# INFORME PARCIAL 1

DANIEL PINZON

NICOLAS VELASQUEZ

Universidad De Antioquia

INFORMATICA ll

ANIBAL JOSE GUERRA SOLER

AUGUSTO ENRIQUE SALAZAR JIMENEZ

31/03/2024

Primero se hizo énfasis en leer las descripciones del programa y sus características, así como las funciones que debe cumplir y los aspectos a tener en cuenta. Después de tener una vista general, se abordó en problema en fracciones para que sea más fácil realizar el análisis y la posterior implementación del proyecto.

Antes de analizar el problema es necesario tener una idea general de lo que debe llevar a cabo, por esta razón es importante entender la funcionalidad que debe cumplir el programa:

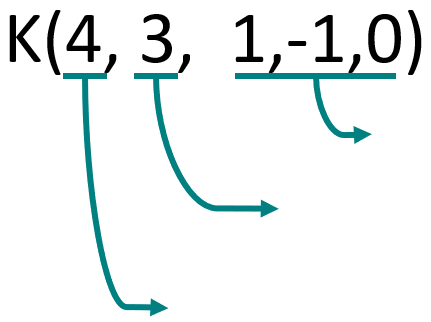
La entrada del programa es una clave K (ejemplo: K(4,3,1,-1,1) ), debemos dar una configuración de cerradura (X) que satisfaga esta clave.

Esto da a entender que es necesario analizar el funcionamiento de los componentes de una clave K y cómo estos interactúan con una cerradura X, además también es necesario comprender los comportamientos de las matrices que componen una cerradura para así interpretar en su totalidad la estructura.

**Análisis y Consideraciones**

1. **Análisis de la clave.**

Al observar la clave de ejemplo, se sacó como conclusión que, aparte de ser una herramienta necesaria para realizar las comparaciones, es útil para determinar el tamaño mínimo de las matrices con ayuda del valor de la fila o el valor de la columna, y la cantidad de matrices asociadas a una cerradura con la cantidad de comparaciones:



Fila de la primera estructura.

Columna de la primera estructura.

Comparaciones | Cantidad de comparaciones

1. **Análisis de las matrices.**

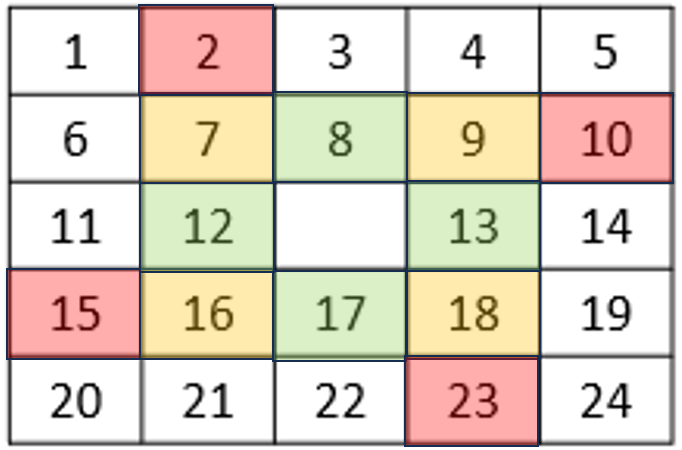
En un primer momento se analizó la estructura de la matriz con el fin de determinar valores universales y fórmulas que se adapten al comportamiento general de las mismas. (Se consideran las divisiones como enteras)

F = Fila | C = Columna | n = 5 | nxn donde n es un número impar mayor que 0.

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

* Valor de esquina superior derecha = n
* Valor de esquina superior izquierda = 1
* Valor de esquina inferior derecha = (nxn)-1
* Valor de esquina inferior izquierda = (n\*n) - n
* Casilla centro = ((nxn)/2 +1)
* Valor casilla antes del centro: C + ((F-1)xn)
* Valor casilla después del centro: (C + ((F-1)xn))-1

Además, se observó al rotar la matriz que cada una de sus casillas solo puede obtener 4 valores como máximo. Por otro lado, la distancia que el conjunto de casillas asociadas por rotación guarda con respecto al borde se conserva. Al conjunto de casillas asociadas por la rotación se les nombró como **familia de casillas**:

Teniendo una fórmula para obtener el valor de una casilla cualquiera, se optó por encontrar una trasformación que permita, a partir de una casilla dada obtener la casilla correspondiente a **una** rotación de la matriz, esto debido a que, existe un patrón de rotación entre las casillas pertenecientes a una familia de casillas:



F = Fila, C = Columna

Transformación de rotación (->) = (F,C) -> (C, n-F+1)

Ejemplo: (2,3) -> (3,6)

* Valor de (2,3) = 10
* Valor de (2,3) rotado **una** vez = Valor de (3,6) = 20

Entonces, si se quiere obtener el valor de la casilla después de una segunda o tercera rotación, es necesario volver a aplicar la transformación a la casilla la cantidad de veces que se quiera rotar la matriz, para después, aplicar la formula correspondiente para hallar su valor. No obstante, para efectos de reducir ciclos de procesamiento, también se expandió el resultado de realizar la transformación de forma consecutiva para hasta tres rotaciones:

Número de rotaciones

(F,C) -> (C, n-F+1) -> (n-F+1, n-C+1) -> (n-C+1, F)

#R =

0R 1R 2R 3R

De esta forma se obtiene que, para conseguir la casilla correspondiente a:

* Una rotación se aplica (F,C) -> (C, n-F+1)
* Dos rotaciones se aplica (F,C) -> (n-F+1, n-C+1)
* Tres rotaciones se aplica (F,C) -> (n-C+1, F)

Es importante notar que si se volviera a aplicar la transformación por cuarta vez consecutiva, la casilla obtenida debe ser la casilla en su forma original (F,C)

1. **Análisis de las comparaciones.**

Para dar solución al problema planteado es importante entender que significa cada comparación y cómo estas, dependiendo de algunos factores, interactúan con las matrices y generan un conjunto de acciones a considerar:

* **Si la comparación es igual a 1.**

Aumentar el tamaño de la primera matriz

* Si F<=(n/2) y C<n

Rotar la segunda matriz una vez.

* F>(n/2) y C<=(n/2 +1)

Rotar la segunda matriz tres veces.

* C=n y F>(n/2)
* C>(n/2 +1) y F>(n/2)



* **Si la comparación es igual a -1.**

Aumentar la dimensión de la segunda matriz.

* F>(n/2 +1) y C>1

Rotar la segunda matriz una vez.

* Si F<=(n/2 +1) y C>(n/2)

Rotar la segunda matriz tres veces.

* Si C=1 F>(n/2 +1)
* Si F<=(n/2 +1) y C<=(n/2)



* **Si la comparación es igual a 0**.
* Si son de diferente dimensión:

Verificar la dimensión de la primera matriz y cambiar la segunda matriz por una de igual dimensión que la primera.

* Si son de igual dimensión:

Verificar el estado de rotación de la primera matriz usando las fórmulas de las esquinas (ver análisis de matrices) y rotar la segunda a dicho estado.



**Solución, Algoritmos y Tareas**

* **[10%]** Desarrollar un módulo que permita crear estructuras de datos de tamaño variable, consistentes con las características descritas en la Consideraciones Iniciales.

Para este punto se creará un módulo que contenga todo lo relacionado a la gestión de las matrices (por lo que se incluirá en este módulo el punto 2: **[10%]** Implementar funciones que permitan realizar las rotaciones a las estructuras.), tanto su creación, su rotación y liberación de la memoria correspondiente, esto debido a que se plantea la utilización de memoria dinámica.

Módulo Matriz

* Función para la creación de una matriz:

1. Tener una dimensión definida.
2. Declarar la matriz dinámica (nxn) con tipo Unsigned Short Int.
3. Rellenar la matriz utilizando ciclo for doble. Añadiendo valores con ayuda de la formula desde la primera y última casilla hacia la casilla centro para reducir los ciclos a la mitad.
4. Asignar a la casilla centro el valor de 0.
5. Devolver el apuntador a la matriz

* Función para rotar una matriz existente:

1. Tener una matriz y su dimensión definida, además de la cantidad de rotaciones a realizar.
2. Recorrer con un doble ciclo for la matriz siguiendo la misma idea de la función anterior. Se realiza la transformación de rotación a cada casilla, se obtiene el valor de la casilla rotada y se reemplaza.

* Función para determinar el estado de rotación de una matriz:

1. Tener una matriz y su dimensión definida.
2. Comprobar a cuál de todas las fórmulas para las esquinas corresponde el valor de la casilla (1,1) y así determinar el estado de rotación.
3. Devolver el estado de la rotación.

* Función para borrar la matriz y liberar la memoria:

1. Tener una matriz y su dimensión definida.
2. Recorrer con un for la cantidad de filas de la matriz, liberando la memoria correspondiente a cada una de ellas.
3. Liberar la memoria del arreglo principal.

* **[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.

Módulo Cerradura

* Función para crear cerradura.

1. Tener un tamaño definido y la posibilidad de solucionar la clave.
2. Declarar la cerradura estática con tamaño definido por la función para obtener la cantidad de matrices.
3. Crear la cantidad de matrices necesarias con un ciclo for (Función para crear matriz) y almacenarlas en el arreglo.
4. Con ayuda del mismo for, se almacenan las dimensiones de cada matriz en otro arreglo.

* Función para obtener la dimensión mínima de las matrices de la cerradura.

1. Tener una posición determinada (clave).
2. Buscar el mayor entre la fila y la columna.
3. Si ese número es par, entones sumarle 1.
4. Devolver el valor.

* Función para obtener la cantidad de matrices necesarias de cerradura.

1. Tener las comparaciones (clave).
2. Contar la cantidad de comparaciones y al valor resultante sumarle 1.
3. Devolver el valor.

* **[10%]** Implementar funciones para validar una regla de apertura sobre una cerradura

Módulo FuncionesComprobación.

* Función que verifica si una cerradura se abre con una clave dada.

1. Tener una cerradura definida, junto a sus matrices y respectivas dimensiones. Una llave.
2. Con un ciclo se compara una casilla específica de un par de matrices, empezando desde las dos últimas, tomando como primera la penúltima y así sucesivamente.
3. Si se cumple la condición se pasa al siguiente par de matrices, si no, se devuelve un valor de falso.

* Función que determina el camino a seguir dependiendo de la comparación entre dos matrices. (Esta va dentro de un for).

1. Tener una cerradura definida, junto a sus matrices y respectivas dimensiones.
2. Separar los casos como en el análisis de las comparaciones.
3. Llevar a cabo el camino de acción para cada caso.

**Detección de Problemas**

En el análisis del programa nos encontramos con los siguientes problemas:

1. En el momento en el que se decidió realizar la creación de la cerradura, se decidió que esta se guardaría en un arreglo estático para tener más facilidad de accesibilidad a sus datos, por lo que, si se deseaba guardar el arreglo, se necesitaba saber específicamente cuantos componentes tenia este arreglo (en este caso de la cerradura serian cuantas matrices).

Después se optó por la solución de dividir en partes la llave que ingresaba el usuario y preguntarle al usuario sobre las comparaciones a realizar, ya que, como se analizó anteriormente, el numero necesario de matrices para la cerradura seria la cantidad de comparaciones más uno.