**DESAFIO 1**

**Informática II**

**Ethan Salomon Parra Reyes**

**Juan Felipe García Bonilla**

**Universidad de Antioquia**

**Medellín**

**2024**

**Recreación del ejemplo propuesto en el Desafio1.**

Imagen que contiene Forma

Descripción generada automáticamente

Un reloj digital

Descripción generada automáticamente con confianza media

Calendario

Descripción generada automáticamente

**Análisis.**

* Se debe implementar un sistema para gestionar la alineación y rotación de las cerraduras X, permitiendo alinear estructuras de diferentes tamaños y realizar rotaciones independientes.
* Se debe desarrollar un algoritmo que pueda validar la regla K en función de las coordenadas y valores de las celdas dentro de las estructuras alineadas. Esto puede requerir un enfoque recursivo o iterativo para evaluar todas las combinaciones posibles de rotaciones.
* Dado que el proceso de apertura implica la rotación de múltiples estructuras, se debe prestar atención a la eficiencia del algoritmo de rotación para minimizar el tiempo de procesamiento.
* Es importante implementar mecanismos para manejar situaciones inesperadas, como entradas de reglas K inválidas o condiciones de cerraduras que no pueden cumplir con la regla especificada.

C**onsideraciones a tener en cuenta para el desarrollo del problema.**

1. Las matrices rotan en sentido antihorario.
2. Los números dentro de la matriz cumplen una secuencia.
3. Para la matriz siguiente, si varía el tamaño de la matriz, varía la posición con respecto a las filas y columnas de la matriz anterior.
4. La fila y columna ingresada en la llave k nos determinara el tamaño mínimo que pueden tener las matrices.
5. Nos encontramos con diferentes combinaciones que cumplen los parámetros dados para llegar a una solución del problema.
6. El mínimo tamaño de las matrices es de 3x3.
7. Dependiendo de los datos ingresados en la llave k, se podrá deducir si es necesario girar la matriz inicial.
8. El usuario no podrá ingresar la fila y columna que lo situé en la mitad de la matriz.
9. El espacio vacío en la matriz se denotara con el número 0 para efectos prácticos del código.
10. Si la matriz es de 5x5 y la siguiente matriz aumenta de tamaño a una de 7x7, y nos encontrábamos en la fila (f) y columna (c), cuando se alineen las dos matrices, la fila y columna en la que estaremos en la matriz de 7x7 será [f+1][c+1] respectivamente, pero si aumenta a una de 9x9, la fila y columna será [f+2][c+2], seguirá aumentado la posición si la matriz sigue creciendo.

**Esquema de tareas para el desarrollo.**

1. Se lee la llave ingresada.

* Con base en los dos primeros números ingresados en la llave se determina el tamaño mínimo posible de las matrices de prueba.

1. Se hace una función que genere una matriz de tamaño impar nxn formada por la secuencia de números especificada.
2. Se hace una función que rotara la matriz nxn en sentido antihorario a conveniencia.
3. Partiendo de lo anterior, habrá un proceso iterativo en el cual se buscará cumplir con las características brindadas por el usuario en la llave.
4. Validar que todas las condiciones brindadas se cumplen, y darle el resultado al usuario.

* Si se determina que la llave tiene una cerradura valida, se imprimirá la cerradura valida y las rotaciones realizadas para que se cumpla la llave.
* Si se determina que la llave no tiene una cerradura valida, termina el programa y se imprime un comentario que exprese que no se encontró una cerradura valida.

1. Se crea un módulo que tendrá dos opciones, validará una cerradura y una llave ingresada por el usuario o validar la cerradura obtenida en el módulo que retorna una cerradura válida para la llave ingresada, este imprimirá las matrices que están en la cerradura y rotadas si es pertinente.
2. Se crea una interfaz que sea más amigable con el usuario.

**Algoritmos implementados.**

**Módulo de funciones.**

El código implementa cuatro funciones de manejo, generar matriz, rotar matriz, imprimir matriz y liberar la memoria asignada dinámicamente.

1. **generarMatriz(int n):**

Esta función genera una matriz cuadrada e impar con números consecutivos, dejando el centro de la matriz vacío.

* Se verifica si el tamaño de la matriz es impar. Si no lo es, se imprime un mensaje de error y se devuelve un puntero nulo.
* Se crea una matriz dinámica de tamaño n x n.
* Se recorren todas las filas y columnas de la matriz, excepto el centro, asignando valores consecutivos al elemento correspondiente.
* Finalmente, se devuelve el puntero a la matriz.

1. **rotarMatriz(int matriz, int n):**

Esta función rota la matriz en sentido antihorario. El proceso de rotación se realiza en capas concéntricas:

* Se itera a través de las capas de la matriz, comenzando desde el borde exterior y avanzando hacia el centro.
* En cada capa, se intercambian los valores de los elementos correspondientes para realizar la rotación.

1. **imprimirMatriz(int matriz, int n):**

Esta función imprime la matriz en la consola. Simplemente recorre todas las filas y columnas de la matriz e imprime cada elemento.

1. **liberarMatriz(int matriz, int n):**

Esta función libera la memoria asignada dinámicamente para la matriz. Recorre todas las filas de la matriz y elimina los arreglos dinámicos asociados a cada fila, luego libera el arreglo de punteros principal que apunta a las filas de la matriz.

**Módulo de configuración llave.**

El código contiene dos funciones llamadas configuracionKey, que son responsables de solicitar al usuario la configuración de una llave para el programa.

1. **configuracionKey(int compare, int &minSizeMat):**

Esta función solicita al usuario la configuración de una llave, incluyendo la fila y columna inicial, así como los valores comparativos.

* Primero, solicita al usuario que ingrese una fila mayor o igual a 1. Verifica que la entrada sea un número válido. Luego, se almacena el valor ingresado en la variable k.
* A continuación, solicita al usuario que ingrese una columna mayor o igual a 1 de manera similar.
* Después, se solicita al usuario que ingrese los valores comparativos para cada etapa, y se almacenan en el arreglo llave, para finalmente retornarlo.

1. **configuracionKey(int compare):**

Esta versión de la función configuracionKey es similar a la anterior, pero no incluye el parámetro minSizeMat.

* Al igual que la función anterior, solicita al usuario la fila y columna inicial, así como los valores comparativos para cada etapa.
* Verifica la validez de las entradas del usuario y almacena los valores en el arreglo llave, para finalmente retornarlo.

**Módulo generar una cerradura valida.**

El código implementa una función llamada lockFinder que busca una cerradura en una matriz generada según una llave ingresada por el usuario.

1. **Operaciones:**

* Se inicializa la matriz y se verifica si es necesario rotarla inicialmente según la llave ingresada.
* Se inicia un bucle principal que recorre cada elemento de la llave.
* Dentro del bucle principal, se determina el tipo de operación a realizar.
* Se comprueba si se ha encontrado una cerradura válida.
* Se almacena el tamaño de la matriz actual en el arreglo lockX.
* Se libera la memoria asignada para la matriz generada.
* Se actualiza el arreglo de rotaciones.
* Finalmente, se devuelve el arreglo lockX.

**Módulo de validación.**

El código proporcionado incluye tres funciones que forman parte del proceso de encontrar y verificar una cerradura en una matriz generada a partir de una llave ingresada por el usuario.

1. **valid:**

Esta función verifica si una cerradura encontrada es válida según la llave ingresada y la matriz generada.

* Se genera la matriz inicial y se imprime su contenido junto con el valor del pin en las coordenadas especificadas.
* A continuación, se itera a través de los elementos de la llave para verificar cada paso de rotación.
* En cada paso, se genera una nueva matriz con el tamaño correspondiente, se realiza el número de rotaciones necesario y se compara el valor del pin antes y después de las rotaciones.
* Dependiendo del tipo de rotación especificada en la llave, se compara el valor del pin y se retorna false si no se cumple la condición requerida.
* Finalmente, se retorna true si todas las condiciones se cumplen.

1. **valid (sobrecargada):**

Esta versión de la función valid tiene un parámetro adicional rotaciones, que indica la cantidad de rotaciones realizadas en cada etapa según la llave.

* La función sigue el mismo proceso que la versión anterior, pero utiliza las rotaciones proporcionadas en lugar de realizarlas internamente.
* En cada iteración, se realiza el número de rotaciones especificado en rotaciones y se verifica la validez de la cerradura de la misma manera que en la función anterior.

1. **cerradura:**

Esta función solicita al usuario que ingrese los tamaños de las matrices en cada etapa para generar la cerradura, para después retornar esta misma.

**Problemas de desarrollo que afrontó.**

1. Desafíos en la implementación y la utilización de repositorios.
2. Desafíos en el manejo de excepciones.
3. Desafíos en la elaboración de un programa eficiente y modular.
4. Dificultades para detectar llaves que no tienen una cerradura viable.
5. Dificultades para establecer condiciones que optimicen la eficiencia del programa y para identificar llaves sin una cerradura posible.

**Evolución de la solución y consideraciones para tener en cuenta en la implementación.**

* Se han desarrollado funciones para el manejo de estructuras de datos, incluyendo la creación de matrices cuadradas impares, su rotación en sentido antihorario y la gestión dinámica de memoria para su liberación.
* Además, se ha implementado un módulo que permite al usuario ingresar una llave. En este módulo, se ha aplicado el manejo de excepciones para evitar que el usuario ingrese valores fuera de los parámetros requeridos por el programa.
* La matriz inicial generada está determinada por un tamaño mínimo, el cual se calcula al comparar la fila y columna ingresadas por el usuario en la llave. Se ha establecido una restricción para evitar que el usuario seleccione una fila y columna que lo posicione en la primera fila y la columna n-1, a menos que el valor de comparación inicial sea 1.
* Se ha observado que los usuarios no pueden seleccionar la fila 2 y la columna 2, ya que esto generaría una matriz de 3x3 y el elemento seleccionado estaría en el centro de la matriz.
* Inicialmente, se emplea una matriz del tamaño mínimo junto con las coordenadas ingresadas, y luego se genera otra matriz con las mismas características y se rota. Sin embargo, se ha descubierto que la matriz puede cambiar de tamaño para cumplir con las condiciones requeridas, pero no puede disminuir más de una matriz 3x3.
* Cada vez que la matriz de comparación cambia de tamaño, las coordenadas del elemento seleccionado se ajustan en función de la diferencia entre el tamaño actual y el anterior de la matriz.
* Se ha determinado que si el usuario ingresa 0 como valor de comparación, es más eficiente guardar el tamaño de la matriz, así como la fila y la columna para generar la misma matriz de tamaño nxn en la fila y columna especificadas.
* Además, se ha verificado que si, después de varias comparaciones, nos encontramos en un valor menor que n y el siguiente valor de comparación es 1, no se podrá satisfacer la llave ingresada.
* Para mejorar la eficiencia del programa, se ha descubierto que si el valor de comparación anterior es -1 o 0 y el nuevo valor de comparación es -1, se deben guardar el tamaño de la matriz y las coordenadas de la fila y la columna especificadas.
* Se implementó la opción de verificar la cerradura recién generada y se cambió el interfaz del programa para además de generar una cerradura, validar una llave y una cerradura ingresada por el usuario.
* Se observó que el código tenia posibles fugas de memoria, para solucionar esto se liberara la matriz antes de generar una nueva.