Universidad Autónoma Metropolitana Unidad Azcapotzalco

División de Ciencias Básicas e Ingeniería

Licenciatura en Ingeniería en Computación

Reporte de proyecto terminal

Implementación de un algoritmo de aproximación para el problema del cartero con restricciones en los arcos

Elaborado por:

Zambrano Gervacio Raymundo
206305264

Trimestre: 13-I

Asesores:

Dr. Francisco Javier Zaragoza Martínez
Profesor titular.
Departamento de Sistemas.

M en C. Dolores Lara Cuevas
Profesora asociada.
Departamento de Sistemas.

Contenido.

- **Resumen.**

El siguiente documento redacta de manera desarrollada el problema del cartero cuando sus arcos tienen restricciones, es decir cuando los arcos (aristas) son tanto dirigidas como no dirigidas y al mismo tiempo cada arco tiene un costo de recorrido.

Para el desarrollo de este problema general tratamos de dividir el problema en problemas más pequeños que podemos atacar de manera más eficaz, es aquí donde nos apoyaremos en problemas de flujos mínimos y en emparejamientos perfectos.

Un problema de flujo es un problema en el que se busca que un grafo G sea conexo, es decir que para todos sus vértices exista un recorrido el cual pase por todos sus vértices al menos una vez [1]. Como nuestro problema contiene restricciones, costos, lo que buscamos es encontrar el flujo mínimo perteneciente al grafo G, es decir encontrar un recorrido que pase por todos los vértices de G al menos una vez y que el costo de hacer este recorrido sea el menor posible.

Un problema de acoplamiento es aquel donde para un grafo G existen pares de vértices tales que los vértices están ligados por una arista en común y los pares de vértices no son adyacentes entre si [1]. Cabe mencionar que un emparejamiento máximo se presenta cuando se encuentran todas las parejas de vértices posibles en el grafo, si además de esto, el emparejamiento que encontramos contiene a todos los vértices del grafo G entonces decimos que el emparejamiento es máximo y perfecto [1].

El problema del cartero chino es un problema en el cual un cartero entrega la correspondencia a su destino y al finalizar regresa a su lugar de salida, por supuesto debe recorrer todas las calles y entregar toda la correspondencia, al salir a hacer la entrega el cartero desea utilizar la ruta por la cual tenga que caminar lo menos posible. Este problema puede ser resuelto de manera eficaz si lo tratamos de resolver buscando un recorrido euleriano para el problema, suponiendo que la ruta que cubra el cartero se puede representar

como un grafo G donde cada lugar donde entrega correspondencia se trate como un vértice de G y cada calle o tramo recorrido se puede representar como una arista del grafo G y que además cada arista tiene un costo asociado, el cual podría ser el tiempo de recorrido o el esfuerzo de recorrer ese tramo [1], así tomando en cuanta esto se puede contemplar el buscar un camino euleriano óptimo para el grafo G.

El problema que nosotros pretendemos resolver es una variación del problema del cartero chino, donde cada arista dentro de nuestro grafo G puede ser o no dirigida, es decir que una arista dirigida es aquella que va de un punto A a un punto B y que no puede recorrerse esa arista del punto B al punto A, por otro lado una arista no dirigida es aquella en la que se puede ir tanto del punto A al punto B y viceversa.

- **Introducción.**

El presente documento describe el proceso en la elaboración del proyecto terminal *Implementación de un algoritmo de aproximación para el problema del cartero con restricciones en los arcos,* así como la descripción de las pruebas realizadas y los resultados obtenidos.

Se describen los algoritmos usados en el desarrollo del proyecto, así como la estructura de los archivos de entrada que se manipulan y los archivos de salida que se generan.

Además se presenta un manual de usuario de los pasos que se deben seguir para obtener los resultados esperados, de la mano de pruebas que se tomaron para la realización del proyecto.

Adjunto a este archivo se hace entrega del código fuente del programa así como los ejecutables y documentos de pruebas para probar el proyecto, así como también las librerías necesarias para el correcto funcionamiento del programa.

- **Retroalimentación.**

Para retomar el proyecto se hará una remembranza tanto del objetivo general como de los objetivos particulares.

Objetivo general

Implementar y probar un algoritmo de aproximación polinomial para el problema del cartero en gráficas mixtas con restricciones en los arcos.

Objetivos específicos

- Diseñar e implementar un módulo para crear gráficas mixtas aleatorias con restricciones en los arcos.

- Implementar un módulo para reducir el problema general a un problema de flujo y un problema de acoplamiento.

- Diseñar e implementar un módulo para el algoritmo de aproximación.

- Realizar pruebas para las funciones de la aplicación desarrollada.

- **Desarrollo.**

Análisis.

El proyecto para un mejor desarrollo y elaboración se plantea en dos (tres) módulos por separado, los cuales se estudian, planean y desarrollan para después ser probados, una vez probados se integran en un solo proyecto.

Módulo 1.

El primer módulo planteado es el modulo que está planeado a diseñar gráficas mixtas conexas, es decir que contengan tanto aristas dirigidas como no dirigidas y que todos sus nodos estén conectados, para el desarrollo de este módulo se tuvieron varias condiciones tomadas en cuenta que se explican a continuación.

El usuario del sistema debe indicar al mismo el número de vértices (nodos) que se contemplaran para la ejecución del programa, el módulo está diseñado para aceptar de 3 a 50 nodos, ya que lo que se espera es que como mínimo se tenga una gráfica de 3 vértices y 2 aristas cumpliendo la condición:

$$n - 1 \leq m \leq 3n - 6$$

Donde $n - 1$ es el número mínimo de aristas que se tendrá (el número de aristas de un árbol), $3n - 6$ el máximo de aristas que se tendrán (el máximo número de aristas de una gráfica plana).

El programa generara coordenadas al azar con valores entre 0 y 50, que se asociaran a cada vértice (nodo) de la gráfica.

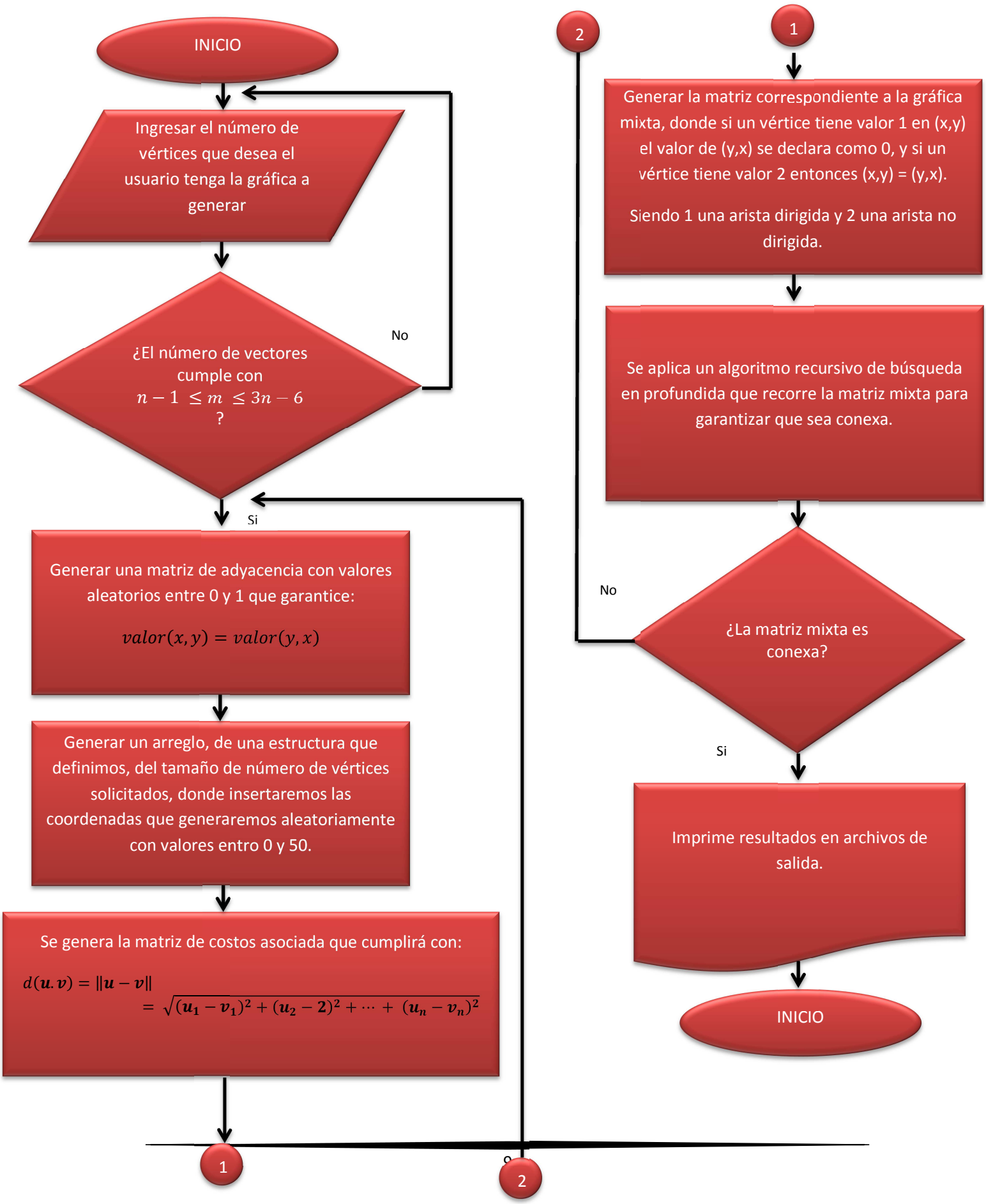
Además se toma en cuenta que cada arista debe tener un costo asociado, el cual se calcula con la longitud dada por la norma euclidiana de las coordenadas generadas que están asociadas entre sí de la siguiente manera:

$$d(\boldsymbol{u}.\boldsymbol{v}) = \|\boldsymbol{u} - \boldsymbol{v}\| = \sqrt{(\boldsymbol{u_1} - \boldsymbol{v_1})^2 + (\boldsymbol{u_2} - \boldsymbol{2})^2 + \cdots + (\boldsymbol{u_n} - \boldsymbol{v_n})^2}$$

El módulo 1 generara de manera automática dos archivos de salida, el primero contendrá el número de vértices que se utilizaron, la matriz de adyacencia generada correspondiente a la gráfica generada y su matriz de costos (pesos) asociados, además este archivo tiene un formato estándar que servirá para que el segundo módulo pueda leerlo sin problemas. El segundo programa generado será un reporte de los resultados arrojados, explicando con detalles el significado de los resultados.

Cabe mencionar que el programa genera gráficas conexas, ya que utiliza un algoritmo recursivo de búsqueda en profundidad que revisa, independientemente de si la gráfica es totalmente dirigida, no dirigida o mixta, que todos sus nodos estén conectados entre sí, si el programa detecta que la gráfica generada no es conexa entonces repite la generación de la gráfica tantas veces sea necesario para obtener una gráfica conexa, esto garantiza que el resultado arrojado en los archivos de salida contengan una gráfica conexa mixta.

Para poder entender mejor el problema a continuación se presenta un diagrama de flujo que ayudara a entender mejor el funcionamiento del módulo.


INICIO
Ingresar el número de vértices que desea el usuario tenga la gráfica a generar
¿El número de vectores cumple con $n-1 \leq m \leq 3n-6$?
No
Si
Generar una matriz de adyacencia con valores aleatorios entre 0 y 1 que garantice:
$valor(x,y) = valor(y,x)$
Generar un arreglo, de una estructura que definimos, del tamaño de número de vértices solicitados, donde insertaremos las coordenadas que generaremos aleatoriamente con valores entro 0 y 50.
Se genera la matriz de costos asociada que cumplirá con:
$d(\boldsymbol{u}.\boldsymbol{v}) = \|\boldsymbol{u}-\boldsymbol{v}\| = \sqrt{(\boldsymbol{u_1}-\boldsymbol{v_1})^2+(\boldsymbol{u_2}-\boldsymbol{2})^2+\cdots+(\boldsymbol{u_n}-\boldsymbol{v_n})^2}$
1
2
1
Generar la matriz correspondiente a la gráfica mixta, donde si un vértice tiene valor 1 en (x,y) el valor de (y,x) se declara como 0, y si un vértice tiene valor 2 entonces (x,y) = (y,x).
Siendo 1 una arista dirigida y 2 una arista no dirigida.
Se aplica un algoritmo recursivo de búsqueda en profundida que recorre la matriz mixta para garantizar que sea conexa.
¿La matriz mixta es conexa?
No
Si
Imprime resultados en archivos de salida.
INICIO

Módulo 2.

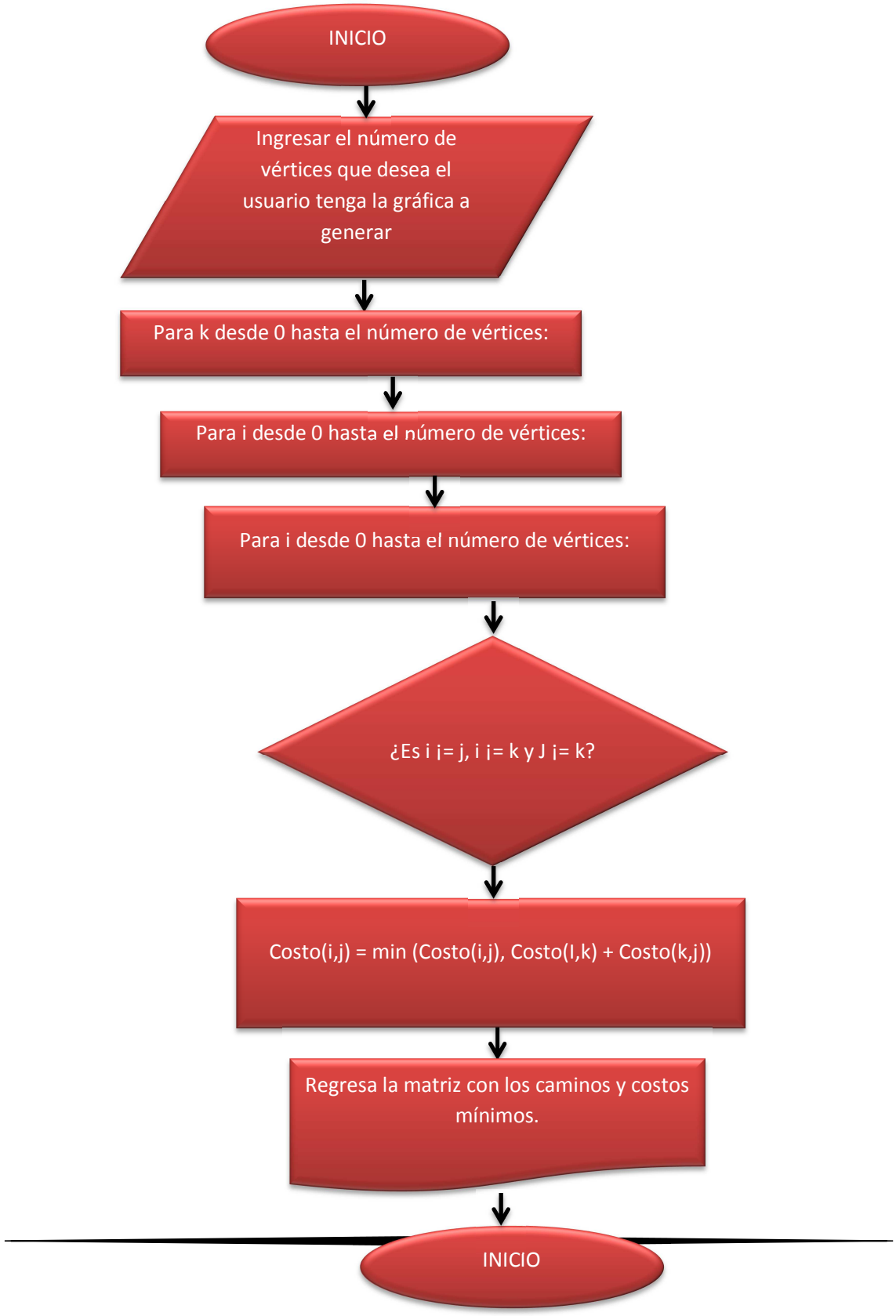
El módulo dos por otra parte se encarga de reducir el problema general a un problema de flujo y un problema de acoplamiento.

En este módulo lo primero que se hace es leer los datos arrojados por el módulo uno, para esto se contempla una función que lee y prepara los datos para utilizarlos de manera adecuada al reducir el problema general.

Por una parte reduciremos el problema a un problema de flujo, para esto es necesario modificar nuestros datos, ya que utilizaremos el algoritmo de Floyd-Warshall [2] el cual nos pide que las aristas que no tienen costo alguno debemos considerarlas con un costo de tamaño muy grande para poder aprovechar el algoritmo, a continuación se de una pequeña explicación del algoritmo Floyd-Warshall [2].

Un ejemplo del algoritmo de Floyd-Warshall se puede ver en [3]

Diagrama de flujo:



Una vez ejecutado el algoritmo de Floyd-Warshal procedemos a buscar el camino más corto dentro de la matriz de costos, que después de usar Floyd-Warshall se modificó, con dos pequeños ciclos for y una condición buscamos el costo menor y procedemos a buscar su recorrido en la matriz de caminos que el programa anteriormente género.

A continuación se procede a resolver el problema de acoplamiento, para esto nos apoyaremos en un algoritmo ya diseñado llamado “Blossom IV” [4], en los documentos anexo se incluye este programa junto con sus librerías y su manual de instalación si se requiere instalar independiente, dentro de los archivos entregados este programa ya está instalado y compilado, listo para su uso.

El programa que se dedica a resolver el problema de acoplamiento debe recibir un archivo de entrada con las combinaciones pares de todos los vértices que contiene nuestra gráfica, el programa se encarga de generar este archivo y hacer la llamada al programa para que realice sus pruebas.

El programa de Blossom IV se puede descargar en [5].

- **Análisis de resultados.**

Una vez acoplados los dos módulos se procede a realizar las pruebas para mostrar la funcionalidad del proyecto.

El proyecto completo está desarrollado bajo lenguaje C++ y C (debido al programa de Blossom IV), para realizar tanto el desarrollo como las pruebas se utilizó software libre, utilizando el programa Geany [6] para realizar la codificación y las pruebas.

Para realizar las pruebas se utiliza una plantilla base, que ya contiene los datos de entrada específicos para demostrar un ejemplo de cómo funciona la aplicación en la parte de la reducción del problema, en la generación de gráficas mixtas se deja que el usuario inspeccione y descubra que los resultados son reales.

Pruebas a módulo 1.

Al ejecutar el modulo que genera las gráficas nos solicita le digamos el número de vértices que deseamos utilizar para nuestra gráfica.

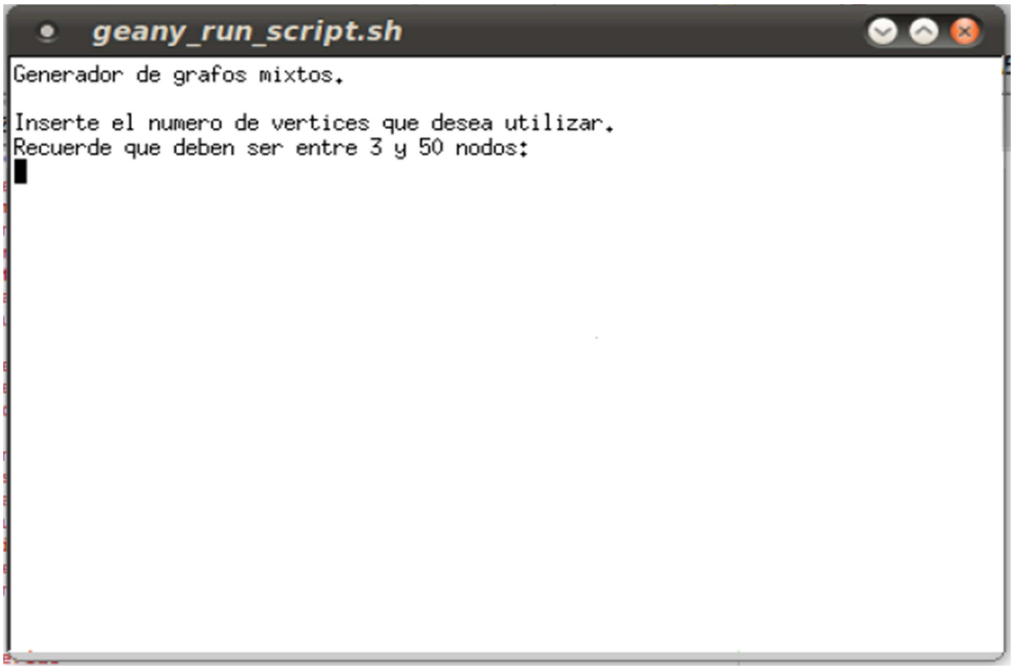


Figura 1. Generador de grafos mixtos, se pide se inserte el número de vértices iniciales.

Si por alguna razón el usuario indica que quiere tener más de 50 o menos de 3 vértices el programa so le indica su error y le solicita de nuevo introduzca el número de vértices.

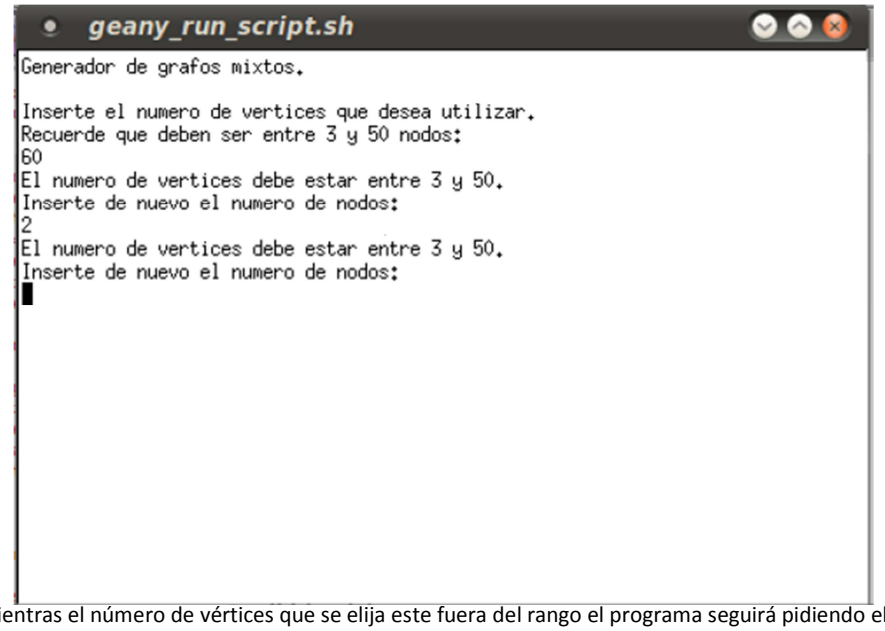


Figura 2. Mientras el número de vértices que se elija este fuera del rango el programa seguirá pidiendo el número de vértices.

Cuando indicamos un numero de vértices aceptado el programa empieza a generar las matrices correspondientes, si la gráfica obtenida resulta ser no conexa se repite la operación hasta obtener una gráfica conexa.

Figura 3. Se finaliza la fase de creación de un grafo mixto.

Los resultados a estas pruebas son:

El archivo de salida que utilizaremos para trabajar el módulo dos es de la siguiente forma y nos muestra el número de vértices, la matriz de adyacencia mixta y la matriz de costos.

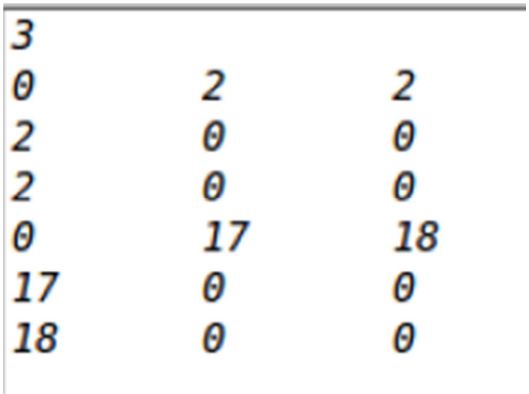


Figura 4. Los resultados arrojados muestran el número de vértices, la matriz de adyacencia y la matriz de costos asociada.

El reporte que nos genera el programa nos muestra la siguiente información:

```
Reporte de la generación del grafo mixto!!

El número de vertices elegidos por el usuario son:
3

La matriz de adyacencia correspondiente al grafo mixto es:

0       2       2
2       0       0
2       0       0

0 significa que no hay arista que conecte los vertices.
1 significa que hay un arista que conecte a los vertices y es dirigida.
2 significa que hay un arista que conecte a los vertices y no es dirigida.

Las coordenadas correspondientes a cada vertice son:

Coordenadas del vertice 0: (28,17)
Coordenadas del vertice 1: (11,14)
Coordenadas del vertice 2: (46,12)


La matriz de pesos correspondientes es:

0       17      18
17      0       0
18      0       0
```

Figura 5. Se muestra el reporte generado tras la ejecución del programa.

Como podemos observar en la pruebas el módulo uno cumple con sus características, crea una gráfica conexa de vértices tanto dirigidos como no dirigidos, el costo correspondiente a cada arista está dado por la norma euclidiana de sus coordenadas.

Pruebas módulo 2.

Para realizar las pruebas de este módulo se necesitan tener los archivos generados del módulo anterior o generar a mano un archivo de prueba que contenga el número de vértices, la matriz adyacente y la matriz de costos.

Ejecutamos el segundo programa el cual genera de manera automatizada todos los pasos de en una solo corrida.

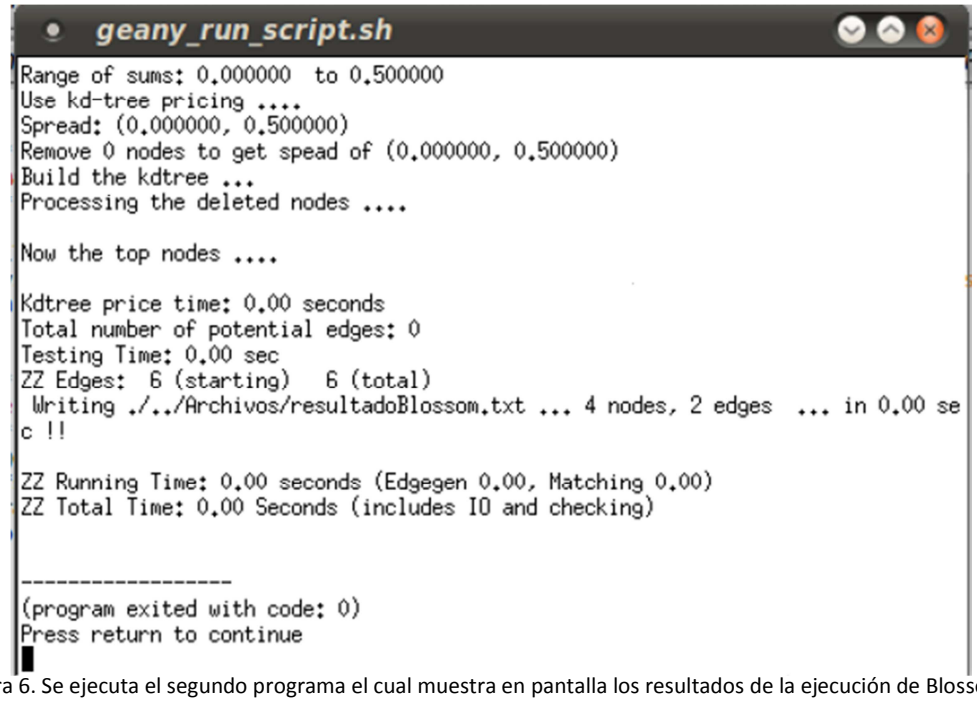
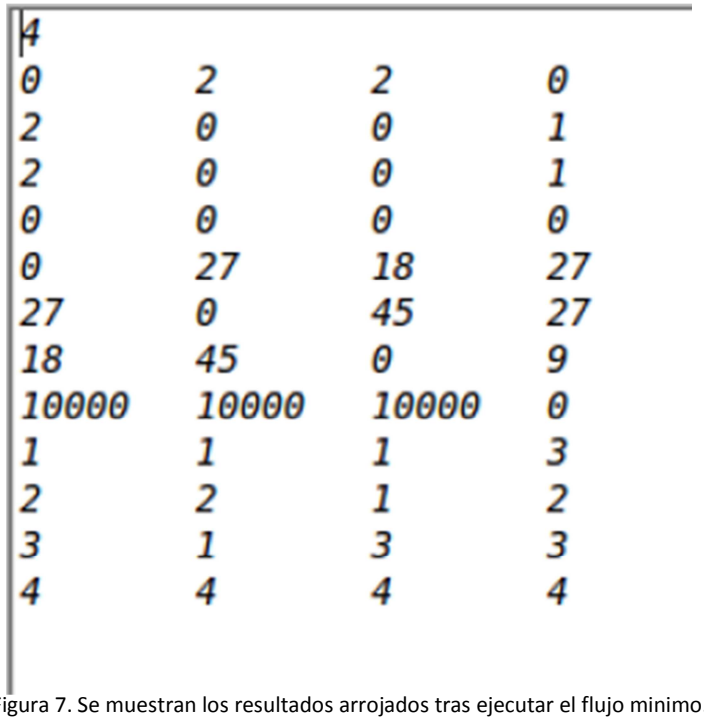


Figura 6. Se ejecuta el segundo programa el cual muestra en pantalla los resultados de la ejecución de Blossom 4.

Los resultados arrojados en pantalla son por parte del programa “Blossom IV”, los resultados que nosotros necesitamos se encuentran en archivos de texto.



```
4
0       2       2       0
2       0       0       1
2       0       0       1
0       0       0       0
0       27      18      27
27      0       45      27
18      45      0       9
10000   10000   10000   0
1       1       1       3
2       2       1       2
3       1       3       3
4       4       4       4
```

Figura 7. Se muestran los resultados arrojados tras ejecutar el flujo minimo.

El resultado arrojado es la matriz adyacente y la matriz de los costos mínimos.

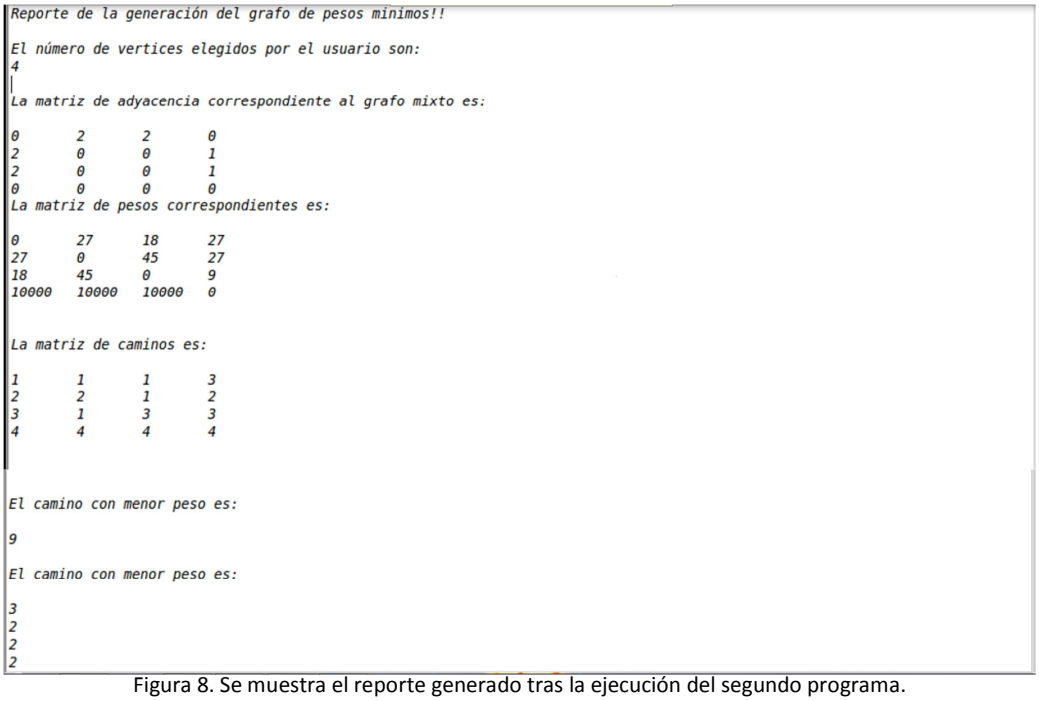


Figura 8. Se muestra el reporte generado tras la ejecución del segundo programa.

Los resultados arrojados en el reporte nos muestran el número de vértices, la matriz de adyacencia, la matriz de costos, la matriz de caminos, el camino con menor peso y los vértices por donde pasa.

- **Instrucciones de uso.**

Para ejecutar el módulo uno debemos entra a la ruta “ProblemaDelCartero-PT/Crear_grafos_mixtos” y abrir el archivo “Crear_grafos_mixtos.cpp” debemos ejecutarlo y compilarlo, se debe ejecutar bajo un entorno de Linux para asegurar su correcto funcionamiento.

Tras ejecutar el programa obtendremos los archivos de salida en la ruta “ProblemaDelCartero-PT/Archivos/” donde podremos ver nuestros archivos creados “grafoMixto.txt” y “reporteGrafoMixto.txt”.

Una vez se tienen los archivo procedemos a compilar y ejecutar nuestro módulo dos, el cual se encuentra en la ruta “ProblemaDelCartero-PT/Transformar_problema” y usaremos el archivo “Tranformar_problema.cpp”, una vez lo ejecutemos obtendremos los archivos de salida “grafoPesosMinimos”, “reporteGrafosPesosMinimos”, “archivoParaBlossom” y “resultadoBlossom”.

- **Conclusiones.**

Un problema que se puede modelar en forma de grafo es un problema que podemos abordar de distintas formas logrando darle una solución óptima, hoy en día muchos de los problemas frecuentes se puede modelar a manera de grafos, sabiendo esto la aplicación de un recorrido de pesos mínimos puede dar una solución óptima, sobre todo si logramos restringir y definir bien este.

Para el problema del cartero chino se tienen varias soluciones que nos pueden dar una solución óptima, sin embargo al tratar de abordar este problema con restricciones en los arcos nos encontramos con un problema NP-duro [7].

En conclusión la mejor manera de abordar un problema es tratar de dividirlo, como lo hicimos en este caso, donde se divide el problema para abordarlo por un lado con un problema de flujo mínimo y por otro lado con un problema de acoplamiento perfecto.

- **Referencias.**

[1] J. A. Bondy, “Graph theory with applications”, 4a Ed. University of Waterloo, Ontario, Canada.

[2]M. Montenegro, “Algoritmo de Floyd-Warshall”, [online]. Madrid: Departamento de sistemas informáticos y computación, Facultad de informática (UCM), Disponible en: http://dalila.sip.ucm.es/~manuel/Informatica/FloydWarshall.pdf

[3] A. Conejero, C. Jordán, “Algoritmo de Floyd-Warshall (modelización)”, [online]. Universidad Politécnica de Valéncia, Disponible en: https://polimedia.upv.es/visor/?id=754309f8-d338-8144-a358-c0c4c18dc5e6

[4] WIKIPEDIA, la enciclopedia libre [online], Disponible en: http://en.wikipedia.org/wiki/Blossom_algorithm

[5] La página de William Cook (http://www2.isye.gatech.edu/~wcook/blossom4/) posee la librería con el programa de Blossom IV y otras librerías útiles.

[6] La página oficial de Geany (http://www.geany.org) posee el entorno recomendado para ejecutar el programa.

[7] J. Veerasamy. “Approximation Algorithms for Postman Problems”. PhD thesis, Universidad de Texas, Dallas, 1999.

- **Anexos.**

A continuación se muestra el código fuente de los módulos desarrollados.

Módulo 1.

```
/*************************************************************************/
//Librerias
/*************************************************************************/
#include<iostream>
#include<stdio.h>
#include<stdlib.h>
#include<fstream>
#include<math.h>
#include<time.h>

/*************************************************************************/
//Definiciones, estructuras y variables globales
/*************************************************************************/
#define MAX 50

using namespace::std;

//Estructura para guardar las coordenadas de cada vertice.
struct coordenadas{
	int coorX;
	int coorY;
}matrizCoordenadas[MAX];

int matriz[MAX][MAX], matrizW[MAX][MAX], i, j;
int matrizConexa[MAX][MAX], Visitados[MAX];
```

```
/*************************************************************************/
//genera_matriz()
/*************************************************************************/
void genera_matriz(int numero_vertices){
	srand (time(NULL));
	for(i=0;i<numero_vertices;i++){
		for(j=i;j<numero_vertices;j++){
			if(i==j){
				matriz[i][j]=0;
			}else{
				matriz[i][j] = matriz[j][i] = rand()%2;
			}
		}
	}
return;
}

/*************************************************************************/
//genera_coordenadas()
/*************************************************************************/
void genera_coordenadas(int numero_vertices){
	srand (time(NULL));
	for(i=0;i<numero_vertices;i++){
		matrizCoordenadas[i].coorX=rand()%51;
		matrizCoordenadas[i].coorY=rand()%51;
	}
return;
}
```

```
/*************************************************************************/
//genera_matriz_de_pesos()
/*************************************************************************/
void genera_matriz_de_pesos(int numero_vertices){
	float Xtotal,Ytotal,Xcuadrada,Ycuadrada,suma,total;
	for(i=0;i<numero_vertices;i++){
		for(j=i;j<numero_vertices;j++){
			if(matriz[i][j]==1){
				Xtotal = matrizCoordenadas[i].coorX - matrizCoordenadas[j].coorX;
				Ytotal = matrizCoordenadas[i].coorY - matrizCoordenadas[j].coorY;
				Xcuadrada = pow(Xtotal,2);
				Ycuadrada = pow(Ytotal,2);
				suma = Xcuadrada + Ycuadrada;
				total = sqrt(suma);
				matrizW[i][j] = matrizW[j][i] = total;
			}else{
				matrizW[i][j] = matrizW[j][i] = 0;
			}
		}
	}
return;
}
```

```
/*************************************************************************/
//genera_grafo_mixto()
/*************************************************************************/
void genera_grafo_mixto(int numero_vertices){
	int tipo_arista = 0;
	srand (time(NULL));
	for(i=0;i<numero_vertices;i++){
		for(j=i;j<numero_vertices;j++){
			if(matriz[i][j] == 1){
				tipo_arista = (1 + rand()%2);
				matriz[i][j] = matriz[j][i] = tipo_arista;
				if(matriz[i][j] == 1){
					matrizW[j][i] = 0;
					matriz[j][i] = 0;
				}
			}
		}
	}
return;
}
```

```
/*************************************************************************/
//es_conexa()
/*************************************************************************/
void visitar(int i, int numero_vertices){
	int j;
	for(j=0;j<numero_vertices;j++){
		if((matrizConexa[i][j] == 1 || matrizConexa[i][j] == 2)){

			matrizConexa[i][j] = 0;
			visitar(j, numero_vertices);
			Visitados[j] = 1;
		}
	}
return;
}

bool es_conexa(int numero_vertices){
	int i,j,numero_vertices_visitados = 0;
	bool conexa;
	for(i=0;i<numero_vertices;i++){
		for(j=0;j<numero_vertices;j++){
			matrizConexa[i][j] = matriz[i][j];
		}
	}
	Visitados[0] = 1;
	visitar(0, numero_vertices);
	for(i=0;i<numero_vertices;i++){
		if(Visitados[i] == 1){
			numero_vertices_visitados++;
		}
	}
	if(numero_vertices_visitados == numero_vertices){
		conexa = true;
	}else{
		conexa = false;
	}
return conexa;
}
```

```
/*************************************************************************/
//genera_archivo_salida()
/*************************************************************************/
void genera_archivo_salida(int numero_vertices){
        ofstream grafoMixto("./../Archivos/grafoMixto.txt",ios::out);
  grafoMixto << numero_vertices << endl;
  for(i=0;i<numero_vertices;i++){
    for(j=0;j<numero_vertices;j++){
      grafoMixto << matriz[i][j] << "\t";
    }
    grafoMixto << endl;
  }
  for(i=0;i<numero_vertices;i++){
    for(j=0;j<numero_vertices;j++){
      grafoMixto << matrizW[i][j] << "\t";
    }
    grafoMixto << endl;
  }
return;
}
```

```
/*************************************************************************/
//genera_reporte()
/*************************************************************************/
void genera_reporte(int numero_vertices){
        ofstream grafoMixto("./../Archivos/reporteGrafoMixto.txt",ios::out);
  grafoMixto << "Reporte de la generaciÃ³n del grafo mixto!!" << endl << endl;
  grafoMixto << "El nÃºmero de vertices elegidos por el usuario son:" << endl;
  grafoMixto << numero_vertices << endl << endl;
  grafoMixto << "La matriz de adyacencia correspondiente al grafo mixto es:" << endl <<
endl;
  for(i=0;i<numero_vertices;i++){
    for(j=0;j<numero_vertices;j++){
      grafoMixto << matriz[i][j] << "\t";
    }
    grafoMixto << endl;
  }
  grafoMixto << endl << "0 significa que no hay arista que conecte los vertices." << endl;
  grafoMixto << "1 significa que hay un arista que conecte a los vertices y es dirigida." <<
endl;
  grafoMixto << "2 significa que hay un arista que conecte a los vertices y no es dirigida."
<< endl << endl;
  grafoMixto << "Las coordenadas correspondientes a cada vertice son:" << endl << endl;
  for(i=0;i<numero_vertices;i++){
            grafoMixto << "Coordenadas del vertice " << i << ": ";
            grafoMixto << "(" << matrizCoordenadas[i].coorX << "," <<
matrizCoordenadas[i].coorY << ") " << endl;
        }
        grafoMixto << endl << endl;
        grafoMixto << "La matriz de pesos correspondientes es:" << endl << endl;
  for(i=0;i<numero_vertices;i++){
    for(j=0;j<numero_vertices;j++){
      grafoMixto << matrizW[i][j] << "\t";
    }
    grafoMixto << endl;
  }
return;
}
```

```
/*************************************************************************/
//main()
/*************************************************************************/
int main(){
	int numero_vertices = 0;
	cout << "Generador de grafos mixtos." << endl << endl;
	cout << "Inserte el numero de vertices que desea utilizar." << endl;
	cout << "Recuerde que deben ser entre 3 y 50 nodos:" << endl;
	cin >> numero_vertices;
	while(numero_vertices < 3 || numero_vertices > 50){
		cout << "El numero de vertices debe estar entre 3 y 50." << endl;
		cout << "Inserte de nuevo el numero de nodos:" << endl;
		cin >> numero_vertices;
	}
	do{
		cout << "Generando matriz de adyacencia..." << endl;
		genera_matriz(numero_vertices);
		cout << "Generando coordenadas..." << endl;
		genera_coordenadas(numero_vertices);
		cout << "Generando matriz de pesos..." << endl;
		genera_matriz_de_pesos(numero_vertices);
		genera_grafo_mixto(numero_vertices);
	}while(!es_conexa(numero_vertices));
	cout << "Generando archivo de salida..." << endl;
	genera_archivo_salida(numero_vertices);
	cout << "Generando reporte..." << endl;
	genera_reporte(numero_vertices);
	cout << "Grafo mixto listo." << endl;
return 0;
}
```

Módulo 2.

```
/*************************************************************************/
//Librerias
/*************************************************************************/
#include<iostream>
#include<stdio.h>
#include<stdlib.h>
#include<fstream>
#include<math.h>
#include<time.h>

/*************************************************************************/
//Definiciones, estructuras y variables globales
/*************************************************************************/
#define MAX 50
#define INFINITO 10000

using namespace::std;

int matriz[MAX][MAX], matrizW[MAX][MAX], matrizCaminos[MAX][MAX], i, j;
int peso_menor = INFINITO, coordenada_x_menor = 0, coordenada_y_menor = 0;
int caminoRecorrido[MAX];
```

```
/*************************************************************************/
//lee_datos()
/*************************************************************************/
int lee_datos(){
   int numero_vertices;
   ifstream grafoMixto("./../Archivos/grafoMixto.txt",ios::in);
   //ifstream grafoMixto("./../Archivos/prueba.txt",ios::in); //usarla para ejemplo
   grafoMixto >> numero_vertices;
   while(!grafoMixto.eof()){
      while(i < numero_vertices){
                        while(j < numero_vertices){
                                grafoMixto >> matriz[i][j];
                                j++;
                        }
                        i++;
                        j = 0;
      }
      i = j = 0;
      while(i < numero_vertices){
                        while(j < numero_vertices){
                                grafoMixto >> matrizW[i][j];
                                if(matrizW[i][j] == 0 && i != j){
                                        matrizW[i][j] = INFINITO;
                                }
                                j++;
                        }
                        i++;
                        j = 0;
      }
   }
   grafoMixto.close();
return numero_vertices;
}
```

```
/*************************************************************************/
//floyd_warshal()
/*************************************************************************/
void floyd_warshall(int numero_vertices){
   int k = 0, suma = 0;
   for(i=0;i<numero_vertices;i++){
      for(j=0;j<numero_vertices;j++){
                              matrizCaminos[i][j] = (i + 1);
                     }
          }
   for(k=0;k<numero_vertices;k++){
      for(i=0;i<numero_vertices;i++){
         for(j=0;j<numero_vertices;j++){
                                    if(i != j && i != k && k != j){
                                             suma = matrizW[i][k] + matrizW[k][j];
                                             if(matrizW[i][j] > suma){
                                                      matrizW[i][j] = suma;
                                                      matrizCaminos[i][j] = matrizCaminos[k][j];
                                             }
                                    }
         }
      }
   }
   return;
}
```

```
/*************************************************************************/
//camino_mas_corto()
/*************************************************************************/
int camino_mas_corto(int numero_vertices){
	int peso_total = 0;
	for(i=0;i<numero_vertices;i++){
		for(j=0;j<numero_vertices;j++){
			if(matrizW[i][j] < INFINITO && matrizW[i][j] > 0){
				peso_menor = matrizW[i][j];
				coordenada_x_menor = i;
				coordenada_y_menor = j;
			}
		}
	}
	//coordenada_x_menor = 4; //usarla para ejemplo
	//coordenada_y_menor = 2; //usarla para ejemplo
	peso_total = matrizW[coordenada_x_menor][coordenada_y_menor];
	for(i=0;i<numero_vertices;i++){
		caminoRecorrido[i] = coordenada_y_menor;
		coordenada_y_menor                                                    =
(matrizCaminos[coordenada_x_menor][coordenada_y_menor] - 1);
	}
return peso_total;
}
```

```
/*************************************************************************/
//genera_archivo_para_blossom()
/*************************************************************************/
void genera_archivo_para_blossom(int numero_vertices){
        ofstream archivoParaBlossom("./../Archivos/archivoParaBlossom.txt",ios::out);
  archivoParaBlossom << numero_vertices << endl;
  for(i=0;i<numero_vertices;i++){
    for(j=0;j<numero_vertices;j++){
      if(matriz[i][j] == 1){
                                archivoParaBlossom << i << " " << j << endl;
                        }
                }
        }
        archivoParaBlossom.close();
return;
}

/*************************************************************************/
//llama_blossom()
/*************************************************************************/

void llama_blossom(){
        system("./../MATCH/blossom4 -x ./../Archivos/archivoParaBlossom.txt -w
./../Archivos/resultadoBlossom.txt");
return;
}
```

```
/*************************************************************************/
//genera_archivo_salida()
/*************************************************************************/
void genera_archivo_salida(int numero_vertices){
        ofstream grafoPesosMinimos("./../Archivos/grafoPesosMinimos.txt",ios::out);
  grafoPesosMinimos << numero_vertices << endl;
  for(i=0;i<numero_vertices;i++){
    for(j=0;j<numero_vertices;j++){
      grafoPesosMinimos << matriz[i][j] << "\t";
    }
    grafoPesosMinimos << endl;
  }
  for(i=0;i<numero_vertices;i++){
    for(j=0;j<numero_vertices;j++){
      grafoPesosMinimos << matrizW[i][j] << "\t";
    }
    grafoPesosMinimos << endl;
  }
  for(i=0;i<numero_vertices;i++){
    for(j=0;j<numero_vertices;j++){
      grafoPesosMinimos << matrizCaminos[i][j] << "\t";
    }
    grafoPesosMinimos << endl;
  }
  grafoPesosMinimos.close();
return;
}
```

```
/*************************************************************************/
//genera_reporte()
/*************************************************************************/
void genera_reporte(int numero_vertices, int peso_total){
        ofstream
grafoPesosMinimos("./../Archivos/reporteGrafoPesosMinimos.txt",ios::out);
  grafoPesosMinimos << "Reporte de la generaciÃ³n del grafo de pesos minimos!!" <<
endl << endl;
  grafoPesosMinimos << "El nÃºmero de vertices elegidos por el usuario son:" << endl;
  grafoPesosMinimos << numero_vertices << endl << endl;
  grafoPesosMinimos << "La matriz de adyacencia correspondiente al grafo mixto es:" <<
endl << endl;
  for(i=0;i<numero_vertices;i++){
    for(j=0;j<numero_vertices;j++){
      grafoPesosMinimos << matriz[i][j] << "\t";
    }
    grafoPesosMinimos << endl;
  }
       grafoPesosMinimos << "La matriz de pesos correspondientes es:" << endl << endl;
  for(i=0;i<numero_vertices;i++){
    for(j=0;j<numero_vertices;j++){
      grafoPesosMinimos << matrizW[i][j] << "\t";
    }
    grafoPesosMinimos << endl;
  }
  grafoPesosMinimos << endl << endl;
  grafoPesosMinimos << "La matriz de caminos es:" << endl << endl;
  for(i=0;i<numero_vertices;i++){
    for(j=0;j<numero_vertices;j++){
      grafoPesosMinimos << matrizCaminos[i][j] << "\t";
    }
    grafoPesosMinimos << endl;
  }
  grafoPesosMinimos << endl << endl;
  grafoPesosMinimos << "El camino con menor peso es:" << endl << endl;
  grafoPesosMinimos << peso_total;
  grafoPesosMinimos << endl << endl;
  grafoPesosMinimos << "El camino con menor peso es:" << endl << endl;
```

```
        for(i=0;i<numero_vertices;i++){
                grafoPesosMinimos << caminoRecorrido[i] << endl;
        }
        grafoPesosMinimos.close();
return;
}
```

```
/*************************************************************************/
//main()
/*************************************************************************/
int main(){
	int numero_vertices = 0, peso_total = 0;
	cout << "Calculo del flujo minimo..." << endl << endl;
	numero_vertices = lee_datos();
	cout << "Calculo Floyd_warshall..." << endl << endl;
	floyd_warshall(numero_vertices);
	peso_total = camino_mas_corto(numero_vertices);
	genera_archivo_para_blossom(numero_vertices);
	cout << "Llamando a Blossom..." << endl << endl;
	llama_blossom();
	cout << "Generando archivos..." << endl << endl;
	genera_archivo_salida(numero_vertices);
	cout << "Generando reportes..." << endl << endl;
	genera_reporte(numero_vertices, peso_total);
	cout << "Proyecto terminado." << endl << endl;
return 0;
}
```