Jorge Gavidia C.I. 18836874 Secc. C1

Planteamiento del problema:

Dado un Número Finito de submatrices (7) llamadas imágenes, que poseen un tamaño mxn llamados ancho y alto de limites 1 ≤ ancho, alto ≤ 3, cada imagen debe tener un nombre único que debe ser un carácter ASCII comprendido en A-Z y '0'-'9'.

Una vez leído los datos anteriores, se debe procesar los datos de las imágenes y buscar la combinación de submatrices dentro de una matriz mayor, para que esta ocupe el mínimo de espacio, y a su vez minimice el número de espacios sin usar si llegase a haberlo dentro de la matriz resultante.

La Salida debe contener el tamaño mxn de la matriz resultante de todas las submatrices ubicadas en ella, e imprimir la matriz con el nombre de la imagen que se ubica en cada espacio de esta matriz, si el espacio no está utilizado por una imagen llevará el signo #. Se debe imprimir la matriz que sea de menor tamaño, contenga todas las imágenes y posea la menos cantidad de espacios no utilizados (#).

Análisis:

El problema se debe hacer bajo el esquema de backtraking, de forma exhaustiva, para encontrar la solución óptima, esta puede ser que no sea única, así que cualquiera de las soluciones es posible imprimir.

Se debe encontrar una o varias estructuras de datos que puedan recabar la información de las imágenes, y la matriz solución. Se debe hacer las validaciones correspondientes a los valores fronteras que se dan en el problema.

Explicación de la solución:

-Utilizando el esquema de programación a objetos utilizando c++ como lenguaje de programación, se crea una clase de Nombre Atlas que posee todos los atributos y métodos a utilizar para resolver el problema. Para trabajar creamos un objeto llamado trabajo del tipo atlas.

Ahora a explicar los métodos de la clase atlas:

inicializar_matriz: en este método inicializamos la matriz con el valor nulo #, esto para saber dónde tengo espacio libre para colocar las imágenes.

pedir_cantidad_de_imagenes: Pedir los datos de entrada de cantidad de imágenes en una variable global de tipo entero llamada cantidad_de_imagenes. Este dato tiene restricciones donde 1 ≤ n≤ 7, en caso de que no esté dentro de estos valores el programa lo pedirá hasta que sea correcto.

llenar_arreglo_DatoImagen: Pedir los datos de cada una de las imágenes que están dentro del problema, para esto cree un tipo de dato llamado DatoImagen, en el cual es un registro que contiene los tres datos de las imágenes a colectar, nombre de imagen, ancho,

largo. También cree un arreglo arreglo_DatoImagen de 7 posiciones del tipo DatoImagen, para así colectar todos los datos en una misma estructura.

Dentro de este método, hacemos la verificación de que ninguna de los nombre que introducimos que son caracteres ASCII comprendido en A-Z y '0'-'9', dentro de este método tenemos una variable c, la cual guardara el número de que representa al carácter ASCII para su comparación entre los límites establecidos y a su vez creamos una variable repetido del tipo booleano, para verificar en el arreglo que las imágenes anteriores no tengan el mismo nombre de la imagen actual. Si el nombre esta repetido, se pide uno nuevo y se hace la verificación. También hacemos la verificación de que los datos ancho y alto estén dentro de los límites que son 1 ≤ ancho, alto ≤ 3, sino están dentro de estos se piden repetidas veces.

mostrar_matriz: En este método se mostrará la matriz resultado.

Backtraking: Es el método donde implementaremos el backtraking. En este método utilizamos el esquema de solución óptima.

```
Procedure Backtracking(paso)
    InicializarAlternativa(alternativa);
    // Me coloco en la primera casilla menos uno del arreglo arreglo_DatoImagen
que contiene todos los datos de las imágenes.
    Repeat
        ObtenerSiguienteAlternativa(alternativa);
        // dentro del arreglo  arreglo_DatoImagen, me muevo una casilla para
obtener los datos de otra imagen.
        If (AlternativaValida(alternativa)) Then // Esta sentencia la omitì ya
que todas las imágenes son válidas.
            IncluirAlternativa(alternativa, solucionParcial);
            // Introduzco la imagen dentro de la matriz
                If EsSolucion(solucionParcial) then
                    If EsMejor(solucionParcial, solucionOptima) Then
                    solucionOptima = solucionParcial;
                    EndIf
                Else
                nuevoPaso = GenerarNuevoPaso(paso);
                Backtracking(nuevoPaso);
                Paso = DeshacerPaso(nuevoPaso);
                EndIf
            ExcluirAlternativa(alternativa, solucionParcial);
            EndIf
    Until SeAcabaronLasAternativas(alternativa) or encontreSolucion==true;
    EndProcedure
```

incluir_alternativa: Introduzco la imagen dentro de la matriz.

Comprobarancho: Verifica que en la casilla (i,j) donde está parado en incluir imagen, quede espacio en el ancho, para su introducción dentro de la matriz.

Seudocodigo:

```
clase atlas {
Publico:
        entero  cantidad_de_imagenes; //# cantidad de imagenes
        char Matriz [20][20]; // matriz solucion
        DatoImagen arreglo_DatoImagen[6];// arreglo que contiene la
informacion de cada una de las imagenes

        Procedimiento pedir_cantidad_de_imagenes (); // Se lee la cantidad de
imagenes
        Procedimiento llenar_arreglo_DatoImagen ();// Se llena las
caracteristicas de la imagen.
        Procedimiento inicializar_matriz (); // Inicializa la matriz con el valor
nulo
        Procedimiento mostrar_matriz();// muestra la matriz
        Procedimiento Backtraking ();
        Procedimiento incluir_alternativa (DatoImagen alternativa);
        Procedimiento comprobarancho(entero ancho, entero
punteroancho,int punterolargo);
};Fin de Clase

procedimiento atlas::pedir_cantidad_de_imagenes (){
    cout << "\n Introduzca el número de imagenes: ";
    leer >> cantidad_de_imagenes;

    si( (cantidad_de_imagenes>7) || (cantidad_de_imagenes<1))entonces{

        mientras ((cantidad_de_imagenes>7)Y(cantidad_de_imagenes<1))
        {
            cout << endl << "El numero de imagenes debe estar comprendido
entre 1 y 7 incluyendolos : ";
            leer >> cantidad_de_imagenes;
        } Fin mientras

    }Fin si
}Fin Procedimiento

Procedimiento atlas::llenar_arreglo_DatoImagen()  {
Entero i,c;
booleano repetido;
    Para (i=0; i<cantidad_de_imagenes; i++)
    {
        cout << "\n Introduzca nombre de la imagen # "<<i ;
        repetido=falso;
            hacer
                {
                    si (repetido==falso) entocnes
                    {
                        cout << endl << "C puede ser cualquier caracter
ASCII comprendido en A-Z y '0'-'9' : ";
                    }
```

```
                                        sino
                                                cout << endl << "C es un caracter que identifica
a la imagen univocamente no se puede repetir : ";

                                        leer >> arreglo_DatoImagen [i].Nombre_Imagen;
                                        c=arreglo_DatoImagen [i].Nombre_Imagen; //
(((c>=65)&&(c<=90))||((c>=48)&&(c=<57)))
                                        cout << c;
                                        repetido=falso;

                                        si (i>0) entocnes//siemre que i>0 verificamos que no se
repita el nombre
                                        {
                                                Para (int j=0; j==(i-1); j++)
                                                {
                                                        si (c==arreglo_DatoImagen
[j].Nombre_Imagen) entocnes//Son nombres iguales
                                                        {
                                                                repetido=true;
                                                                c=0;
                                                        }
                                                }
                                        }

                                }
                         mientras (!((c>=65)&&(c<=90))&& !((c>=48)&&(c<=57)))          ;

                cout << "\n Introduzca ancho de la imagen # "<<i ;
                        hacer
                                {
                                        cout << endl << "El ancho debe ser 1<= ancho <=3 : ";
                                        leer >> arreglo_DatoImagen [i].ancho;
                                }
                        mientras ((arreglo_DatoImagen [i].ancho<1)|| (arreglo_DatoImagen
[i].ancho>3) )    ;

                cout << "\n Introduzca largo de la imagen # "<<i ;
                        hacer
                                        {
                                                cout << endl << "El largo debe ser 1<= largo
<=3 : ";
                                                leer >> arreglo_DatoImagen [i].largo;
                                        }
                        mientras ((arreglo_DatoImagen [i].largo<1)|| (arreglo_DatoImagen
[i].largo>3));

        }
}
```

```
procedimiento atlas::inicializar_matriz()  { //Inicializamos la Matriz en #
entero i,j;
	para (i=0; i<22 ; i++)
	{
		para (j=0; j<21 ; j++)
		{
			atlas::Matriz [i][j]='#';
		}
	}

}

procedimiento atlas::mostrar_matriz()  { //Inicializamos la Matriz en #
entero i,j;
	para (i=1; i=7 ; i++)
	{
		para (j=1; j=(7) ; j++)
		{
			cout << Matriz [i][j];
		}
	}

}
procedimiento atlas::incluir_alternativa (DatoImagen alternativa){

entero punteroalto,punteroancho;
booleno comprobarancho;

	para (punteroalto=0; (((atlas::cantidad_de_imagenes)*3)-1); punteroalto ++)
	{
			para (punteroancho=0; (((atlas::cantidad_de_imagenes)*3)-1);
punteroancho ++)
			{
				si (Matriz[punteroancho][punteroalto]=='#') {

					comprobarancho=
atlas::comprobarancho(alternativa.ancho, punteroancho, punteroalto);// comprueba q donde
voy a poner la imagen se puede

				}
			}

	}
}


Funcion atlas::comprobarancho(entero ancho, entero punteroancho,entero
punterolargo):Booleano {

booleano verificador;
entero i;
```

```
	para (i=0; i==(ancho); i++)
	{
		si ( (Matriz [punteroancho+i][punterolargo]) == '#' ){
			verificador=true;
		}
		sino
		i=ancho;//sale
	}
retorna (verificador);
}

procedimiento atlas::Backtraking () {
DatoImagen alternativa;
entero ialter;
ialter=(-1);
	alternativa= arreglo_DatoImagen[ialter];		//iniciar alternativa, es escoger
cada uno de las imagenes para meter.

	hacer{
		alternativa= arreglo_DatoImagen[ialter+1];	//obtener sig alternativa;

		incluir_alternativa(alternativa); //incluir_alternativa
()//IncluirAlternativa(alternativa, solucionParcial);

	}
	mientras(ialter=2);

}

int main (){

atlas trabajo;
trabajo.inicializar_matriz();
entero opcion=0;
menu:
mientras( opcion != 3)
 {
 //muestra el  menu
 cout << endl << " Menu: " << endl;
 cout << "_____________________________________________" << endl;
 cout << " 1. Cantidad de Imagenes \n 2. Llenar datos de imagenes \n 3. Salir\n";

 //leemos de teclado
 cout << "\n\n\n Que desea hacer: ";
 cin >> opcion;

	switch (opcion)
	{
		case 1: //leer cantidad de imagenes

			trabajo.pedir_cantidad_de_imagenes ();
```

```
                    break;

                     case 2: //llenar arreglo de imagenes

                                    trabajo.llenar_arreglo_DatoImagen();
                    break;

                     case 3: //Salir


                     break;

                     default:
                              cout << "Solo se puede introducir opciones validas comprendidas entre
(1-3) !!!";
                              goto menu;
                    }

}

return(0);
}
```