en M\_Base debe ser menor al número que este alineado del siguiente arreglo Matriz\_B y dependiendo si se cumple o no la condición se hará el mismo proceso descrito anteriormente.
* Caso2: cuando la condición es 0 quiere decir que el número que vamos a comparar en M\_Base debe ser igual al número que este alineado del siguiente arreglo Matriz\_B y dependiendo si se cumple o no la condición se hará el mismo proceso descrito anteriormente.
* Caso3: cuando la condición es 1 quiere decir que el número que vamos a comparar en M\_Base debe ser mayor al número que este alineado del siguiente arreglo Matriz\_B y dependiendo si se cumple o no la condición se hará el mismo proceso descrito anteriormente.

**Estructura del algoritmo:**

**Algoritmos implementados:**

* **MODULO1:**

|  |
| --- |
| 1. **Int\*\* CrearM (unsigned int n):** Esta función tiene como parámetro un entero positivo que es la dimensión del arreglo que quiero crear y me retorna un doble puntero a un arreglo bidimensional de enteros. 2. **Void rotarM (int\*\* matriz, unsigned int orden):** esta función tiene como parámetros un doble puntero a un arreglo bidimensional de enteros y un entero positivo que sería la dimensión del arreglo bidimensional. Esta función no retorna nada sin embargo modifica directamente el arreglo bidimensional a través del doble puntero y rota el arreglo 90 grados en contra de las manecillas del reloj. 3. **Void cambiar\_matriz (int\*\*& matriz1, int\*\* matriz2, unsigned dimension2):** esta funcione tiene como parámetro la dirección de memoria de un primer puntero doble, un segundo puntero doble a un arreglo bidimensional y la dimensión de este segundo arreglo. Esta función mediante la dirección de memoria del primer puntero, reserva memoria para crear un nuevo arreglo bidimensional que va a tener los mismos datos que el segundo arreglo. |

* **MODULO2:**

|  |
| --- |
| 1. **Void liberar\_memoria (int\*\* matriz, unsigned int dimensión):** esta función tiene como parámetro un doble puntero a un arreglo bidimensional de enteros y la dimensión de este arreglo. esta función no retorna nada, pero se encarga de liberar la memoria reservada. 2. **Int\* validar ():** esta función no tiene ningún parámetro, pero se encarga de pedir al usuario que ingrese la clave y valida cada una de las entradas. Al final retorna un puntero a un arreglo de enteros, donde están todos los datos de la clave. 3. **Bool verificar\_condicion (int\*\* matriz\_base, unsigned int dimensión, unsigned int numero comprobar):** esta función tiene como parámetros un doble puntero a un arreglo bidimensional de enteros, la dimensión de este arreglo y un entero que es el numero a comparar. Esta función determina si el número que se va a comparar puede cumplir la condición “1” porque los números ubicados en la esquina superior izquierda de mi arreglo base no pueden cumplirla y esto ayuda a evitar y aborto de la ejecución. Se retorna “true” si el número se encuentra en la esquina superior izquierda del arreglo base y “false” si no se encuentra, esto quiere decir que se puede cumplir la condición “1” sin problemas. 4. **Bool comparar\_arreglos (int\*\* matriz1, int\*\* matriz2, unsigned int fila, unsigned int columna, unsigned int fila2, unsigned int columna2, int condición):** esta función recibe como parámetros 2 punteros dobles a arreglos bidimensionales con sus respectivas ubicaciones para comparar los dos números y un entero que sería la condición. Esta función retorna “true” o “false” dependiendo de si se cumple o no la condición dada. |

**Problemas en el desarrollo:**

1. **Saber el tamaño del arreglo dinámico:** durante el proceso de ingresar la clave y guardar todo en un arreglo dinámico, luego se tuvo el inconveniente de que para iterar sobre cada elemento de este arreglo necesitábamos saber hasta qué punto debía detenerse, por lo que se llegó a la conclusión de introducir el tamaño del arreglo en la posición cero.
2. **Ubicación incorrecta:** aunque no es un problema muy grave, si costo un poco darse cuenta. Cada vez que se iban a comparar los números de dos arreglos, arrojaba resultados incorrectos, esto se debía a que no estaba tomando los números en las ubicaciones correctas de cada arreglo, luego de buscar se dio con que no se le había restado -1 a cada ubicación porque en estas estructuras de datos se comienza desde la posición 0.
3. **Aborto de la ejecución:** según la lógica implementada. Cada vez que se cumplía una condición se volvía a crear el segundo arreglo con las características del primer arreglo base y luego se iba cambiando, dependiendo de las condiciones, entonces había un segmento del arreglo que no podía cumplir la condición “1” en el arreglo base, este segmento estaba delimitado para las celdas cuya fila era menor al número de ubicación del centro y cuya columna era menor o igual al número de ubicación del centro.

***Arreglo de ubicaciones***

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **1,1** | **1,2** | **1,3** | **1,4** | **1,5** |
| **2,1** | **2,2** | **2,3** | **2,4** | **2,5** |
| **3,1** | **3,2** | **3** | **3,4** | **3,5** |
| **4,1** | **4,2** | **4,3** | **4,4** | **4,5** |
| **5,1** | **5,2** | **5,3** | **5,4** | **5,5** |

En el arreglo anterior se puede observar las posiciones de mi arreglo base que no pueden satisfacer la condición “1”. Se opto por crear una función para verificar cada número y así saber si se encuentra en ese segmento del arreglo, y si es así, entonces se imprimirá por pantalla un mensaje “No se puede crear una cerradura con la clave ingresada”, luego se liberar toda la memoria reservada y se terminará de ejecutar el programa.

1. **Arreglo mal creado:** se tenía previsto que una vez se cumpliera una condición entre dos arreglos, entonces el primer arreglo tomaría todos los valores del segundo arreglo y el segundo arreglo se iniciaría de nuevo con la dimensión original, sin embargo al pasar los datos del arreglo dos al arreglo uno, no se estaban copiando los datos correctamente, esto se debía que la función encargada de este trabajo tenía como parámetros los dobles punteros de los dos arreglos, sin embargo, para copiar los datos del arreglo dos al uno, primero había que liberar la memoria reservada, por lo que al pasar ese puntero que estaba apuntando a nulo, no se copiaban los datos en ninguna dirección de memoria de ningún arreglo bidimensional. Por lo que se tuvo que pasar la dirección de memoria del primer puntero para así reservar nueva memoria y copiar los datos correctamente.

**Evolución de la solución:**

1. Se pedirá al usuario que ingrese la clave y se guardará todo en un arreglo dinámico.
2. Separar la fila y columna para crear los dos primeros arreglos.
3. Iterar sobre la clave para tener cada condición y hasta que no se cumpla una, no se pasa a la siguiente.
4. Si la condición es “1” verificar que el número del primer arreglo no se encuentra en la esquina superior izquierda. si el número se encuentra en este segmento se liberará la memoria reservada y se acabara el programan. Si no se encuentra en este segmento se continuará con normalidad.
5. Se compararán los dos números de los dos arreglos. Si no cumplen entonces se rotará el segundo arreglo hasta un máximo de 2 veces y si todavía sigue sin cumplir, entonces, se aumentará la dimensión del segundo arreglo en +2, y también la ubicación de fila y columna en +1, para luego repetir el paso 4.
6. Si cumple la condición entonces lo que estaba en el segundo arreglo se copiara en el primero y el segundo arreglo se iniciara de nuevo con las dimensiones originales y se comenzara desde el paso 3 iterando sobre la siguiente condición.
7. Si al final del programa no hay nada inusual, se liberara toda la memoria reservada anteriormente.

**Consideraciones para tener en cuenta en la implementación:**

Debido a la lógica utilizada para esta solución se debe tener presente que hay un número significativo de claves que no se puede crear su respectiva cerradura, y es por eso que verificamos esos casos para que no se aborte la ejecución y no se pueda liberar correctamente la memoria, antes de que esto suceda aparecerá un mensaje por pantalla de que no se puede crear una cerra dura para la clave ingresada, luego se liberara la memoria reservada y por último finalizara el programa.