Universidad Rafael Landívar
Facultad De Ingeniería
Matemática Discreta II
Ing. Juan Carlos Soto Santiago

Manual técnico

PROYECTO MATEMATICA DISCRETA II: ISOMORFISMO

INDICE

Contenido

INTRODUCCION

En el siguiente reporte vamos a presentar como fue elaborado la función de isomorfismo, los proceso y las implicaciones que esta misma conlleva para poder dar la respuesta esperada. Se va a presentar el funcionamiento a detalle para lograr comprender cuál fue el proceso para hacer posible el funcionamiento de dicha función.

Para la solución de este problema, fue necesario usar el lenguaje de alto nivel C# y Visual Studio 2013. Se hizo la verificación completa de todos los posibles errores y llegamos a la conclusión que al finalizar nuestro proyecto logramos alcanzar las metas planteadas.

Este reporte introduce también los conceptos básicos sobre los temas que son totalmente necesarios de dominar para poder lograr el desarrollo del proyecto de forma efectiva, alcanzando metas y conocimientos que nos van a permitir desarrollarnos en el ámbito de la ingeniería de sistemas.

Se logró entender cómo funcionan las funciones y las matrices de adyacencia y las permutaciones para dar la función o las funciones que se encuentren a lo largo del desarrollo del proyecto.

También vamos a presentar como es la elaboración del diseño del programa.

MARCO TEORICO

Teoría de grafos

Los grafos son una estructura de datos no lineal, la cual se puede usar para modelar diversas aplicaciones. Es una parte importante de la teoría combinatoria de matemáticas discreta, lo cual le da un carácter bastante amplio y complejo.

Los grafos se representan con frecuencia en la vida real, tal es el caso de una red de carreteras que enlace a cierto grupo de ciudades; aquí los nodos de la red o las ciudades representan los vértices del grafo, las carreteras que unen las ciudades representan los arcos o las aristas.

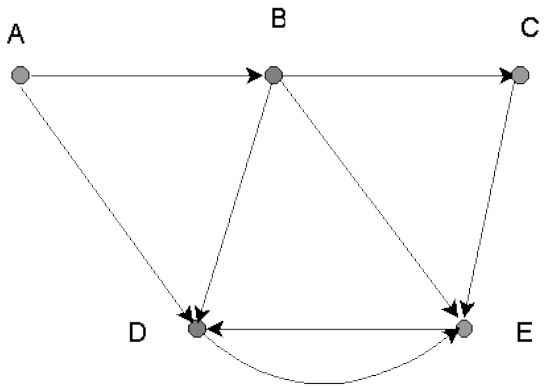


Figura 1: Representación gráfica de un grafo

Isomorfismo

Dos grafos tendrán la misma “forma matemática” cuando la única diferencia entre ambos, en cuanto a su estructura, sea que hemos usado cualquier tipo de representación distinta para sus vértices, mientras que las conexiones entre ellos sean las mismas. En este caso diremos que son isomorfos.

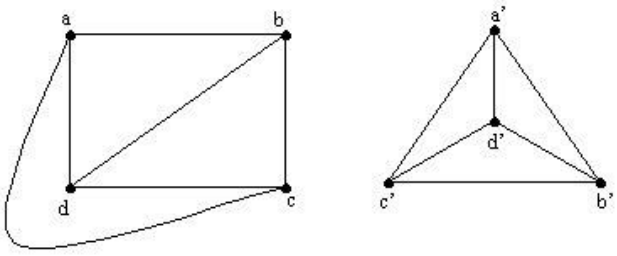


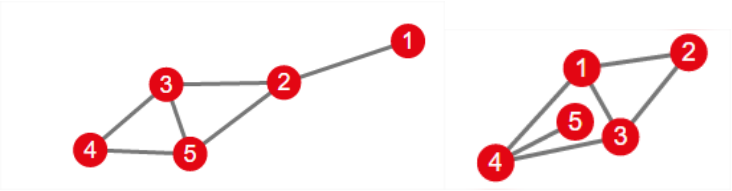
Figura 2: Ejemplo de isomorfismo

Matriz de adyacencia

Todo grafo simple puede ser representado por una matriz, que llamamos matriz de adyacencia. Se trata de una matriz cuadrada de n filas \times n columnas (siendo n el número de vértices del grafo). Para construir la matriz de adyacencia, cada elemento a_ {i, j} vale {{1}} cuando haya una arista que una los vértices i y j. En caso contrario el elemento a_ {i, j} vale 0. La matriz de adyacencia, por tanto, estará formada por ceros y unos.

Como se genera una matriz de adyacencia

Sea G1 el grafo de la figura izquierda y G2 el grafo de la figura derecha.



Sea V1, el conjunto de vértices de G1.

$$V_1 = \{1, 2, 3, 4, 5\}$$

Sea V2, el conjunto de vértices de G2

$$V_2 = \{1, 2, 3, 4, 5\}$$

Sea C el conjunto de permutaciones de V2

$$C = \{\{1,2,3,4,5\}, \{1,2,3,5,4\}, \dots, \{5,4,1,3,2\}, \dots, \{5,4,3,2,1\}\}$$

1. Se toma la primera permutación y se genera la matriz de V1 mapeado en el elemento tomado de C.

Permutación tomada: $\{1,2,3,4,5\}$

Matriz generada (Filas representan los elementos de V_1 y columnas los elementos de la permutación tomada.

$$P = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

2. Se genera la matriz de adyacencia del segundo grafo.

$$A^H = \begin{pmatrix} 0 & 1 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \end{pmatrix}$$

3. Se genera la matriz transpuesta de la matriz generada en el paso 1.

$$P = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

4. Se realiza el producto de $P * A^H * P$. En este caso se obtiene A^H por ser multiplicada por la matriz identidad:

$$\begin{pmatrix} 0 & 1 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \end{pmatrix}$$

Si la matriz que se obtiene es igual a la matriz de adyacencia del primer grafo, entonces se ha encontrado una función de isomorfismo.

Se procede con la siguiente permutación de V2 hasta encontrar una función, de existir.

PLANTIAMIENTO Y DESARROLLO DEL PROBLEMA

En el desarrollo del problema nos encontramos con la elaboración de las permutaciones pertinentes, esto nos permite tener más exactitud al momento de desarrollar el proyecto, brevemente se encontrará de qué forma se llevó a cabo este proceso.

Las permutaciones nos permiten no solo encontrar una función de isomorfismo sino varias, llevando a cabo la comparación de todos los vértices de los grafos, con sus respectivos grados y los vértices a los cuales están unidos.

ANALISIS

En el desarrollo del problema nos encontramos con la elaboración de las permutaciones pertinentes, esto nos permite tener más exactitud al momento de desarrollar el proyecto, brevemente se encontrará de qué forma se llevó a cabo este proceso.

Las permutaciones nos permiten no solo encontrar una función de isomorfismo sino varias, llevando a cabo la comparación de todos los vértices de los grafos, con sus respectivos grados y los vértices a los cuales están unidos.

Pero para llegar a esta solución se realizó un análisis previo del problema a resolver, para ello se utilizaron las herramientas de los pasos de Polya, diagrama de clases y diagramas de flujo para generar una idea del proceso que se debía llevar para resolver el problema.

Problema planteado:

Se requiere que implemente una aplicación que sea capaz de validar si 2 grafos (no multi-grafos, no dirigidos) son isomorfos, en cuyo caso deberá de generar una función de isomorfismo, en caso contrario indicar que no son isomorfos y el criterio que no cumplen:

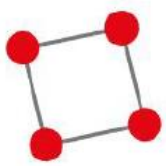
- No tienen la misma cantidad de vértices
- No tienen la misma cantidad de aristas
- No concuerdan los grados de los vértices
- No es posible determinar una función de isomorfismo.

Los grafos deberán ser leídos desde archivos de texto con el siguiente formato, uno por cada grafo.

Formato de entrada (.txt):
de vértices (n)
de vértice inicial A (0..n-1), # de vértice final A (0..n-1)
de vértice inicial B (0..n-1), # de vértice final B (0..n-1)
de vértice inicial C (0..n-1), # de vértice final C (0..n-1)

Ejemplo:

```
1  4
2  0,1
3  0,2
4  1,3
5  2,1
```



Análisis General – Pasos de Pólya

Paso 1: Comprender el problema

El primer paso y el más importante fue entender de que trataba el problema y que resultados se debían obtener.

Basados en el problema planteado se desgloso una lista de ideas importantes:

i. Se necesita una aplicación que sea capaz de dar solución al problema.
ii. Como se necesita una aplicación se debe hacer uso de la programación en algún lenguaje. (Queda a nuestro criterio elegir sobre que lenguaje programar).
iii. De entrada, se tienen 2 grafos.
iv. La cantidad de vértices y como se conectan los vértices entre sí, están almacenados en un archivo de texto con el formato establecido por el problema.
v. Se tiene un archivo de texto por cada grafo.
vi. Los grafos son no dirigidos y no son multi-grafos.
vii. La cantidad de vértices y aristas no tienen restricciones.
viii. Se necesita validar que estos 2 grafos sean isomorfos o no.
ix. De salida se tendrán 2 casos, en caso sean isomorfos y en el caso de que no lo sean.
x. Si los grafos son isomorfos se debe generar alguna función de isomorfismo.
xi. Si los grafos no son isomorfos se debe indicar porque no lo son.

Paso 2: Formular un plan

“Divide y vencerás”

Maquiavelo (Julio Cesar)

Se decidió tomar el problema general y dividirlo en sub-problemas más sencillos de resolver, de este modo tendríamos varias partes para resolver y de este modo es más fácil reconocer patrones, encontrar errores, etc.

Paso 3: Ejecutar el plan

Para llevar a cabo la solución se decidió utilizar el lenguaje de alto nivel C#, en el cual por medio del uso de POO se crearon las clases y métodos necesarios para llevar a cabo la solución al problema.

Basándonos en el problema general, se ha dividido en los siguientes sub problemas:

1. **Interacción con el usuario:** Tiene la tarea de tener un entorno en el cual el usuario pueda interactuar con la aplicación, para cargar los archivos y mostrar el resultado del problema.

   Solución: Para ello se hizo uso de una aplicación creada en Windows Forms, la cual posee un entorno gráfico que permite interactuar con la app por medio de accesorios, como lo son los botones, labels, picture box, etc. Por medio de los botones permite cargar los archivos de los dos grafos que se van a analizar, por medio de un botón ejecuta la sección de código que se encarga de determinar si los grafos son isomorfos o no, y muestra el resultado usando datagridview o MessageBox, además se le agregó una función para que por medio de Picture Box muestre una representación gráfica de los grafos ingresados.

2. **Lector de grafos:** Se debe encargar de leer de los archivos de texto y de ellos obtener la información de cada grafo, la información que contienen los archivos son: número de vértices y la configuración de cómo se conectan entre sí los vértices.

   Solución: Haciendo uso de POO se creó una clase llamada CargarArchivo que posee métodos que permiten ingresar un archivo que el usuario elija, luego recorre el archivo obteniendo la información del grafo, si en caso el archivo no posee el formato correcto, genera error al hacer la carga, esta información obtenida es almacenada en una clase llamada Grafo y dentro de esta clase existe una lista del tipo Vértice (Vértice es una clase creada que

permite almacenar la información el ID del vértice y con que otros vértices conecta) y otra lista del tipo Arista (Arista es una clase creada que permite almacenar el vértice de origen y a que vértice se dirige), en estas listas se almacena la información obtenida de los archivos de texto.

3. **Analizar grafos:** Se encarga de recorrer la nuevamente la información que esta almacenada en las clases Grafo, y va realizando comparaciones, la primera comparación es en la cantidad de vértices de cada grafo, la segunda comparación es en la cantidad de aristas de cada grafo, la tercera comparación es en el grado de cada vértice, si las anteriores comparaciones son iguales entonces se dice que existe la posibilidad de que los grafos sean isomorfos. Por lo que se pasa a analizar si los grafos poseen una función de isomorfismo, de ser esto verdadero se muestra que los grafos son isomorfos y se muestran las posibles funciones de isomorfismo, de lo contrario se mostrara porque no son isomorfos los grafos.

   Solución: Se creó una clase isomorfismo que entra en ejecución cuando el usuario presiona el botón encargado de generar la ejecución en el entorno gráfico, esta clase isomorfismo posee métodos que realizan las comparaciones mencionadas en el problema, para ello se hizo uso de la función foreach, e if's para realizar los recorridos de los grafos e ir comparando contra la información de un grafo contra otro grafo. Al llegar a determinar una función de isomorfismo se realizan otras operaciones. Para mostrar el resultado, se utilizan MessageBox para mostrar con un mensaje lo que ha sucedido, si fue isomorfo y si no lo fue, muestra las razones por la cual no.

4. **Determinar función de isomorfismo:** Entra en ejecución cuando las comparaciones anteriores se cumplieron, por lo tanto, es necesario analizar si los grafos poseen una función de isomorfismo.

   Solución: Para la solución se implementaron varias clases, la primera llamada PermutadorVertice, la segunda PermutadorArista, la función de estas clases es obtener todas las combinaciones posibles de vértices y aristas, llamándose así mismo recursivamente, para luego ir comparando contra el otro grafo para verificar si existen similitudes, en este caso de existir similitudes, es posible que exista una función de isomorfismo, por lo que pasa a apoyarse de la clase Matriz que se encarga de generar listas de vértices adyacente y luego comparar con la lista del otro grafo para determinar si existe una función de adyacencia entre los grafos, de ser esto verdadero se dice que existe una función de isomorfismo, por lo tanto los grafos son

isomorfos y al usuario se le muestra en un DataGridView las funciones de isomorfismo encontradas.

5. **Dibujar grafos ingresados (*):** Como grupo se decidió agregar una función a la aplicación para que además de mostrar si son isomorfos o no los grafos, el usuario tenga la opción de ver una representación gráfica de los grafos ingresados. Es importante mencionar que la app dibuja los grafos basándose en colocar los vértices en una forma circular, para luego conectar los vértices según indique la información cargada de los archivos de texto. Para dibujar aristas que van del mismo vértice de origen al mismo vértice de llegada, se hizo utilizando otro color y cambiando el grosor de la línea de esta arista.

   Solución: Utilizando mapas de bits y la clase Graphics de C# se pudo representar el grafo en un PictureBox, para ello se utilizó una clase llamada Coordenadas, que almacena la coordenada tanto en el eje X como en el eje Y de un vértice.

   Para determinar en qué posición del PictureBox estaría cada vértice, se utilizó la idea de dibujar los vértices en un círculo, para esto se determinó el radio en base a la altura del PictureBox, luego se determinó el centro del PictureBox tanto en X como en Y, luego se dividió 360 entre la cantidad de vértices del grafo, esto indicaba el grado en el cual debía ser dibujado el grafo, con el ángulo y el radio por medio de ley de senos se determinó la coordenada de Y y de X respectivamente. También se hizo uso de la librería Math, ya que era necesario usar la función matemática Sin(), además de la constante PI para hacer transformaciones de grados a radianes.

   La función se habilita por medio de un botón que se encuentra en el entorno gráfico, el usuario puede decidir si ver o no la representación gráfica del grafo.

(*): Esta parte han sido extras que se le decidió agregarle al proyecto.

Paso 4: Examinar la solución obtenida

Una vez teníamos ya todo finalizado era necesario generar todos los posibles casos y escenarios para verificar respuestas, para este paso fue necesario usar el libro de “Matemática Discreta y Combinatoria” del autor Ralph Grimaldi, específicamente en el capítulo 11 que es donde se toca el tema de grafos. Tomamos varios grafos de ese libro en la parte de isomorfismo y comparamos nuestra respuesta con la que el libro plantea.

DISEÑO

Diagrama de clases

El diagrama de clases se encuentra en la siguiente dirección dentro del proyecto: \Documentación\Diagrama de clases.png

Diagrama de flujo y Algoritmo de la matriz de adyacencia

Algoritmo:

1. Sea V1 la lista de vértices del primer grafo.
2. Sea V2 una de las permutaciones de la lista del segundo grafo.
3. Construir una matriz con el mapeo de los elementos de V1, V2, a esta matriz la vamos a llamar V3.
4. Multiplicar la matriz creada en el paso anterior (V3) con la matriz de adyacencia del segundo grafo. A esta matriz la llamaremos (V4)
5. Multiplicar la matriz generada con la multiplicación anterior por la matriz transpuesta de la matriz construida en el paso 3.
6. Si la matriz obtenida es igual a la matriz de adyacencia del primer grafo, entonces el mapeo de V1 en V2 es una función de isomorfismo.
7. Se repite el proceso con la siguiente permutación de la lista del segundo grafo.

Diagrama de flujo: