UNIVERSIDAD DE ANTIOQUIA
Facultad de Ingeniería

Informática II
Semestre 2024 – 1
Desafío 1. Marzo 21 de 2024

NOMBRES:

Alejandro Naranjo Naranjo
Juan Pablo R.

Puntos del informe

a. Análisis del problema y consideraciones para la alternativa de solución propuesta.
b. Esquema donde describa las tareas que usted definió en el desarrollo de los algoritmos.
c. Algoritmos implementados.
d. Problemas de desarrollo que afrontó.
e. Evolución de la solución y consideraciones para tener en cuenta en la implementación.

A) ANÁLISIS DEL PROBLEMA Y CONSIDERACIONES PARA LA ALTERNATIVA DE SOLUCIÓN Y PROPUESTA:

Por objetivos generales, lo primero que pensamos fue en como hacer el cambio de las posiciones, ya que todas las matrices están alineadas por el centro y son de distinto tamaño, por lo que la posición dada en la matriz 1 no iba a ser igual al de la matriz 2, 3, 4, etc. Luego, nos enfocamos en las rotaciones de cada una de las matrices que mas adelante se explicara con más detalle y detenimiento.

Luego, otro de los retos que se nos presentaba, era que matrices íbamos a utilizar en el candado, es decir, de que tamaño, ya que estas pueden ser iguales, mayores o menores según sea el caso.

y llegamos a la conclusión de que esto era según las condiciones nos lo exigían, es decir, si por ejemplo en las condiciones del ejemplo del desafío, que tenemos que el valor en la matriz A debe ser mayor que el valor en la matriz B, pues por lógica tendremos que usar una matriz mas grande para la matriz A, por ejemplo, una 9x9, y en la matriz B una 3x3 o 5x5, ya que la 9x9 tiene valores mas grandes que la 5x5. Y así sucesivamente hasta examinar todas las condiciones que se nos revelan en la llave.

Ahora, la reserva de memoria de las matrices, pensamos en hacer uso del triple puntero, en el cual este, va a reservar los doble punteros de las matrices a través de un bucle for.

Cabe aclarar, que estas ideas pueden ir cambiando debido a que contamos con otra semana para seguir con el desafío, por lo que estas ideas podrían mejorar o cambiar.

B) ESQUEMA DONDE DESCRIBA LAS TAREAS QUE USTED DEFINIÓ EN EL DESARROLLODE LOS ALGORITMOS.

1. Cuatro funciones para la matriz (neutra, 1, 2, 3,)
2. Una función que busque los valores dentro de las matrices o sea un valor en específico
3. Una función que reciba los valores (A, B, C Y D) y los compare para ver si pueden abrir el candado
4. Una función que almacene los posibles valores de x
5. Una función que verifique que los valores de k pueden abrir el candado o sea x

C) ALGORITMOS IMPLEMENTADOS:

Implementamos varios algoritmos, como el movimiento o rotación de las matrices:

```
53
54  void rotar(int **puntero_matriz, int numero_filas){
55
56      //transponer matriz:
57      for(int i=0; i<numero_filas; i++){
58          for(int j=i+1; j<numero_filas; j++){
59              int temp = *(*(puntero_matriz + i) + j);
60              *(*(puntero_matriz + i) + j) = *(*(puntero_matriz + j) + i);
61              *(*(puntero_matriz + j) + i) = temp;        }
62      }
63
64      for(int i=0; i<numero_filas/2; i++){
65          swap(*(puntero_matriz + i), *(puntero_matriz + numero_filas - i - 1));
66      }
67
68      for(int i=0; i<numero_filas; i++){
69          for(int j=0; j<numero_filas; j++){
70              if(*(*(puntero_matriz+i)+j)>=10){
71                  cout<<*(*(puntero_matriz+i)+j)<<"    ";
72
73              }
74              else{
75                  cout<<*(*(puntero_matriz+i)+j)<<"     ";
76
77              }
78          }
79          cout<<endl<<endl<<endl;
80      }
81  }
82
```

también, los que rellenan la matriz:

```
21
22  void RellenarMatriz(int **puntero_matriz, int numero_filas){
23      int contador=1;
24      for(int i=0; i<numero_filas ; i++){
25          for(int j=0; j<numero_filas; j++){
26              if((i==numero_filas/2) && (j==numero_filas/2)){
27                  //rellenar la matriz central con 0
28                  *(*(puntero_matriz+i)+j)= 0;
29              }
30              else{
31                  *(*(puntero_matriz+i)+j)= contador++;
32              }
33          }
34      }
35  }
36
```

Para los límites de las matrices:

```
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
      }
      else{
         return false; //la posicion dada no es compatible en la segunda matriz
      }
}
```

Para definir cuáles son las posiciones alineadas dependiendo de si es mayor o menor la siguiente matriz:

```
fila_segunda= fila_primera + (tamaño_segunda - tamaño_primera)/2.

columna_segunda= columna_primera + (tamaño_segunda - tamaño_primera)/2.
```

```
fila_segunda= (fila_primera - (tamaño_primera - tamaño_segunda)/2)

columna_segunda= (columna_primera - (tamaño_primera - tamaño_segunda)/2)
```

Estas son algunas de los códigos generales que hemos venido haciendo, y como se entenderá, aun nos queda mucho código por programar antes del 5 de abril

D) PROBLEMAS DE DESARROLLO QUE SE AFRONTARON y
E) EVOLUCION DE LA SOLUCION Y CONSIDERACIONES PARA TENER EN CUENTA EN LA IMPLEMENTACION:

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”.

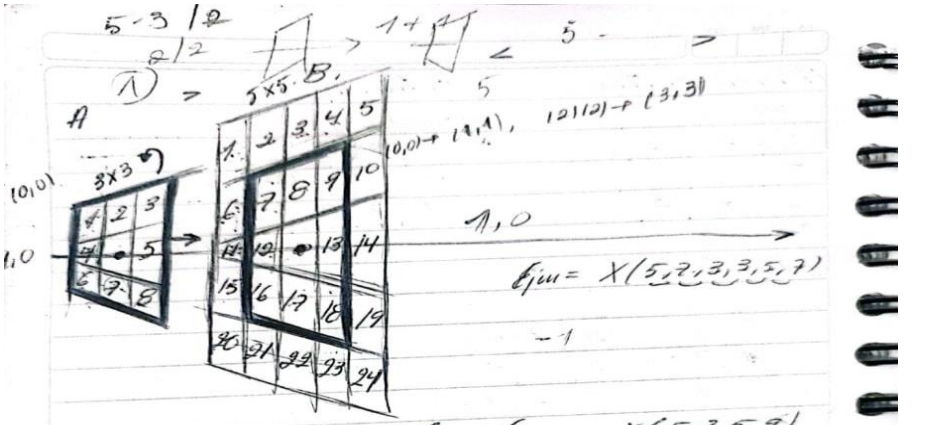
Para saber cuántas estructuras o matrices debe tener el “candado”, tenemos que ver cuantas condiciones nos dan en la llave, es decir, por ejemplo, en la llave que nos dan: K(4,3,1,-1,1): Como nos dicen, las 2 primeras revelan la primera matriz, y las otras 3, nos dicen las condiciones que deben de tener para que este “candado” se abra.
Por lo cual, el candado va a tener: #condiciones + 1, es decir:
número 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 cuáles 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:



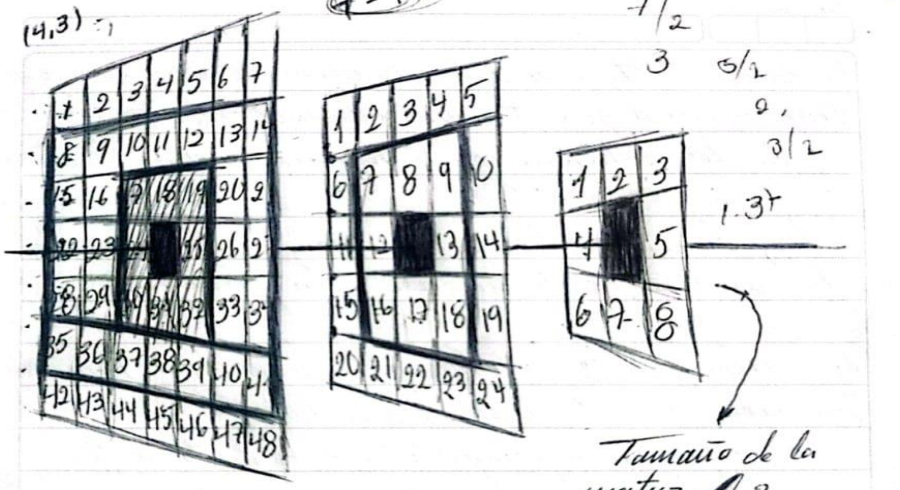
Ejm = Tenemos una Cerradura X(5,7,5,9)
* 4 Estructuras Alineadas, de 5x5, 7x7, 5x5, 9x9.

Una posible entrada K a X(5,7,5,9) puede ser
X(4, 3, 1, -1, -1)

Posición Fila primera matriz

Posición Columna primera Matriz.

Matriz1[4][3] > Matriz2[Posición Alineada].
Matriz2[P. Alineada] < Matriz3[P. Alineada]
Matriz3[P. Alineada] > Matriz4[P. Alineada].



[4][3] Matriz 1 = 31.
→ [3][2] Matriz 2 = 17
→ [2][1] Matriz 3 = 7.

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:

```
3
4  bool esCompatible(int fila, int columna, int tamano_primera, int tamano_segunda){
5      // Verificar que la posición dada esté dentro de los límites de la segunda matriz
6      if (fila >= 0 && columna >= 0 && fila < tamano_segunda && columna< tamano_segunda) {
7          return true; // La posición dada en la primera matriz es compatible con la segunda matriz
8      }
9      else{
10         return false; //la posicion dada no es compatible en la segunda matriz
11     }
12 }
13
```

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:

```
15
16 fila_segunda= fila_primera + (tamaño_segunda - tamaño_primera)/2.
17
18 columna_segunda= columna_primera + (tamaño_segunda - tamaño_primera)/2.
19
```

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:

```
23
24 fila_segunda= (fila_primera - (tamaño_primera - tamaño_segunda)/2)
25
26 columna_segunda= (columna_primera - (tamaño_primera - tamaño_segunda)/2)
27
28
```

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:

```
3
4  bool esCompatible(int fila, int columna, int tamano_primera, int tamano_segunda){
5      // Verificar que la posición dada esté dentro de los límites de la segunda matriz
6      if (fila >= 0 && columna >= 0 && fila < tamano_segunda && columna< tamano_segunda) {
7              return true; // La posición dada en la primera matriz es compatible con la segunda matriz
8      }
9      else{
10         return false; //la posicion dada no es compatible en la segunda matriz
11     }
12 }
13
```

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.

Por ultimo y creo, lo más importante, cual seria el tamaño de las matrices:

Ya que como lo dijimos en el item A, en el candado pueden ser utilizadas varias matrices de distintos tamaños, solo con la única condición de que fueran impares, pero estas podrían ser iguales, mayores o menores según se diera el caso.

y llegamos a la conclusión de que esto era según las condiciones nos lo exigían, es decir, si por ejemplo en las condiciones del ejemplo del desafío, que tenemos que el valor en la matriz A debe ser mayor que el valor en la matriz B, pues por lógica tendremos que usar una matriz más grande para la matriz A, por ejemplo, una 9x9, y en la matriz B una 3x3 o 5x5, ya que la 9x9 tiene valores más grandes que la 5x5. Y así sucesivamente hasta examinar todas las condiciones que se nos revelan en la llave.

Este tamaño es importante, sobre todo en la creación de la memoria dinámica que sería utilizada en cada uno de los doble punteros que se necesitan para cada matriz.
esta reserva se hace en una función y al final, se hace la liberación de esta memoria dinámica.

CABE ACLARAR QUE, COMO LO DIJE ANTES, ESTE NO ES LA ENTREGA FINAL DEL DESAFIO Y/O PROYECTO, POR LO QUE SE PUEDEN PRESENTAR CAMBIOS.

Muchas Gracias!!!.