Proyecto Final
Estructuras de Datos, 2023-2
Departamento de Electrónica y Ciencias de la Computación
Pontificia Universidad Javeriana Cali

Profesor:
Carlos Alberto Ramírez Restrepo
carlosalbertoramirez@javerianacali.edu.co

El presente proyecto tiene como objetivo enfrentar a los estudiantes del curso:

- Al diseño e implementación de Tipos Abstractos de Datos.
- Al análisis de la complejidad computacional de la implementación de operaciones primitivas de un Tipo Abstractos de Datos.
- Al análisis y comparación en términos de complejidad computacional de diferentes estrategias de representación para un mismo Tipo Abstracto de Datos.
- Al diseño e implementación de operaciones generales sobre un Tipo Abstracto de Datos.
- Al uso de la biblioteca STL del lenguaje de programación C++ y al análisis de sus ventajas en términos de eficiencia, practicidad y generalidad con respecto a estructuras y funciones definidas directamente.
- Al uso de Tipos Abstractos de Datos para la solución de problemas computacionales.

Contexto General

Una matriz dispersa corresponde a un tipo de matriz de gran tamaño en la cuál la cantidad de información relevante es baja en comparación con las dimensiones de la matriz. Típicamente, en este tipo de matrices la información no relevante se representa por el valor 0 (cero) y la información relevante por valores diferentes a 0. De esta manera, en este tipo de matrices es posible predecir que el porcentaje de ceros es alto. En general, considerar si una matriz es dispersa o no, osea, si la cantidad de ceros en la matriz es lo suficientemente alta, depende del contexto específico en el cuál se trabaje.

Las matrices son estructuras de datos que en general son muy eficientes en cuanto al uso de memoria y de procesador. Sin embargo, la representación computacional de las matrices dispersas mediante el enfoque general tiene como consecuencia un uso ineficiente de la memoria y del tiempo de procesamiento. Esto se debe a que se utilizan muchas posiciones de memoria para almacenar ceros y se requiere procesar todas las posiciones de la matriz innecesariamente.

En consecuencia, teniendo como objetivo mejorar la complejidad espacial y temporal de las operaciones y de la implementación específica de matrices dispersas, se han propuesto diversas representaciones. De esta manera, las matrices dispersas pueden ser definidas y especificadas como un TAD. El propósito de dichas representaciones es almacenar únicamente la información relevante de cada fila y columna de la matriz original con el propósito de utilizar menos memoria y agilizar las operaciones. En la siguiente sección se describen los principales enfoques de representación de matrices dispersas.

Representaciones

En las siguientes secciones se describirán los enfoques de representación más comúnes para matrices dispersas. En adelante, se utilizará `n` y `m` para denotar el número de filas y columnas de la matriz completa, en tanto que se utilizará `ne` para denotar el número de elementos con valor diferente a cero en la matriz completa. Además, cada una de las representaciones será ilustrada utilizando como referencia la matriz dispersa de la Tabla 1. En dicha matriz, se tiene que `n = 8`, `m = 7`, `ne = 13`.

0	2	0	0	0	0	4
0	8	9	0	0	1	0
0	0	0	3	0	0	0
0	0	0	0	0	0	0
5	0	0	0	0	6	0
1	2	0	0	0	0	0
4	0	0	0	0	0	0
0	0	7	0	0	11	0

Table 1: Ejemplo Matriz Dispersa

Formato Coordenado La representación de matrices dispersas mediante el formato coordenado almacena los datos distintos de cero de la matriz dispersa junto con su ubicación en la matriz, i.e., el índice de su fila y su columna. Para esto se utilizan 3 estructuras secuenciales (arreglos, vectores, listas, etc.) de tamaño `ne`. En el primer vector se almacen los valores distintos de cero de la matriz original mientras que en el segundo y tercer vector se almacenan los índices de la fila y la columna para cada uno de los valores distintos de cero.

Los siguientes serían los valores en las estructuras internas para la representación en formato coordenado de la matriz dispersa de la Tabla 1:

```
valores = [2 4 8 9 1 3 5 6 1 2 4 7 11]
  filas = [0 0 1 1 1 2 4 4 5 5 6 7 7]
columnas = [1 6 1 2 5 3 0 5 0 1 0 2 5]
```

Formato Comprimido por Fila El formato comprimido por fila (Compressed Sparse Row CSR) representa una matriz dispersa por medio de 3 estructuras secuenciales (arreglos, vectores, listas, etc.). La primera estructura tiene tamaño `ne` y almacena los valores distintos de cero de la matriz original organizados fila por fila. En la segunda estructura, que también tiene tamaño `ne`, se almacenan los índices de las columnas en las que están cada uno de los valores del primer vector en la matriz original. En tanto que, la tercera estructura tiene tamaño `n + 1` y almacena la posición donde empiezan los valores de cada fila en la segunda estructura (estructura de columnas).

Los siguientes serían los valores en las estructuras internas para la representación en el formato CSR para la matriz de la Tabla 1:

```
valores = [2 4 8 9 1 3 5 6 1 2 4 7 11]
   cols = [1 6 1 2 5 3 0 5 0 1 0 2 5]
 cfilas = [0 2 5 6 6 8 10 11 13]
```

Observe que la tercera estructura (`cfilas`) se puede interpretar asociando el valor `cfilas[i]` y el valor `cfilas[i + 1] - 1` como el índice inicial y el índice final en estructura de columnas asociadas a la fila `i`. De esta manera, para el ejemplo anterior se tiene:

Fila	Límites	Columnas asociadas	Fila	Límites	Columnas asociadas
0	`cfilas[0]` a `cfilas[1]-1`	`cols[0]`, `cols[1]`	4	`cfilas[4]` a `cfilas[5]-1`	`cols[6]`, `cols[7]`
1	`cfilas[1]` a `cfilas[2]-1`	`cols[2]`, `cols[3]`, `cols[4]`	5	`cfilas[5]` a `cfilas[6]-1`	`cols[8]`, `cols[9]`
2	`cfilas[2]` a `cfilas[3]-1`	`cols[5]`	6	`cfilas[6]` a `cfilas[7]-1`	`cols[10]`
3	`cfilas[3]` a `cfilas[4]-1`	-	7	`cfilas[7]` a `cfilas[8]-1`	`cols[11]`, `cols[12]`

Es importante resaltar que en esta representación cuando una fila no contiene ningún valor distinto a cero entonces el valor asociado a dicha fila en la tercera estructura (`cfilas`) es el mismo de la fila anterior. Además, el último valor en la tercera estructura corresponde al último índice válido en el segunda estructura más 1.

Formato Comprimido por Columna El formato comprimido por columna (Compressed Sparse Column CSC) también representa una matriz dispersa por medio de 3 estructuras secuenciales (arreglos, vectores, listas, etc.). Este formato es análogo al formato CSR. La primera estructura tiene tamaño `ne` y almacena los valores no nulos en la matriz original organizados por columna. La segunda estructura tiene tamaño `ne` y almacen los índices de las filas en las que están cada uno de los valores de la primera estructura. Mientras que la tercera estructura tiene tamaño `m + 1` y almacena la posición donde empiezan los valores de cada columna en la segunda estructura (estructura de filas).

Los siguientes serían los valores en las estructuras internas para la representación en el formato CSC para la matriz de la Tabla 1:

```
valores = [5 1 4 2 8 2 9 7 3 1 6 11 4]
  filas = [4 5 6 0 1 5 1 7 2 1 4 7 0]
    ccolumnas = [0 3 6 8 9 9 12 13]
```

Observe que el tercer vector (vector `ccolumnas`) se puede interpretar asociando el valor c_i y el valor $c_{i+1} - 1$ como el índice inicial y el índice final en el vector de filas asociados a la columna i. De esta manera, para el ejemplo anterior se tiene:

Fila	Límites	Columnas asociadas	Fila	Límites	Columnas asociadas
0	`ccolumnas[0]` a `ccolumnas[1]-1`	`filas[0]`, `filas[1]` `filas[2]`	4	`ccolumnas[4]` a `ccolumnas[5]-1`	-
1	`ccolumnas[1]` a `ccolumnas[2]-1`	`filas[3]`, `filas[4]`, `filas[5]`	5	`ccolumnas[5]` a `ccolumnas[6]-1`	`filas[9]`, `filas[10]` `filas[11]`
2	`ccolumnas[2]` a `ccolumnas[3]-1`	`filas[6]`, `filas[7]`	6	`ccolumnas[6]` a `ccolumnas[7]-1`	`filas[12]`
3	`ccolumnas[3]` a `ccolumnas[4]-1`	`filas[8]`			

Como en la representación CSR, cuando una columna no contiene ningún valor distinto a cero entonces el valor asociado a dicha columna en la tercera estructura (`ccolumnas`) es el mismo de la columna anterior. Además, el último valor en la tercera estructura corresponde al último índice válido en el segunda estructura más 1.

Listas Enlazadas por Fila Una matriz dispersa puede ser representada mediante listas enlazadas. De esta manera, en esta representación se tiene una estructura secuencial que contiene listas enlazadas. Cada una de las listas enalazadas representa los elementos en una fila o en una columna de la matriz original.

La Figura 1 muestra la representación mediante listas enlazadas de la matriz dispersa de la Tabla 1. En la figura se asume que se utilizan listas doblemente enlazadas aunque también es posible utilizar listas simples o listas circulares. Observe que las filas que no tienen valores distintos a cero son representadas mediante listas vacías. En cada nodo de la lista se almacena un valor distinto de cero de la matriz original y el índice de la columna en donde está ubicado dicho valor. Finalmente, es posible construir una representación análoga en la que cada lista enlazada esté asociada a cada columna en la matriz original.

En este proyecto usted tiene como tarea diseñar e implementar el tipo abstracto `DisperseMatrix` que permite almacenar matrices dispersas de una forma compacta y realizar operaciones sobre estas matrices. Para esto, usted deberá elegir alguna de las representaciones descritas anteriormente. En esta decisión deberá tener en cuenta elementos asociados a la complejidad de las operaciones al usar una representación u otra.

Por otro lado, en adición a lo anterior, usted deberá usar su implementación en la solución de un problema en la arena de programación.

Requerimientos

Operaciones Sobre el TAD `DisperseMatrix`

El TAD `DisperseMatriz` debe contener las siguientes operaciones:

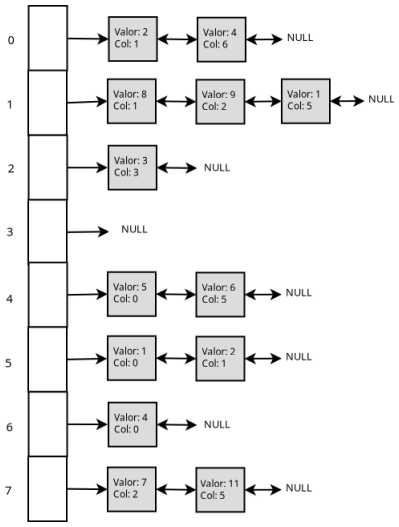

Figure 1: Representación Listas Enlazadas Matriz Tabla 1

1. Una operación constructora que tome como parámetro una matriz completa representada como un arreglo de dos dimensiones.
2. Una operación constructora que tome como parámetro una matriz completa representada como un vector de vectores.
3. Una operación constructora que tome como parámetro otra instancia de `DisperseMatrix` y copie sus valores en el objeto que se está creando.
4. La operación `rebuild` que reconstruya la matriz completa asociada a la instancia de una matriz dispersa.
5. La operación `get` que permita obtener el valor que está en la posición i, j en la matriz. Si la posición i, j corresponde a un cero, este valor debe ser retornado.
6. La operación `getRow` que permite obtener la fila j de la matriz dispersa como un vector o como una lista enlazada (se deben considerar ambas variantes).
7. La operación `getCol` que permite obtener la columna j de la matriz dispersa como un vector o como una lista enlazada (se deben considerar ambas variantes).
8. La operación `getDisperseRow` que permite obtener la fila j de la matriz dispersa incluyendo los ceros como un vector o como una lista enlazada (se deben considerar ambas variantes).
9. La operación `getDisperseCol` que permite obtener la columna j de la matriz dispersa incluyendo los ceros como un vector o como una lista enlazada (se deben considerar ambas variantes).
10. La operación `assign` que permite modificar el valor en la posición i, j de la matriz. Es importante resaltar que si el valor anterior es un cero en la matriz original o si el nuevo valor es un cero, se debe modificar la estructura de la matriz dispersa para incluir el nuevo valor o para eliminar la posición que pasa a contener un cero.
11. La operación `add` que recibe otra matriz dispersa de iguales dimensiones y le suma al objeto actual la matriz que se recibe como parámetro.
12. La operación `printMatrix` que recibe una cadena de separación e imprime cada fila de la matriz en una línea con los elementos separados por dicho caracter.
13. La operación `productVector` que recibe un vector y multiplica el objeto actual por el vector recibido.

14. Adicionalmente, es necesario sobrecargar los siguientes operadores binarios +, *,==. El operador + permite obtener la suma entre dos matrices y el operador * permite obtener la múltiplicación. Puede asumir que en ambos casos si las matrices no tienen las dimensiones apropiadas se llenará la matriz más pequeña con ceros para completarla. Estos operadores no deben modificar los objetos sobre los que son aplicados sino que deben producir un nuevo objeto en el caso de los operadores aritméticos y un valor booleano en los demás casos.
15. La operación `getMax` que debe retornar el mayor elemento en la matriz.
16. La operación `getTranspose` que construye la matriz dispersa correspondiente a la transpuesta del objeto actual.
17. La operación estática `addMatrixList` que toman una lista de instancias del TAD `DisperseMatrix` y retorna el resultado de sumar la lista de matrices recibidas.

Problema en la Arena de Programación

Además de la implementación del TAD `DisperseMatrix` con las operaciones solicitadas, en este proyecto se solicita resolver en la arena de programación uno entre los siguientes problemas:

- Having Fun with Matrices
- Flip-Flop the Squarelotron

Es importante tener en cuenta que es necesario usar el TAD `DisperseMatrix` para resolver el problema elegido. Cualquier solución que no use el TAD no será tenida en cuenta en la calificación.

Entrega

En el presente proyecto se espera que usted implemente el TAD `DisperseMatrix` como una clase en C++ utilizando y explotando las características de la **POO** y de STL.Más especificamente, cada grupo de trabajo debe:

1. Implementar el TAD `DisperseMatrix` con la representación en el formato de listas enlazadas por fila o columna y en uno de los otros formatos presentados en este documento.
2. Incluir en cada implementación todas las operaciones descritas en este documento. En su diseño e implementación debe utilizar apropiadamente los conceptos de POO (herencia, polimorfismo, sobrecarga de métodos, sobrecarga de operadores, etc).
3. Analizar la complejidad de cada operación. No es necesario que se analice detalladamente cada linea de cada operación pero se debe explicar claramente la complejidad que se reporte para cada una de ellas.
4. Explicar claramente las decisiones de implementación realizadas y probar cada una de las operaciones implementadas.

Informe

Se debe elaborar un documento tipo informe en un archivo PDF en el que se debe reportar y comparar las complejidades de las operaciones para cada implementación.

Aclaraciones

1. El proyecto se puede realizar en equipos de trabajo de **2 personas**.Cualquier indicio de copia causará la anulación del proyecto y la ejecución del proceso disciplinario que corresponda de acuerdo a la normativa de la universidad.
2. La entrega debe realizarse a través del servidor de Discord mediante un mensaje al profesor. Debe enviarse de forma comprimida una carpeta llamada `proyecto` y en esa carpeta deben haber subcarpetas para cada implementación. Cada subcarpeta debe contener el archivo `.h` y el archivo `.cpp` asociados a la implementación del TAD `DisperseMatrix`. Así mismo, se debe incluir un archivo llamado `main.cpp` en el que debe implementarse la solución al problema seleccionado mediante el uso del TAD.

3. Los criterios de evaluación del proyecto son los siguientes:
   - Implementaciones del TAD `DisperseMatrix`: 45%
   - Implementación solución problema arena de programación: 20%
   - Correcta utilización POO, abstracción y STL: 10%
   - Estilo de programación: 7.5%
   - Reporte Complejidades: 17.5%
4. El proyecto tendrá dos entregas. La primera entrega debe contener **el prototipo de la clase asociada al TAD `DisperseMatrix`** así como evidenciar avances en la implementación de algunas operaciones. Esta entrega debe ser enviada a más tardar el día **21 de Octubre a las 23:59**.

   La segunda entrega debe contener **todos los elementos adicionales** además de posibles mejoras que haya identificado sobre la primera entrega. La entrega final se debe enviar a más tardar el día **20 de Noviembre a las 8:00 a. m.**. La sustentación del proyecto se llevará a cabo los días **20 y 21 de Noviembre**.

   La nota final del proyecto dependerá de la sustentación de cada grupo de trabajo. Cada grupo de trabajo, después de la sustentación tendrá asignado un número real (el factor de multiplicación) entre 0 y 1, correspondiente al grado de calidad de la sustentación de ambos integrantes del grupo de trabajo. Su nota definitiva será la nota obtenida con las ponderaciones indicadas anteriormente, multiplicada por ese valor. Si su asignación es 1, su nota será la del proyecto. Pero si su asignación es 0.9, su nota será 0.9 por la nota del proyecto. La no asistencia a la sustentación tendrá como resultado una asignación de un factor de 0.

   Tenga muy en cuenta esta aclaración. El propósito de la sustentación es que ambos estudiantes en cada grupo de trabajo demuestren que elaboraron completamente y equitativamente el proyecto. Por esta razón, trabajen a conciencia y preparen muy bien su sustentación. Durante la sustentación se harán preguntas sobre los detalles del proyecto y es posible que sea necesario explicar sus implementaciones en el tablero.
5. Recuerde, en caso de dudas y aclaraciones puede preguntar al inicio de las clases, asistir al horario de monitoria o enviar mensajes con sus consultas al profesor.

A - Having Fun with Matrices

Source file name: `fun.cpp`
Time limit: x seconds

We have a matrix of size $N \times N$. Each value of the matrix occupies an integer from $[0, 9]$. A few operations are going to be performed on this matrix. We would like to know how the matrix looks like after these operations are performed sequentially.

There could be five different types of operations.

- `row` *a b*: In this operation, row *a* is interchanged with row *b*.
- `col` *a b*: In this operation, column *a* is interchanged with column *b*.
- `inc`: In this operation, every cell value is increased by 1 (modulo 10). That is, if after adding 1, a cell value becomes 10 we change it to 0.
- `dec`: In this operation, every cell value is decreased by 1 (modulo 10). That is, if after subtracting 1, a cell value becomes -1 we change it to 9.
- `transpose`: In this operation, we simply transpose the matrix. Transposing a matrix, denoted by A^T, means turning all the rows of the given matrix into columns and vice-versa.

  Example:

```
1 2 3                    1 4 7
4 5 6  after transposing 2 5 8
7 8 9                    3 6 9
```

Input

The input file starts with an integer T ($T < 50$) that indicates the number of test cases. Each case starts with a positive integer N ($N < 5000$) that represents the size of the matrix. The next N lines contain N integers each. The value of each integer is in the range $[0, 9]$. Next there is a line with an integer M ($M < 50$). Each of the next M lines contain an operation each. If the command is `row` *a b* or `col` *a b*, then you can assume $1 \leq a, b \leq N$ and $a \neq b$.

The input must be read from standard input.

Output

For each case, output the case number on the first line. Then on the next N lines output the content of the final matrix. Print a blank line after each case (even after the very last one).

The output must be written to standard output.

Sample Input	**Sample Output**
2 4 1234 5678 1234 5678 1 transpose 3 000 111 000 2 row 1 2 inc	Case #1 1515 2626 3737 4848 Case #2 222 111 111

B - Flip-Flop the Squarelotron

Source file name: `squarelotron.cpp`
Time limit: x seconds

Anita is the baby siter of Baron Von Hauser' kids, a famous Physics Assistant of ITESM Campus Monterrey Physics department. As such Von Hauser kids have weird toys, all of which Anita has to master to be able to effectively entertain Baron' Kids.

While Anita was cleaning the bathtub she found a new toy. It is extremely weird, and posses a lot of mathematical symmetry, it is a Squarelotron game. She is determined to understand this new toy, otherwise she won't be able to play with Von Hauser' kids. However the complexity of such extreme toy makes it dificult to play. Thats why Anita asked the judges of this ITESM Campus Monterrey internal ACM programming contest eliminatory to put this problem, so that answers could be given by the best sttudents of computer science and engineering of this Campus.

A Squarelotron consist basically of a matrix of numbers. However this matrix can be decomposed as squared rings which can rotate independently in 4 diferent ways, Upside-Down, Left-Rigt, through the Main Inverse Diagonal and through the Main Diagonal.

For example Consider the following Squarelotrons.

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	16

1	2	3	4	5
6	7	8	9	10
11	12	13	14	15
16	17	18	23	21
25	22	19	24	20

Squarelotron a) as 2 rings while squarelotron b) has 3.

A Upside-Down Flip of the outmost ring of Squarelotron a) yields:

13	14	15	16
9	6	7	12
5	10	11	8
1	2	3	4

A Left-Rigth Flip of the 2 ring of squarelotron b) yields:

1	2	3	4	5
6	9	8	7	10
11	14	13	12	15
16	23	18	17	21
25	22	19	24	20

A Flip through the Main Inverse Diagonal of the second ring of squarelotron a) yields:

1	2	3	4
5	11	7	8
9	10	6	12
13	14	15	16

A Flip through the Main Diagonal of the outermost ring of squarelotron b) yields:

1	6	11	16	25
2	7	8	9	22
3	12	13	14	19
4	17	18	23	24
5	10	15	21	20

Anita wants you to do a program which performs the following. She will give you a Squarelotron and your program will perform several of the flips described earlier for each of the rings of the given Squarelotron. The output of your program should be the final state of the Squarelotron.

Input

The first line contains the number M of different cases to process, consisting of blocks of lines. Each of these blocks consist of the following. The first line of each block contains a number N which describes the order $N \times N$ of the Squarelotron. Following comes N lines of N numbers each, which describes the Squarelotron itself. Next comes an equal number of lines as number of rings of the Squarelotron. Each of these lines begins with a number T, followed by T numbers C which identifies the move to perform on the ring. All these numbers are in the following ranges:

- $0 < M \leq 1000$
- $1 \leq N < 5001$
- $0 < T \leq$ Number of Rings
- $1 \leq C < 5$

The numbers in the squarelotron are smaller than 2^{16}. . The Moves are identifies as follows. `'1'` means Upside-Down Flip, `'2'` means Left-Right Flip, `'3'` means flip through the Main Diagonal, `'4'` means a flip through the Main Inverse Diagonal.

The input must be read from standard input.

Output

For each of the M cases, output a $N \times N$ squarelotron at the state it is supposed to be after all the moves. This squarelotron should be N lines with N numbers each. No blank line in between each case should be output.

The output must be written to standard output.

Sample Input	**Sample Output**
4 3 1 2 3 4 5 6 7 8 9 2 1 2 4 1 2 3 4 4 1 2 3 4 5 6 7 8 9 1 2 3 4 5 6 6 2 1 2 2 3 4 5 1 2 3 4 5 6 7 8 9 1 2 3 4 5 6 7 8 9 1 2 3 4 5 6 7 2 3 4 2 1 3 4 1 3 2 4 6 1 2 3 4 5 6 7 8 9 1 2 3 4 5 6 7 8 9 1 2 3 4 5 6 7 8 9 1 2 3 4 5 6 7 8 9 2 4 1 2 2 3 1 1	9 8 7 6 5 4 3 2 1 6 6 5 4 3 2 1 9 8 7 6 5 4 3 2 1 7 6 5 4 3 2 8 3 7 7 6 9 4 8 2 1 1 5 9 6 5 4 3 2 1 4 7 1 4 7 1 5 2 8 5 2 2 6 1 3 4 1 3 7 9 6 7 9 4 8 8 5 2 8 5 9 3 6 9 3 6