Proyecto Isomorfismo, Manual Técnico

José Alvarez, Oscar Lemus y Juan Aragón

Universidad Rafael Landívar

"Divide y vencerás…"

-Maquiavelo (Julio Cesar)

ÍNDICE

INTRODUCCIÓN

El presente manual pretende proporcionar la estrategia con la que se subdividió el problema para resolverlo de eficientemente, explicar los algoritmos desarrollados por los autores para buscar un isomorfismo además los detalles de la implementación de estos.

ANÁLISIS

Problema General: Como determinar si dos grafos son isomorfos, y en caso lo sean, como encontrar su función de isomorfismo. En caso no, las razones.

La complejidad de este problema radica en encontrar un algoritmo que sea capaz de hallar una función de isomorfismo sin necesitar permutar de forma incoherente o exagerada las opciones posibles. (Consultar algoritmo JJO más adelante en esta sección, desarrollado por los autores para resolver dicho problema)

ENTRADAS

- Dos grafos no dirigidos, no multigrafos: (G1 y G2) => los cuales son un conjunto de vértices y aristas: G1(V, E) & G1(V, E)

SALIDA

- Determinación de si el grafo es isomorfo o no.
    - En caso lo sea se obtiene la función de isomorfismo.
    - En caso no lo sea se obtienen las razones por las cuales no puede serlo.

BREAK DOWN DEL PROBLEMA

- Realizar una comparación de vértices.
- Realizar una comparación de aristas.
- Realizar un conteo de número de vértices con un determinado grado.
- Realizar una búsqueda de una función de isomorfismo
    - Cuadrar un par de vértices y sus respectivos adyacentes para cada vértice en el grafo.

CRITERIOS APLICADOS PARA DETERMINAR ISOMORFISMO

- Coincidencia en cantidad de vértices de ambos grafos.
- Coincidencia en cantidad de aristas de ambos grafos.
- Coincidencia en *tabla de número de vértices para determinado grado* de ambos grafos.
- Criterios aplicados en función JJO como coincidencia de adyacentes, entre otros; descritos más adelante en esta misma sección. (Estos criterios son más "pesados" y propios del algoritmo JJO también descrito más adelante).
- Existencia de función de isomorfismo. (Obtenida luego de aplicar JJO)

ANALISIS DEL **ALGORITMO JJO**[1] (PARA **BUSCAR UNA FUNCIÓN DE ISOMORFISMO**)

DESCRIPCIÓN INFORMAL

El algoritmo JJO es un algoritmo para encontrar una función de isomorfismo concebido y desarrollado por los autores de este proyecto.

Para un par de vértices **A** y **B** pertenecientes a G1 y G2 respectivamente, el algoritmo busca cuadrar sus adyacentes primero tomando un adyacente **a** de **A** y un adyacente **x** de **B.** (Primero llamando a un JJO(G1, G2) que llama internamente a JJO(A,B), el primero es una sobrecarga que recibe grafos de parámetros y el segundo vértices)

Recursivamente, aplica JJO a **a** y **x**. Si es posible encontrar una función de Isomorfismo retorna verdadero, a la vez que marca en cada iteración los dos vértices que va "cuadrando" en una tabla de isomorfismo. Si no es posible encontrar una función de Isomorfismo, retorna falso, e intenta aplicar JJO a **A** y **otro vértice de G2.** Este procedimiento lo realice hasta que se acaben los vértices de G2 (esto se debe a que A debe "cuadrar" con por lo menos un vértice de G2 si existe una función de isomorfismo).

Si la primera llamada de JJO retorna falso, no fue posible encontrar una función de isomorfismo. Si retorna verdadero, se habrá logrado modificar una tabla de isomorfismo que represente dicha función.

Las validaciones y criterios formales con los que valida la existencia de una función de isomorfismo se describen en las siguientes tres secciones.

CRITERIOS BASE APLICADOS POR JJO PARA DETERMINAR ISOMORFISMO

- **Chequear disponibilidad:** Se refiere a si los vértices adyacentes a un par de vértices A (en G1) y B (en G2) coinciden en el número de vértices no marcados previamente (disponibles o "azul") en la tabla de isomorfismo hasta ahora modificada/obtenida en la ejecución de las recursiones previas de JJO.
- **Chequear "*Red-Pairs*[2]":** Se refiere a si los vértices adyacentes que se encuentran "rojos" (ya marcados previamente en función de isomorfismo hasta ahora obtenida por JJO) de un par de vértices A (en G1) y B (en G2) ya encuentran una correspondencia en la tabla de isomorfismo.
  En otras palabras, se busca que la correspondencia en tabla de isomorfismo de un adyacente rojo de A, se encuentre en adyacentes a B.
- **Chequear grados:** Se refiere a cuadrar si un par de vértices A (en G1) y B (en G2) poseen el mismo grado.

[1] JJO significa "Juan", "José" y "Oscar" en honor a los autores.
[2] *"Red-Pairs" significa vértices marcados (no disponibles) o vértices "rojos" a lo largo de este documento. Los contrarios se les denomina "azules" o disponibles. Términos que son parte del algoritmo JJO. Ver porción de iteraciones de JJO en anexo A.*

ALGORITMO FORMAL

A partir de ahora se le llamará **"JJO Global"** a la **sobre carga que recibe grafos (G1 y G2) de parámetros**, y **"JJO Específico"** a la **sobrecarga que recibe vértices (Ay B) de parámetros.**

JJO Global:

(Diagrama de flujo en Anexo A ilustración 2)

Se asume que se tiene una única tabla de Isomorfismo disponible en cualquier nivel de recursión. (Consultar detalles de implementación en sección Diseño)

1. Tomar el primer vértice de G1 y el primer vértice de G2.
2. Aplicar JJO Específico a vértices seleccionados.
3. Si JJO Específico retorna falso, se verifica si existe otro vértice en G2 y se vuelve al paso 2 seleccionando ahora ese siguiente vértice de G2 (se conserva seleccionado el primer vértice de G1.
4. Si JJO Específico retorna verdadero, se ha encontrado una función de isomorfismo, por lo que JJO Global retorna también verdadero.

JJO Específico:

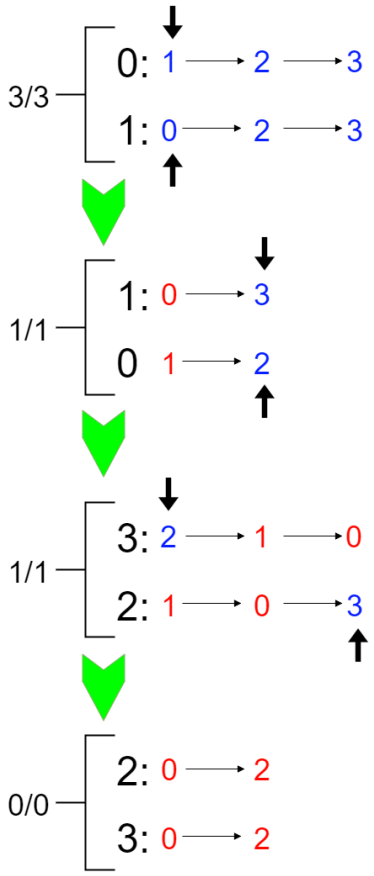
(Diagrama de flujo en Anexo A 3)

Nuevamente se asume que se tiene la misma única tabla de Isomorfismo disponible en cualquier nivel de recursión.

1. Comparar cantidad de vértices disponibles (adyacentes no marcados en tabla) entre A y B.
2. Verificar que disponibilidades sean iguales y disponibilidades no sean 0. Si NO se cumple lo anterior, ir al **paso 3**. Si se cumple, ir al **paso 5**.
3. Verificar que disponibilidad de ambos vértices sea igual a 0. Si NO se cumple, ir al **paso 11**. **Si se cumple (empieza probable caso trivial), continuar.**
4. Verificar que parejas de vértices no disponibles ("Red-Pairs") cuadren o encuentren una correspondencia. Si NO se cumple lo anterior, ir al **paso 11**. **Si se cumple (caso trivial), ir al paso 12.**

5. Verificar que parejas de vértices no disponibles ("*Red-Pairs*") cuadren o encuentren una correspondencia (**No es caso trivial**, es distinta la forma de llegar a 5). Si NO se cumple lo anterior, ir al **paso 11**. Si se cumple, continuar.
6. Modificar disponibilidad de vértices actuales (ahora se consideran "rojos") y marcarlos en la tabla de isomorfismo.
7. Seleccionar el primer vértice disponible ("azul") de A más no el último del listado.
8. Seleccionar el primer vértice disponible de B y verificar que sea disponible ("azul"). Si NO lo es, ir al **paso 10**. Si lo es, continuar.
9. Aplicar JJO Específico (aquí aplica la recursividad) a vértices seleccionados. Si JJO Especifico retorna verdadero, ir al **paso 11**. Si retorna falso, ir al **paso 10**.
10. Si existe algún elemento restante en B, seleccionarlo y volver al **paso 9**.
11. **Retornar falso** en este nivel de recursión. Fin de este nivel de recursión.
12. **Retornar verdadero** en este nivel de recursión. Fin de este nivel de recursión.

EJEMPLO DE UNA PORCIÓN DE ITERACIÓN DE JJO



VENTAJAS DE ESTE ANALISIS Y DE JJO

La ventaja del enfoque presentado radica en como los criterios aplicados permiten determinar una función de isomorfismo de forma inteligente, sin necesidad de permutar todas las posibilidades.

DISEÑO

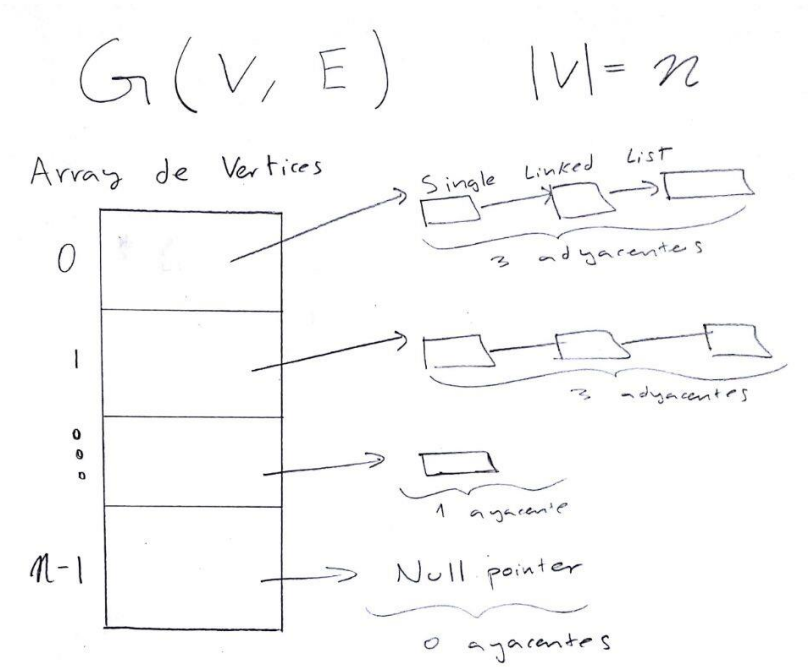
DETALLES DE LA TECNOLOGÍA UTILIZADA

- SOFTWARE UTILIZADO: Visual Studio 2015 Community
- LENGUAJE UTILIZADO: C++

PATRÓN DE DISEÑO

REPRESENTACIÓN DEL GRAFO COMO UNA LISTA DE ADYACENCIA

La representación de un grafo como una lista de adyacencia consiste en un arreglo de punteros hacia listas simplemente enlazadas, donde cada posición del arreglo representa un vértice en el grafo, y cada nodo de las listas representa los adyacentes al vértice que representa la posición del arreglo.



La principal operación ejecutada por una estructura de datos de lista de adyacencia es reportar los vecinos de un vértice dado, entre otros métodos como determinar si un vértice es adyacente o devolver un vértice en un índice especifico (estos últimos índices son propios del concepto de una lista simplemente enlazada y dependen del orden en que se añaden las aristas, no del grafo; sin embargo, es útil tener este método para recorrer los adyacentes)

PARADIGMA DE PROGRAMACIÓN

Programación orientada a objetos.

DIAGRAMA DE CLASES

Consultar anexo A página vii.

FUNCIONES Y SUBRUTINAS IMPORTANTES

CGraph bool Isomorphism(G1, G2): Determina si dos grafo son isomorfos.

CGraph bool CompareVerticesQuantity(G1, G2):Determina si dos grafos poseen la misma cantidad de vertices. Utilizada en Isomorphism().

CGraph bool CompareEdgesQuantity(G1, G2): Determina si dos grafos poseen la misma cantidad de aristas. Utilizada en Isomorphism().

CGraph bool CompareVerticesPerDegreeTables(G1, G2): Determina si las *tablas de número de vértices para determinado grado* coinciden. Utilizada en Isomorphism().

CGraph int* (array) GetVerticesQuantityPerDegreeTableClassification(G): Devuelve una *tabla de número de vértices para determinado grado* de un grafo. Utilizada en CompareVerticesPerDegreeTables().

CGraph bool JJO(G1, G2): Esta sobrecarga de JOO arranca el algoritmo JOO. Se le considera "JOO Global" . // Sobrecarga "Global"

CGraph bool JJO(vertexArrayA, vertexArrayB, vertexA, vertexB): Esta sobrecarga ejecuta el corazón del algoritmo JJO, se le llama desde JOO Global. Se le considera "JJO local"

CGraph int AvailableAdjacents(vertexV, int table): Devuelve la cantidad de vértices disponibles (no marcados o "azuls") de un vértice. Utilizada en algoritmo JJO. (Consultar análisis de JJO).

CGraph bool Assigned(vertexV, int table): Determina si un vértice ya está marcado en la table de isomorfismo estatica (Consultar sección variables y constantes más adelante). Utilizada en algoritmo JJO.

CGraph bool CheckRedPairs(vertexA, vertexB): Determina si los vértices adyacentes de dos vértices A y B ya marcados encuentran una correspondencia en la tabla de isomorfismo estatica. Utilizada en JJO.

CGraph void CrossOutInTable(vertexA, vertex): Marca en tabla de isomorfismo (static isomorphismTable) y en las (static CrossOutTable) 's. (Consultar sección de variables y constants más adelante). Utilizada en JJO.

CGraph void UncoverInTable(vertexA, vertexB): Desmarca de tabla de isomorfismo lo ya marcado por CrossOutInTable(). Utilizada en JJO.

VARIABLES Y CONSTANTES DEL PROGRAMA, Y SU UTILIZACIÓN

Tabla de isomorfismo estática (static int* isomorphismTable):

Esta tabla o arreglo es de mucha importancia para la implementación de JJO, cada recursión intenta generar y modificarla hasta lograr la función de isomorfismo de la manera más determinista y eficiente posible.

Las posiciones representan vértices de G1 y los valores en dichas posiciones su correspondiente vértice isomorfo en G2.

Es estática por el hecho de que debe ser independiente a cualquier nivel de recursión.

Tabla de marcados estática de G1 (static bool* G1_CrossOutTable)

Y Tabla de marcados estática de G2 (static bool* G2_CrossOutTable)

Esta tabla o arreglo es prácticamente una forma de consultar atributos de un vértice sin relacionarlos al vértice. Se hizo independiente al objeto vértice porque su utilidad radica únicamente dentro del algoritmo JJO.

Es estática por el hecho de que de que debe ser independiente a cualquier nivel de recursión.

ANEXO A

DIAGRAMA DE FLUJO

PROCESO GLOBAL DEL PROGRAMA

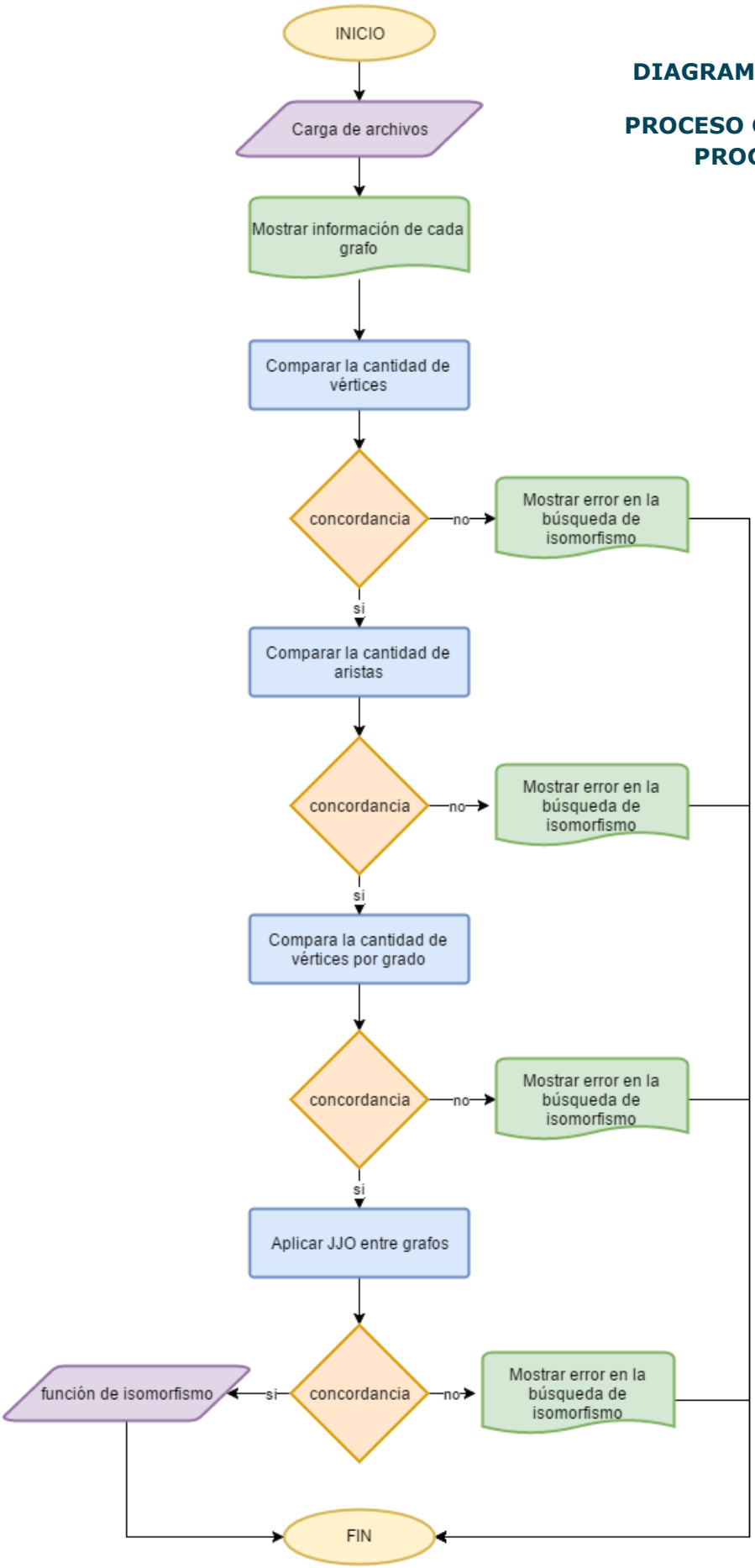


ILUSTRACIÓN 1

DIAGRAMA DE FLUJO

JOO "Global"

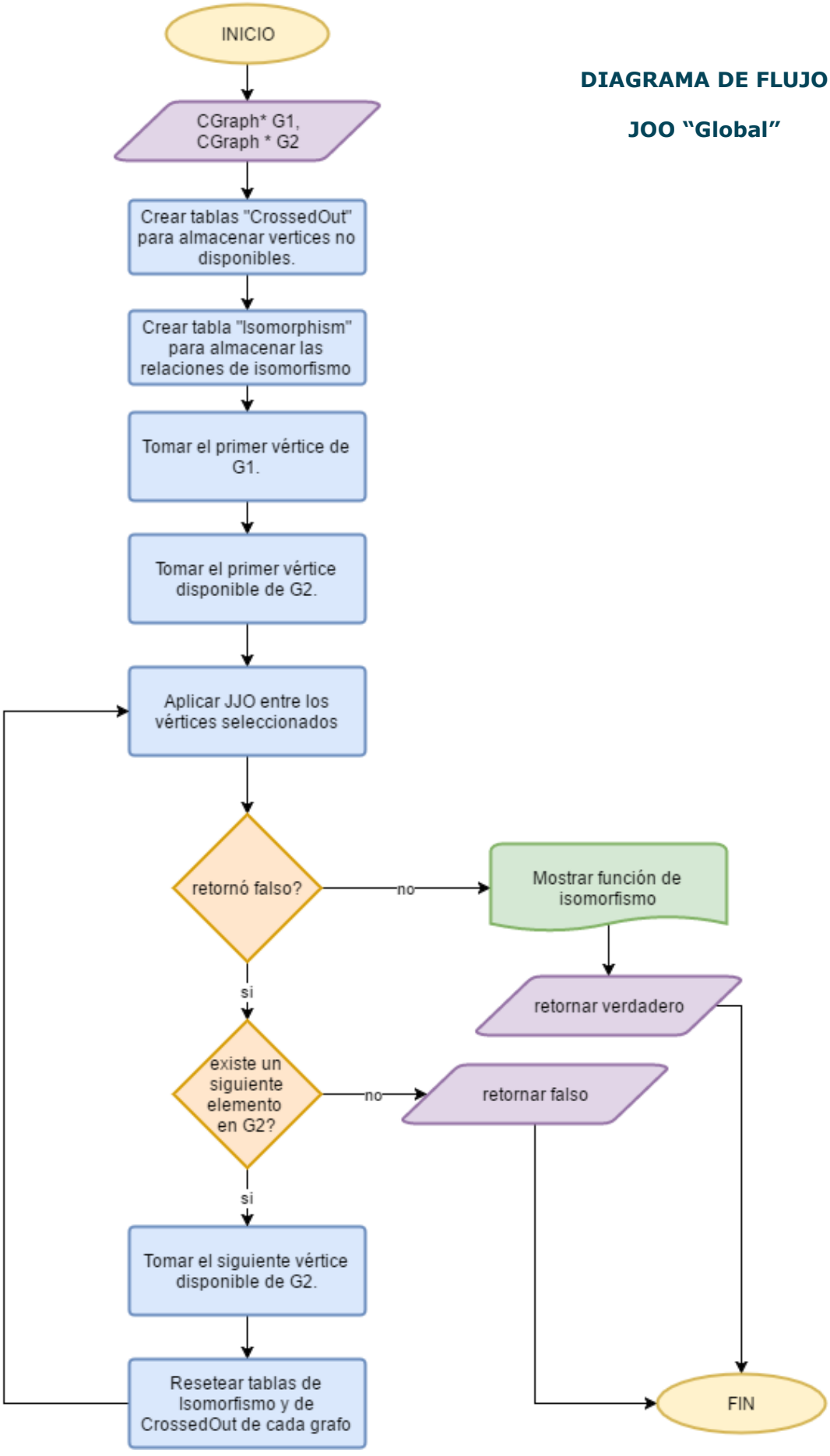


ILUSTRACIÓN 2

DIAGRAMA DE FLUJO

JOO "Específico"

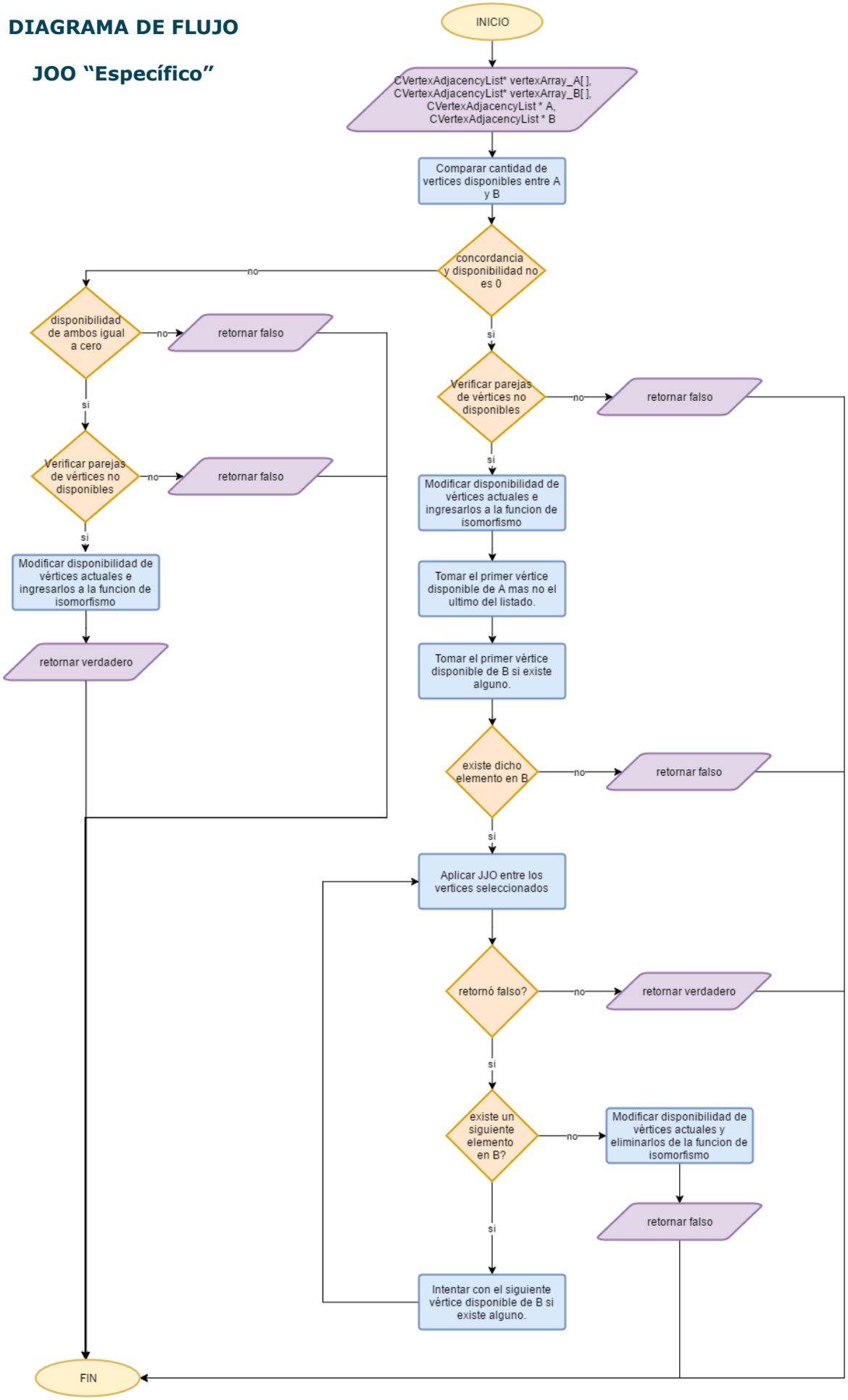
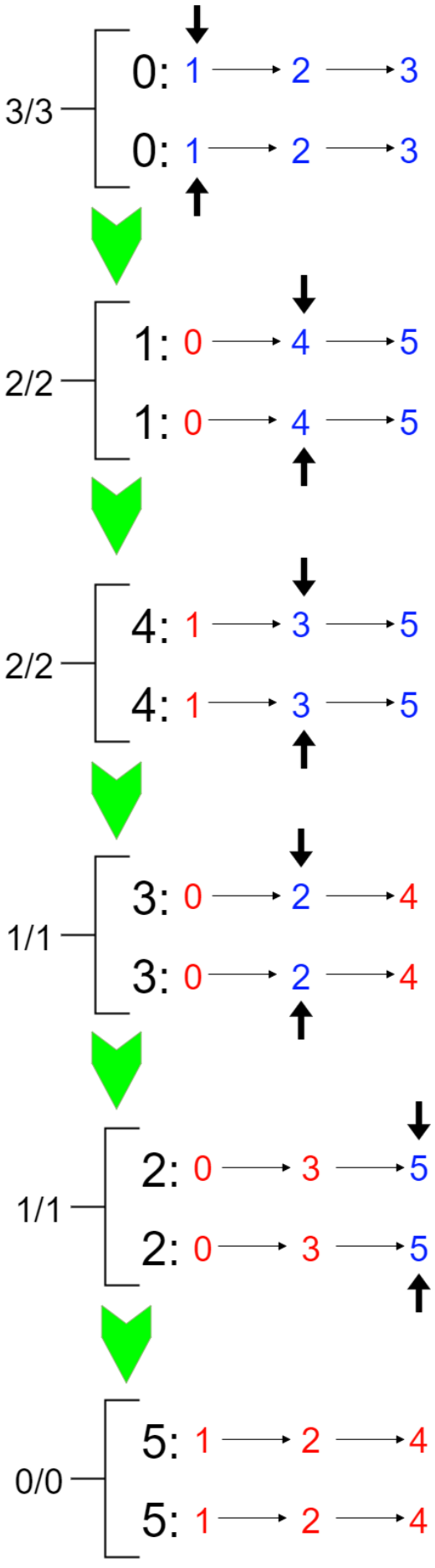


ILUSTRACIÓN 3

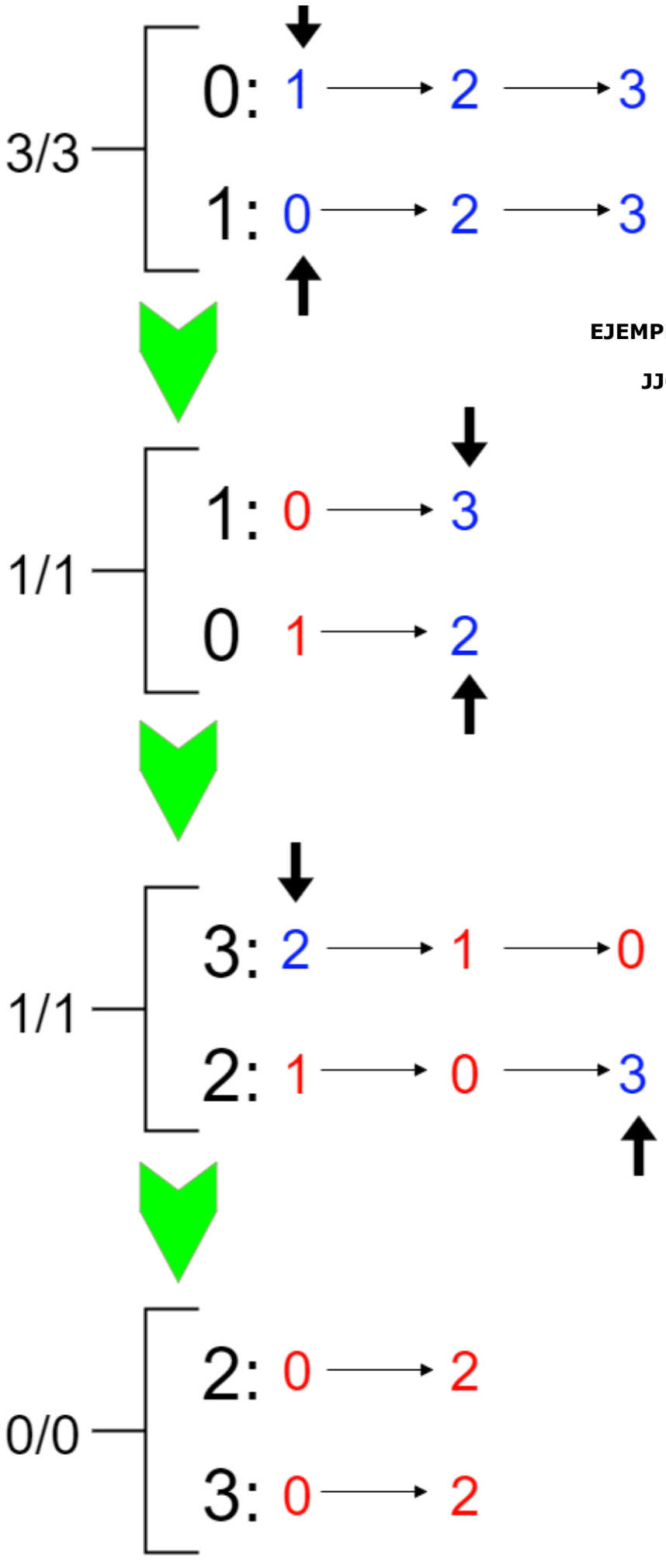
El algoritmo se comportaría de esta forma si se ingresa el archivo **GrafoOriginalA.txt dos veces**. (Dos grafos totalmente iguales). Naturalmente, esta iteración es con fines ilustrativos, ya que no representa mayor complejidad ingresar el mismo archivo dos veces.



EJEMPLO DE LLAMADA A JJO ESPECIFICO

ILUSTRACIÓN 4

Una porción del algoritmo se comportaría de esta forma si se ingresan los archivos: Grafo11A.txt y Grafo11B.txt



EJEMPLO DE LLAMADA A JJO ESPECIFICO

DIAGRAMA DE CLASES

CGraph **Clase**
Campos
- *vertexArray* - *vertexQuantity*
Métodos
- *~CGraph* - *AddEdge* - *Adjacent* - *Assigned* - *AvailableAdjacents* - *CGraph* - *CheckIsomorphismTable* - *CheckIsomorphismTableAdjacentVertices* - *CheckRedPairs* - *CompareEdgesQuantity* - *CrossOutInTable* - *Degree* - *GetEdgesQuantity* - *GetSumOfVerticesDegrees* - *GetVertexDegreeTable* - *GetVerticesQuantityPerDegreeTableClasification* - *Isomorphism* - *JJO (+1 overload)* - *LeastDistanceVertex* - *Neighbors* - *PrintGraphsInfo* - *PrintVertexDegreeTable* - *RemoveEdge* - *UncoverInTable* - *VertexQuantity*

CVertexAdjacencyList
Clase

Campos

- *head*
- *size*
- *value*

Métodos

- *~CVertexAdjacencyList*
- *Add*
- *CVertexAdjacencyList*
- *ElementAtIndex*
- *IsAnElement*
- *PrintList*
- *SetVertexValue*
- *Size*
- *Value*

CReadInput
Clase

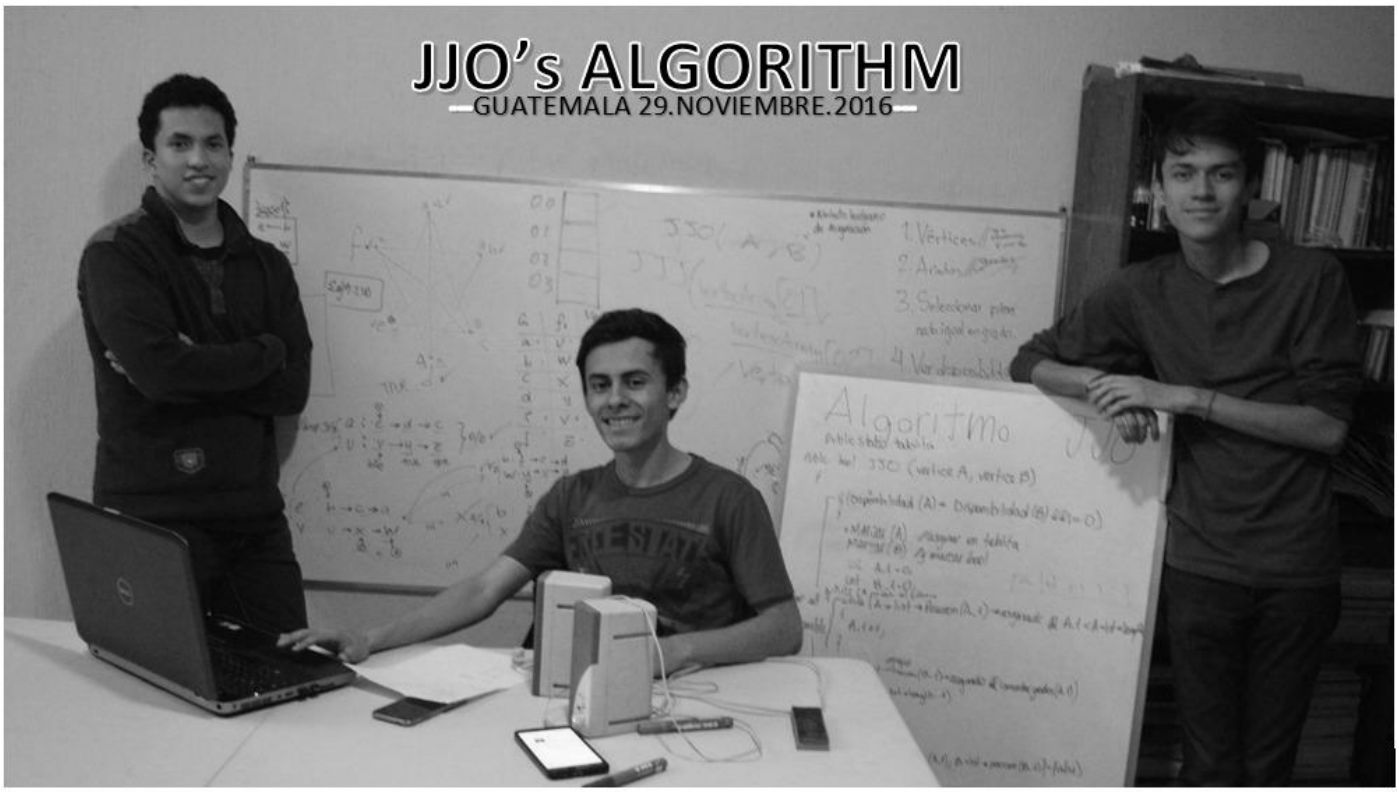
Métodos

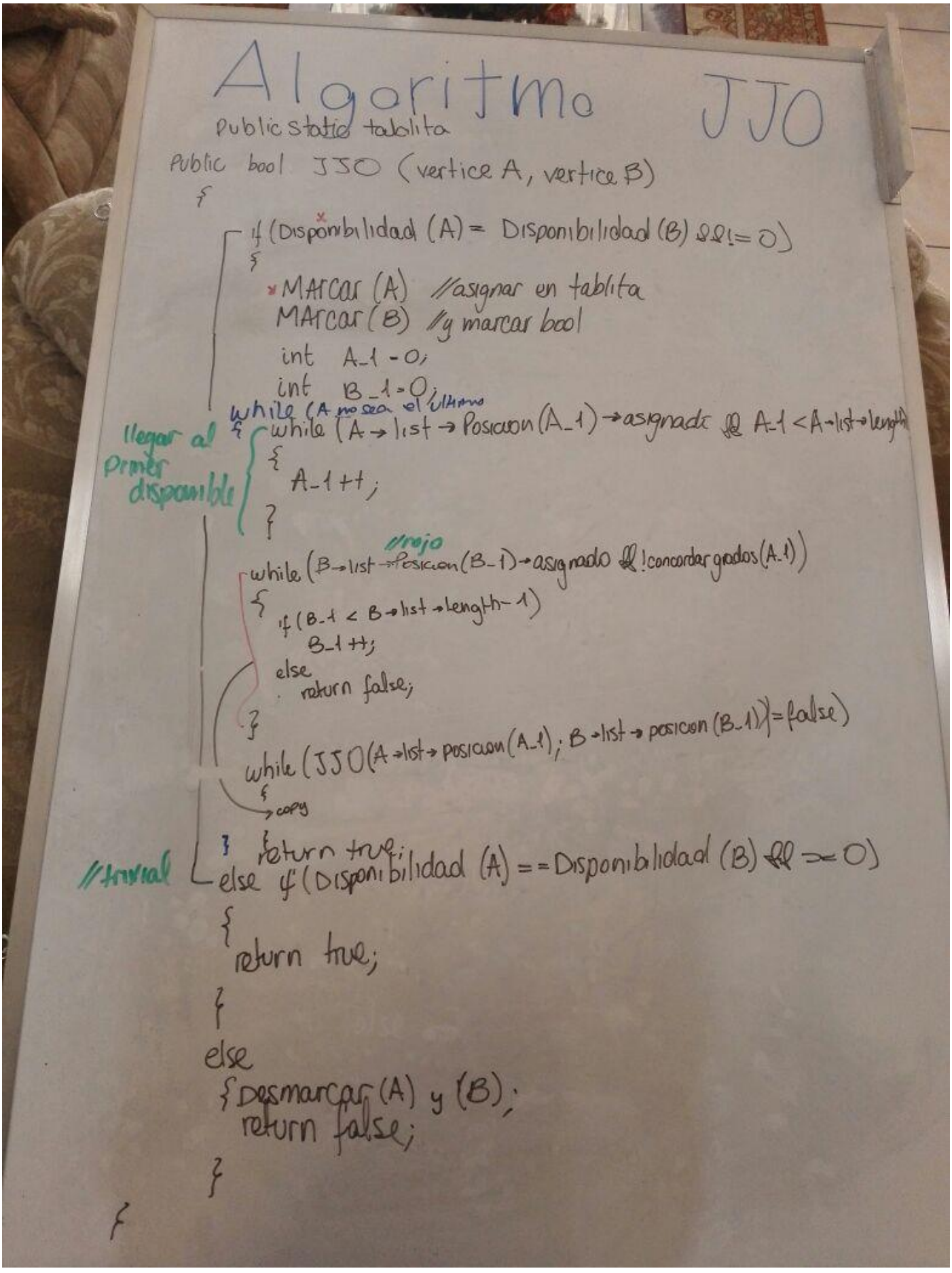
- *~CReadInput*
- *CReadInput*
- *FileSize*
- *GetVertexQuantity*
- *ReadInput*

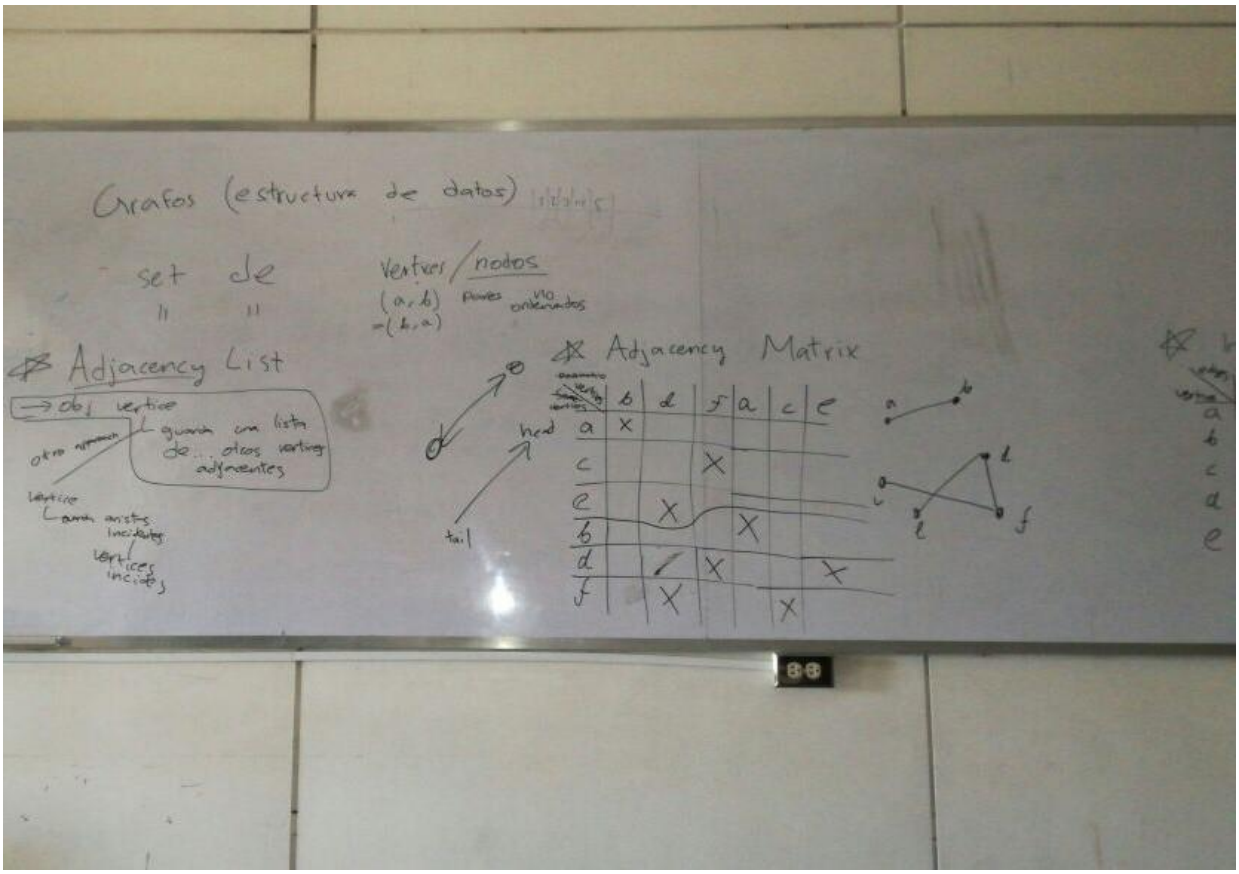
ANEXO B

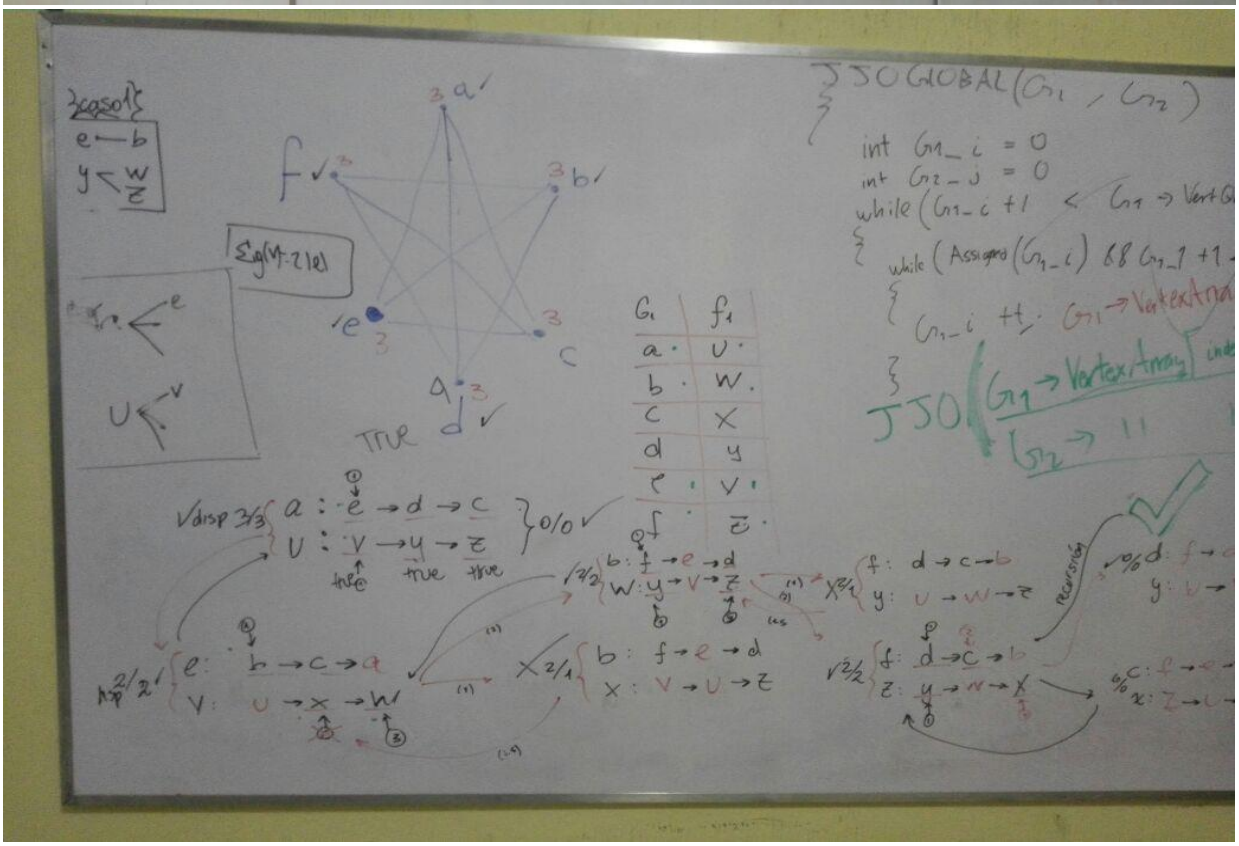
(Borradores del análisis y la implementación)

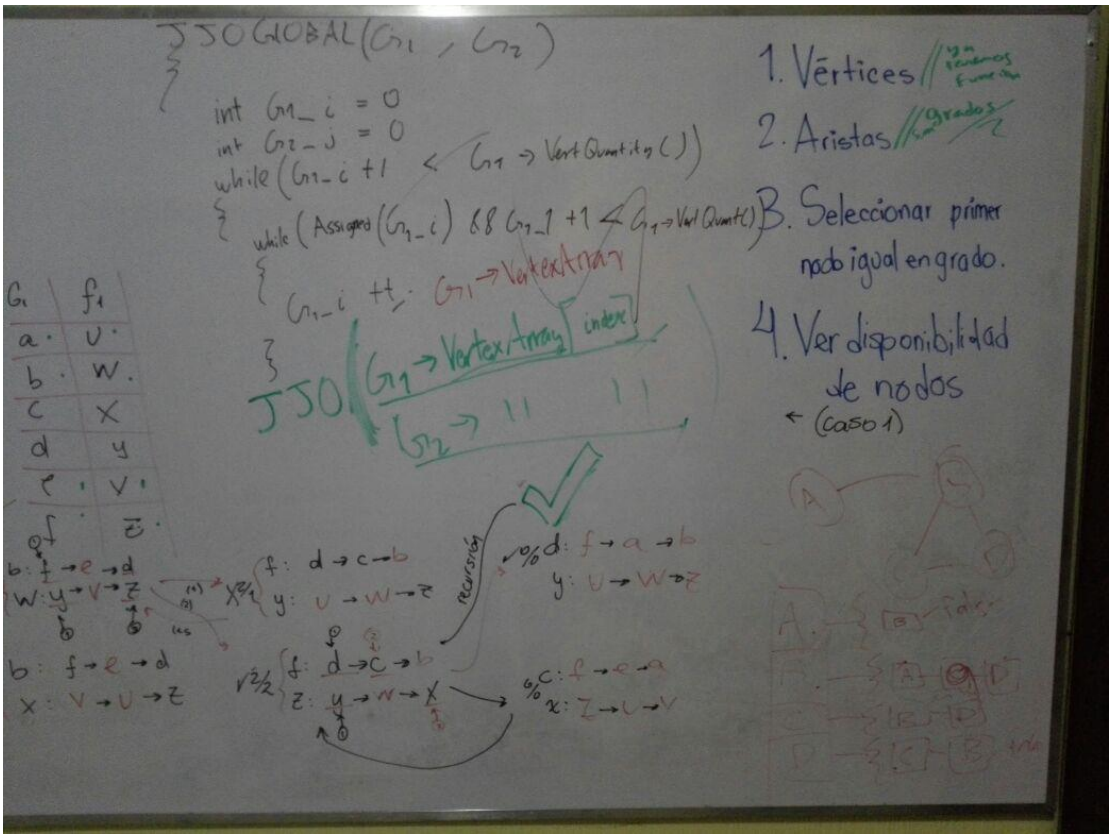
x

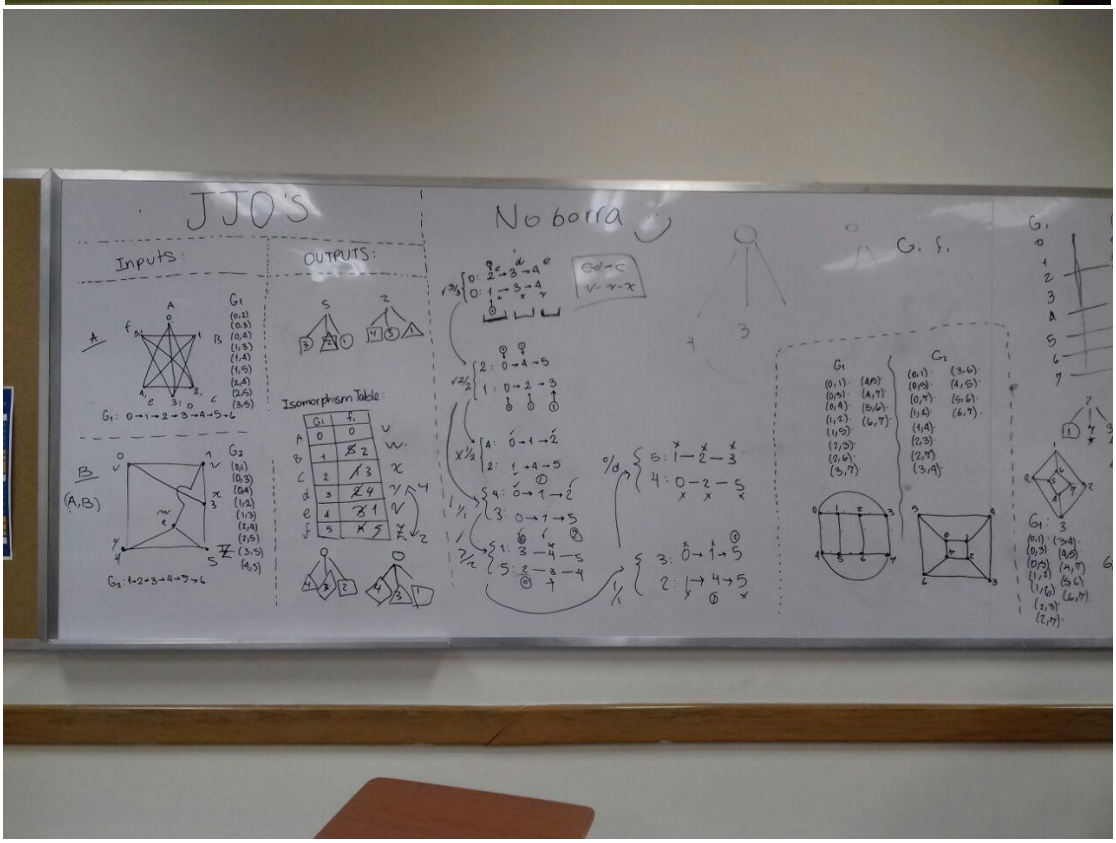


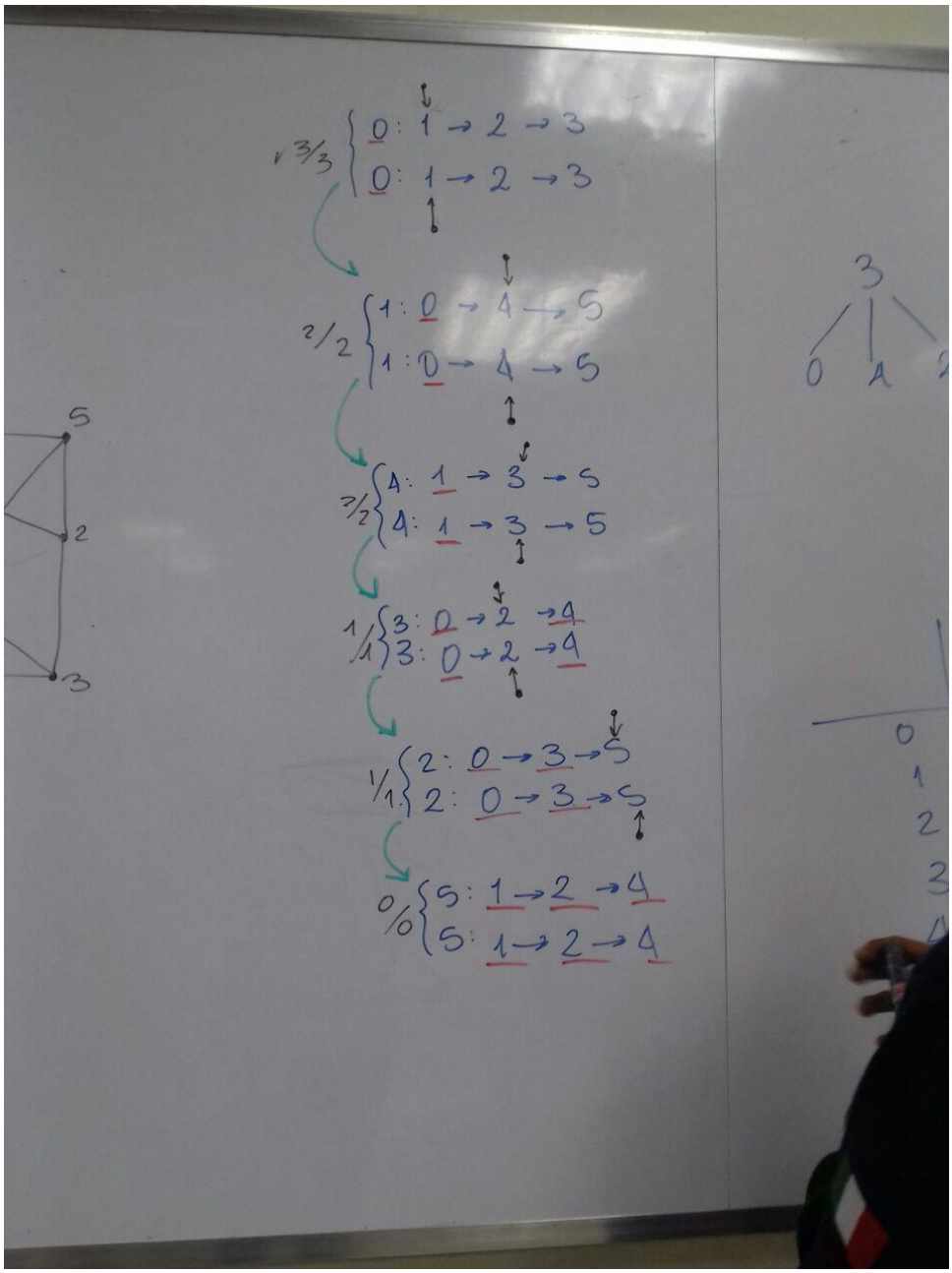

Algoritmo JJO
Public static tablita
Public bool JJO (vertice A, vertice B)
{
if (Disponibilidad (A) = Disponibilidad (B) && != 0)
{
MArcar (A) //asignar en tablita
MArcar (B) //y marcar bool
int A_1 = 0;
int B_1 = 0;
while (A no sea el último
llegar al primer disponible
while (A->list->Posicion(A_1)->asignado && A_1 < A->list->length
{
A_1++;
}
//rojo
while (B->list->Posicion(B_1)->asignado && !concordar grados(A_1))
{
if (B_1 < B->list->length-1)
B_1++;
else
return false;
}
while (JJO(A->list->posicion(A_1); B->list->posicion(B_1)) != false)
{
copy
}
return true;
//trivial
else if (Disponibilidad (A) == Disponibilidad (B) && >= 0)
{
return true;
}
else
{Desmarcar (A) y (B);
return false;
}
}


Grafos (estructura de datos)
set de
Vertices / nodos
(a,b) pares no ordenados
=(b,a)
Adjacency List
vértice
guarda una lista de... otros vertices adyacentes
otro vértice
Cuenta aristas incidentes
Vértices incidentes
Adjacency Matrix
head
tail
a b c d e f


caso1
TRUE
JJO GLOBAL
int
while
JJO


JJO GLOBAL (G1, G2)
int G1_i = 0
int G2_j = 0
1. Vértices
2. Aristas
3. Seleccionar primer nodo igual en grado.
4. Ver disponibilidad de nodos
(caso 1)
JJO
recursión


JJO'S
Inputs:
OUTPUTS:
Isomorphism Table:
No borra
G. S.


0: 1 → 2 → 3
0: 1 → 2 → 3
1: 0 → 4 → 5
1: 0 → 4 → 5
4: 1 → 3 → 5
4: 1 → 3 → 5
3: 0 → 2 → 4
3: 0 → 2 → 4
2: 0 → 3 → 5
2: 0 → 3 → 5
5: 1 → 2 → 4
5: 1 → 2 → 4
5
2
3
0
A