# INFORME PARCIAL 1

Daniel Esteban Pinzón Martínez

Nicolás Velásquez

Universidad De Antioquia

Informática II

Aníbal José Guerra Soler

Augusto Enrique Salazar Jiménez

7/04/2024

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:

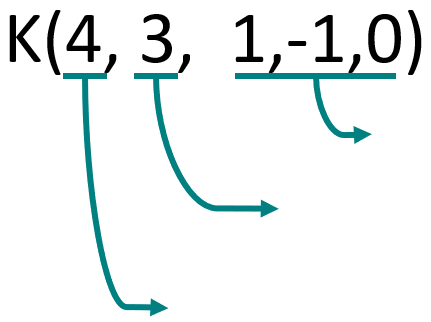
La entrada del programa es una clave K (ejemplo: K(4,3,1,-1,1) ), debemos dar una configuración de cerradura (X) que satisfaga esta clave.

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.**

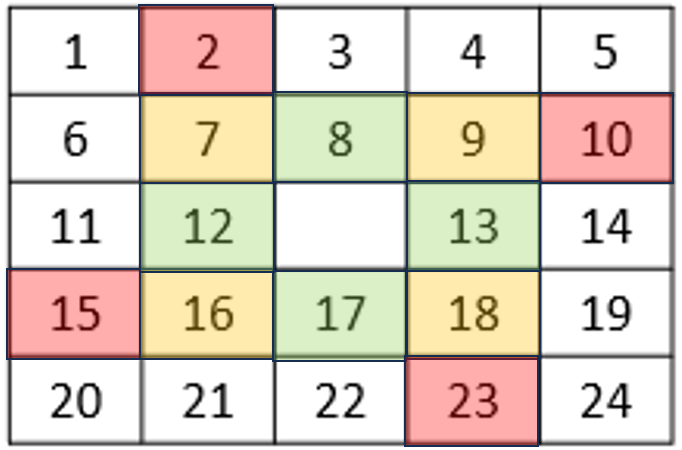
En un primer momento se analizó la estructura de la matriz con el fin de determinar valores universales y fórmulas que se adapten al comportamiento general de las mismas. (Se consideran las divisiones como enteras)

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). Por otro lado, es posible obtener la casilla a la que pertenece un valor siempre y cuando el valor no sea divisible por la dimensión:

* Fila asociada a un valor: ((V-1)//n)+1 | (V//n)+1
* Columna asociada a un valor: V%n | (V+1)%n

1. **Análisis de las comparaciones.**

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

* **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:

Aumentar la dimensión de la matriz de menor dimensión a la de mayor dimensión

* 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 crear una matriz.
* Función para rotar una matriz existente.
* Función para determinar el estado de rotación de una matriz.
* Función para borrar la matriz y liberar la memoria.
* Función para transformar la fila a una determinada rotación.
* Función para transformar la columna a una determinada rotación.
* Función para obtener el valor asociado a una fila y columna.
* Función que determina si una casilla esta detrás o delante del centro.
* Función que imprime la matriz.
* **[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.
* Función para obtener la dimensión mínima de las matrices.
* Función para obtener la diferencia entre la fila y columna de dos matrices de dimensiones diferentes
* Función para igualar la rotación de la segunda matriz al de la primera.
* Función para reemplazar una matriz por otra de dimensión ajustable.
* Función para eliminar la cerradura.
* **[10%]** Implementar funciones para validar una regla de apertura sobre una cerradura | **[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.

Módulo FuncionesComprobación.

* Función que verifica si una cerradura se abre con una clave dada.
* Función que realiza los cambios a la cerradura.

**Detección de Problemas**

Durante el desarrollo del parcial se afrontaron los siguientes problemas:

1. Al momento de crear la cerradura es necesario conocer cuantas comparaciones se van a realizar, estas últimas se almacenarán en un arreglo estático cuya cantidad de elementos se debe definir previamente. El obtener la cantidad de elementos de este arreglo estático representa un problema debido a que encontrar una forma de saber cuántas comparaciones se llevarán a cabo sin preguntarle directamente al usuario y al mismo tiempo utilizar la cantidad de memoria justa es complicado.

Una posible solución es crear un arreglo dinámico previamente definido con un numero considerable de espacios disponibles, pedirle al usuario las comparaciones (hasta que se ingrese un número diferente de 1, -1 y 0) y al mismo tiempo contarlas, después se crea un nuevo arreglo dinámico con la cantidad exacta.