GUÍA DE APRENDIZAJE PARA CONCURSANTES ICPC Y IOI: MATRICES O ARREGLOS BIDIMENSIONALES

1. Introducción

En varios problemas debemos trabajar con varias informaciones de forma grupal y dichas informaciones tienen en común que comporten el mismo tipo de dato. Una variante bastante trivial es declarar un variable por cada información con que necesito trabajar. Por ejemplo si tengo el salario mensual de un trabajador durante un año con declarar 12 variables me sería suficiente para luego realizar un grupo de operaciones como el promedio salarial, el mínimo y máximo salario del trabajador. Estas operaciones aunque se pueden implementar no cabe duda que tienen su complejidad en cuanto a la implementación que puede ser un tanto tediosas. Bueno imaginemos que ahora tengamos que hacer ese mismo trabajo con un quinquenio o decada de trabajo del trabajador la complejidad de implementación aumentería casi que literal en cino o diez veces más.

Ya habíamos tratado en otras guías que la solución a estos era el trabajo con arreglo. Pero que pasaría si quiero almacenar el salario mensual de varios años de forma que pueda luego calcular de forma cómoda el promedio del salario anual. Veremos en esta guía que una forma muy comoda de organizar los datos bajo determinadas circunstancia es utilizando las **matrices**.

2. Conocimientos previos

2.1. Estructura de datos

Una estructura de datos es una forma de organizar un conjunto de datos elementales con el objetivo de facilitar su manipulación. Un dato elemental es la mínima información que se tiene en un sistema. Una estructura de datos define la organización e interrelación de estos y un conjunto de operaciones que se pueden realizar sobre ellos. Cada estructura ofrece ventajas y desventajas en relación a la simplicidad y eficiencia para la realización de cada operación. De está forma, la elección de la estructura de datos apropiada para cada problema depende de factores como la frecuencia y el orden en que se realiza cada operación sobre los datos.

2.2. Estructuras de datos estáticas compuestas

Las estructuras de datos estáticas son aquellas que el tamaño de las mismas no puede ser modificadas en la ejecución del algoritmo. Así su tamaño tiene que ser defino en la creación de la misma. Está estructura almacena varios elementos del mismo tipo en forma lineal.

2.3. Operador *new*

C++ y Java permite a los programadores controlar la asignación de memoria en un programa, para cualquier tipo integrado o definido por el usuario. Esto se conoce como administración dinámica de memoria y se lleva a cabo mediante el operador *new*. Podemos usar el operador *new* para asignar (reservar) en forma dinámica la cantidad exacta de memoria requerida para contener cada nombre en tiempo de ejecución. La asignación dinámica de memoria de esta forma hace que se cree un arreglo (o cualquier otro tipo integrado o definido por el usuario) en el almacenamiento libre (algunas veces conocido como el heap o montón): una región de memoria asignada a cada programa para almacenar los objetos que se asignan en forma dinámica. Una vez que se asigna la

memoria para un arreglo en el almacenamiento libre, podemos obtener acceso a éste si apuntamos un apuntador al primer elemento del arreglo.

2.4. Esctructura de repetición (bucles o ciclos)

Un **bucle** se utiliza para realizar un proceso repetidas veces. Se denomina también **lazo** o **loop**. El código incluido entre las llaves {} (opcionales si el proceso repetitivo consta de una sola línea), se ejecutará mientras se cumpla unas determinadas condiciones. Hay que prestar especial atención a los bucles infinitos, hecho que ocurre cuando la condición de finalizar el bucle (booleanExpression) no se llega a cumplir nunca. Se trata de un fallo muy típico, habitual sobre todo entre programadores poco experimentados. En el caso de los lenguajes de programación de C++ y Java se cuenta con varias instrucciones de tipo bucle o ciclo como son el **for**,**while** y **do ... while**

3. Desarrollo

Una matriz es una estructura de datos que consiste en filas y columnas. En otras palabras, tiene múltiples filas y columnas, cada una con más de dos elementos. Las intersecciones de filas y columnas se denominan celdas, y cada celda puede contener información simple o compleja. Se representa como ($n \times m$), donde n es el número de filas y m es el número de columnas.

Las matrices en programación pueden tener una o varias dimensiones, como unidimensional, multidimensional o escalonada. Se componen de un conjunto de números, símbolos, letras u otro contenido matemático organizado en filas y columnas. El número y la longitud de cada dimensión se establecen al crear la instancia de matriz y no se pueden cambiar durante la vigencia de la instancia. La característica principal de una matriz es que contiene filas y columnas, siendo las filas variables y las columnas casos.

En programación, existen varios tipos de matrices, entre los cuales se encuentran la rectangular, la fila, la columna, la nula, la cuadrada de orden n, la diagonal, la escalar, la identidad, la opuesta, la traspuesta, el triángulo superior y el triángulo inferior.

4. Implementación

4.1. C++

4.1.1. Declaración y creación de arreglos

Las matrices se pueden declarar según la situación. Ahora veremos cada una de ellas:

```
/* Sabemos la cantidad de elementos a priori, esta manera es estatica
<tipo_de _dato> <nombre_matriz> [<cantidadFila>][<cantidadColumna>];*/
bool mark[100][50];

/*La cantidad de elementos puede variar y depende del valor de una
variable.Es la mas usada y esta se conoce como dinamica
<tipo_de _dato> ** <nombre_matriz>= new * <tipo_de _dato> [<cantidadColumna>];
```

```
for(int i=0;i<cantidadColumna;i++){
   <nombre_matriz>[i] = new [nfilas];
}
*/
int ncolumns = 100;
int nfilas = 50;
int ** mat;
mat = new int * [ncolumns];
for(int i=0;i<ncolumns;i++)
   mat[i]=new int [nfilas];

/*Cuando conocemos los valores que integran la matriz*/
double carrots[3][4] {{2.5, 3.2, 3.7, 4.1},// primera fila
                      {4.1, 3.9, 1.6, 3.5},// segunda fila
                      {2.8, 2.3, 0.9, 1.1} // tercera fila
                     };
```

4.1.2. Almacenar valor en la matriz

Para almacenar un valor en la matriz solo debemos inidicar la posición en la matriz en que se va almacenar dicha posición esta definida por una fila y una columna debe ser un valor o variable entera y debe estar en rango $[0, fila - 1][0, columna - 1]$.

```
/* <nombre matriz>[<fila>][<columna>] = <valor>; donde:
<fila> puede ser una variable entera, valor entero literal, expresion cuyo
   resultado sea entero e indicada la fila a la que quiero acceder en la
   matriz
<columna> puede ser una variable entera, valor entero literal, expresion cuyo
   resultado sea entero e indicada la columna a la que quiero acceder en la
   matriz
<valor> puede ser una variable entera , valor entero literal, expresion cuyo
   resultado sea entero */
notas[2][3]=cantidadMaxima;
notas[cantidadMaxima-50][23+cantidadMaxima]=34;
notas[10+5][3]=2*cantidadMaxima;
```

4.1.3. Obtener valor en la matriz

Para obtener un valor en una matriz solo debemos inidicar la posición en la matriz en que se va almacenar dicha posición esta definida por una fila y una columna debe ser un valor o variable entera y debe estar en rango $[0, fila - 1][0, columna - 1]$.

```
/* <variable> = <nombre matriz>[<fila>][<columna>]; donde:
<fila> puede ser una variable entera, valor entero literal, expresion cuyo
   resultado sea entero e indicada la fila a la que quiero acceder en la
   matriz
```

```
<columna> puede ser una variable entera, valor entero literal, expresion cuyo
    resultado sea entero e indicada la columna a la que quiero acceder en la
    matriz
<variable> va ser la el lugar donde se almacenara una copia del valor
    solicitado al arreglo, debe ser del mismo tipo de dato del arreglo */
int a =notas[2][3];
int b;
b = notas[cantidadMaxima-50][a+1];
```

4.1.4. Recorrer todos los elementos de la matriz

Para recorrer todos los elementos o parte de los elementos almacenados en la matriz vamos utilizar dos instrucciones **for** de forma anidada cada una asociada a una de las dimensiones de la matriz es importante hacer notar que dependiendo de como se definan dichas intrucciones el orden y sentido de recorrido dentro de la matriz puede variar lo cual puede ser útil o engorroso dependiendo de la situación

```
/*for( int i=<fila_inicial>; i < <fila_final>; i++ ){
     for( int j=<columna_inicial>; j < <columna_final>; j++ ){

     }
}
i++ si <fila_inicial> <= <fila_final>
i-- si <fila_inicial> > <fila_final>
i <= <fila_final> si deseo incluir en el rango la ultima fila pero tiene que
    ser una fila valida de la matriz
j++ si <columna_inicial> <= <columna_final>
j-- si <columna_inicial> > <columna_final>
j <= <columna_final> si deseo incluir en el rango la ultima columna pero tiene
    que ser una columna valida de la matriz
*/
for(int i=0;i<cFilas;i++){
   for(int j=0;j<cColumnas;j++){
      //notas[i][j] accedo al valor almacenado en la i-nesima fila y j-nesima
         de la matriz
   }
}
```

4.1.5. Leer los valores y almacenarlos en la matriz

Para leer los valores consola y almacernalos directamente en la matriz vamos a utilizar una combinación de recorrer todos los elementos y almacenar en la matriz visto anteriormente

```
/* En cada iteracion de los fors el cin lee un valor y dicho valor es
   almacenado en la matriz en la posicion que indique el valor de las
   variable i y j en esa iteracion, en la primera el valor seria (0;0), en la
    segunda (0;1) y asi sucesivamente. Tener en cuenta que los valores de i
```

```
    esten en rango de posiciones validas de la filas y la j en el rango de
    posiciones validas de la columnas de la matriz */
for(int i=0;i<cFilas;i++){
    for(int j=0;j<cColumnas;j++){
        cin>>matriz[i][j];
    }
}
```

4.1.6. Imprimir los valores almacenados en el matriz

Para imprimir los valores en consola de la matriz vamos a utilizar una combinación de recorrer todos los elementos y obtener en una matriz visto anteriormente

```
/* En cada iteracion de los for el cout imprime un valor que se corresponde
    con el almacenado en la matriz en la posicion que indique el valor de las
    variables i (fila) y j (columna)  en esa iteracion, en la primera el valor
     seria (0;0), en la segunda (0;1) y asi sucesivamente. Tener en cuenta que
     los valores de i esten en rango de posiciones validas de la filas y la j
    en el rango de posiciones validas de la columnas de la matriz*/
for(int i=0;i<cFilas;i++){
    for(int j=0;j<cColumnas;j++){
        cout<<matriz[i][j]<<" ";
    }
}
cout<<endl;
for(int i=0;i<cFilas;i++){
        for(int j=0;j<cColumnas;j++){
                cout<<matriz[i][j]<<" ";
        }
    cout<<endl;
}
```

En primer bucle anidado se imprime todos los valores de la matriz en una sola linea separados por espacios, mientras en el segundo bucle anidado cada valor almacenado en la matriz se imprime acorde a la fila y columna que le corresponde en la matriz.

4.2. Java

4.2.1. Declaración y creación de matrices

La matriz se puede declarar según la situación. Ahora veremos cada una de ellas:

```
/*La cantidad de elementos puede variar y depende del
valor de dos variables (una para fila y la otra para columna).Es la mas usada
<tipo_de _dato> [] <nombre_matriz>= new <tipo_de _dato> [<cant_fila>][<
    cant_columna>];*/
int cantidadEvaluaciones=15;
```

```
int cantidadEstudiantes=35;
int [][] notas;
notas=new int [cantidadEstudiantes][cantidadEvaluaciones];
double [][] temperaturas=new double [12][31];

/*Cuando conocemos los valores que integran la matriz*/
String [][] nombres ={ {"Luis", "Ernesto", "Susana"}, //primera fila
                       {"Luisa", "Yinela", "Rosa"}, //segunda fila
                     };
```

4.2.2. Almacenar valor en la matriz

Para almacenar un valor en la matriz solo debemos inidicar la posición en la matriz en que se va almacenar dicha posición esta definida por una fila y una columna debe ser un valor o variable entera y debe estar en rango $[0, fila - 1][0, columna - 1]$.

```
/* <nombre matriz>[<fila>][<columna>] = <valor>; donde:
<fila> puede ser una variable entera, valor entero literal, expresion cuyo
   resultado sea entero e indicada la fila a la que quiero acceder en la
   matriz
<columna> puede ser una variable entera, valor entero literal, expresion cuyo
   resultado sea entero e indicada la columna a la que quiero acceder en la
   matriz
<valor> puede ser una variable entera , valor entero literal, expresion cuyo
   resultado sea entero */
notas[2][3]=cantidadMaxima;
notas[cantidadMaxima-50][23+cantidadMaxima]=34;
notas[10+5][3]=2*cantidadMaxima;
```

4.2.3. Obtener valor en la matriz

Para obtener un valor en una matriz solo debemos inidicar la posición en la matriz en que se va almacenar dicha posición esta definida por una fila y una columna debe ser un valor o variable entera y debe estar en rango $[0, fila - 1][0, columna - 1]$.

```
/* <variable> = <nombre matriz>[<fila>][<columna>]; donde:
<fila> puede ser una variable entera, valor entero literal, expresion cuyo
   resultado sea entero e indicada la fila a la que quiero acceder en la
   matriz
<columna> puede ser una variable entera, valor entero literal, expresion cuyo
   resultado sea entero e indicada la columna a la que quiero acceder en la
   matriz
<variable> va ser la el lugar donde se almacenara una copia del valor
   solicitado al arreglo, debe ser del mismo tipo de dato del arreglo */
int a =notas[2][3];
int b;
b = notas[cantidadMaxima-50][a+1];
```

4.2.4. Recorrer todos los elementos de la matriz

Para recorrer todos los elementos o parte de los elementos almacenados en la matriz vamos utilizar dos instrucciones **for** de forma anidada cada una asociada a una de las dimensiones de la matriz es importante hacer notar que dependiendo de como se definan dichas intrucciones el orden y sentido de recorrido dentro de la matriz puede variar lo cual puede ser útil o engorroso dependiendo de la situación

```
/*for( int i=<fila_inicial>; i < <fila_final>; i++ ){
     for( int j=<columna_inicial>; j < <columna_final>; j++ ){
     }
  }
i++ si <fila_inicial> <= <fila_final>
i-- si <fila_inicial> > <fila_final>
i <= <fila_final> si deseo incluir en el rango la ultima fila pero tiene que
    ser una fila valida de la matriz
j++ si <columna_inicial> <= <columna_final>
j-- si <columna_inicial> > <columna_final>
j <= <columna_final> si deseo incluir en el rango la ultima columna pero tiene
    que ser una columna valida de la matriz
*/
for(int i=0;i<cFilas;i++){
   for(int j=0;j<cColumnas;j++){
      //notas[i][j] accedo al valor almacenado en la i-nesima fila y j-nesima
         de la matriz
   }
}
```

4.2.5. Leer los valores y almacenarlos en la matriz

Para leer los valores consola y almacernalos directamente en la matriz vamos a utilizar una combinación de recorrer todos los elementos y almacenar en la matriz visto anteriormente

```
/* En cada iteracion de los fors el cin lee un valor y dicho valor es
   almacenado en la matriz en la posicion que indique el valor de las
   variable i y j en esa iteracion, en la primera el valor seria (0;0), en la
    segunda (0;1) y asi sucesivamente. Tener en cuenta que los valores de i
   esten en rango de posiciones validas de la filas y la j en el rango de
   posiciones validas de la columnas de la matriz */
Scanner in = new Scanner(System.in);
for(int i=0;i<cFilas;i++){
   for(int j=0;j<cColumnas;j++){
      matriz[i][j] = in.nextInt();
   }
}
```

4.2.6. Imprimir los valores almacenados en la matriz

Para imprimir los valores en consola de la matriz vamos a utilizar una combinación de recorrer todos los elementos y obtener en una matriz visto anteriormente

```
/* En cada iteracion de los for el cout imprime un valor que se corresponde
   con el almacenado en la matriz en la posicion que indique el valor de las
   variables i (fila) y j (columna)  en esa iteracion, en la primera el valor
    seria (0;0), en la segunda (0;1) y asi sucesivamente. Tener en cuenta que
    los valores de i esten en rango de posiciones validas de la filas y la j
   en el rango de posiciones validas de la columnas de la matriz*/
for(int i=0;i<cFilas;i++){
   for(int j=0;j<cColumnas;j++){
      System.out.print(matriz[i][j]+" ");
   }
}
System.out.println();
for(int i=0;i<cFilas;i++){
   for(int j=0;j<cColumnas;j++){
      System.out.print(matriz[i][j]+" ");
   }
   System.out.println();
}
```

En primer bucle anidado se imprime todos los valores de la matriz en una sola linea separados por espacios, mientras en el segundo bucle anidado cada valor almacenado en la matriz se imprime acorde a la fila y columna que le corresponde en la matriz.

5. Aplicaciones

Las matrices son utilizadas ampliamente en la programación de concursos, por su facilidad y liviandad para manipular información. En este contexto, son una buena forma para representar grafos, y son muy utilizadas en el cálculo númerico.

También pueden almacenar múltiples valores en una sola variable, información necesaria en sistemas y ayudar a resolver circuitos eléctricos. Las matrices también se utilizan para analizar fallas en las telecomunicaciones y las probabilidades de los corredores de bolsa, y pueden encriptar códigos y graficar funciones cruzadas.

6. Complejidad

La complejidad de las matrices esta dada por el uso de memoria que siempre va ser O($N \times M$) donde N y M son las dimensiones de la matriz. En cuanto la complejidad de tiempo va estar en gran medida del trabajo que hagamos con ella aunque el recorrido de por todos los elementos de la matriz va ser igualmente O($N \times M$).

7. Ejercicios propuestos

A continuación una lista de ejercicios que se resuelven utilizando matrices:

- DMOJ - Entubando el Estanque.
- DMOJ - Crucigrama
- MOG E - Encoding Matrices
- MOG B - Encoding Matrices