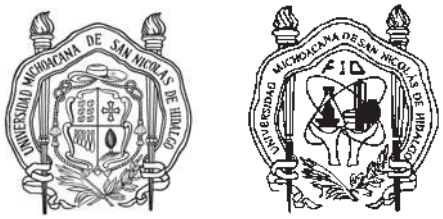
UNIVERSIDAD MICHOACANA DE SAN NICOLAS DE HIDALGO

FACULTAD DE INGENIERIA QUIMICA



"IMPLEMENTACION EN COMPUTADORA DE ALGORITMOS DE RASGADO DE DIAGRAMAS DE FLUJO DE PROCESOS QUIMICOS"

TESIS

que para obtener el Título de

INGENIERO QUIMICO

Presenta

HORTENCIA MARTINEZ CORREA

Asesor de Tesis

DR. MEDARDO SERNA GONZALEZ

Marzo del 2011

AGRADECIMIENTOS

Agradezco inmensamente a **Dios** el haberme permitido llegar a este día con mucho ánimo y con muchas ganas de seguir adelante para lograr muchas metas más, por las experiencias amargas y gratas, por las decisiones grandes que tuve que tomar para poder estar aquí, más sin embargo llegar al momento de culminación de esta tesis lo justifica todo. Gracias por las bendiciones de las que he gozado a lo largo de mi vida.

A mi asesor **Dr. Medardo Serna González**, por depositar su confianza en mí para la realización de este trabajo y por sus palabras de aliento cuando más las necesitaba para no desistir. Agradezco también sus fines de semana empleados en la revisión de mi trabajo y por no dejarme sola en este proceso a pesar de sus muchas responsabilidades. No sabe lo agradecida que estoy por la paciencia que me tuvo cada vez que me aparecía después de largos periodos de ausencia. Mil gracias por compartir sus conocimientos conmigo y por creer en mí.

A la **Dra. Alicia Ortiz** por haber estado siempre al pie del cañón y pendiente de mí desde que la conozco, pero sobre todo en esta etapa que no fue nada fácil y que hubiera sido muy complicada si no hubiera contado con su valiosísimo apoyo. Gracias por confiar y creer en mí. Ni con todo el dinero del mundo le podría pagar todo lo que hasta el día de hoy me ha brindado. Le estaré eternamente agradecida. La quiero mucho.

A mis amigos **Luis F. Guzmán Nateras** y **Sergio J. Domínguez González** por haberme dedicado parte de su tiempo y esfuerzo para poder avanzar en mi tesis. Sin su ayuda chicos esto no habría sido posible al día de hoy. Gracias por haber alumbrado mi camino con sus conocimientos y por la disponibilidad que mostraron a pesar de sus ocupaciones. Esta tesis también es suya.

DEDICATORIAS

Para mi mami **Concepción Correa** quien ha pagado con creces el costo de mi sueño de realizarme profesionalmente pero que a pesar de eso no deja de brindarme su apoyo, de ser mi consejera y, sobre todo, no deja de ser mi madre, aquella que siempre me espera con los brazos abiertos. Para mi representas fortaleza, fuiste muy valiente y tenaz cuando te viste sola con 6 hijos y sin importar eso nos diste lo mejor que pudiste. Eres maravillosa. Te amo linda.

Para mi papá **Benigno Martínez** quien a pesar del poco tiempo que pudimos compartir me dejó una enseñanza muy importante y muy valiosa que el día de hoy me llevó a culminar uno de mis sueños más grandes. Siempre te vi como un luchador, como un gran hombre, de pie ante los peores momentos y esa imagen nadie la podrá borrar de mi mente. Fuiste muy admirado y un ejemplo a seguir para mucha gente, para mí fuiste y seguirás siendo mi ídolo. Espero que en donde quiera que estés te sientas orgulloso de mí.

Para mi hermana **Bety** que nunca dejó de confiar en mí y que me brindó tantas cosas como pudo y que hasta la fecha lo hace. Mil gracias hermosa por haberme dedicado parte de tu vida y por haberte entregado por mucho tiempo a nosotros, tus hermanos, en cuerpo y alma. Quiero decirte que tú también eres mi ejemplo a seguir por ser una luchadora

A mis hermanas **Lalis y Cristy** y a mi hermano **Martín**, quienes me hacen el día cada vez que los veo. Tantas cosas que hemos compartido. Somos una familia algo extraña pero somos una gran familia. Gracias por estar conmigo a pesar de la enorme distancia que a veces nos separa y por quererme tanto. Sé que siempre contaré con ustedes y sepan que también ustedes pueden contar conmigo en cualquier momento. Los amo.

A la **Dra. Alicia Ortiz** que ha sido como una segunda madre para mí durante estos últimos años que me he encontrado fuera de mi hogar. Me ha tendido la mano en muchos momentos difíciles y lo ha hecho desinteresadamente, por eso y muchas cosas más estoy más que agradecida con usted y le agradezco a Dios haberla puesto en mi camino, no pudo haberme mandado un ángel mejor. Que Dios la bendiga siempre.

A mi hermana **Car** y a su esposo **Arturo** quienes me abrieron las puertas de su casa y me brindaron el apoyo que necesitaba para comenzar a trabajar en la realización de este sueño brindándome las herramientas necesarias para hacerlo. Muchas gracias por su apoyo. Que Dios los colme de bendiciones.

A ti **Julio César López Malfabón** por ser mi compañero, confidente, amigo, por brindarme palabras de aliento para que no tirara la toalla cuando me sentía cansada de todo, por tu confianza y apoyo incondicionales y, sobre todo, por darme tu amor y recibir el mío. Gracias también a ti, corazón, he llegado aquí. Este logro es tuyo también.

A las **DIDTH´s** y a los **Trones**, mis grandes amigos, que me hacían olvidarme un rato de todo y que compartieron muchas cosas conmigo. Los quiero mucho y espero poder reunirme con ustedes pronto. Gracias por alentarme a llegar hasta aquí.

A mi amigo y paisano **Rafael Piña** quien con una llamada, sin él saberlo, le dio fuerza a mi decisión y me hizo ver la importancia de dar este paso.

A todos mis amigos y personas que de una u otra forma me brindaron su apoyo y estuvieron conmigo a lo largo de este camino.

RESUMEN

En este trabajo de tesis se presenta un procedimiento implementado en computadora para determinar el conjunto mínimo de corrientes de corte, que permite el rompimiento de todos los ciclos de procesos químicos con recirculaciones, lo cual es necesario para llevar a cabo la simulación de los mismos usando la estrategia modular secuencial. Este procedimiento consiste en dos etapas. En la primera etapa se identifican todos los ciclos simples del diagrama de flujo de información (DFI) del proceso bajo estudio, usando un algoritmo basado en la técnica de búsqueda en profundidad prioritaria. En la segunda parte, el conjunto mínimo de corrientes de corte es obtenido usando un modelo de programación entera y un método heurístico. Ambos algoritmos están basados en la matriz de ciclos. Las corrientes de corte así determinadas rompen todos los ciclos y, por consiguiente, generan procesos con estructuras acíclicas que tienen asociado un orden de precedencia de las unidades de proceso, el cual se puede identificar a fin de establecer la secuencia en la que éstas se deben calcular de manera secuencial e iterativa. Para tal efecto, las corrientes de corte se dividen en dos partes que están vinculadas por un módulo de convergencia. En cada iteración, la salida de este módulo se convierte en la corriente supuesta, mientras que la corriente de entrada pasa a ser la corriente calculada. Las variables independientes de todas las corrientes de salida de los módulos de convergencia se deben suponer para iniciar los cálculos iterativos de simulación, los cuales concluyen cuando las corrientes calculadas son iguales a las corrientes supuestas que se actualizan en cada iteración de conformidad al método de convergencia utilizado. Por lo tanto, esta aproximación simplifica la solución del problema de simulación de procesos con recirculaciones.

La identificación de ciclos se realiza usando un algoritmo de búsqueda en profundidad, que se implementó en computadora usando el lenguaje JAVA. Por otro lado, la selección de corrientes de corte, que se formula como un problema de programación entera, se resolvió usando el paquete GAMS, mientras que el método heurístico y el programa para determinar el orden de cálculo también se implementaron en computadora usando el lenguaje JAVA.

Se presentan catorce casos de estudio que han sido reportados previamente en la literatura. Los resultados de estos ejemplos se muestran en tablas para permitir su fácil interpretación. La selección de corrientes de corte reduce la complejidad del análisis

matemático de procesos químicos con recirculaciones, lo que presenta ventajas como la optimización de recursos y la disminución del tiempo de los estudios de gabinete.

INDICE

INDICE DE FIGURAS

INDICE DE TABLAS

CAPITULO 1
INTRODUCCCION

1.1 Generalidades

Una de las tareas más importantes encomendada a los ingenieros químicos es el diseño preliminar de un proceso o el análisis de un proceso ya existente, con el propósito de explorar distintas condiciones de operación y estructuras de proceso a fin de encontrar los mejores diagramas de flujo de procesos nuevos o los reajustes a realizar para mejorar el funcionamiento de procesos existentes.

La mayoría de los procesos se caracterizan por la presencia de corrientes de recirculación debido a que, en general, los reactores químicos no operan al 100% de conversión. Por lo tanto, por razones económicas y ambientales es preciso recuperar y recircular los reactivos no consumidos. En algunos casos, la operabilidad de los procesos demanda el uso de corrientes de purga para evitar la acumulación de sustancias indeseables. Por consiguiente, los procesos químicos son, comúnmente, sistemas complejos formados por diversas unidades de proceso interconectadas por corrientes que forman ciclos o lazos de recirculación.

La división de un sistema complejo de gran dimensión en subsistemas más pequeños o bloques, que se deben resolver por separado en una secuencia global determinada, es una estrategia acertada para simplificar la solución del problema de simulación de procesos químicos en estado estacionario empleando la técnica modular secuencial.

Cuando ninguno de los bloques encontrados en la etapa de descomposición contiene más de un nodo, el correspondiente diagrama de flujo de información resultante no tiene ciclos y su simulación es simple. Si éste no es el caso, no hay modo de dividir el problema de simulación en subproblemas más pequeños. Esto se debe a que el bloque resultante está constituido por dos o más nodos que están conectados por corrientes de recirculación y, por consiguiente, son interdependientes.

La solución simultánea de este tipo de bloques aparentemente es sencilla haciendo uso de la computadora y técnicas numéricas actualmente disponibles; no obstante, implica un esfuerzo de cálculo considerable si se compara con una estrategia de cálculo secuencial, debido a que la dificultad de resolver simultáneamente un conjunto de ecuaciones aumenta exponencialmente con el número de ellas (Mah, 1990). Además, si los valores de los estimados iniciales de las variables independientes están lejos de la solución, entonces el método numérico utilizado puede tener problemas de convergencia. Por lo tanto, es deseable encontrar un conjunto mínimo de corrientes de corte que permitan el rompimiento de todos los ciclos de cada bloque con nodos interdependientes, a fin de simular en forma secuencial e iterativa los nodos que lo conforman. Este tipo de nodos representan, de acuerdo a la teoría de grafos, a los procesos químicos con recirculaciones. Es por ello que este problema es comúnmente conocido como rasgado y ordenamiento de procesos cíclicos.

La solución del problema de rasgado y ordenamiento de cada bloque incluye:

- La generación de todos los ciclos simples.
- La determinación y selección del número mínimo de corrientes de corte para romper todos los lazos de recirculación.
- El ordenamiento del sistema acíclico resultante, a fin de establecer la secuencia de cálculos iterativos de los nodos o unidades del proceso.

En este trabajo, la identificación de ciclos se realiza utilizando un algoritmo de búsqueda en profundidad, mientras que la selección de corrientes de corte se determina usando dos formulaciones: la primera es representada por un modelo de programación entera (Pho y Lapidus, 1973) y la segunda se propone en este trabajo de tesis como un método heurístico, que se basa en los parámetros rango de ciclo y frecuencia de arco de la matriz de ciclos. Para el ordenamiento de los cálculos secuenciales del proceso acíclico se aplica un algoritmo basado en la eliminación de nodos sumidero.

1.2 Antecedentes

Debido a la necesidad que ha surgido de contar con una herramienta que permita optimizar el tiempo dedicado al análisis de los procesos químicos, a lo largo de los años se han venido desarrollando algoritmos para determinar el número mínimo de corrientes de corte de un diagrama de flujo de procesos con recirculaciones. Sargent y Westerberg (1964)

aplicaron la programación dinámica para encontrar un conjunto de corrientes de corte con el mínimo número de variables de recirculación; sin embargo, esta aproximación no pudo superar adecuadamente el problema de dimensionalidad. En 1965, Norman utilizó la *matriz de adyacencia* para representar los nodos de los diagramas de flujos de información y encontrar corrientes de recirculación (Norman, 1965). Otro de los primeros trabajos (Lee y Rudd, 1966) presenta procedimientos simples para determinar el número mínimo de parámetros de recirculación en procesos complejos. Por otro lado, las aportaciones realizadas en los años 70´s (Tiernan, 1970; Weinblatt, 1972; Johnson, 1975; Read y Tarjan, 1975; Reingold, 1977) relacionadas con algoritmos de búsqueda en profundidad prioritaria de ciclos simples, han resultado ser muy eficientes y hoy en día forman parte de las ideas y conceptos bajo los cuales se ha formulado el algoritmo de búsqueda de ciclos simples que se describe en el presente trabajo de tesis. La esencia de este método consiste en seleccionar un nodo del Diagrama de Flujo de Información (DFI) y formar una lista de nodos al trazar un camino en la dirección del flujo, de nodo a nodo, hasta encontrar un nodo que no tenga arcos de salida.

El desarrollo de los métodos aplicados para la solución del problema de identificación del conjunto mínimo de corrientes de corte se ha venido dando gracias a los avances matemáticos relacionados con este tema. Los algoritmos propuestos por Barkley y Motard (1972) y Pho y Lapidus (1973) usaron un engorroso método que consiste en convertir un digrafo en un diagrama de flujo y, posteriormente, formar una matriz de adyacencia. Los algoritmos de rasgado desarrollados por Upadhye y Grens II (1972, 1975) y Lee y Rudd (1966) requieren el desarrollo de una matriz de ciclos-corrientes y, por lo tanto, la información de todos los ciclos en un digrafo. Gundersen y Hertzberg (1982, 1983) mostraron que el método de Upadhye y Grens es muy apropiado para sistemas con un número grande de equipos pero con un número moderado de ciclos.

Entre los algoritmos más populares están los propuestos por Kehat y Shacham (1973) y Murthy y Husain (1981, 1983). En 1986, se demostró que el algoritmo de Kehat y Shacham no era muy eficiente, ya que requería de más tiempo para la simulación en computadora del diagrama de flujo para el conjunto de corte encontrado (Husain, 1986). El algoritmo de Motard y Westerberg (1981) está basado en el trabajo de Upadhye y Grens

(1975), el cual emplea una matriz de ciclos y parece no ser muy eficiente para problemas complejos.

Gundersen y Hertzberg (1982, 1983) realizaron una revisión del estado del arte de los algoritmos de rasgado desarrollados hasta esa fecha, el cual fue acompañado de un minucioso estudio para encontrar las ventajas y desventajas de los mismos. Estos autores también propusieron un método de descomposición que consiste de dos fases. La primera fase involucra la partición del DFI usando el algoritmo de Tarjan (1972) y, en la segunda fase, se selecciona un conjunto de corrientes de corte. En 1983, Ollero y Amselem (1983) propusieron un método basado en el grafo de flujo dirigido. Luego, Lien y Hertzberg, (1990) presentaron un algoritmo que mejora la idea original del método de Gundersen y Hertzberg con buenos resultados. Este método involucra la aplicación repetida del algoritmo de partición (Tarjan, 1972) y el algoritmo para la identificación de todos los ciclos del sistema (Tarjan, 1973). Zhou Li (1988) desarrolló un nuevo método para la descomposición de sistemas; sin embargo, el requerimiento de almacenamiento de este método era grande para sistemas con muchos nodos. Una buena exposición de la teoría del orden de precedencia y rasgado, así como una revisión de algoritmos de descomposición y rasgado fue presentada por Mah (1990).

Lakshminarayanan y col. (1992) presentaron una nueva estrategia heurística para la secuencia de cálculo de los módulos y la determinación de las corrientes de corte. Desarrollaron un algoritmo que a diferencia de los anteriores, no implicaba el uso del digrafo dirigido, ni la información sobre los ciclos del sistema. El algoritmo propuesto encontró el número óptimo de corrientes de corte en los primeros doce casos estudiados en el presente trabajo, mientras que para los últimos dos casos este algoritmo obtuvo una corriente más que el número mínimo.

A diferencia del algoritmo desarrollado por Lakshminarayanan y col., en este trabajo se necesita información acerca de los ciclos simples del sistema para posteriormente obtener la selección de las corrientes de corte y finalmente determinar el orden de precedencia en que el sistema será resuelto. Es importante hacer notar que se obtienen resultados óptimos en los catorce casos bajo estudio.

1.3 Justificación

El enfoque de cálculos por computadora más comúnmente utilizado para simular procesos químicos estacionarios es la estrategia modular secuencial, la cual requiere que cada unidad de proceso sea representada en términos de uno o más módulos de simulación. En esta aproximación también es necesario representar formalmente como módulos de simulación a los equipos virtuales, tales como mezcladores y divisores de corrientes, y consignar los nombres de los módulos de simulación (rutinas de cálculo) en los bloques de los diagramas en lugar de los nombres de los equipos de proceso. Todos los flujos y módulos son numerados y las conexiones entre los módulos describen la transferencia de materia y energía entre los módulos. Un ejemplo de esta transformación se muestra en la Figura 1.1b, que es el diagrama de simulación del diagrama de flujo simplificado de una planta de ácido nítrico. De este diagrama se deduce que, en la estrategia modular secuencial, un proceso completo se puede modelar en términos de un conjunto de módulos de simulación conectados por los flujos de materia y energía que existen entre ellos. De esta manera se obtiene el **diagrama modular** para la simulación de un proceso.

Cuando un diagrama modular no contiene corrientes de recirculación, la solución del problema de simulación mediante la estrategia modular secuencial es muy simple; los cálculos comienzan en el módulo al que se introduce la primera alimentación y proceden módulo por módulo en la secuencia que fija el flujo físico del proceso, hasta que los productos son obtenidos. Sin embargo, la mayoría de los procesos químicos son grandes y se caracterizan por tener estructuras complejas que involucran un gran número de unidades de proceso y varias corrientes de recirculación. Para estos casos, el orden en el que se realizan los cálculos de los módulos ya no es directo. La información fluye en serie en el diagrama de flujo modular, de un módulo al siguiente en la secuencia corriente abajo, hasta que se presenta una recirculación y un módulo retorna alguna de sus corrientes de salida a sí mismo o a un módulo corriente arriba. Por lo tanto, el funcionamiento de los módulos corriente arriba depende del funcionamiento de los módulos corriente abajo. Ahora la retroalimentación de información existente impide la resolución directa en algún orden de los módulos de simulación. Para estos casos, la descomposición y el rasgado de diagramas de flujo modulares son dos técnicas ampliamente utilizadas para determinar los conjuntos de módulos independientes que tiene que ser resueltos simultáneamente e identificar las

mejores corrientes de corte de cada uno de estos conjuntos, respectivamente, a fin de desarrollar una secuencia de cálculo acíclica del problema de simulación de procesos.

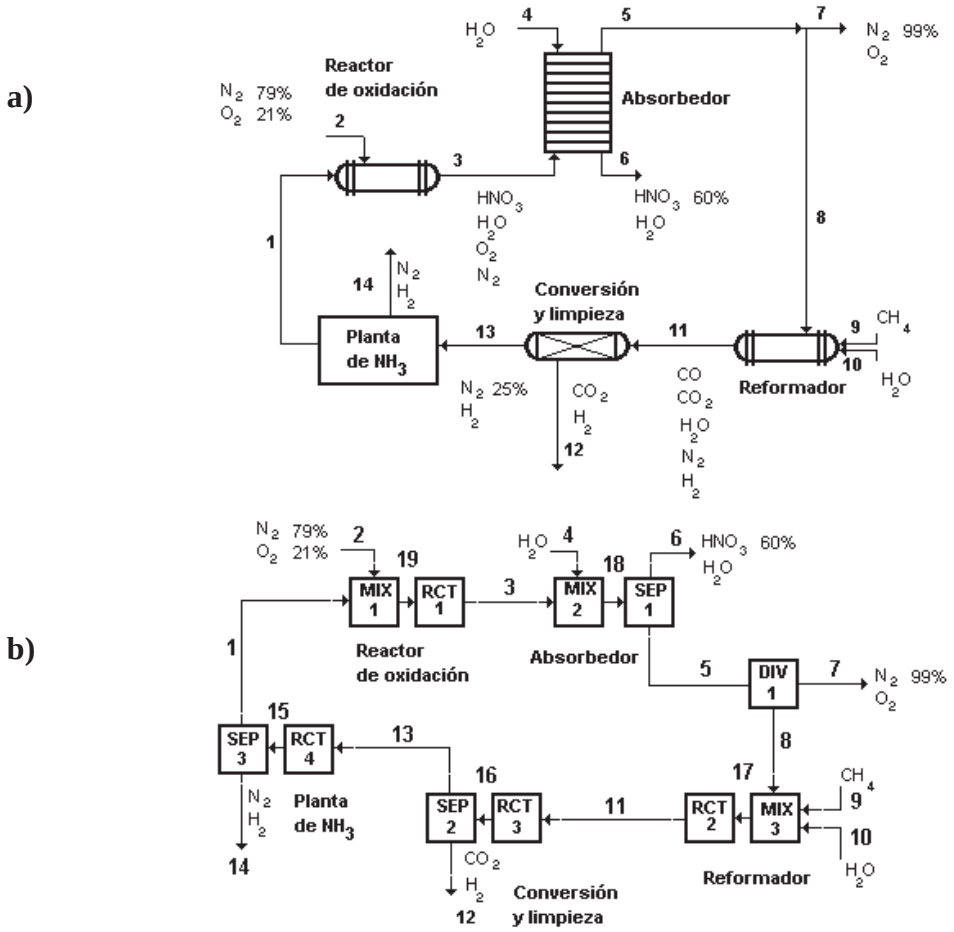


Figura 1.1 Transformación del diagrama de flujo convencional (a) en un diagrama de flujo modular (b)

La descomposición de los diagramas modulares tiene por objeto identificar los grupos de módulos que deben resolverse simultáneamente y por separado, con el propósito de simplificar la solución del problema de simulación de procesos (Norman, 1965; Sargent y Westerberg, 1964). Para lograr este objetivo se buscan, en primer lugar, los conjuntos más pequeños de módulos interconectados por corrientes de recirculación y, en segundo lugar, se establece el orden de resolución de tales grupos independientes.

Por otro lado, para resolver por separado un grupo de módulos involucrados en uno o más lazos de recirculación es necesario determinar el número mínimo de corrientes de

corte sobre las cuales se deberá iterar y seleccionar las corrientes de corte (Barkley y Motard, 1972; Upadhye y Grens, 1975). Esta etapa es conocida como el rasgado de procesos químicos con recirculaciones.

En este trabajo se propone la implementación en computadora de un procedimiento para determinar un conjunto mínimo de corrientes de corte, que consiste de dos etapas. En la primera etapa se identifican todos los ciclos simples del diagrama de flujo de información bajo estudio, usando un algoritmo basado en la técnica de búsqueda en profundidad prioritaria. En la segunda parte, un conjunto mínimo de corrientes de corte es obtenido usando un algoritmo de programación entera y, como una segunda opción, también se resuelve mediante un método heurístico. Tales corrientes rompen todos los lazos de recirculación del proceso al suponer valores iniciales para todas las variables independientes de las mismas, por lo que hacen posible obtener procesos con estructuras acíclicas. Esta característica permite establecer un orden de precedencia de las unidades del proceso y, por consiguiente, determinar la secuencia en la que éstas se deben calcular de manera iterativa. Por lo tanto, esta aproximación simplifica la solución del problema de simulación de procesos con recirculaciones.

1.4 Hipótesis

Los algoritmos de búsqueda de ciclos e identificación de corrientes de corte implementados en computadora; permiten transformar procesos complejos con recirculaciones en procesos acíclicos. De esta manera se simplifica el análisis y se determina un orden de cálculo secuencial, módulo a módulo y en la corriente de flujo, que facilita la simulación y el entendimiento del comportamiento de este tipo de procesos.

CAPITULO 2

OBJETIVOS

2.1 Objetivo General

Formular e implementar en computadora, utilizando el lenguaje JAVA y el paquete GAMS, algoritmos de identificación de ciclos, rasgado y ordenamiento de procesos químicos con recirculaciones.

2.2 Objetivos Específicos

1. Desarrollo de un algoritmo de identificación de ciclos simples de estructuras cíclicas basado en la técnica de búsqueda en profundidad en digrafos. Este algoritmo se implementará en el lenguaje JAVA.

2. Formulación del problema de identificación de corrientes de corte como un problema de programación entera, el cual se resolverá usando la matriz de ciclos y el paquete GAMS.

3. Formulación del problema de identificación de corrientes de corte usando la matriz de ciclos y una búsqueda heurística de las mismas, así como la implementación de este algoritmo en el lenguaje JAVA.

4. Desarrollo de un algoritmo de orden de precedencia basado en un método de búsqueda de nodos sumidero e implementado en el lenguaje JAVA.

5. Aplicación de los algoritmos de identificación de ciclos simples y de corrientes de corte a catorce diagramas de flujo cíclicos de diferente complejidad, previamente reportados en la literatura.

CAPITULO 3

RASGADO Y ORDENAMIENTO DE DIAGRAMAS DE PROCESOS QUIMICOS

3.1 Importancia de las corrientes de corte

La simulación de DFI sin ciclos es muy simple. Sin embargo, éste no es el caso que comúnmente se presenta en la industria química. Los procesos con recirculaciones son representados por digrafos o bloques conteniendo dos o más nodos (unidades de proceso) que son interdependientes, debido a que cada nodo en el grupo influencia a todos los demás nodos del bloque a través de corrientes de recirculación. Por consiguiente, en este caso no hay modo de dividir el problema de simulación en subproblemas más pequeños.

Actualmente, existen técnicas numéricas y computadoras más rápidas que permiten la solución simultánea de este tipo de problemas, sin embargo, requieren un esfuerzo de cálculo considerable comparado con una estrategia de cálculo secuencial, pues la dificultad para resolver un conjunto de ecuaciones aumenta exponencialmente con el número de ellas que deben manejarse simultáneamente (Mah, 1990). Cuando los valores iniciales están lejos de la solución el método numérico utilizado puede sufrir problemas de convergencia. Debido a esto, es preferible encontrar un conjunto mínimo de corrientes de corte que permita el rompimiento de todos los ciclos en un bloque con el fin de realizar una simulación en forma secuencial e iterativa de los nodos que lo conforman. Dicho problema se conoce comúnmente como rasgado y ordenamiento de procesos cíclicos.

3.2 Identificación de ciclos simples

El primer paso para la solución del problema de rasgado y ordenamiento de procesos cíclicos es la identificación de los ciclos simples. Para ello es necesario hacer la transformación del DFI en un diagrama de flujo modular o digrafo dirigido, como se

muestra en la Figura 3.1, donde se presenta un ejemplo simple cíclico de una columna de absorción de cuatro platos teóricos y su correspondiente diagrama de flujo modular.

Cabe señalar que es importante eliminar del digrafo las corrientes de entrada y salida del sistema, debido a que dichas corrientes son innecesarias en esta etapa como se podrá apreciar más adelante.

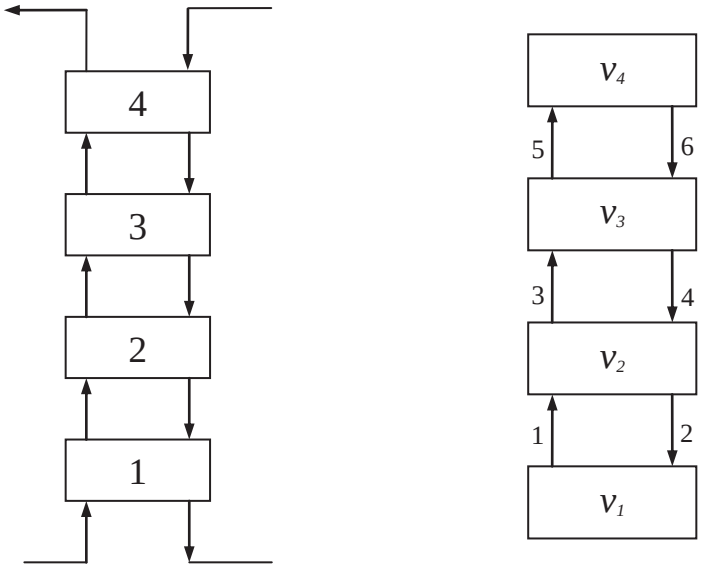


Figura 3.1 Columna de absorción de cuatro platos y su correspondiente DFI reducido (digrafo)

El tipo de ciclos que se deben identificar en el sistema estudiado son los ciclos simples. A continuación se hablará acerca de los tipos de ciclos y se explica su concepto en forma gráfica para su fácil entendimiento.

3.2.1 Tipos de ciclos

Para poder realizar de una forma más detallada y gráfica la explicación de los tipos de ciclos, se utiliza como ejemplo la Figura 3.1.

Ese digrafo tiene 6 arcos y 4 nodos. Por inspección uno puede identificar fácilmente los seis ciclos de este sistema. Estos ciclos pueden ser descritos por los caminos cerrados que son definidos como se indica a continuación:

Ciclo 1 $v_1 \rightarrow v_2 \rightarrow v_1$
Ciclo 2 $v_2 \rightarrow v_3 \rightarrow v_2$
Ciclo 3 $v_3 \rightarrow v_4 \rightarrow v_3$
Ciclo 4 $v_1 \rightarrow v_2 \rightarrow v_3 \rightarrow v_2 \rightarrow v_1$

Ciclo 5 $v_2 \rightarrow v_3 \rightarrow v_4 \rightarrow v_3 \rightarrow v_2$
Ciclo 6 $v_1 \rightarrow v_2 \rightarrow v_3 \rightarrow v_4 \rightarrow v_3 \rightarrow v_2 \rightarrow v_1$

Se observa que todos los ciclos son secuencias de nodos y arcos que comienzan y terminan en el mismo nodo. Sin embargo, en los ciclos 1 a 3 todos los otros nodos del ciclo aparecen una vez y sólo una vez en las secuencias, razón por la cual son conocidos como ciclos simples. Por el contrario, los ciclos 4 a 6 se denominan ciclos compuestos, debido a que en las secuencias hay nodos intermedios que se repiten y, por consiguiente, incluyen ciclos internos.

La enumeración por inspección de los seis ciclos en el digrafo de la Figura 3.1 fue una tarea fácil. Sin embargo, es difícil identificar por inspección todos los ciclos en un sistema complejo fuertemente interconectado, debido a que el número de posibles ciclos se incrementa al menos como $N!$ (Ulanowicz, 1983), donde N es el número de nodos en el digrafo. Por lo tanto, el número total de ciclos en un digrafo puede crecer más rápido con N que el exponencial 2^N, por lo que puede alcanzar proporciones astronómicas aún para sistemas de tamaños modestos.

3.2.2 Algoritmo de búsqueda de ciclos simples

La identificación de los ciclos simples de digrafos con reciclos es un problema importante en el área de la teoría de grafos lineales y en otras aplicaciones. De los métodos generales de enumeración de ciclos en grafos dirigidos existentes, los algoritmos de búsqueda en profundidad prioritaria, que incorporan métodos de eliminación de caminos de búsqueda inútiles, son las técnicas más eficientes para la mayoría de los problemas (Mateti y Deo, 1976). Diferentes algoritmos de búsqueda en profundidad prioritaria de ciclos simples han sido reportados en la literatura (Johnson, 1975; Read y Tarjan, 1975; Reingold, 1977; Tiernan, 1970; Weinblatt, 1972). Con base en las ideas y conceptos propuestos en estos trabajos, se ha formulado el algoritmo de búsqueda de ciclos simples que se describe a continuación. La teoría de grafos requerida para el desarrollo de este algoritmo se presenta en el Apéndice A de este trabajo de tesis. Se observa que dicha teoría es simple de entender y, como se mostrará enseguida, es también simple de aplicar.

Datos de entrada. Sea V el conjunto de nodos y A el conjunto de arcos del digrafo reducido $G = (V, A)$, que representa a un bloque cíclico dado. V y A contienen solamente un número entero y finito de miembros denotado por N y M, respectivamente.

Paso 1. Enumerar los arcos del digrafo reducido G desde 1 hasta M. Asimismo, identificar los nodos de G por v_j, $1 \leq j \leq N$. Hacer $r = 1$.

Paso 2. Comenzar en el nodo de referencia v_r, que se marca como el nodo activo, la construcción del camino simple.

Paso 3. Antes de continuar el recorrido en el digrafo:

Paso 3.1. Eliminar los nodos fuente, que sólo tienen arcos saliendo de ellos, junto con sus arcos incidentes del digrafo.

Paso 3.2. Eliminar los nodos sumidero, que sólo tienen arcos dirigidos hacia ellos, junto con sus arcos incidentes del digrafo.

Paso 3.3. Repetir los pasos 3.1 y 3.2 hasta que no queden nodos fuente y sumidero en el digrafo remanente.

Si el nodo de referencia v_r es uno de los nodos eliminados, ir al paso 8.

Paso 4. Continuar el trazado recursivo del camino simple en curso desde el nodo activo, siguiendo arcos no explorados en la dirección del flujo del grafo dirigido, hasta encontrar un ciclo simple o que no sea posible su extensión.

Este paso funciona de la siguiente manera: Suponiendo que se ha llegado al nodo v_k por el camino simple en construcción, la secuencia de nodos distintos conectados por arcos que se ha obtenido es: $v_r \rightarrow v_{r+1} \rightarrow ... \rightarrow v_i \rightarrow v_{i+1} \rightarrow ... \rightarrow v_k$. Este camino comienza en v_r y no contiene nodos más pequeños que v_r. Para el siguiente arco (v_k, v_{k+1}) a explorar en la búsqueda en profundidad, hay tres casos generales a considerar:

Paso 4.1. El nodo v_{k+1} extiende el camino actual, pasando a ser el nodo activo, si no ha sido visitado previamente. Luego se inserta en el camino el nodo v_{k+2}, adyacente a v_{k+1} y que todavía no se haya visitado. Se toma v_{k+2} como el nodo activo del camino. Después se realiza el recorrido en profundidad partiendo del nodo activo para encontrar un nuevo nodo no visitado. Se procede así sucesivamente para extender el camino bajo consideración, adicionándole un nodo no visitado a la vez, hasta que se encuentra un

nodo cuyo(s) adyacente(s) haya(n) sido visitado(s), es decir, que cumple(n) la condición 4.2 ó la 4.3.

Paso 4.2. El camino finaliza en el nodo activo v_k cuando reaparece el nodo de referencia en la secuencia, esto es, cuando el nodo $v_{k+1} = v_r$. El nodo de referencia y los nodos que se encuentran entre el nodo de referencia de la secuencia forman el ciclo simple $v_r \rightarrow v_{r+1} \rightarrow ... \rightarrow v_i \rightarrow v_{i+1} \rightarrow ... \rightarrow v_k \rightarrow v_r$. Marcar como explorado el arco (v_k, v_{k+1}) e ir al paso 5.

Paso 4.3. El recorrido termina en el nodo activo v_k cuando el nodo v_{k+1} se encuentra en el camino y $v_{k+1} \neq v_r$. Marcar como explorado el arco (v_k, v_{k+1}) e ir al paso 6.

Paso 5. El ciclo identificado es localizado en la lista de ciclos en términos del conjunto de arcos numerados que lo forman.

Paso 6. Si los arcos de salida del nodo activo en el camino actual han sido explorados en su totalidad, ir al paso 7; en caso contrario, seleccionar otro arco de salida no explorado de dicho nodo y regresar al paso 4.

Paso 7. Volver hacia atrás por el camino desde el nodo activo al nodo previo y así sucesivamente, borrando los nodos y arcos por los que se va retrocediendo, hasta el último nodo de la secuencia que tiene, al menos, un arco de salida no explorado que cumple la condición 4.1 ó la 4.2. El nodo así detectado se convierte en el nuevo nodo activo del camino. Ir al paso 4 para reiniciar la búsqueda de ciclos.

Si se ha retornado al nodo de referencia v_r y éste ya no tiene arcos de salida sin explorar, borrar del digrafo al nodo v_r junto con todos sus arcos incidentes e ir al paso 8.

Paso 8. El algoritmo termina cuando el subgrafo remanente es vacío o acíclico. En caso contrario, hacer $r = r + 1$ y regresar al paso 2.

Comentario 1. La eliminación de nodos fuente y sumidero, que se realiza en el paso 3 del algoritmo, es una técnica de reducción de grafos que simplifica la identificación de ciclos simples. Esto se debe a que un ciclo es un subgrafo conectado en el que cada nodo tiene un arco de entrada y un arco de salida, por lo que claramente un nodo fuente y un nodo sumidero, así como sus arcos incidentes, no pueden aparecer en un ciclo.

Comentario 2. En el paso 4.1 se establece que para extender el camino simple en construcción, ningún nodo intermedio puede ser repetido en el camino. En términos matemáticos, esta condición puede ser expresada como la restricción: $v_i > v_r$ y $r \leq i \leq k$, que se aplica a cada uno de los nodos del camino $v_r \rightarrow v_{r+1} \rightarrow ... \rightarrow v_i \rightarrow v_{i+1} \rightarrow ... \rightarrow v_k$. De esta manera, se evita la generación de ciclos duplicados desde diferentes nodos del mismo ciclo, por lo que se asegura que cada uno de los ciclos simples del digrafo se identificará solamente una vez.

Comentario 3. En el paso 7 del algoritmo se borra el nodo de referencia v_r, después de haber generado todos los ciclos simples asociados a dicho nodo. Esta acción produce un subgrafo que ya no contiene tal nodo, lo cual decrece la dimensión de la búsqueda subsecuente de ciclos simples.

Este algoritmo se basa en la construcción de caminos simples desde un nodo inicial para identificar todos los ciclos simples de un digrafo, cuyo número es al menos igual al número de sus ciclos independientes. Todos los ciclos del digrafo se pueden generar a partir de los ciclos simples que son una base para el espacio vectorial de ciclos (Zhu y col., 1996). Es importante hacer notar que el algoritmo anterior también es capaz de identificar autociclos.

3.2.2.1 Ejemplo del algoritmo de búsqueda de ciclos simples

Se considera el digrafo reducido, mostrado en la Figura 3.2 (Rubin, 1962), para ilustrar la aplicación del algoritmo de búsqueda de ciclos simples. Este ejemplo es considerado el caso de estudio número uno de los catorce casos de estudio que serán presentados en el siguiente capítulo.

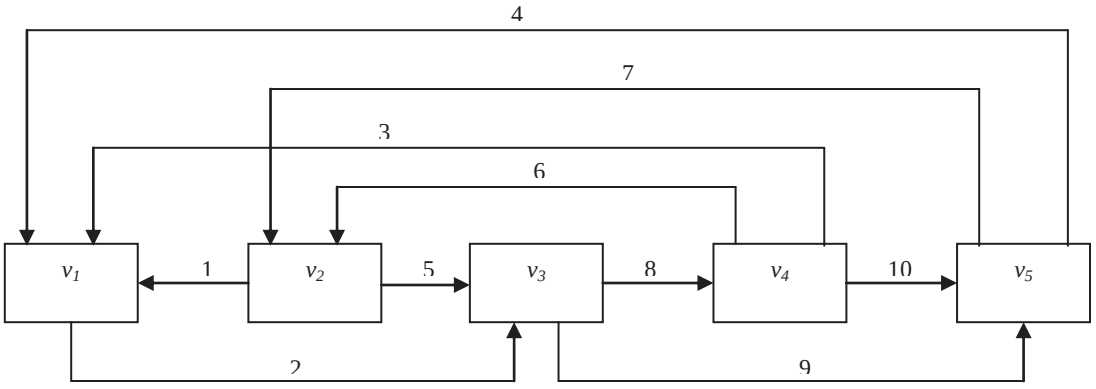


Figura 3.2 Digrafo reducido de Rubin (1962)

Solución.

Los pasos del algoritmo de identificación de ciclos se ilustran detalladamente a continuación para el primer ciclo simple asociado al nodo v_1.

Paso 1. Se asigna un número de identificación a cada uno de los nodos y arcos del digrafo original, como se muestra en la figura 3.2. Se hace $r = 1$. Se avanza al paso 2.

Paso 2. Se inicia la construcción del camino en el nodo de referencia v_1. Se va al paso 3.

Paso 3. El digrafo original no tiene nodos fuente o sumidero, por lo que permanece sin cambio. Se va al paso 4.

Paso 4. En primer lugar, se continúa el trazado del camino dirigido en la dirección del flujo del digrafo de conformidad a la condición 4.1.

Paso 4.1. A partir del nodo v_1 se llega por el arco 2 al nodo v_3 no visitado previamente y se añade al camino. Este nodo se conecta mediante los arcos 8 y 9 con los nodos no visitados v_4 y v_5, respectivamente. Se puede elegir, indistintamente, cualquier arco; en este caso, se opta por recorrer el arco 9 para extender el camino con el nodo v_5. De este nodo salen los arcos 7 y 4 y, de igual manera que se hizo anteriormente, se elige al arco 7 para continuar con el camino a través del nodo v_2. Este nodo conduce, a través de los arcos 5 y 1, a los nodos v_3 y v_1, respectivamente, y debido a que ambos nodos ya se encuentran en el camino, se analizan las condiciones dándole prioridad a la condición 4.2 por ser el paso consecutivo del algoritmo, lo cual dirige a la selección del arco 1. Por lo tanto, este camino finaliza en el nodo v_2. De esta forma se obtiene el camino simple mostrado en la figura 3.3a, que va del nodo v_1 al nodo v_2, pasando por los nodos v_3 y v_5 los cuales están conectados por los arcos 2, 9 y 7, respectivamente. Se avanza al paso 4.2

Paso 4.2. El nodo de referencia v_1 y los nodos que se encuentran entre el nodo de referencia del camino forman el primer ciclo simple: $v_1 \longrightarrow v_3 \longrightarrow v_5 \longrightarrow v_2 \longrightarrow v_1$. El arco de salida (1 ó $v_2 \longrightarrow v_1$) de v_2 se marca como explorado por una línea discontinua y se va al paso 5.

Paso 5. El conjunto de arcos {2, 9, 7, 1} que forman el primer ciclo encontrado se coloca en la lista de ciclos. Se va al paso 6.

Paso 6. El nodo activo v_2 cuenta con el arco 5 inexplorado que conduce nuevamente al paso 4.

Paso 4. Se observa que el arco 5 conduce al nodo v_3, que es un nodo intermedio del camino actual, por lo que se cumple la condición 4.3 y se avanza al paso 6.

Paso 6. El nodo activo v_2 ya no tiene arcos de salida inexplorados, por lo que se va al paso 7.

Paso 7. Como se muestra en la figura 3.3b, la búsqueda retrocede hasta el nodo v_5 que es el último nodo en el camino actual que tiene un arco de salida sin explorar, lo cual permite continuar por otro camino. En la figura 3.3b se observa que el arco 7 está marcado con una X, para denotar que fue borrado del camino durante el retroceso al nodo v_5. Esta acción de eliminación actualiza el camino y deja disponibles los arcos borrados para ser nuevamente explorados en caso necesario. Se va al paso 4.

Después, en el paso 4.1 y considerando la figura 3.2, se reinicia la construcción del camino a partir del nodo activo v_5, pero ahora por su arco 4 no explorado que a su vez vuelve nuevamente al nodo v_1. Termina, por lo tanto, este camino al cumplirse la condición 4.2 y el arco 4 se marca como explorado. Luego, en el paso 5, el ciclo encontrado {2, 9, 4} se coloca en la lista de ciclos. Como ya se han examinado todos los arcos de salida del nodo activo v_5, del paso 6 se avanza directamente al paso 7, donde se vuelve por el camino hasta el nodo v_3, por ser éste el último nodo de la secuencia bajo consideración que tiene un arco de salida (arco 8) inexplorado que cumple la condición 4.1. Como se muestra en la figura 3.3c, al retroceder se borra el arco 9.

El algoritmo va al paso 4, donde continúa el recorrido a partir del nodo activo v_3, visitándose sucesivamente los nodos v_4, v_5 y v_2 como resultado de la correspondiente exploración de los arcos 8, 10 y 7 de la figura 3.2. Al recorrer el único arco de salida (arco 1) viable de explorar del nodo activo v_2 de este camino, reaparece el nodo de referencia v_1. Por lo tanto, se va al paso 4.2 para terminar el camino actual, identificando el tercer ciclo simple asociado al nodo inicial v_1 y marcando como explorado el arco 1. Luego, en el paso 5, el conjunto de arcos del tercer ciclo {2, 8, 10, 7, 1} del digrafo se coloca en la lista de ciclos.

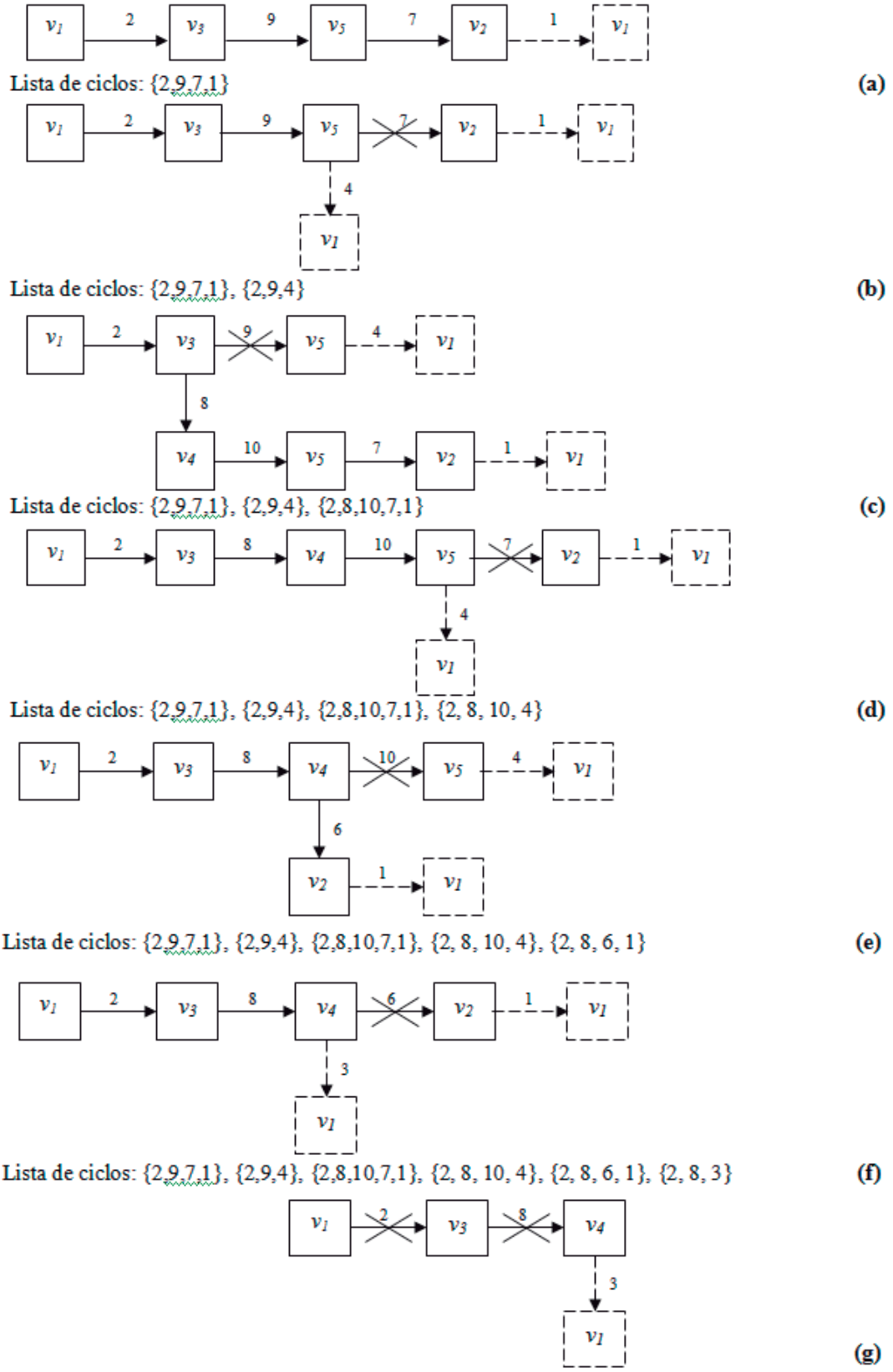


Figura 3.3 Identificación de los ciclos simples asociados al nodo v_1

A continuación, como v_2 ya no tiene arcos de salida viables, del paso 6 se avanza al paso 7, donde se retrocede desde el nodo activo v_2 hasta el nodo v_5, puesto que éste es el último nodo del camino que tiene un arco de salida (arco 4) por explorar para construir otro camino simple. El arco 7 se borra a medida que se vuelve hacia atrás por el camino, como se muestra en la figura 3.3d. Se va al paso 4.1, donde se expande el camino actual tanto como sea posible a partir del nodo v_5. Para tal efecto, se recorre el arco 4 que conduce nuevamente al nodo de referencia v_1. En este punto se va al paso 4.2, donde el camino actual concluye, se identifica el cuarto ciclo simple del digrafo y se marca como explorado el arco 4. Se va al paso 5, donde el conjunto de arcos {2, 8, 10, 4} del ciclo detectado se coloca en la lista de ciclos. En el paso 6 se determina que el nodo activo ya no tiene ningún vecino no visitado, por lo que se va al paso 7. Ahora, el algoritmo se mueve hacia atrás a través de los arcos del camino actual, pasando del nodo activo v_5 al v_4 debido a que éste es el último nodo del camino al que le restan dos arcos de salida por explorar para construir otros caminos simples. El arco 10 se borra cuando se retrocede por el camino como se muestra en la figura 3.3e. Se avanza al paso 4.1, donde se extiende el camino actual tanto como sea posible a partir del nodo v_4. Para ello, de los dos arcos que salen de este nodo, se selecciona indistintamente el arco 6 para avanzar y llegar así al nodo v_2 no visitado y el cual se incorpora al camino simple en construcción. Luego, se selecciona el arco 1, siendo éste el único arco viable de explorar que sale del nodo activo v_2 para extender el camino en construcción, lo que conduce de nuevo al nodo de referencia v_1. Se va al paso 4.2, donde el camino actual concluye, se identifica el quinto ciclo del digrafo y se marca como explorado el arco 1. Se sigue el paso 5, donde el conjunto de arcos {2, 8, 6, 1} del ciclo encontrado se coloca en la lista de ciclos. Siguiendo el paso 6 se determina que el nodo activo ya no tiene ningún arco no visitado, por lo que se va al paso 7. Enseguida, el algoritmo recorre hacia atrás a través de los arcos del camino actual, pasando del nodo activo v_2 al v_4 ya que éste es el último nodo del camino que tiene un arco de salida sin explorar para construir otro camino simple. El arco 6 se elimina a medida que se vuelve hacia atrás en el camino, como se muestra en la figura 3.3f. Se va al paso 4.1, extendiendo así el camino actual tanto como sea posible a partir del nodo v_4. En este caso se recorre el arco 3, que finalmente conduce al nodo de referencia v_1. Se avanza al paso 4.2, donde concluye el camino actual, se identifica el sexto ciclo del digrafo y se marca como explorado el arco 3. Se va al paso 5, donde el

conjunto de arcos {2, 8, 3} del ciclo detectado se coloca en la lista de ciclos. En el paso 6 se determina que el nodo activo ya no tiene ningún vecino no visitado, por lo que se va al paso 7. Ahora, el algoritmo se mueve hacia atrás a través de los arcos del camino actual, pasando desde el nodo activo v_4 al v_3 y, finalmente, regresa al nodo de referencia v_1 debido a que ya han sido explorados todos los arcos de salida de esos nodos. Durante este retroceso se van borrando, en forma sucesiva, los arcos 8 y 2, como se muestra en la figura 3.3g. Por consiguiente, se borra el nodo v_1 junto con sus arcos incidentes del digrafo original, para obtener el subgrafo de la figura 3.4. Luego se hace r = r + 1 = 1 + 1 = 2 y se reinicia el proceso de búsqueda de ciclos simples en el paso 2, siendo ahora v_2 el nodo inicial de un nuevo camino simple en construcción.

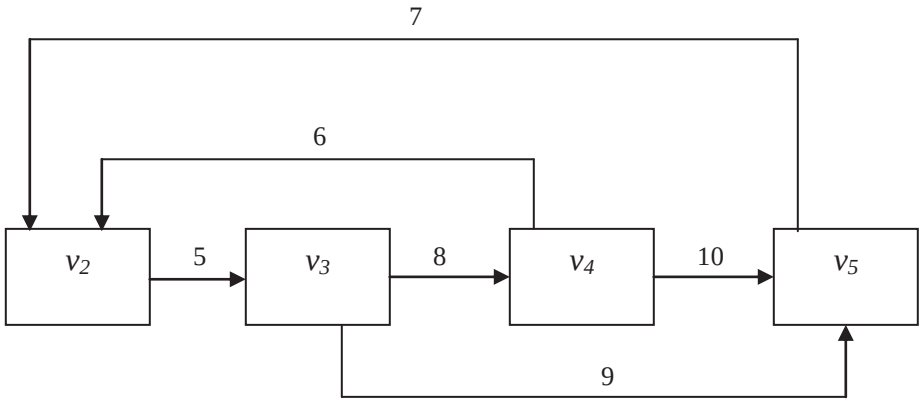


Figura 3.4 Subgrafo remanente para el nodo de referencia v_2

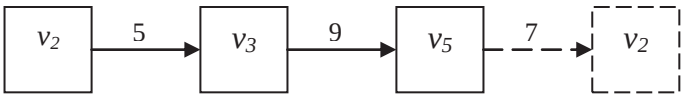
De acuerdo a lo indicado en el paso 3 del algoritmo, antes de continuar el trazado del camino se observa, en la figura 3.4, si alguno de los nodos son del tipo fuente o sumidero. Dado que ninguno de ellos son nodo fuente o sumidero, no se borran del subgrafo; en el caso contrario, es decir, si alguno de estos nodos correspondiera con la descripción anterior, se borrarían del subgrafo junto con sus arcos incidentes. Y si esto sucediera con el nuevo nodo de referencia v_2, se haría $r = r + 1 = 2 + 1 = 3$ y los siguientes caminos a explorar comenzarían en el nodo v_3. Pero como este no es el caso, por lo que el nodo de referencia para continuar trazando los ciclos faltantes es el nodo v_2 y el subgrafo mostrado anteriormente es el que se utiliza para continuar con el procedimiento.

Aclarado esto, lo que prosigue es ir al paso 4.1, donde se visitan en forma sucesiva los nodos v_3 y v_5 y se vuelve al nodo v_2. En este punto se repite el nodo de referencia, por lo

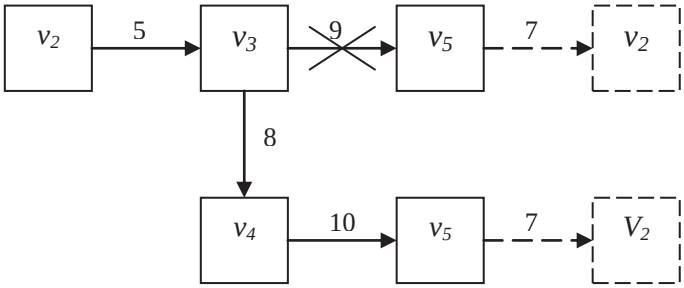
que se identifica el séptimo ciclo simple del digrafo. Se va al paso 4.2 para concluir el camino mostrado en la figura 3.5a, marcando el arco 7 como explorado. Luego, en el paso 5, el conjunto de arcos {5, 9, 7} del ciclo encontrado se coloca en la lista de ciclos. Se va al paso 6, donde se determina que el nodo activo no tiene ningún vecino sin visitar, por lo que se avanza al paso 7. El retroceso comienza en el nodo v_5 y concluye en el nodo v_3, como se muestra en la figura 3.5b, porque este nodo tiene un arco de salida inexplorado (arco 8). Se va al paso 4.1, se recorren en forma sucesiva los nodos v_4 y v_5 y se vuelve al nodo v_2 a través del arco 7. En esta parte se repite el nodo de referencia y así se identifica el octavo ciclo simple del digrafo. Para concluir el camino se va al paso 4.2 y se marca el arco 7 como explorado. Ya en el paso 5, el conjunto de arcos {5, 8, 10, 7} del ciclo encontrado se colocan en la lista de ciclos. Luego se va al paso 6.

Se determina que el nodo activo no cuenta ya con arcos sin visitar, por lo que se sigue al paso 7. Se retrocede en el camino desde el nodo v_5 al nodo v_4, como se muestra en la figura 3.5c, puesto que este nodo tiene un arco de salida (arco 6) viable de explorar que cumple con la condición 4.2. Luego, en el paso 4.2, en un solo recorrido por el arco 6 se alcanza de nuevo al nodo v_2 desde el nodo activo v_4. De esta manera se marca el arco 6 como explorado. Se avanza al paso 5, donde el conjunto de arcos {5, 8, 6} que forman el noveno ciclo del digrafo se coloca en la lista de ciclos. Se va al paso 6, donde se determina que, en esta etapa, el nodo activo v_4 no tiene ningún arco adyacente no visitado, por lo que se va al paso 7. A continuación, como se muestra en la figura 3.5d, el algoritmo retrocede hasta el nodo de referencia v_2, el cual ya no tiene más arcos de salida sin explorar. Por tal motivo, el nodo v_2 y sus corrientes incidentes se borran del subgrafo correspondiente, mostrado en la figura 3.4. Se va al paso 8. Se hace $r = r + 1 = 2 + 1 = 3$ y el algoritmo continúa en el paso 2.

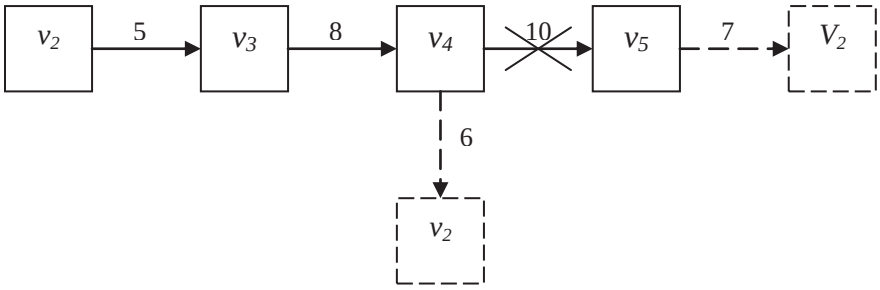
Por lo tanto, ahora v_3 es el nodo de referencia. Su correspondiente subgrafo es acíclico, como se muestra en la figura 3.6. Por consiguiente, el algoritmo de identificación de ciclos simples ha terminado. De hecho, la implementación en este subgrafo del proceso de eliminación de nodos fuente y sumidero del paso 2, genera un grafo vacío.



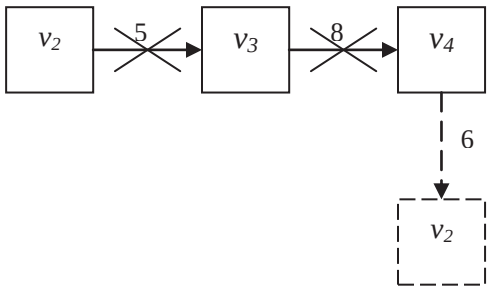
Lista de Ciclos: {5, 9, 7} **(a)**



Lista de Ciclos: {5, 9, 7}, {5, 8, 10, 7} **(b)**



Lista de Ciclos: {5, 9, 7}, {5, 8, 10, 7}, {5, 8, 6} **(c)**



(d)

Figura 3.5 Identificación de los ciclos simples asociados al nodo v_2

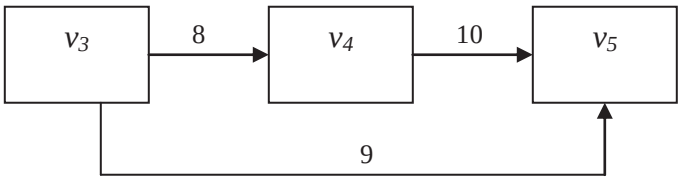


Figura 3.6 Subgrafo acíclico que resulta de la aplicación del algoritmo de búsqueda de ciclos

Como resultado de la aplicación del algoritmo se han identificado nueve ciclos simples, que representan la estructura de recirculación del digrafo original: {2,9,7,1}, {2,9,4}, {2,8,10,7,1}, {2, 8, 10, 4}, {2, 8, 6, 1}, {2, 8, 3}, {5, 9, 7}, {5, 8, 10, 7}, {5, 8, 6}.

En la práctica, como se mostró en este ejemplo, el proceso de eliminación de nodos y arcos frecuentemente provoca que otros nodos se conviertan en fuentes o sumideros en los grafos remanentes y, por lo tanto, los convierte en candidatos para una posterior eliminación antes de iniciar el trazado de nuevos caminos simples. De esta manera se evita la búsqueda inútil de ciclos, por lo que este proceso de eliminación hace más eficiente el algoritmo de identificación de ciclos simples.

3.3 Selección del conjunto mínimo de corrientes de corte

Una vez identificados todos los ciclos simples de un digrafo, es posible determinar cuáles y cuántos arcos tienen que ser borrados del digrafo para eliminar todos sus ciclos.

Dado un digrafo conectado *G (V, A)*, un *conjunto de corte* de *G* es un subconjunto de arcos $A' \subseteq A$ conteniendo al menos un arco de cada ciclo en *G*, de modo tal que el digrafo *G'(V, A-A')* resultante de la eliminación de los elementos de A' no tiene ciclos. Por consiguiente, un conjunto de corte produce una estrategia de cálculo secuencial en el contexto de simulación de procesos (Mah, 1990). Para un DFI cíclico dado, aquellos conjuntos de corte que tengan el número mínimo de arcos son llamados *conjuntos mínimos de corte*.

Hay diversas técnicas para determinar los *conjuntos mínimos de corte* para un DFI; no obstante, el método basado en la matriz de ciclos es recomendado por ser general y fácil de aplicar. El método consiste en dos etapas:

- ✓ Primera etapa: Construcción de la matriz de ciclos simples.
- ✓ Segunda etapa: Identificación de los *conjuntos mínimos de corte*.

3.3.1 Matriz de ciclos

Las filas *i* y columnas *j* de la matriz de ciclos **C** representan los ciclos simples y los arcos del DFI reducido, respectivamente. Los elementos c_{ij} de **C** toman los siguientes valores [11]:

$$\boldsymbol{C} = [c_{ij}] = \begin{cases} 1, & \text{si el arco j forma parte del ciclo i} \\ 0, & \text{en caso contrario} \end{cases} \tag{3.1}$$

A manera de ejemplo, en la Tabla 3.1 se muestra la matriz de ciclos simples para el DFI de la figura 3.2, donde se observa que en la fila del ciclo simple *i* donde no participa el arco *j*, en el correspondiente elemento c_{ij} aparece un 0. Nótese que varios de los arcos aparecen en más de un ciclo.

Tabla 3.1 Matriz de ciclos simples para la figura 3. 2

	ARCOS									
CICLOS	1	2	3	4	5	6	7	8	9	10
1	1	1	0	0	0	0	1	0	1	0
2	0	1	0	1	0	0	0	0	1	0
3	1	1	0	0	0	0	1	1	0	1
4	0	1	0	1	0	0	0	1	0	1
5	1	1	0	0	0	1	0	1	0	0
6	0	1	1	0	0	0	0	1	0	0
7	0	0	0	0	1	0	1	0	1	0
8	0	0	0	0	1	0	1	1	0	1
9	0	0	0	0	1	1	0	1	0	0

3.3.2 Formulación lineal entera para la selección del conjunto mínimo de corrientes de corte

El problema de encontrar un conjunto mínimo de corte puede ser formulado y resuelto por una técnica de programación entera. Para tal efecto, es necesario conocer las relaciones que existen entre los elementos del conjunto de ciclos *C* y el conjunto de arcos *A* de un digrafo conectado. Esta información es proporcionada por la matriz de ciclos, como lo indica la ecuación (3.1).

En términos de la matriz de ciclos, una solución a este problema es un subconjunto de arcos (columnas) *S* de modo tal que cada ciclo (fila) de *G* contenga al menos un elemento de *S*. En otras palabras, el objetivo es cubrir[1] todos los ciclos de *G* por un conjunto mínimo de arcos.

[1] Es decir, que al menos un arco de cada uno de los ciclos de G aparezca en un conjunto mínimo de arcos.

A fin de representar matemáticamente este problema, para cada arco se introduce una variable binaria y_j, la cual puede tomar sólo los valores 0 ó 1 de acuerdo a:

$y_j = 1$ si el arco de la columna j es seleccionado en la solución del problema

$y_j = 0$ en caso contrario.

Por lo tanto, este problema puede ser formulado como un problema de programación entera de la siguiente manera (Pho y Lapidus, 1973):

$$\min \sum_{j=1}^{N_S} y_j \tag{3.2}$$

$$\text{sujeto a } \sum_{j=1}^{N_S} c_{ij} y_j \geq 1 \qquad \text{para toda } i = 1...N_C \tag{3.3}$$

$$y_j = \{0,1\} \qquad \text{para toda } j = 1...N_S \tag{3.4}$$

donde N_S es el número de arcos (o de corrientes) y N_C es el número de ciclos.

La restricción (3.3) asegura que cada ciclo (fila) es cubierto por al menos un arco (columna). La restricción (3.4) asegura que las variables asociadas con los arcos sean variables binarias.

La solución de este problema de optimización da un conjunto mínimo de corrientes de corte, que incluye sólo aquellos arcos para los que la correspondiente variable binaria toma un valor de 1.

3.3.2.1 Ejemplo de la formulación lineal entera para la selección del conjunto mínimo de corrientes de corte

Para hacer más clara esta formulación se considera la matriz de ciclos presentada en la tabla 3.1. Sea Y = {y_1, y_2, y_3, y_4, y_5, y_6, y_7, y_8, y_9, y_{10}}.

El problema de optimización asociado a este caso es como sigue:

Minimizar $y_1 + y_2 + y_3 + y_4 + y_5 + y_6 + y_7 + y_8 + y_9 + y_{10}$

Sujeto a $y_1 + y_2 + y_7 + y_9 \geq 1$

$y_2 + y_4 + y_9 \geq 1$

$y_1 + y_2 + y_7 + y_8 + y_{10} \geq 1$

$y_2 + y_4 + y_8 + y_{10} \geq 1$

$y_1 + y_2 + y_6 + y_8 \geq 1$

$y_2 + y_3 + y_8 \geq 1$

$y_5 + y_7 + y_9 \geq 1$

$y_5 + y_7 + y_8 + y_{10} \geq 1$

$y_5 + y_6 + y_8 \geq 1$

y_j son variables binarias

Una solución óptima de este problema corresponde a un conjunto mínimo de corte. En el presente ejemplo, la solución Y = {0, 1, 0, 0, 1, 0, 0, 0, 0, 0} corresponde al conjunto mínimo de corrientes de corte {2, 5}.

El modelo de programación lineal entera proporciona una representación exacta del problema de encontrar un conjunto mínimo de corte. Por tal motivo, es reconocido como el mejor algoritmo de selección de arcos de corte para cualquier tipo de grafos dirigidos, completamente reductibles o no. Esto es de particular importancia para problemas con números grandes de ciclos y arcos.

3.3.3 Método heurístico para elegir las corrientes de corte

Este método también examina la matriz de ciclos para identificar aquellos conjuntos de arcos que puedan ser conjuntos de corte. Una condición básica que debe cumplir cada arco de un conjunto de corte es que sea miembro de al menos un ciclo.

El método heurístico utiliza los parámetros *frecuencia de arco* y *rango de ciclo.* El primero es definido como el número de ciclos en los que aparece un arco dado y es igual a la suma de los elementos no nulos de la correspondiente columna (Kusiak y Wang, 1993). El segundo parámetro se refiere al número de arcos involucrados en un ciclo, lo que es igual a la suma de los elementos no nulos en la correspondiente fila de la matriz de ciclos.

Para romper un ciclo cuyo rango de ciclo es igual a 1, necesariamente el único arco de ese ciclo debe ser uno de los elementos del conjunto mínimo de corte. De igual manera, para romper un ciclo que tiene un rango de ciclo igual a 2, al menos uno de sus dos arcos debe ser incluido en el conjunto mínimo de corte. Este análisis es válido para ciclos con rangos mayores. Por otro lado, es conveniente notar que el rompimiento de un ciclo automáticamente romperá todos los ciclos más grandes de los que el arco seleccionado forme parte. Por tal motivo, se recomienda elegir como elementos del conjunto de corte a los arcos con mayor frecuencia presentes en los ciclos de rango más pequeño. Usando esta

heurística, se puede desarrollar el siguiente método para encontrar conjuntos mínimos de corte:

Paso 1. Calcular la frecuencia de arco y el rango de ciclo. Presentar estos parámetros para cada arco y cada ciclo en la última fila y última columna, respectivamente, de una matriz de ciclos aumentada.

Paso 2. Identificar el ciclo con el rango más pequeño. Luego, seleccionar al arco de este ciclo que tenga la frecuencia más grande. En caso de haber varias opciones que cumplan estas características, es decir, diferentes arcos con la más grande frecuencia formando parte de diferentes ciclos con el rango más pequeño, de entre ellas seleccionar al ciclo que aparezca primero en la matriz de ciclos de izquierda a derecha.

Paso 3. Adicionar el arco seleccionado al conjunto mínimo de corte.

Paso 4. Eliminar de la matriz de ciclos al arco seleccionado junto con todos los ciclos de los que forma parte.

Paso 5. Si en la matriz remanente no quedan ciclos, terminar; en caso contrario regresar al paso 1.

Los arcos así seleccionados, forman parte de un conjunto de corte que rompe todos los ciclos del digrafo.

Debido a que con este método se presentan varias opciones en igualdad de circunstancias (misma frecuencia, rango de ciclo más pequeño, sin pertenencia a un ciclo previamente eliminado, etc.) es posible generar diversas combinaciones de conjuntos de corte.

3.3.3.1 Ejemplo del método heurístico para elegir las corrientes de corte

Considere la tabla 3.2 que ilustra la matriz de ciclos aumentada para ilustrar el método heurístico.

Solución

Paso 1. Una nueva columna es adicionada a la derecha de la matriz de ciclos así como una nueva fila al final de la misma, como se muestra en la tabla 3.2. Los elementos en esta nueva columna son los rangos de ciclos, que se obtienen al sumar los elementos no

nulos de cada fila; en tanto que, los elementos de la nueva fila son las frecuencias de arco, las cuales se obtienen sumando los elementos no nulos de cada columna.

Tabla 3.2 Matriz de ciclos simples aumentada para la figura 3. 2

	ARCOS										
CICLOS	1	2	3	4	5	6	7	8	9	10	Rango de ciclo
1	1	1	0	0	0	0	1	0	1	0	4
2	0	1	0	1	0	0	0	0	1	0	3
3	1	1	0	0	0	0	1	1	0	1	5
4	0	1	0	1	0	0	0	1	0	1	4
5	1	1	0	0	0	1	0	1	0	0	4
6	0	1	1	0	0	0	0	1	0	0	3
7	0	0	0	0	1	0	1	0	1	0	3
8	0	0	0	0	1	0	1	1	0	1	4
9	0	0	0	0	1	1	0	1	0	0	3
Frecuencia de arco	3	6	1	2	3	2	4	6	3	3	

Paso 2. Los ciclos 2, 6, 7 y 9 tienen el menor rango de ciclo (3). En los ciclos 2 y 6 se encuentra la corriente 2 con la mayor frecuencia de arco (6), mientras que en los ciclos 6 y 9 participa la corriente 8 que también tiene la mayor frecuencia de arco (6). Estos dos arcos están en igualdad de condiciones, por lo que para elegir al primer elemento del conjunto mínimo de corte se utiliza el criterio de seleccionar al que aparezca primero de izquierda a derecha en la matriz de ciclos aumentada, en este caso, el arco 2.

Paso 3. Se adiciona el arco 2 al conjunto mínimo de corte.

Paso 4. Se elimina de la matriz de ciclos al arco 2 junto con todos los ciclos de los que forma parte, tal como se muestra en la tabla 3.3.

Tabla 3.3 Matriz de ciclos simples aumentada después de la eliminación del arco 2

	ARCOS									
CICLOS	1	3	4	5	6	7	8	9	10	Rango de ciclo
7	0	0	0	1	0	1	0	1	0	3
8	0	0	0	1	0	1	1	0	1	4
9	0	0	0	1	1	0	1	0	0	3
Frecuencia de arco	0	0	0	3	1	2	2	1	1	

Paso 5. Se regresa al paso 1, ya que aún existen ciclos en la matriz.

En el paso 1, otra vez, se calculan la frecuencia de arco y el rango de ciclo ahora de la nueva matriz de ciclos aumentada. Se va al paso 2, en donde se selecciona el arco 5 que tiene la mayor frecuencia y el cual forma parte de los ciclos con el menor rango, ciclos 7 y 9. En el paso 3, se agrega dicho arco al conjunto mínimo de corte y se avanza al paso 4 en donde se elimina de la matriz el arco 5 junto con los ciclos en los cuales forma parte, dando como resultado una matriz vacía por lo que el algoritmo termina en este punto.

Los arcos seleccionados que forman parte del conjunto de corte que rompe todos los ciclos del digrafo son: {2, 5}

Como se puede observar, en este ejemplo el resultado es exactamente el mismo que el de la formulación lineal entera; sin embargo, cabe mencionar que no en todos los casos de estudio se repite el mismo resultado, debido a que en el método heurístico se pueden presentar varias opciones en igualdad de circunstancias (misma frecuencia, rango de ciclo más pequeño, sin pertenencia a un ciclo previamente eliminado, etc.). Por lo tanto, este método puede generar diversas combinaciones de conjuntos de corte, como se podrá observar en la tabla de resultados. Esta posibilidad puede ofrecer ventajas, ya que es obvio que un conjunto mínimo de corte puede tener atributos más deseables que otros para una aplicación particular.

Es importante mencionar que el método heurístico proporciona conjuntos mínimos de corte para todos los casos de estudio de este trabajo, al igual que la formulación lineal entera. Además, cuando se implementa en computadora, también proporciona resultados en corto tiempo. Por lo tanto, este método es una buena opción para la selección de conjuntos mínimos de corte. Por otro lado, en algunos casos, da diferentes conjuntos mínimos de corte que los obtenidos usando la formulación lineal entera, por lo que al analizar los arcos de ambos conjuntos se podría seleccionar el más adecuado para resolver el caso de estudio planteado.

3.4 Secuencia de Cálculo

La eliminación de los arcos de cualquier conjunto mínimo de corte de un digrafo conectado *G* de *N* nodos produce un subgrafo conectado *G'* que contiene todos los *N* nodos de *G* y no tiene ciclos.

En el caso de los procesos químicos, un arco de corte es una corriente cuyas variables independientes deben ser estimadas. Por consiguiente, para efectos de cálculo, un arco de corte se transforma en una corriente conocida, razón por la cual se puede eliminar del DFI reducido. De esta manera, como se observa en la figura 3.2 para el conjunto de corte {2, 5} del ejemplo bajo estudio, desaparecen conexiones en el DFI dando lugar a digrafos acíclicos, lo que permite desarrollar un procedimiento de cálculos secuenciales e iterativos. Tal procedimiento se determina usando un método de búsqueda de nodos sumidero en el digrafo acíclico, tal como se explica a continuación.

3.4.1 Algoritmo para determinar la secuencia de cálculo

La secuencia de cálculos se obtiene cuando se aplica al digrafo original el siguiente algoritmo simple para encontrar el ordenamiento de nodos:

Datos de entrada. Sea Y el conjunto mínimo de corrientes de corte que contiene solamente un número entero y finito de miembros denotados por M.

Paso 1. Identificar la existencia de algún nodo sumidero e ir al paso 2. Si no hay este tipo de nodos, ir al paso 5.

Paso 2. El nodo identificado es localizado en la lista de la secuencia inversa de nodos.

Paso 3. Eliminar del digrafo el nodo identificado con sus respectivas corrientes de entrada.

Paso 4. Si ninguno de los nodos existentes en el digrafo remanente es un nodo sumidero, ir al paso 6; en caso contrario, regresar al paso1.

Paso 5. Eliminar del digrafo las M corrientes pertenecientes al conjunto mínimo de corrientes de corte Y y regresar al paso 1.

Paso 6. Si lo único que resta del digrafo es un nodo, localizarlo al final de la lista de la secuencia inversa de nodos e ir al paso 7; en caso contrario aplicar el paso 5.

Paso 7. Reacomodar de manera inversa en la lista de la secuencia de cálculo los nodos obtenidos en la lista de la secuencia inversa de nodos, de tal modo que el último nodo de esta lista sea el primero de la nueva lista generada.

3.4.1.1 Ejemplo del algoritmo para determinar la secuencia de cálculo

Considerando el resultado obtenido en el ejemplo 3.3.2.1, esto es, {2, 5} como conjunto mínimo de corrientes de corte, la aplicación del algoritmo de ordenamiento a la figura 3.2 da los siguientes resultados:

Paso 1. El digrafo original no tiene nodos sumidero como se observa en la figura 3.7a, por lo que permanece sin cambio. Se va al paso 5.

Paso 5. Se eliminan del digrafo las M corrientes mínimas de corte pertenecientes al conjunto Y, como se establece en la figura 3.7b, y se va al paso 1.

Paso 1. El nodo v_1 es el primero identificado como nodo sumidero. Se avanza al paso 2.

Paso 2. El nodo v_1 es localizado en la lista de la secuencia inversa de nodos.

Paso 3. Se elimina del digrafo el nodo v_1 con sus respectivas corrientes de entrada 1, 3 y 4, como se observa en la figura 3.7c.

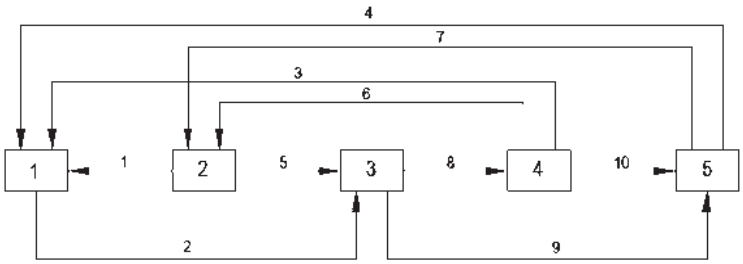
Paso 4. Aún existen nodos sumidero, por lo que se va al paso 1.

En el paso 1 es identificado ahora el nodo v_2 como nodo sumidero, el cual es ubicado en la lista de la secuencia inversa de nodos de acuerdo al paso 2; luego, en el paso 3, este nodo es eliminado del digrafo con sus respectivas corrientes de entrada 6 y 7, como se muestra en la figura 3.7d. De forma automática, del paso 4 se avanza al paso 1 para continuar con la identificación de nodos sumidero. Es identificado entonces el nodo v_5 y localizado, de acuerdo al paso 2, en la lista de la secuencia inversa de nodos. Dicho nodo, como se establece en el paso 3, es eliminado del digrafo junto con sus corrientes de entrada 10 y 9, como se observa en la figura 3.7e. En seguida, del paso 4 se avanza al paso 1 y se identifica como último nodo sumidero al nodo v_4, el cual es localizado en la lista de la secuencia inversa de nodos de acuerdo al paso 2 y eliminado del digrafo con su corriente de entrada 8 en base a lo establecido en el paso 3. El resultado de esta eliminación se puede observar en la figura 3.7f.

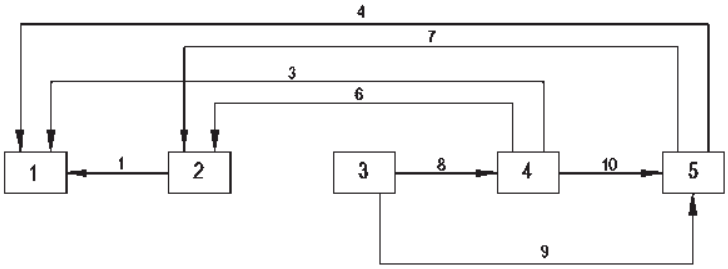
A continuación, como el nodo restante no es un nodo sumidero, del paso 4 se avanza al paso 6 y se localiza al final de la lista de la secuencia inversa de nodos al nodo v_3. Se avanza al paso 7, el cual indica el reacomodo de los nodos en una nueva lista, para obtener

como resultado final la secuencia de cálculo en la que deberán de ser resueltos los nodos del digrafo bajo estudio, obteniendo así el siguiente resultado:

Orden de cálculo para el conjunto de corte (2,5):

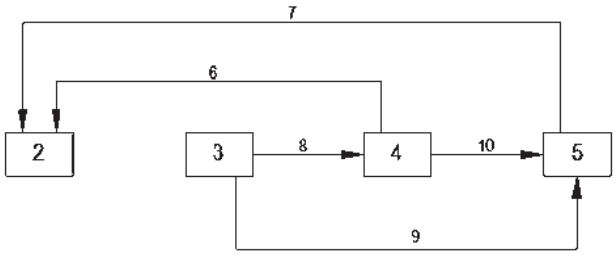


(a)



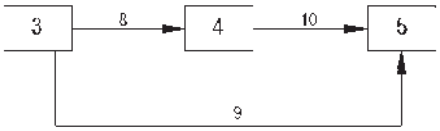
(b)

Lista de la secuencia inversa de nodos: v_1



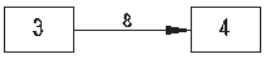
(c)

Lista de la secuencia inversa de nodos: v_1, v_2



(d)

Lista de la secuencia inversa de nodos: v_1, v_2, v_5



(e)

Lista de la secuencia inversa de nodos: v_1, v_2, v_5, v_4

(f)

Lista de la secuencia inversa de nodos: v_1, v_2, v_5, v_4, v_3

Lista de la secuencia de cálculo: v_3, v_4, v_5, v_2, v_1 (g)

Figura 3.7 Algoritmo para determinar la secuencia de cálculo aplicada al digrafo reducido de Rubin (1962)

Lista de la secuencia de cálculo: v_3, v_4, v_5, v_2, v_1

$$v_3 \longrightarrow v_4 \longrightarrow v_5 \longrightarrow v_2 \longrightarrow v_1$$

Este conjunto de corte rompe todos los ciclos y representa un número mínimo de corrientes de corte.

Los algoritmos de rasgado y ordenamiento presentados anteriormente han sido aplicado a varios procesos con el fin de identificar las corrientes de corte. Estos procesos varían en complejidad desde 5 nodos y 10 corrientes hasta 109 nodos y 174 corrientes.

3.5 Comprobación de los resultados obtenidos para el digrafo de Rubin (1962)

Una forma de comprobar que los resultados obtenidos son ciertos y confiables es resolviendo manualmente el digrafo bajo estudio, partiendo del conjunto mínimo de corrientes de corte obtenido, como se hace a continuación con el digrafo de Rubin (1962).

Se marcan las corrientes del conjunto mínimo de corte en el digrafo, para este caso las corrientes 2 y 5.

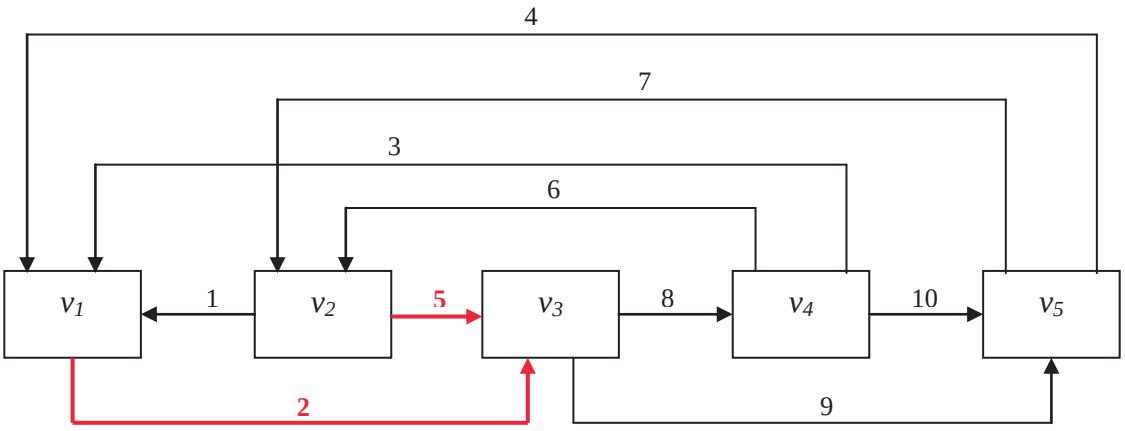


Figura 3.8 Ubicación del conjunto mínimo de corrientes de corte en el digrafo reducido de Rubin (1962)

Estas corrientes, por el hecho de formar parte del conjunto mínimo de corrientes de corte, se convierten en corrientes conocidas mediante la suposición de sus variables independientes. Esto permite transformar un digrafo cíclico en uno acíclico, como ya se explicó anteriormente.

Dicho esto, lo que prosigue es observar el digrafo para verificar realmente que, conociendo los datos de las corrientes 2 y 5, el nodo v_3, ubicado como primer nodo a resolver en la lista de la secuencia de cálculo obtenida, puede ser resuelto.

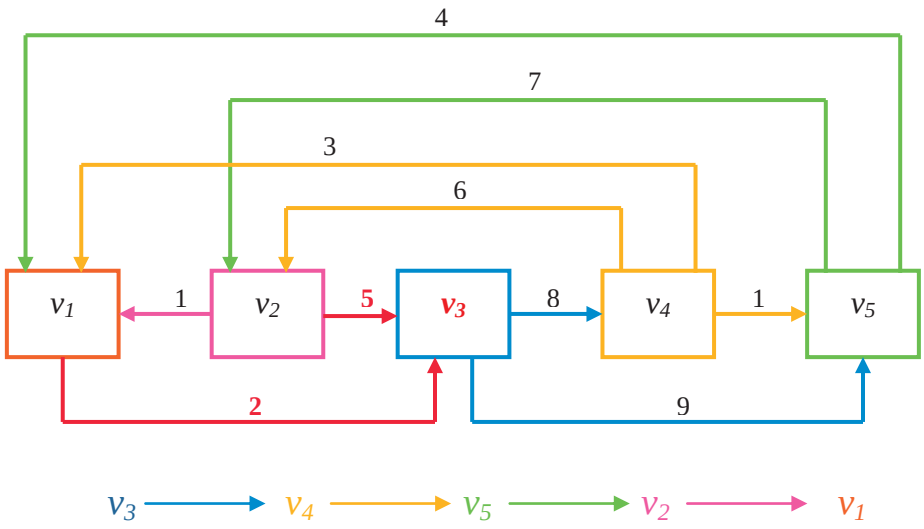


Figura 3.9 Comprobación del conjunto mínimo de corrientes de corte y de la secuencia de cálculo obtenidas para el digrafo reducido de Rubin (1962)

Es comprobable, en la figura 3.8, que a partir de las corrientes de corte 2 y 5 (color rojo) es posible resolver el nodo v_3 y que gracias a esto se pueden conocer las corrientes 8 y 9 (color azul), las cuales hacen posible la solución del nodo v_4 y, por ende, conocer las corrientes 1, 3 y 6 (color amarillo). Nos podemos percatar de que el nodo v_5 puede resolverse a continuación, ya que ahora se conocen sus corrientes de entrada, con lo que automáticamente quedan especificadas las corrientes 4 y 7 (color verde). Finalmente, tomando en consideración todas las corrientes conocidas hasta el momento y analizando los nodos aún sin resolver, se observa que el nodo v_2 ya puede resolverse y, por lo tanto, también su corriente de salida 1 (color rosa). Esto hace posible simular el último nodo sin resolver, es decir, v_1.

Con este análisis se llega a la conclusión de que los resultados obtenidos del conjunto mínimo de corrientes de corte y la secuencia de cálculo para el digrafo de Rubin (1962) son correctos.

3.6 Software

Hasta aquí se sabe que el procedimiento para la identificación de un conjunto mínimo de corrientes de corte consiste en dos pasos principalmente: la identificación de los ciclos simples del sistema y la selección del conjunto mínimo de corrientes de corte. El primer paso se lleva a cabo usando un algoritmo de búsqueda en profundidad prioritaria, el cual es implementado en el lenguaje JAVA. Por otro lado, la selección de corrientes de corte se formula como un problema de programación entera, el cual se resuelve usando el paquete GAMS.

El último algoritmo, el cual es utilizado para establecer el orden de cálculo, también es implementado en el lenguaje JAVA.

La ventaja de la implementación en computadora de estos algoritmos reside en la reducción del tiempo requerido para resolver manualmente estos problemas, además de que también se pueden eliminar errores ocasionados por distracciones durante los cálculos manuales. Una ventaja importante de estos programas de cómputo es que pueden ser usados para determinar el orden de cálculo secuencia de DFI cíclicos muy grandes y complejos, que incrementarían drásticamente el trabajo manual que se requeriría en caso de no contar con estas herramientas computacionales. Por lo tanto, estos programas permiten resolver problemas complejos invirtiendo poco tiempo.

CAPITULO 4

CASOS DE ESTUDIO

4.1 Aplicación de los algoritmos implementados en computadora

A continuación se presentan catorce diagramas de flujo cíclicos de diferente complejidad, previamente reportados en la literatura (Lakshminarayanan y Col., 1992), con sus respectivos resultados después de la aplicación de los algoritmos de rasgado y ordenamiento implementados en computadora. Se explicará primeramente a detalle el caso 1 aplicando dichos algoritmos, con el objetivo de mostrar claramente los pasos a seguir para llegar a los resultados finales.

4.1.1 Acerca del algoritmo implementado en computadora para la identificación de ciclos simples

El algoritmo para la identificación de los ciclos simples se implementó en computadora mediante el lenguaje JAVA. Uno de los datos que es indispensable para poder trabajar con este programa es la matriz de adyacencia de nodos, en la cual, en lugar de utilizar el número 1 para identificar la conexión entre un nodo y otro, se utilizan los números de los arcos que conectan a los nodos, para así obtener la matriz de ciclos expresada en arcos que es necesaria para la segunda parte de la determinación del conjunto mínimo de corrientes de corte. El siguiente ejemplo muestra de forma gráfica y detallada cómo utilizar este programa.

4.1.1.1 Aplicación del algoritmo implementado en computadora con el lenguaje JAVA, para la identificación de ciclos simples en el caso de estudio 1

Teniendo como dato de partida el digrafo reducido del caso bajo estudio, sin nodos fuente o sumidero (figura 4.1), se procede a realizar los siguientes pasos.

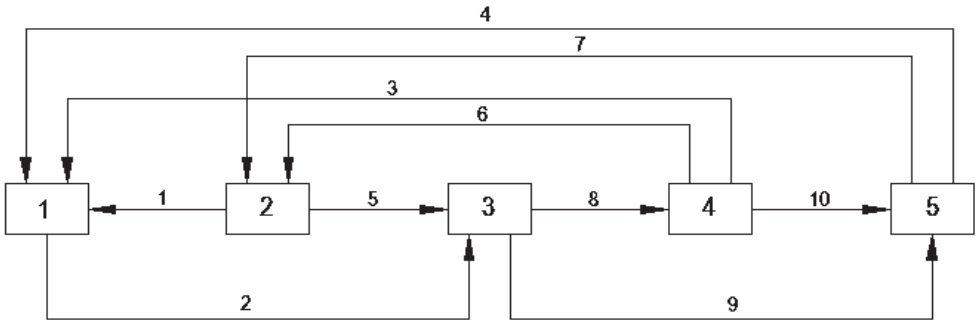


Figura 4.1 Digrafo reducido de Rubin (1962) sin nodos fuente o sumidero

Paso 1. Generar la matriz de adyacencia de nodos. Esta se puede crear en un archivo de Excel, copiarla y pegarla en JAVA, o se puede generar directamente en el programa. En la figura 4.2 se observa cómo se debe ingresar la matriz de adyacencia de nodos para que el programa genere los ciclos. Las filas son los nodos emisores y las columnas son los nodos receptores; esto se puede entender de una mejor forma si se observa la Figura 4.1. Los ceros indican la no conectividad entre nodos y, por el contrario, los números diferentes de cero, que se encuentran dentro de la matriz, indican la conectividad entre ellos pues corresponden al número de arco que une dos nodos. Por ejemplo, en el caso de la primera fila, el número 2, ubicado debajo de la columna tres, indica que el nodo 1 y el nodo 3 están unidos por el arco 2 y que este arco se dirige del nodo 1 al nodo 3.

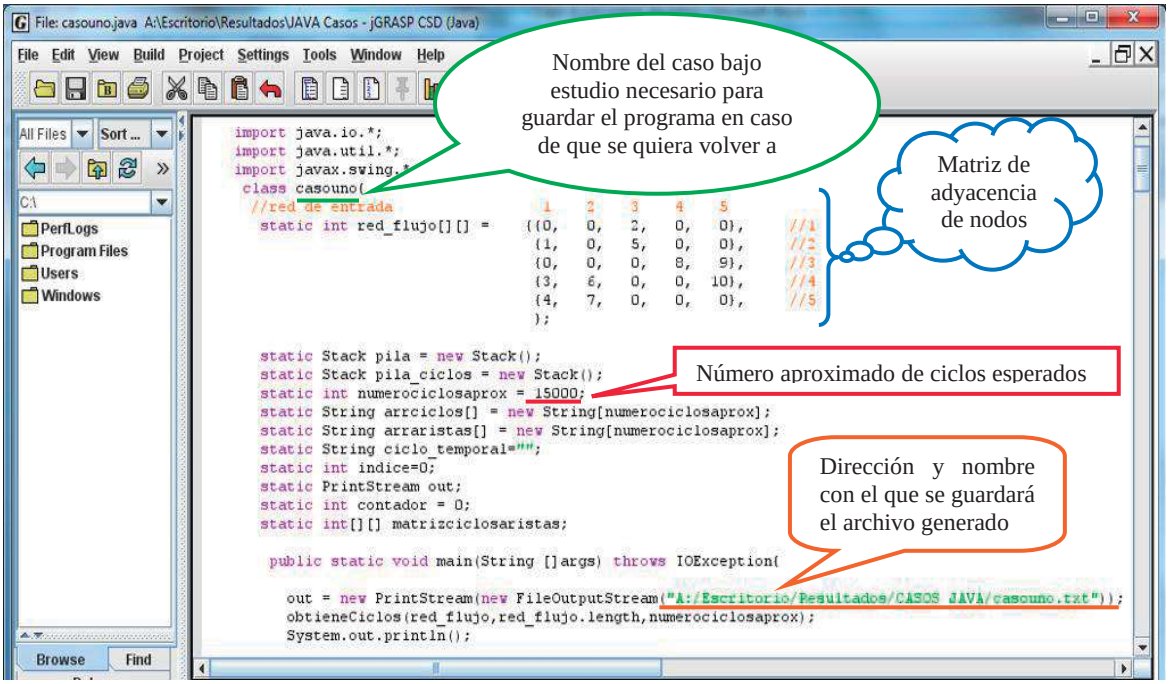


Figura 4.2 Datos necesarios para correr el programa de identificación de ciclos simples en JAVA

Paso 2. Asentar la cantidad de ciclos aproximados que se espera sean arrojados por el programa. Esto se realiza en el tercer renglón, después de la matriz de adyacencia de nodos, como se observa en la figura 4.2. Este dato puede ser o no modificado, esto dependerá de si el digrafo a analizar es demasiado grande y si se espera de él una cantidad de ciclos mayor a 15000. Esta cantidad fue considerada para el caso de estudio número 14, ya que se trata de un sistema muy grande, como se verá más adelante; dicha cantidad no afecta a ninguno de los casos anteriores a éste, por lo que no se modificó en ninguno de los casos de estudio.

Paso 3. Especificar la ruta en donde se desea guardar el archivo que generará el programa con los resultados. Para lo cual, el siguiente renglón a modificar es el doceavo después de la matriz de adyacencia de nodos, como se muestra en la figura 4.2, en donde se escribirá la ruta, el nombre con el que se quieren guardar los resultados y al final la extensión .txt. Dicho documento se generará cuando el programa termine de correr y en él arrojará los resultados de todos los ciclos simples existentes en el digrafo estudiado.

Paso 4. Ingresar la cantidad de corrientes que conforman el digrafo. Se modifica, por último, el catorceavo renglón ubicado después de la ruta anteriormente señalada, tal como se observa en la figura 4.3. Después de la letra "c" y el signo "=" se ingresa el número de corrientes del caso bajo estudio.

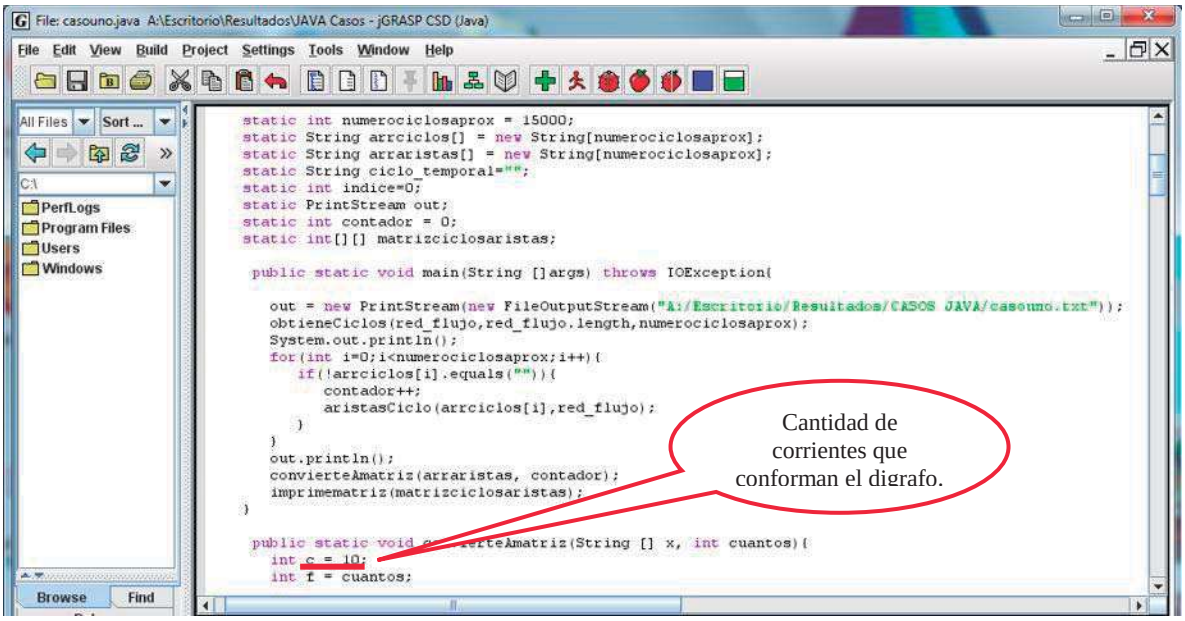


Figura 4.3 Último dato necesario para correr el programa de identificación de ciclos simples en JAVA

Paso 5. Guardar los datos modificados dando clic sobre el ícono ya conocido. Es conveniente que para cada caso estudiado se guarde este programa con un nombre diferente al nombre de origen, para conservar los datos del mismo y utilizarlos las veces que sean necesarias. Si esto último se lleva a cabo, se deberá modificar el nombre que aparece un renglón antes de ingresar la matriz de adyacencia de nodos entre la leyenda "class" y el signo "{" , como se muestra en la Figura 4.2, por el nombre del caso bajo estudio y, enseguida de esto, dar clic en la opción File, después en "Save as", en donde se va a seleccionar la ubicación deseada para este programa; aquí también se escribe, en el recuadro "nombre de archivo", el nombre del caso bajo estudio. De no se escribe el mismo nombre jGRASP, no permitirá que éste se guarde.

Paso 6. Compilar. Este es un paso importante para hacer que el programa corra, pues al momento de hacer modificaciones este paso nos permite condensar la información en JAVA para que reconozca los comandos e instrucciones dadas. Compilar es un paso muy sencillo que consiste en dar un clic sobre el ícono , como se observa en la figura 4.4. Cuando el programa termina de compilar arroja el texto "----jGRASP: operation complete", dicho texto se puede observar en el recuadro blanco que aparece en la parte inferior de la pantalla. Esto indica que el programa fue compilado con éxito y que no hubo errores en esta etapa.

Paso 7. Correr el programa. Después de compilar se procede a ejecutar el programa, lo que se logra dando un clic sobre el ícono . Enseguida sólo hay que esperar a que el programa arroje los resultados; cuando esto pase y el programa haya terminado de operar, en el recuadro inferior de JAVA, aparecerán todos los ciclos generados y al final de éstos la leyenda "*----jGRASP: operation complete"*, como en el paso anterior, tal como se muestra en la figura 4.5.

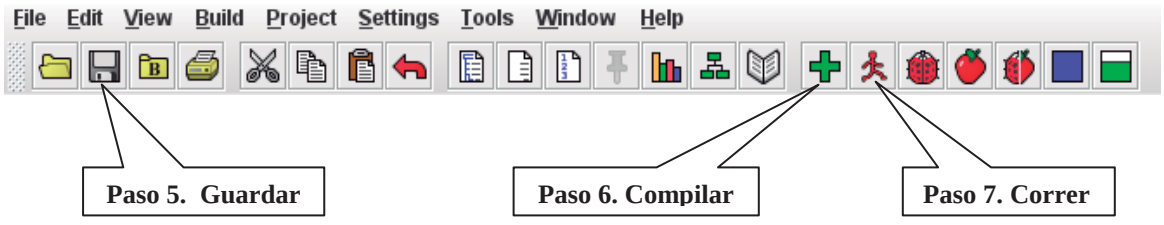


Figura 4.4 Iconos importantes para correr el programa

Los resultados también se imprimen en un archivo de texto (bloc de notas) de dos formas, tal como se muestra en la figura 4.6. Al inicio de la hoja aparecen enlistados todos los ciclos simples en términos de los arcos que lo conforman. Más adelante aparece una matriz de unos y ceros, en la cual las columnas representan los arcos del digrafo y las filas los ciclos simples del mismo. Esta matriz es la matriz de ciclos simples: los unos indican la aparición de un arco en un ciclo, mientras que los ceros indican lo contrario.

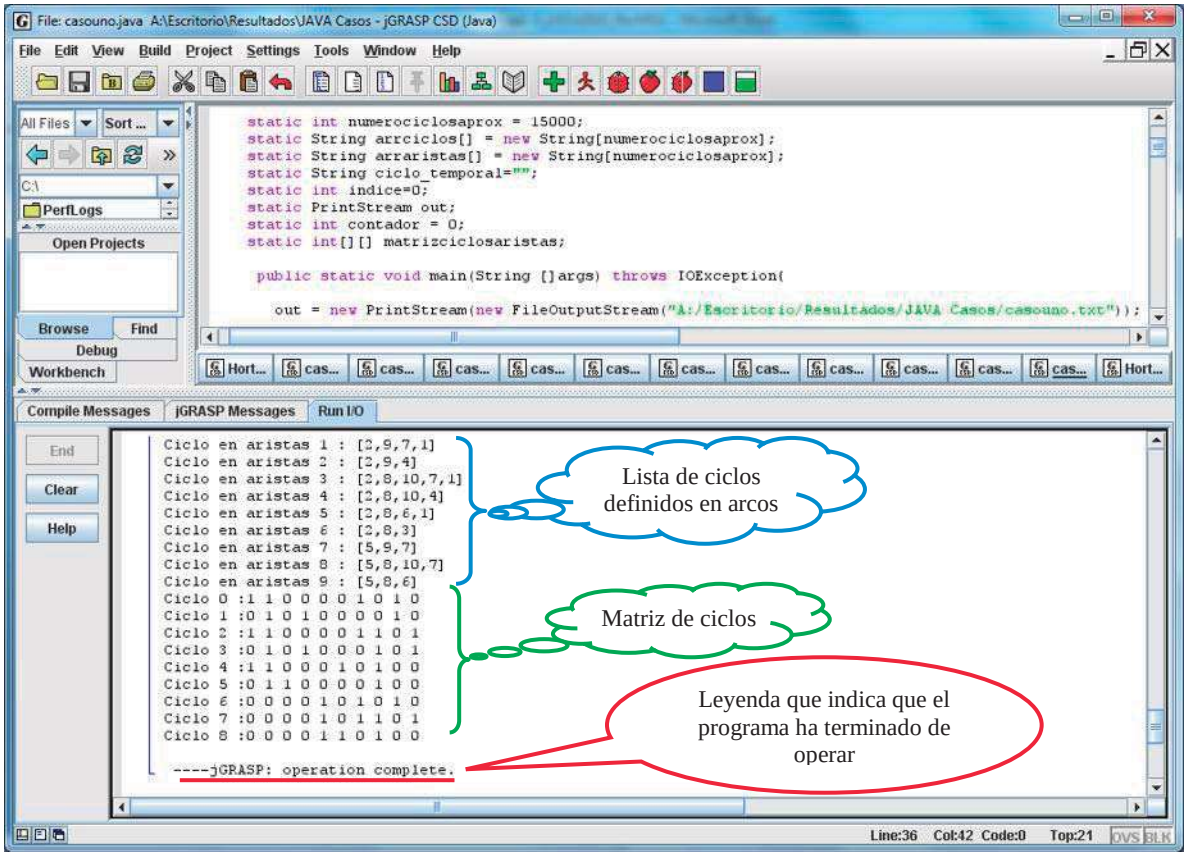


Figura 4.5 Resultados y confirmación de operación finalizada

La matriz de ciclos es la parte más importante de estos resultados, ya que minimiza el trabajo en el ingreso de datos de los programa desarrollados para la identificación del conjunto mínimo de corte.

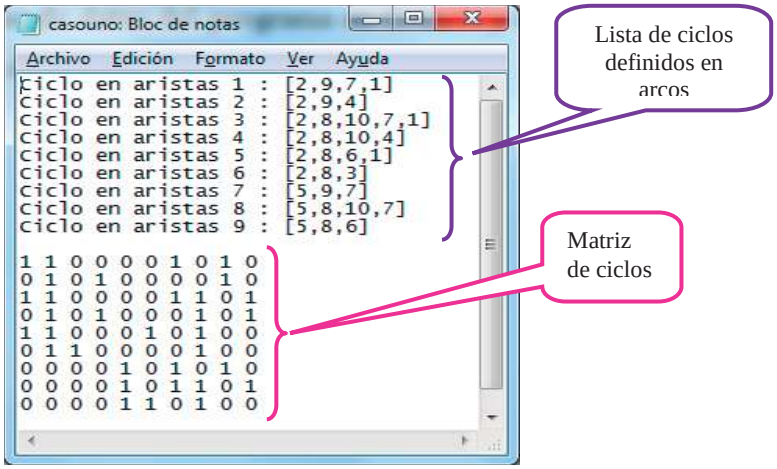


Figura 4.6 Resultados impresos por JAVA en bloc de notas

4.1.2 Acerca del algoritmo implementado en computadora para la selección del conjunto mínimo de corrientes de corte

En lo que a la selección del conjunto mínimo de corrientes de corte se refiere, en el presente trabajo se presentan dos formulaciones para su determinación. La primera opción consiste en un algoritmo de programación entera que se resolvió usando el paquete GAMS, mientras que la segunda opción es el método heurístico, descrito en el capítulo anterior y que fue implementado en el lenguaje JAVA. Ambos algoritmos requieren como punto de partida la matriz de ciclos simples generada por el programa anterior. A continuación se expone, de forma detallada, cómo correr ambos programas.

4.1.2.1 Aplicación del algoritmo implementado en computadora empleando el paquete GAMS, para la selección de un conjunto mínimo de corrientes de corte en el caso de estudio 1

La información más importante para ejecutar este programa es la matriz de ciclos simples generada con el programa anterior. A continuación se describen a detalle los datos que dentro del programa, deberán ser modificados para obtener el conjunto mínimo de corrientes de corte.

Paso 1. Definir el número de ciclos que conforman la matriz de ciclos del caso bajo estudio. Este dato se incluye en el programa modificando el número que aparece después de la leyenda "RENGLONES DE C", justamente como se señala en la figura 4.7.

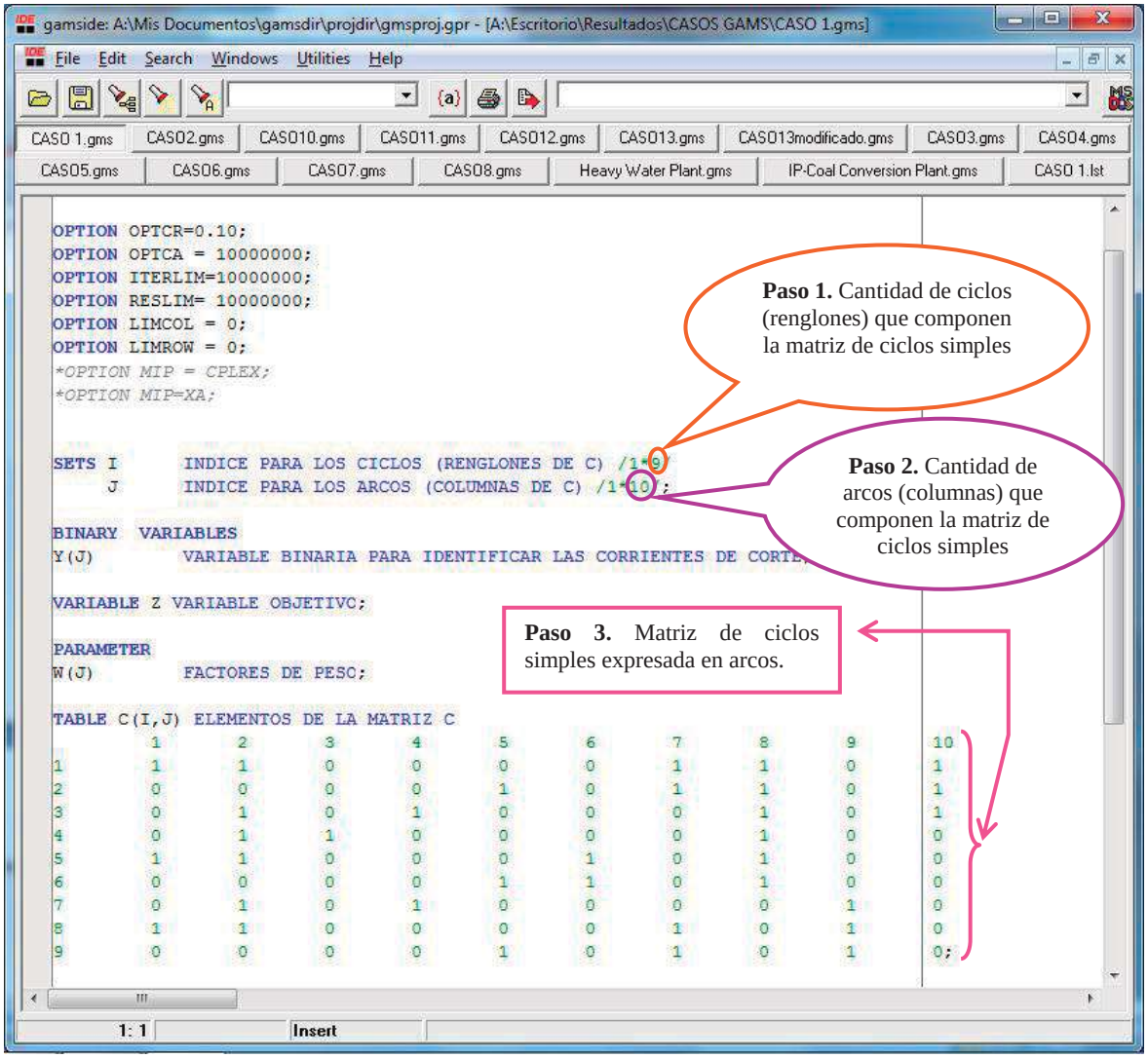


Figura 4.7 Datos necesarios para correr el programa para la selección del conjunto mínimo de corte en GAMS

Paso 2. Definir el número de arcos o columnas que componen la matriz de ciclos simples. Al igual que el paso anterior, este paso sólo incluye la modificación de un número que es el que aparece después de la leyenda "COLUMNAS DE C", también indicado en la figura 4.7.

Paso 3. Introducir al programa la matriz de ciclos simples. Este paso se puede llevar a cabo copiando la matriz de ciclos simples generada en un archivo de texto con el

algoritmo anterior y pegándola en el espacio adecuado tal como se observa en la figura 4.7. También es importante hacer notar que es indispensable darle el formato adecuado a la matriz de ciclos para que el programa funcione (ver figura 4.7, matriz de ciclos simples).

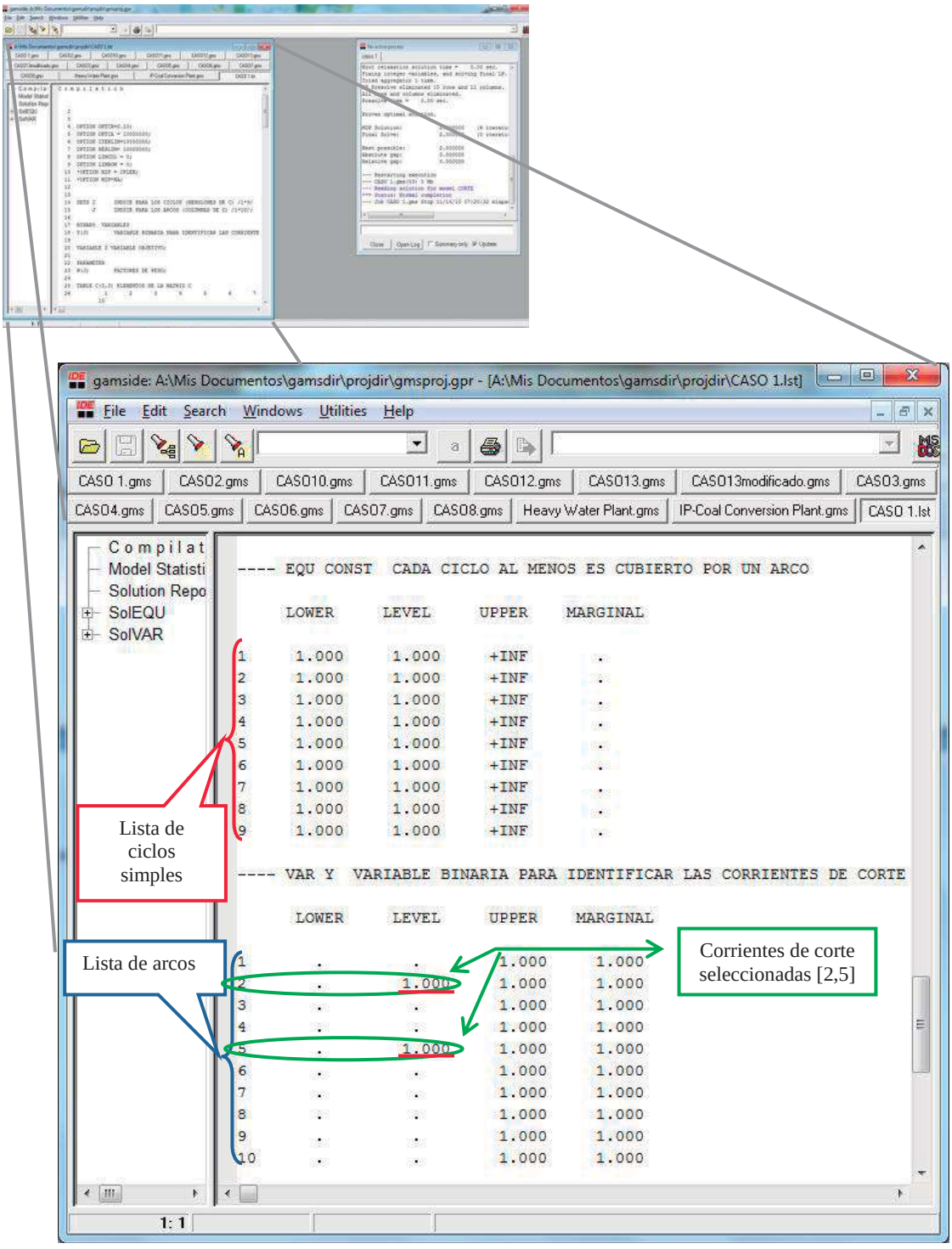


Figura 4.8 Ventana de resultados en GAMS

Paso 4. Guardar las modificaciones hechas al programa. Esto se realiza igual que en otros programas como Word o Excel.

Como se puede observar en la figura 4.8, la solución para el caso 1 es:

Corrientes de corte (2, 5)

4.1.2.2 Aplicación del algoritmo implementado en computadora empleando el lenguaje JAVA, para la selección de un conjunto mínimo de corrientes de corte en el caso de estudio 1.

Esta es la segunda formulación propuesta para la selección de un conjunto mínimo de corrientes de corte. El punto de partida para este programa es también la matriz de ciclos. Para esta opción se deben de seguir los siguientes pasos.

Paso 1. Copiar y pegar únicamente la matriz de ciclos simples, generada con el programa anterior de JAVA, en un archivo de texto nuevo (bloc de notas), como se puede observar en la figura 4.9. Enseguida guardar el archivo con el nombre que se prefiera.

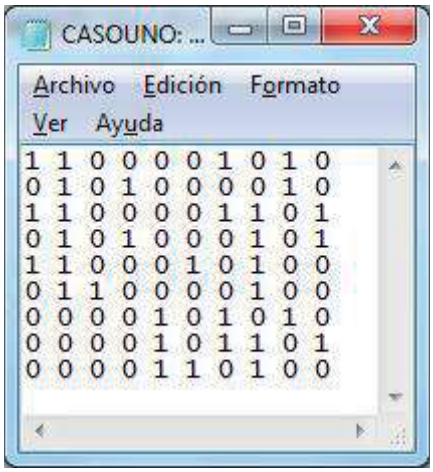


Figura 4.9 Matriz de ciclos simples necesaria para correr el programa en JAVA

Paso 2. Introducir, entre comillas, el nombre del archivo o la ruta del mismo en la línea 165 del programa. En caso de tener el programa en la misma carpeta que el archivo de texto con la matriz de ciclos simples, sólo será necesario escribir el nombre del documento, como se observa en la figura 4.10. Si éste no es el caso se tendrá entonces que anexar la ruta completa en donde se encuentra ubicado el archivo de texto, por ejemplo:

"A:/Escritorio/Resultados/metodo heurístico mejorado JAVA/CASOUNO". El número de la línea se puede observar en la parte inferior derecha de JAVA (ver figura 4.10).

Paso 3. Compilar el programa. Como ya se mencionó anteriormente, este es un paso importante para que JAVA condense la información y así reconozca los comandos e instrucciones dadas, así como las modificaciones realizadas. Solo se da un clic sobre el ícono  . Cuando el programa termina de compilar arroja el texto "----jGRASP: operation complete", el cual se observa en el recuadro blanco que aparece en la parte inferior de la pantalla. Esto indica que el programa fue compilado con éxito y que no hubo errores en esta etapa.

Paso 4. Ejecutar el programa. Dar clic sobre el ícono  . Cuando el programa haya terminado de operar, en el recuadro inferior de JAVA aparecerá el conjunto de corte seleccionado y enseguida la leyenda "*----jGRASP: operation complete"*, como se puede apreciar en la figura 4.10.

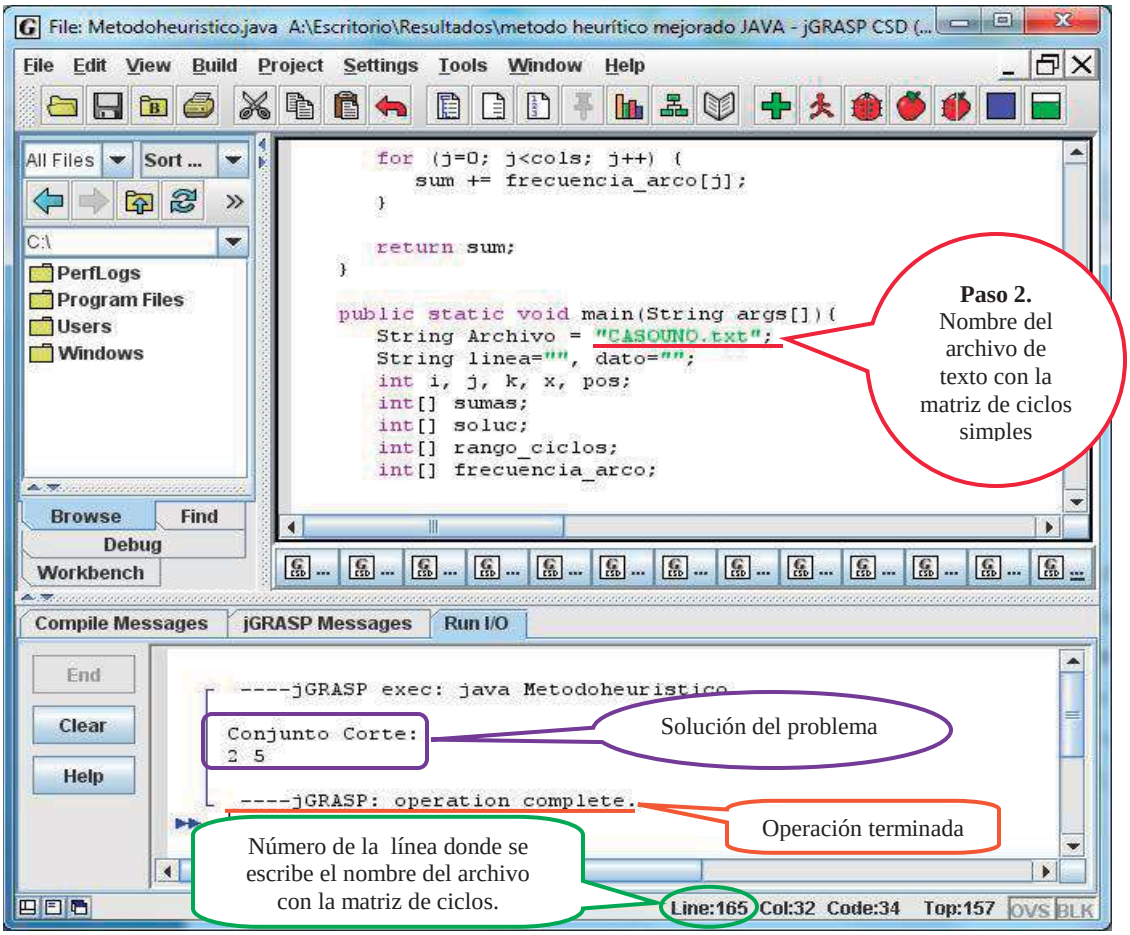


Figura 4.10 Paso 2 y resultado del conjunto de corte seleccionado en JAVA

Como se puede notar, el resultado arrojado por el programa en JAVA es el mismo que el calculado por el programa en GAMS. Esto no se repite en todos los casos de estudio pero si en la mitad de ellos, como se observa en la tabla 4.2, por lo que este programa se puede considerar como una opción para una mejor selección de un conjunto mínimo de corrientes de corte que mejor se adecue a las necesidades del caso a analizar. Cabe mencionar que ambos programas generan conjuntos mínimos de corte con la cantidad de datos deseados, lo cual se podrá apreciar más a detalle en el cuadro comparativo de resultados y en las conclusiones.

4.1.3 Acerca del algoritmo implementado en computadora para determinar el orden de cálculo

Este es el último paso para la solución del problema de rasgado y ordenamiento. En esta etapa se obtiene el orden de cálculo en el cual el problema cíclico inicial puede ser resuelto de manera acíclica suponiendo las variables independientes de las corrientes de corte. Cabe mencionar que existen varias opciones de orden de cálculo para algunos casos y que la solución propuesta por este programa es sólo una de ellas.

4.1.3.1 Aplicación del algoritmo implementado en computadora empleando el lenguaje JAVA, para determinar el orden de cálculo en el caso de estudio 1.

Los datos necesarios para que este programa funcione adecuadamente son: un archivo de texto que contenga la matriz de adyacencia de nodos, como la que se ingresó en el programa de JAVA para la identificación de los ciclos simples, y las corrientes de corte del caso de estudio a analizar. Para lo cual los pasos a seguir son los siguientes.

Paso 1. Generar un archivo de texto en el cual se ingrese la matriz de adyacencia de nodos, dándole a la matriz el formato que se muestra en la figura 4.11.

Paso 2. Indicar la ubicación de la matriz de adyacencia de nodos. En la línea 121 del programa se escribe, entre comillas, el nombre o la ruta de ubicación del texto en donde se guardó la matriz de adyacencia del paso 1, el nombre en caso de que el programa se encuentre dentro de la misma carpeta que la matriz de adyacencia o la ruta en caso de

encontrarse en carpetas distintas. En la figura 4.12 se puede observar a detalle cómo ingresar este dato.

Figura 4.11 Matriz de adyacencia de nodos

Paso 3. Ingresar las corrientes de corte. En el recuadro que aparece en la parte superior de la pantalla de JAVA, en el que lee "Run Arguments" (ver figura 4.12), señalar las corrientes del conjunto mínimo de corrientes de corte obtenidas usando cualquiera de los dos programas desarrollados para la selección de las mismas, separadas por un espacio en blanco. En caso de no ver este recuadro en la parte superior, anexarlo dando clic sobre la pestaña "Build" y enseguida se desplegará una lista de la cual se debe seleccionar la opción "Run Arguments"; con esto ya contamos con la herramienta para ingresar los datos de las corrientes de corte.

Paso 4. Compilar el programa. Como en los programas anteriores de JAVA, se da un clic sobre el ícono  . Y de igual forma, cuando el programa termina de compilar arroja el texto "----jGRASP: operation complete", el cual se puede observar en el recuadro blanco que aparece en la parte inferior de la pantalla, indicando que el programa fue compilado con éxito y que no hubo ningún error.

Paso 5. Correr el programa. Se hace clic sobre el  ícono. Cuando el programa termina de operar, en el recuadro inferior de JAVA aparece el conjunto de corte ingresado y enseguida se muestra, en forma de lista y numerado, el orden de cálculo encontrado por el programa. Para corroborar que el programa terminó de operar, se debe de cerciorar que aparezca, al final de esta lista, la leyenda "*----jGRASP: operation complete*", tal como se aprecia en la figura 4.12.

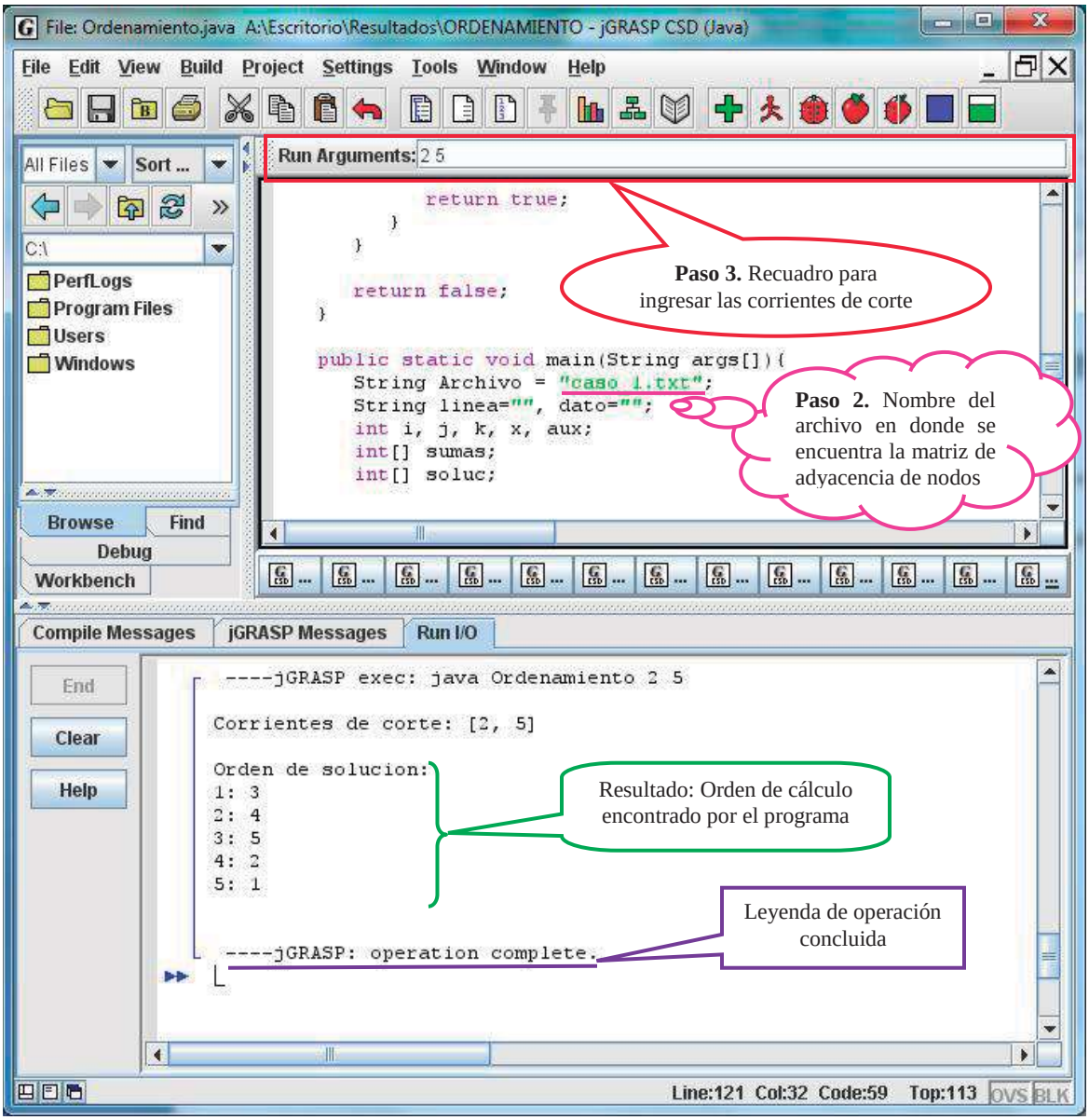


Figura 4.12 Paso 2, paso 3 y resultado del orden de cálculo encontrado en JAVA

Para el caso bajo estudio, caso 1, el orden de cálculo, de acuerdo al programa en JAVA, es: 3, 4, 5, 2 y 1. Por lo tanto, éste es el orden en el que se podrían calcular secuencialmente los nodos del digrafo estudiado, de acuerdo al conjunto mínimo de corrientes de corte [2, 5].

Para todos los casos de estudio se siguen los pasos anteriormente descritos en cada programa.

4.2 Resultados de los catorce casos de estudio

Se consideraron catorce casos de estudio para poner a prueba los algoritmos desarrollados para la solución del problema de rasgado y ordenamiento. Estos casos de estudio se tomaron de los trabajos de Lakshminarayanan y col. (1992), Zhou Li (1988), Gundersen y Hertzberg (1983) y Sood (1979).

En la tabla 4.1 se muestran las características de cada caso de estudio, tales como el número de nodos y de arcos. En ella se observa que el problema número catorce es el caso de mayor complejidad debido a la cantidad de nodos y arcos que lo conforman. De hecho, por su complejidad este proceso tiene 13746 ciclos simples.

Tabla 4.1 Complejidad de los catorce casos de estudio

Caso No.	Número de			Nombre del digrafo
	Nodos	Corrientes	Ciclos	
1	5	10	9	Digrafo de Rubin
2	6	11	7	Digrafo de Forder y Hutchinson
3	5	8	4	Digrafo de Christensen y Rudd (I)
4	4	6	3	Digrafo de Upadhye y Grens
5	19	31	20	Digrafo de Sargent y Westerberg
6	25	32	10	Digrafo de Christensen y Rudd (II)
7	30	42	31	Digrafo de Christensen y Rudd (III)
8	41	61	100	Digrafo de una planta de ácido sulfúrico
9	51	84	20	Digrafo de una planta de conversión de carbón
10	12	21	22	Digrafo de Pho y Lapidus
11	15	35	27	Digrafo de Barkley y Motard
12	29	37	11	Digrafo de una planta de alquilación de HF
13	50	79	22	Digrafo de un proceso de hidrogenación para la extracción de aceite vegetal
14	109	174	13746	Digrafo de planta de agua pesada

Los catorce digrafos de los casos de estudio considerados se presentan a continuación.

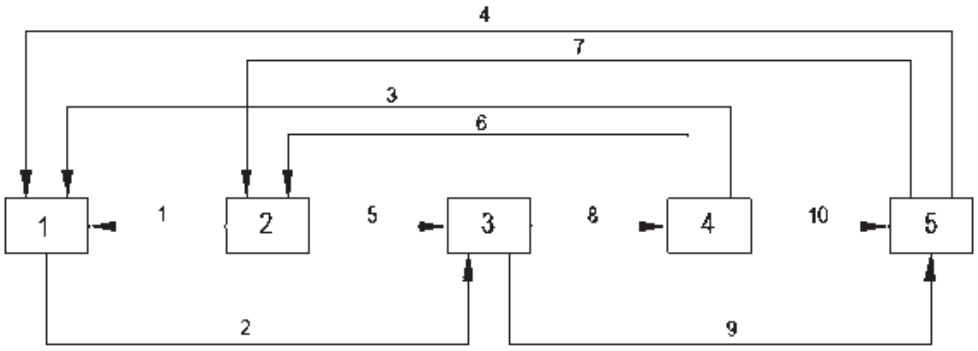


Figura 4.13 Caso 1. Digrafo de Rubin

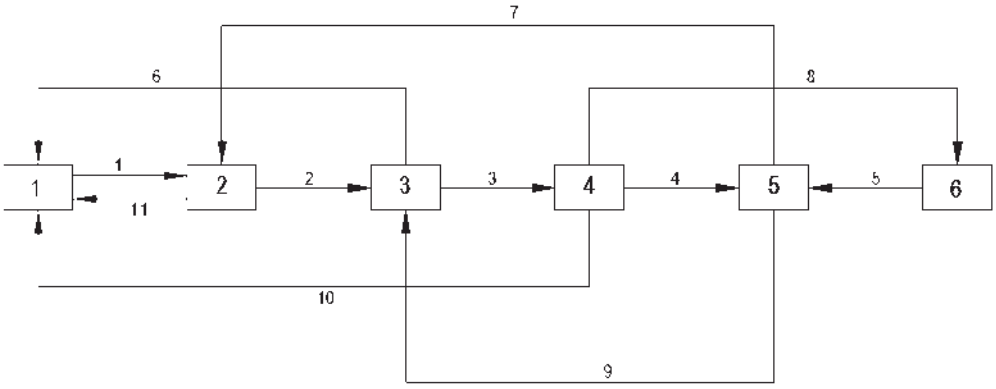


Figura 4.14 Caso 2. Digrafo de Forder y Hutchinson

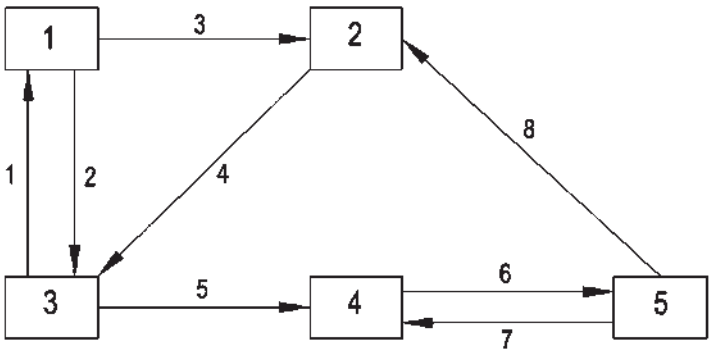


Figura 4.15 Caso 3. Digrafo de Christensen y Rudd (I)

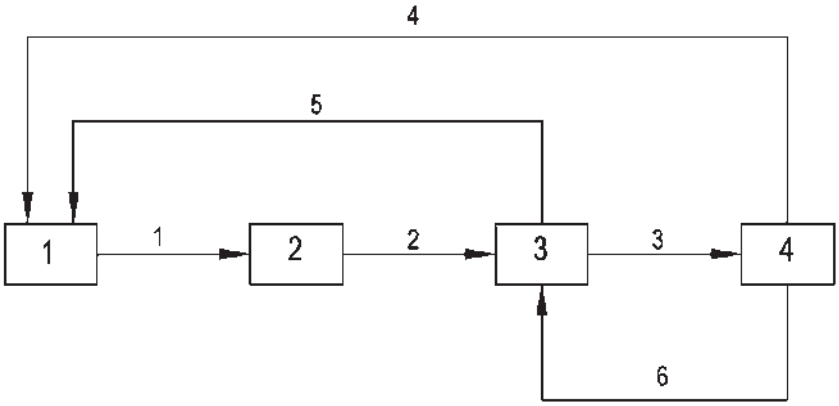


Figura 4.16 Caso 4. Digrafo de Upadhye y Grens

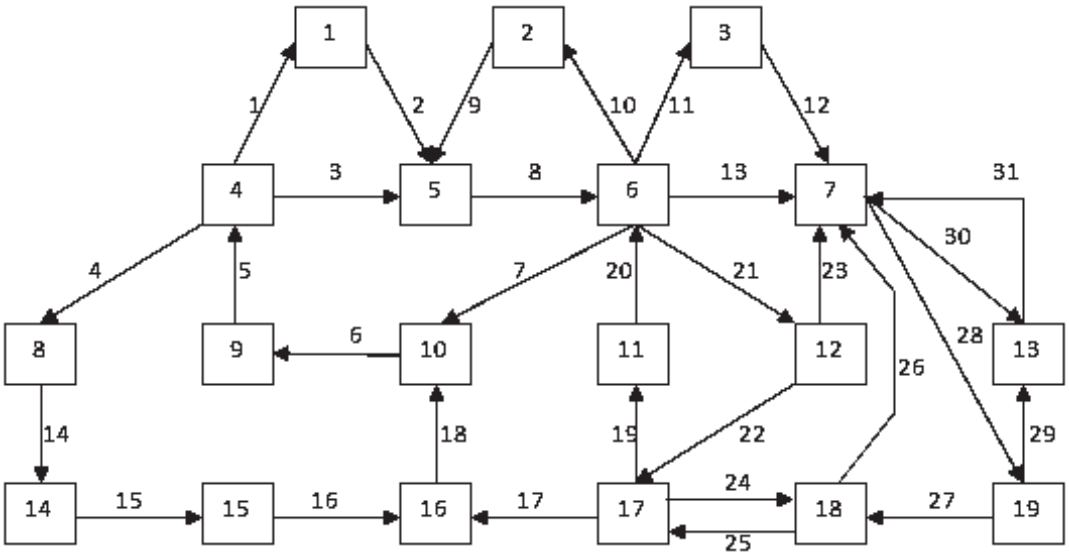


Figura 4.17 Caso 5. Digrafo de Sargent y Westerberg

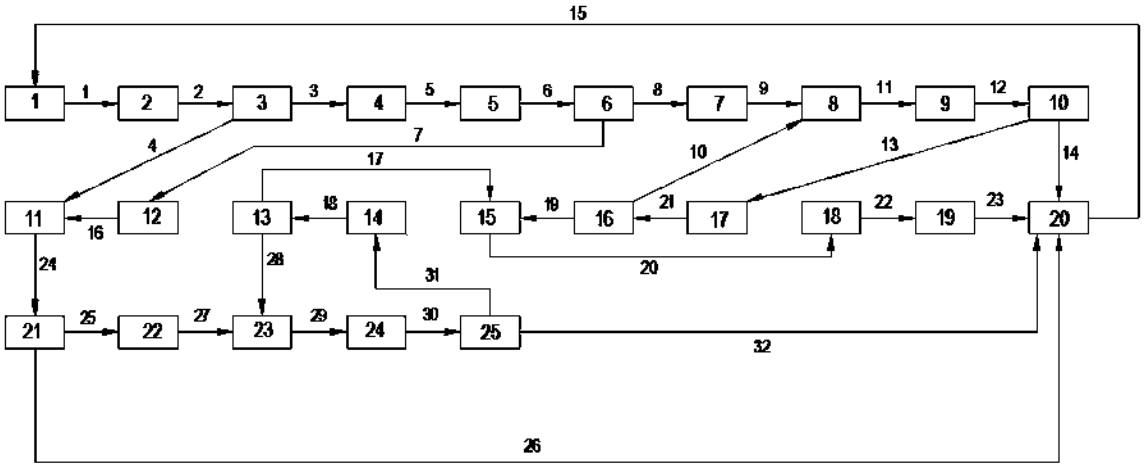


Figura 4.18 Caso 6. Digrafo de Christensen y Rudd (II)

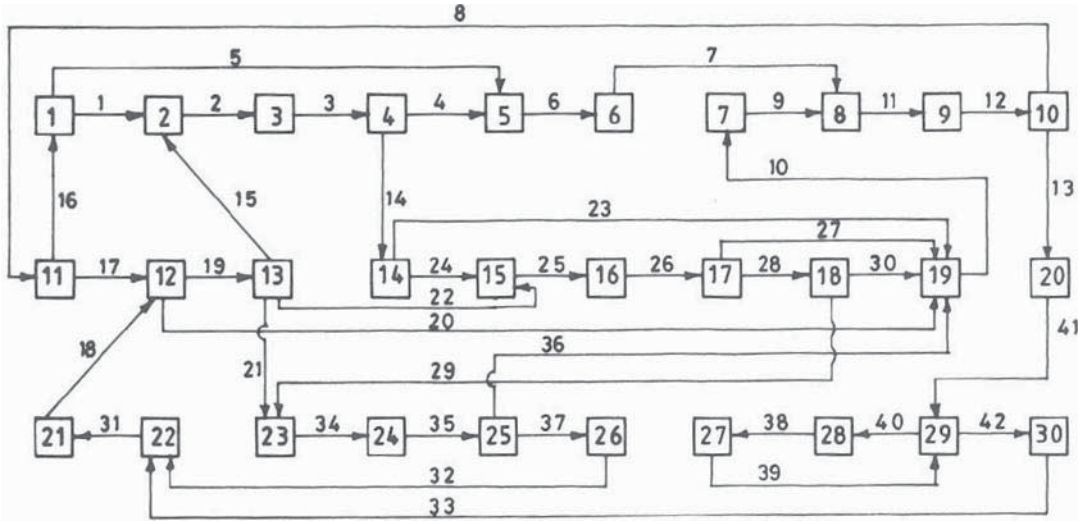


Figura 4.19 Caso 7. Digrafo de Christensen y Rudd (III)

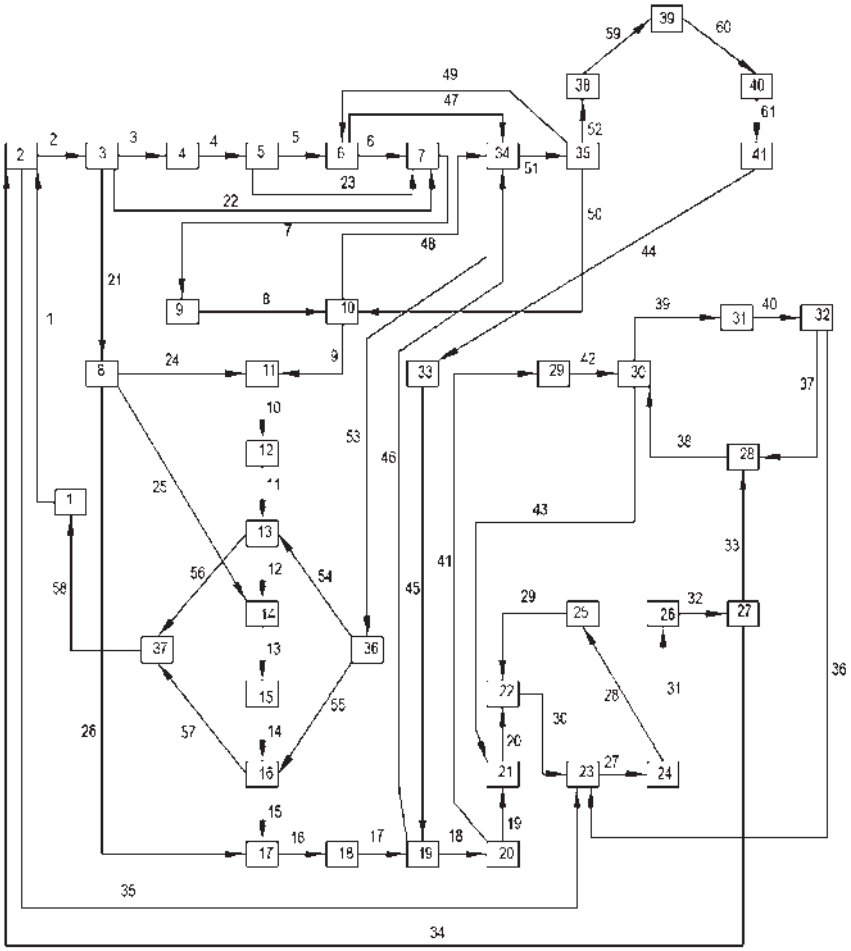


Figura 4.20 Caso 8. Digrafo de una planta de ácido sulfúrico

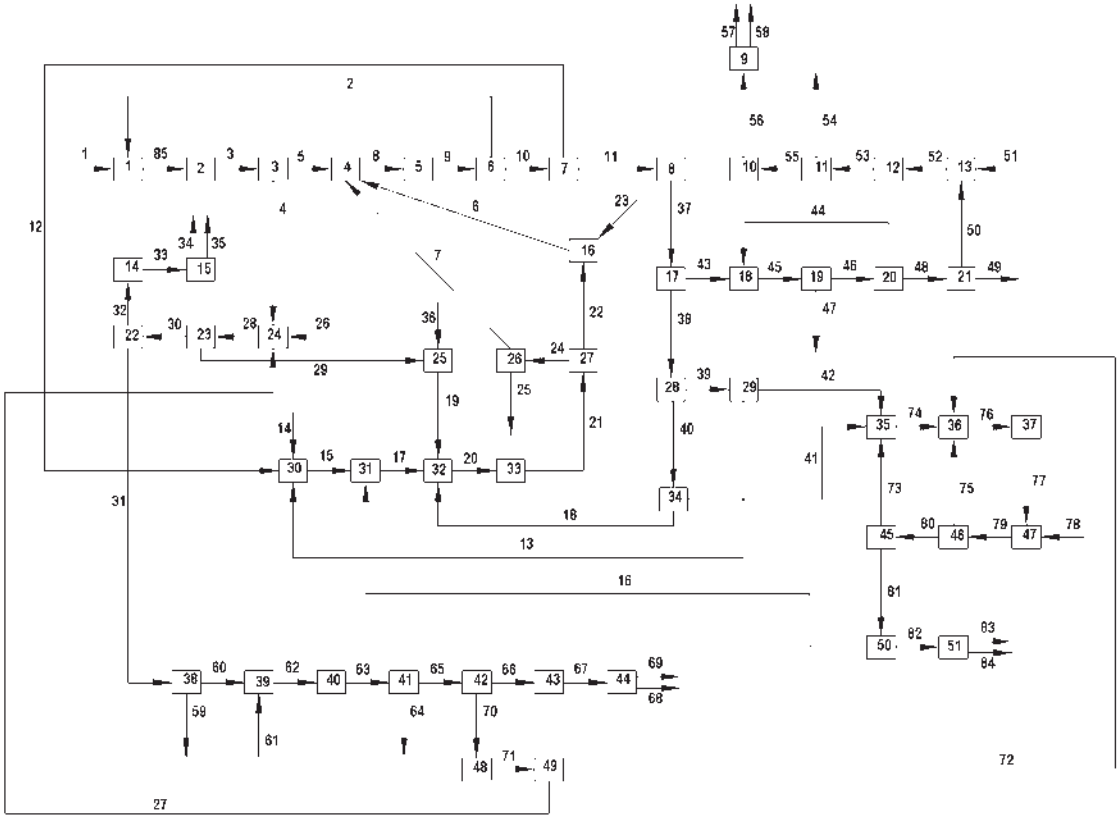


Figura 4.21 Caso 9. Digrafo de una planta de conversión de carbón

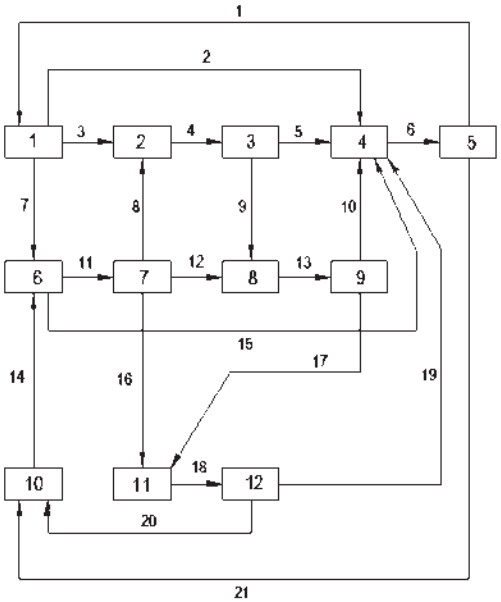


Figura 4.22 Caso 10. Digrafo de Pho y Lapidus

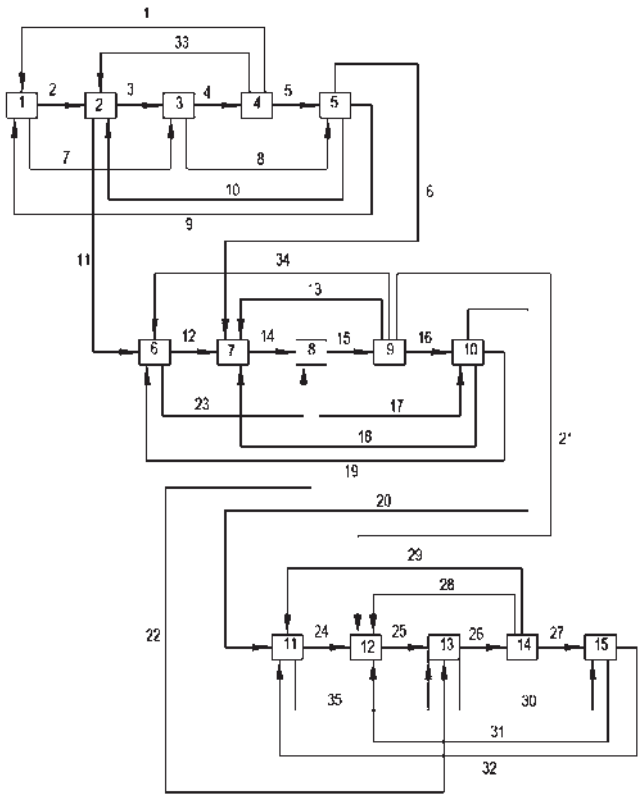


Figura 4.23 Caso 11. Digrafo de Barkley y Motard

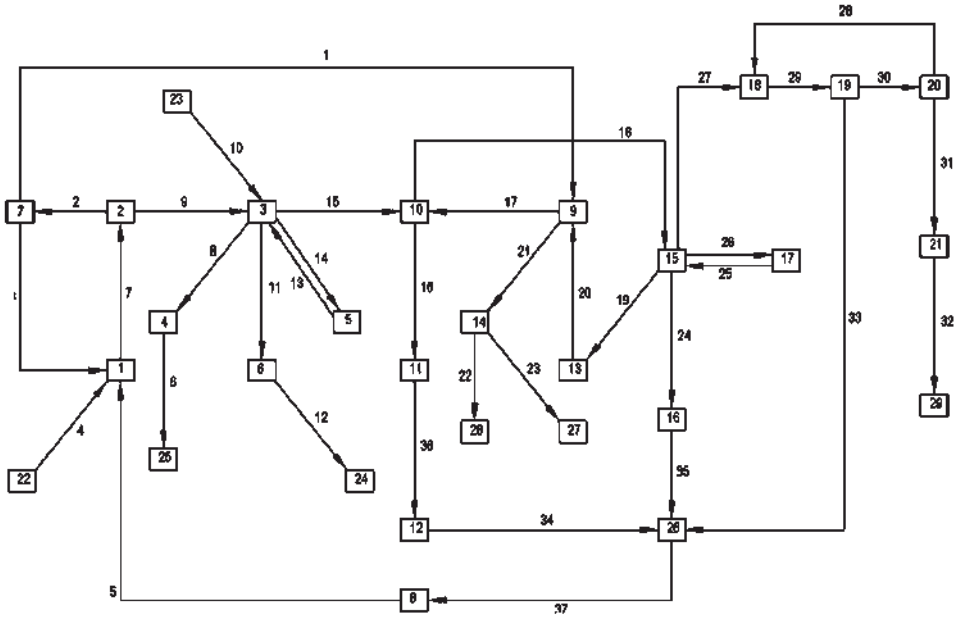


Figura 4.24 Caso 12. Digrafo de una planta de alquilación de HF

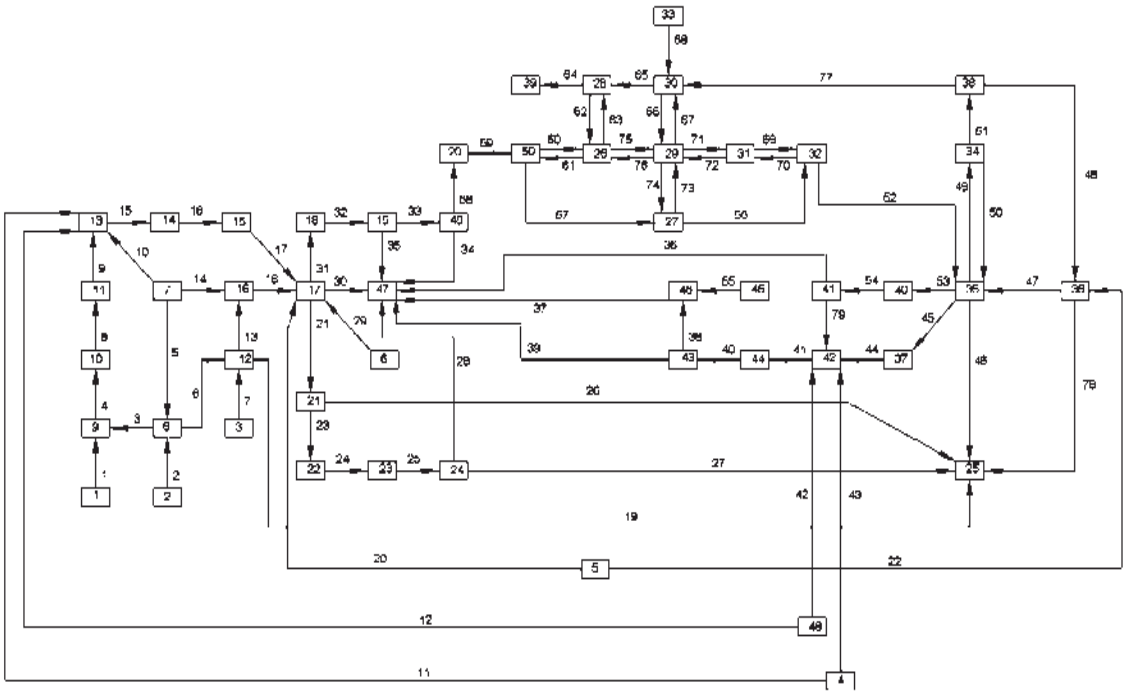


Figura 4.25 Caso 13. Digrafo de proceso de hidrogenación para extracción de aceite vegetal

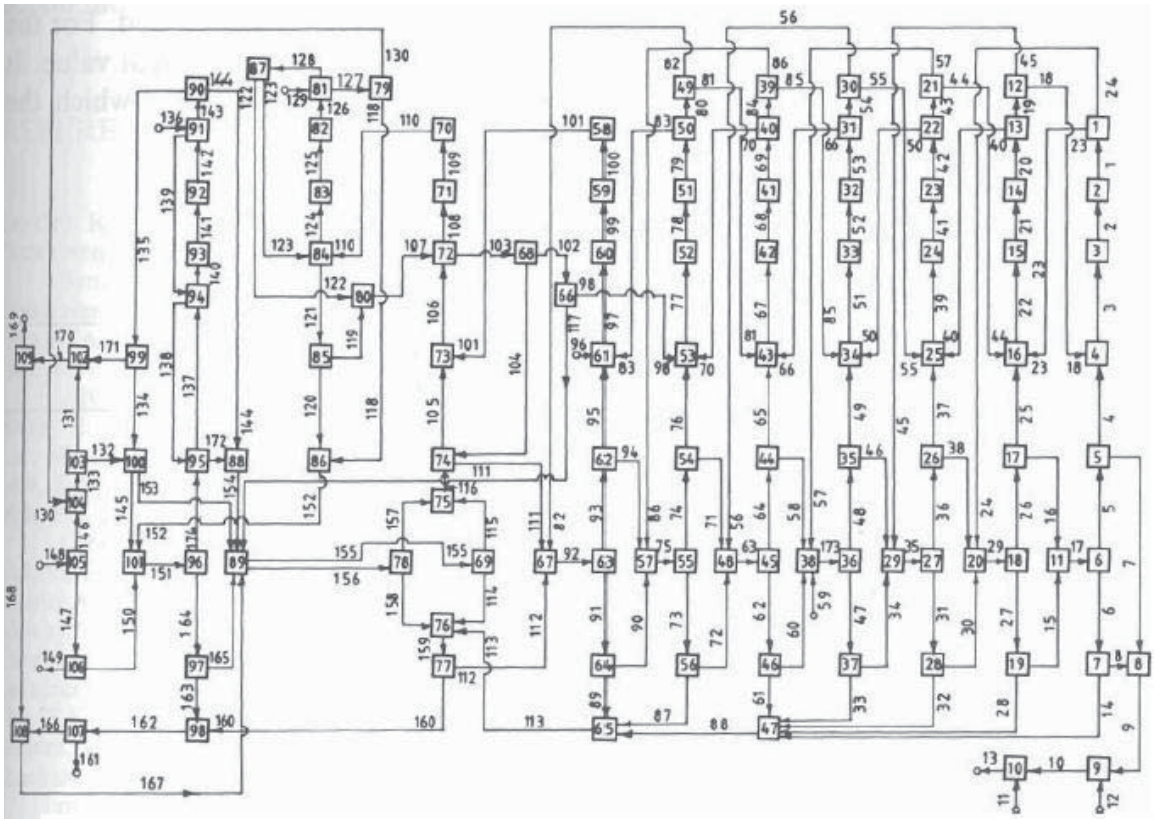


Figura 4.25 Caso 14. Digrafo de una planta de agua pesada

En la tabla 4.2 se presentan los resultados de los conjuntos mínimos de corrientes de corte generados por los algoritmos en JAVA y GAMS. Es importante hacer notar que en todos los casos de estudios ambos algoritmos arrojan el mismo número de corrientes de corte.

Tabla 4.2 Resultados de los conjuntos mínimos de corrientes de corte obtenidos en GAMS y JAVA

Caso No.	Número de		Corrientes de Corte	
	Ciclos	Corrientes de corte en GAMS (Programación Entera) y JAVA (Método Heurístico)	Conjunto Mínimo en GAMS (Programación Entera)	Conjunto Mínimo en JAVA (Método heurístico)
1	9	2	2 y 5	2 y 5
2	7	2	1 y 3	1 y 3
3	4	2	1 y 6	1 y 6
4	3	2	1 y 3	1 y 3
5	20	6	48, 19, 25, 26 y 31	48, 19, 25, 26 y 31
6	10	3	1, 10 y 30	1, 11 y 29
7	31	3	11, 18 y 38	11, 18 y 38
8	100	5	1, 15, 2, 40 y 51	1, 27, 37, 51 y 53
9	20	4	8, 28, 45 y 77	8, 27, 45 y 76
10	22	2	6 y 20	6 y 11
11	27	6	4, 8, 15, 17, 26 y 30	3, 7, 14, 23, 25 y 35
12	11	5	7, 13, 19, 25 y 29	7, 13, 17, 25 y 28
13	22	8	49, 61, 62, 66, 69, 71, 74 y 75	49, 61, 62, 66, 70, 71, 73 y 75
14	13746	12	20, 29, 51, 75, 77, 106, 107, 123, 137, 141, 159 y 173	1, 19, 39, 48, 67, 92, 106, 107, 124, 137, 139 y 159

La tabla 4.3 muestra el número de corrientes de corte obtenido para cada caso, que es comparado con la cantidad obtenida al aplicar un método heurístico basado únicamente en el parámetro “frecuencia de arco” y, por otro lado, con el resultado obtenido por

Lakshminarayanan y col. (1992). Cabe mencionar que, en el artículo mencionado, también se presenta una tabla con el número de corrientes de corte que ellos obtienen para los catorce casos de estudio comparado con los resultados óptimos esperados, los cuales no se cumplen para el caso trece y catorce pues obtienen una corriente más que el resultado óptimo. En este trabajo sí se obtienen las corrientes de corte óptimas para todos los casos de estudio.

Tabla 4.3 Tabla comparativa de la cantidad de corrientes de corte seleccionadas por cuatro métodos diferentes

Caso No.	No. Ciclos	# de corrientes de corte			
		Método heurístico (Frecuencia de arco)	Método heurístico (Rango y frecuencia de arco)	Artículo Lakshminarayanan y col. (1992)	Formulación Lineal Entera
1	9	4	2	2	2
2	7	5	2	2	2
3	4	3	2	2	2
4	3	2	2	2	2
5	20	10	6	6	6
6	10	6	3	3	3
7	31	8	3	3	3
8	100	8	5	5	5
9	20	11	4	4	4
10	22	6	2	2	2
11	27	12	6	6	6
12	11	6	5	5	5
13	22	9	8	9	8
14	13746	29	12	13	12

La tabla 4.4 muestra las corrientes de corte que fueron seleccionadas por cada método.

Tabla 4.4 Tabla comparativa de los conjuntos de corte obtenidos, para cada caso, por cuatro métodos distintos

Caso No.	Corrientes de corte seleccionadas			
	Método heurístico (Frecuencia de arco)	**Método heurístico(Rango y frecuencia de arco)**	**Artículo Lakshminarayanan y col. (1992)**	**Formulación Lineal Entera**
1	3, 6, 9 y 10	2 y 5	8 y 9	2 y 5
2	4, 5, 6, 10 y 11	1 y 3	1 y 3	1 y 3
3	1, 5 y 7	1 y 6	1 y 6	1 y 6
4	1 y 6	1 y 3	1 y 6	1 y 3
5	4, 7, 10, 11, 13, 22, 23, 24, 26 y 31	4, 8, 19, 25, 26 y 31	6, 10, 19, 25, 28 y 31	4, 8, 19, 25, 26 y 31
6	11, 15, 21, 27, 29 y 33	1, 11 y 29	11, 15 y 28	1, 10 y 30
7	4, 5, 20, 21, 22, 23, 24 y 38	11, 18 y 38	11, 31 y 40	11, 18 y 38
8	1, 19, 29, 35, 40, 43, 46 y 49	1, 27, 37, 51 y 53	4, 18, 31, 46 y 58	1, 15, 27, 40 y 51
9	5, 12, 13, 18, 23, 27, 29, 46, 72, 74 y 75	8, 27, 45 y 76	8, 28, 44 y 76	8, 28, 45 y 77
10	2, 5, 9, 12, 15 y 16	6 y 11	6 y 14	6 y 20
11	2, 7, 10, 13, 18, 19, 24, 28, 31, 33, 34 y 35	3, 7, 14, 23, 25 y 35	4, 8, 15, 17, 26 y 30	4, 8, 14, 23, 26 y 30
12	3, 5, 13, 19, 25 y 28	7, 13, 17, 25 y 28	7, 13, 19, 25 y 29	7, 13, 19, 25 y 29
13	48, 50, 60, 62, 66, 69, 72, 73 y 75	49, 61, 62, 66, 70, 71, 73 y 75	49, 52, 61, 63, 67, 70, 72, 74 y 76	49, 61, 62, 66, 70, 72, 74 y 75
14	6, 18, 24, 28, 32, 33, 44, 45, 54, 57, 61, 81, 82, 84, 87, 89, 102, 104, 118, 119, 120, 122, 123, 131, 138, 139, 145, 153 y 160	1, 19, 39, 48, 67, 92, 106, 107, 124, 137, 139 y 159	22, 29, 51, 75, 77, 92, 105, 108, 124, 137, 140, 159 y 173	20, 29, 51, 75, 77, 106, 107, 123, 137, 141, 159 y 173

Todos los conjuntos de corte que se muestran en la tabla anterior son corrientes que cortan todos los ciclos de cada caso, pero los conjuntos mínimos de corte para todos los casos de estudio se encuentran solamente en las columnas 3 y 5.

Por último se presenta la tabla 4. 5, que muestra los resultados obtenidos por el programa en JAVA para determinar el orden de cálculo. Las corrientes de corte que se muestran corresponden a las obtenidas por el programa implementado en GAMS.

Tabla 4.5 Resultados del orden de cálculo obtenidos en JAVA para cada caso de estudio, partiendo del conjunto mínimo de corrientes de corte obtenido en el programa en GAMS

CASO No.	Corrientes de corte	ORDEN DE SOLUCION
1	2 y 5	3, 4, 5, 2 , 1
2	1 y 3	4, 6, 5, 2, 3, 1
3	1 y 6	5, 1, 2, 3, 4
4	1 y 3	4, 2, 3, 1
5	4, 8, 19, 25, 26 y 31	11, 8, 6, 14, 12, 17, 15, 16, 10, 9, 3, 7, 4, 19, 2, 1, 18, 13, 5
6	1, 10 y 30	2, 3, 4, 5, 6, 7, 8, 9, 25, 10, 17, 14, 16, 13, 12, 15, 11, 21, 18, 22, 19, 23, 20, 24, 1
7	11, 18 y 38	9, 10, 11, 12, 13, 1, 2, 3, 4, 14, 15, 16, 17, 18, 23, 27, 24, 20, 29, 25, 30, 26, 19, 5, 22, 7, 6, 28, 21, 8
8	1, 15, 27, 40 y 51	24, 26, 27, 2, 35, 3, 38, 4, 39, 5, 40, 8, 6, 41, 17, 7, 33, 18, 9, 19, 10, 34, 11, 36, 12, 32, 20, 13, 29, 28, 14, 30, 15, 25, 21, 16, 37, 22, 31, 23, 1
9	8, 28, 45 y 77	5, 6, 1, 2, 3, 24, 7, 23, 8, 47, 22, 17, 46, 38, 28, 45, 39, 29, 50, 40, 30, 41, 34, 31, 25, 42, 32, 48, 33, 49, 35, 27, 19, 36, 26, 20, 16, 37, 18, 4, 21, 13, 12, 11, 43, 14, 10, 51, 44, 15, 9
10	6 y 20	5, 10, 1, 6, 7, 2, 3, 8, 9, 11, 12, 4
11	4, 8, 14, 23, 26 y 30	8, 4, 14, 9, 5, 15, 10, 1, 11, 2, 12, 6, 13, 7, 3
12	7, 13, 19, 25 y 29	2, 23, 13, 7, 9, 3, 10, 15, 11, 19, 16, 12, 26, 22, 20, 8, 18, 17, 5, 1, 21, 14, 6, 4, 29, 28, 27, 25, 24
13	49, 61, 62, 66, 70, 72, 74 y 75	7, 2, 8, 1, 9, 10, 48, 11, 4, 13, 14, 16, 15, 6, 5, 17, 18, 19, 49, 20, 50, 27, 34, 29, 38, 33, 31, 36, 32, 30, 26, 35, 28, 40, 41, 37, 42, 21, 44, 22, 45, 43, 23, 3, 46, 24, 12, 47, 39, 25
14	20, 29, 51, 75, 77, 106, 107, 123, 137, 141, 159 y 173	72, 71, 70, 84, 83, 82, 92, 81, 105, 91, 79, 104, 90, 103, 99, 85, 106, 100, 86, 101, 96, 97, 77, 102, 98, 94, 109, 107, 95, 68, 108, 88, 66, 36, 33, 89, 52, 37, 35, 32, 18, 12, 78, 69, 55, 51, 31, 29, 19, 17, 75, 56, 54, 50, 30, 27, 11, 74, 49, 48, 26, 13, 6, 67, 45, 25, 5, 63, 44, 24, 4, 62, 43, 23, 3, 61, 42, 22, 2, 60, 46, 41, 28, 21, 7, 1, 64, 59, 47, 40, 16, 87, 65, 58, 39, 15, 93, 80, 76, 73, 57, 53, 38, 34, 20, 14, 8, 9, 10

CAPITULO 5

CONCLUSIONES

5.1 Conclusiones Generales

Se llevó a cabo la implementación en computadora de algoritmos de identificación de ciclos simples, rasgado y ordenamiento de procesos químicos con recirculaciones con ayuda del lenguaje JAVA y el paquete GAMS. A fin de comprobar su validez, los programas desarrollados se utilizaron para obtener el conjunto mínimo de corrientes de corte de 14 casos de estudio, que fueron reportados con anterioridad en la literatura, así como el orden de cálculo en el que deberán de ser resueltos los nodos que conforman cada uno de los digrafos. La comparación de los resultados obtenidos en este trabajo con los reportados en la literatura permite concluir que son correctos los procedimientos implementados en computadora en esta tesis. Además, para dos de los casos de estudio en los que trabajos previos han fallado en la determinación de un conjunto mínimo de corriente de corte, el método heurístico da resultados correctos. Por consiguiente, este método se recomienda ampliamente ya que además de ser simple de entender y sencillo de aplicar proporciona resultados correctos, es decir, conjuntos verdaderamente mínimos de corrientes de corte para todos los casos de estudio.

5.2 Conclusiones Particulares

El algoritmo desarrollado en el presente trabajo para la identificación de ciclos simples se basa en la técnica de búsqueda en profundidad y fue implementado en computadora utilizando el lenguaje JAVA. Como el número de ciclos no es conocido a priori, no es posible predecir de antemano el tiempo de cómputo. Como es obvio, el tiempo que tarda en imprimir resultados depende de la dimensión del problema a resolver; por ejemplo, para la mayoría de los casos aquí estudiados, el programa arrojó resultados en pocos segundos; sin embargo, para el caso 14, que es el que tiene el mayor número de ciclos (13746), el programa requirió dos días de cómputo. Sin embargo, el algoritmo desarrollado es robusto y correcto, ya que en todos los casos no tuvo fallas durante su ejecución y encontró los mismos ciclos simples reportados previamente en estudios previos.

De los cuatro programas que se desarrollaron para la identificación de ciclos simples, el rasgado y ordenamiento de procesos químicos, el primero es el más tardado debido al tiempo que implica la captura de los datos de entrada. Los resultados que arroja el programa de identificación de ciclos simples se utilizan como datos de entrada de los programas de identificación de corrientes de corte; esta situación disminuyó drásticamente el tiempo requerido para la entrada de datos de éstos últimos.

Los programas implementados en JAVA y GAMS para la selección del conjunto mínimo de corrientes de corte utilizan diferentes métodos para obtener los resultados deseados. El de JAVA utiliza un método heurístico que se basa en los conceptos de rango de ciclo y frecuencia de arco, mientras que el programa implementado en GAMS utiliza un método basado en la programación entera. Ambos programas proporcionan resultados óptimos, pues imprimen un conjunto mínimo de corrientes de corte. En la mayoría de los ejemplos aquí expuestos los resultados fueron los mismos; no obstante, en pocos casos los resultados diferían un poco en la selección de las corrientes de corte más no en la cantidad de ellas, lo cual se puede justificar por el uso de diferentes aproximaciones que usa cada programa para encontrar una solución. Esto se puede considerar una ventaja, ya que da mayor flexibilidad para seleccionar las corrientes de corte más apropiadas del proceso químico bajo estudio. Entre estos dos programas la única diferencia es que es un poco más tardado el ingreso de datos en GAMS, debido a que se debe de ingresar la matriz de ciclos en el programa, mientras que en el caso del programa en JAVA la matriz de ciclos simples se guarda en un archivo de texto (Bloc de notas) en donde no es necesario indicar los ciclos (filas) y arcos (columnas), por lo que sólo se requiere copiar, pegar y guardar la matriz para después indicar su ruta de almacenamiento y que el programa trabaje.

En cuanto al programa para obtener el orden de cálculo, este no requiere de muchos datos de entrada y los resultados para todos los casos, ya sean digrafos chicos, medianos o grandes, se obtienen en cuestión de segundos.

Hasta la fecha, todos los algoritmos que han sido desarrollados con este fin enfrentaron problemas de dimensionalidad por lo cual no se habían obtenido con un solo procedimiento conjuntos mínimos de corrientes de corte óptimos para todos los casos de estudio aquí presentados, en especial para los casos 13 y 14. Los resultados obtenidos en la experimentación computacional fueron muy satisfactorios con la implementación de los

algoritmos en computadora con el lenguaje JAVA y el paquete GAMS ya que, para los catorce casos de estudio, se obtuvieron los conjuntos mínimos de corrientes de corte óptimos esperados en el artículo de Lakshminarayanan y col., 1992. En dicho artículo se establece para el caso 13 un conjunto de corte óptimo de 8 corrientes mientras que para el caso 14 uno de 12 corrientes.

Lo único que falta por comprobar y analizar es qué tan práctico es suponer las corrientes de corte de estos conjuntos mínimos sugeridos por dichos programas en cada caso de estudio, pues si bien es una opción correcta para la solución de un problema podría revisarse si es realmente la mejor opción desde el punto de vista de tiempo de cómputo. Sin embargo, este último análisis implica la simulación de todo el diagrama de flujo, que es la tarea posterior al pre-procesamiento de procesos químicos que se estudia en este trabajo de tesis.

APENDICE A

CONCEPTOS BASICOS DE GRAFOS

La teoría de grafos puede ser usada para representar sistemas compuestos de objetos discretos y las relaciones entre estos objetos (Mah, 1990). Los objetos son comúnmente conocidos como *vértices* o *nodos* y las relaciones como *arcos* o *aristas*.

Un grafo G consiste de un conjunto discreto de *n nodos V* relacionados por un conjunto discreto de *m arcos A*, de modo tal que un arco dado conecta sólo un par de nodos dando lugar a asociaciones binarias entre nodos y arcos. Cuando las relaciones entre dos nodos tienen una dirección (por ejemplo, flujos de materia y flujos de calor que conectan dos módulos de simulación en un proceso), las representaciones gráficas de sistemas se denominan grafos dirigidos o digrafos.

A.1 Representación de un grafo dirigido

La Figura A.1 muestra la forma general de un digrafo con 8 nodos y 11 arcos. Tal como se indica, un digrafo se representa empleando círculos[2] para especificar los nodos y flechas (líneas dirigidas) para trazar los arcos; las cabezas de las flechas indican la dirección de los arcos. Esta característica permite representar comúnmente a los arcos por medio de pares ordenados de nodos $a_k = (v_i, v_j)$, donde se dice que v_i es el **nodo de origen** (la cabeza) y v_j el **nodo de destino** (la cola) del arco a_k. Este arco también se puede representar por la notación $v_i \rightarrow v_j$, que también se empleará en este texto. Por lo tanto, los arcos muestran que entre un par de vértices existe una relación unívoca; por ejemplo, el arco a_k parte de v_i y llega a v_j, pero no lo contrario.

$G = N, A \ \ con:$

$N = 1,2,3,4,5,6,7,8$

$A = \ 1,8 \ , \ 2,1 \ , \ 3,2 \ , \ 3,4 \ , \ 4,5 \ , \ 5,3$

$\quad 5,6 \ , \ 5,7 \ , \ 5,8 \ , \ 6,7 \ , \ 8,2$

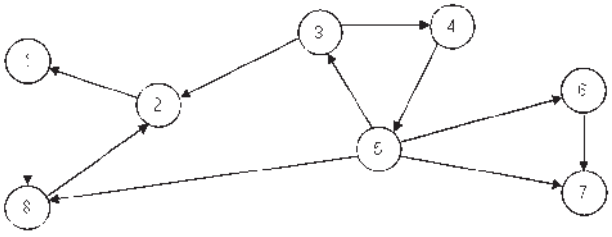


Figura A.1 Representación de un digrafo

[2] En vez de círculos se pueden emplear rectángulos para representar los nodos.

Un grafo dirigido puede representarse simbólicamente como $G = (V, A)$, donde $V = \{v_1, v_2, \cdots, v_n\}$, $A = \{a_1, a_2, \cdots, a_m\}$ y $a_k = (v_i, v_j)$ tal que $v_i, v_j \in V$. En dicho digrafo se entiende que $(v_i, v_j) \neq (v_j, v_i)$ y, en muchos casos, sólo existe uno de los pares ordenados de nodos. Como se observa, los arcos pueden ser representados por pares ordenados de nodos consecutivos.

A.2 Nodo fuente y sumidero

Un nodo que sólo tiene arcos saliendo de él se denomina fuente y un nodo que sólo tiene arcos dirigidos hacia él se denomina sumidero.

A.3 Relaciones de adyacencia

Sea $G = (V, A)$ un grafo dirigido. Un nodo v_j se dice que es **adyacente** al nodo v_i si existe el arco (v_i, v_j) en dicho grafo. Este concepto es de particular importancia, dado que los grafos suelen representarse en la computadora por medio de matrices de adyacencias de nodos o de arcos.

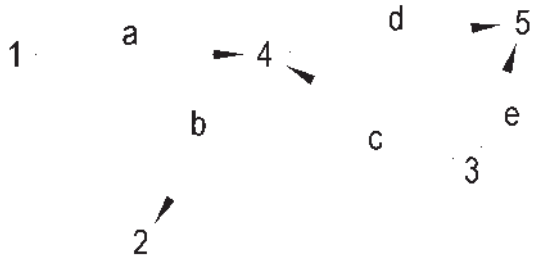
A.3.1 Matriz de adyacencia de nodos.

La matriz de adyacencia de nodos de *G* es la matriz $A = (a_{ij})$ de orden $n \times n$ definida por:

$$a_{ij} = \begin{cases} 1, & \text{si } v_j \text{ es adyacente en } v_i \\ 0, & \text{en caso contrario} \end{cases} \qquad \text{(A.1)}$$

En esta matriz los nodos están representados tanto por las filas como por las columnas, por lo que es una matriz cuadrada con dimensión igual al número de nodos *n* del digrafo y con *m* elementos no-nulos. La Figura A.2 es un ejemplo de la correspondencia entre un grafo dirigido de 5 nodos y su matriz de adyacencia 5 por 5. Una forma simple de ver la información guardada en la matriz de adyacencia de nodos es que los renglones de la misma representan el origen y las columnas el destino de cada arco en el grafo. De acuerdo a la ecuación (A.1), como se muestra en la Figura A.2, se pone un cero en la celda del renglón *i*, columna *j* de la matriz cuando no hay conexión entre los nodos *i* y *j*; por otro lado, se introduce un uno cuando existe un arco en el grafo en la dirección *i* a *j*. Por esta

característica, la matriz de adyacencia de nodos también se denomina matriz de conexión del digrafo. Por simplicidad, se puede dejar vacía la celda a_{ij} cuando no hay conexión entre los nodos *i* y *j*.



Nodos – Origen \ Nodos - Destino	1	2	3	4	5
1	0	0	0	1	0
2	0	0	0	0	0
3	0	0	0	1	1
4	0	1	0	0	1
5	0	0	0	0	0

Figura A.2 Grafo dirigido y su Matriz de Adyacencia de Nodos

En el presente trabajo, se presenta una variante de esta matriz ya que en lugar de introducir un uno en la coordenada de conexión entre nodos se introduce el número del arco que los conecta.

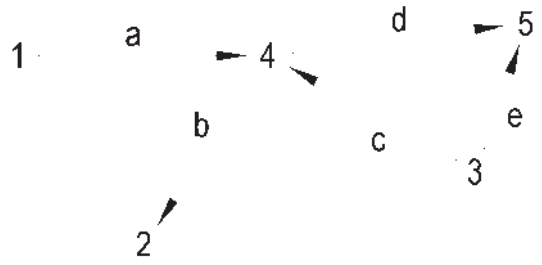
Dado que la matriz de adyacencia representa las dependencias existentes entre los nodos, la lectura descendente de una columna *j* revela todos los nodos cuyas salidas son requeridas para realizar la tarea correspondiente a dicha columna. Por ejemplo, la lectura descendente de la columna 4 indica que dicho nodo requiere información de los nodos 1 y 3. Una columna vacía representa un nodo sin entradas y, por tanto, a un nodo independiente que no requiere información de otros nodos. Por otro lado, la lectura a lo largo de una fila *i* revela qué nodos reciben información del nodo correspondiente a la fila; en consecuencia, una fila vacía representa un nodo sin salidas, es decir, que no proporciona información a cualquier otro nodo del digrafo.

A.3.2 Matriz de adyacencia de arcos (Mah, 1990).

La matriz de adyacencia de arcos de *G* es la matriz $B = (b_{ij})$ de orden *m x m* definida por:

$$b_{ij} = \begin{cases} 1, & \text{si } a_i \text{ entra a un nodo y } a_j \text{ sale del mismo nodo} \\ 0, & \text{en caso contrario} \end{cases} \qquad \text{(A.2)}$$

Las filas y las columnas de la matriz representan los arcos del grafo. Al elemento b_{ij} de la matriz se le asigna un valor de uno si el arco *i* es incidente (entra) a un nodo y el arco *j* sale de ese mismo nodo. Por ejemplo, la Figura A.3 muestra un digrafo y su matriz de adyacencia de arcos. La principal ventaja de esta matriz es que permite la representación de autociclos y múltiples arcos entre dos nodos (Mah, 1990).



Arcos – Origen \ Arcos - Destino	A	b	c	D	e
a	0	1	0	1	0
b	0	0	0	0	0
c	0	1	0	1	0
d	0	0	0	0	0
e	0	0	0	0	0

Figura A.3 Grafo Dirigido y su correspondiente Matriz de Adyacencia de Arcos

A.4 Caminos

Un camino de longitud *k* desde *u* a *u'* en un grafo $G = \langle V, A \rangle$ es una secuencia finita de nodos consecutivos $\langle v_0, v_1, ..., v_k \rangle$ tal que:

- $v_0 = u$ y $v_k = u'$.
- $\forall i : 1 ...k : (v_{i-1}, v_i) \in A$ (v_i es adyacente a v_{i-1})
- La longitud *k* del camino es el número de arcos.

Si hay un camino *P* desde *u* hasta *u'*, se dice que *u'* es alcanzable desde *u* vía *P*.

A.5 Caminos simples y ciclos

Un **ciclo** es un camino $\langle v_0, v_1, ..., v_k \rangle$ que:

- Empieza y acaba en el mismo vértice ($v_0 = v_k$).
- Contiene al menos un arco.

Un camino es **simple** si todos sus nodos son distintos.

Un **bucle** es un ciclo de longitud 1.

Un grafo es **acíclico** si no contiene ciclos.

A.6 Subgrafo, subgrafo fuertemente conectado y componente fuerte

Un **subgrafo** *SG* de *G* es un grafo constituido por un subconjunto *SV* de *V* y un subconjunto *SA* de *A* que conectan los nodos de *SV*.

Se dice que un digrafo *G* es **fuertemente conectado** si existe un camino desde *u* a *v* y un camino desde *v* a *u* para cada par distinto de nodos *u* y *v*, con $u, v \in V$. Si un digrafo no está fuertemente conectado entonces puede ser partido en subgrafos fuertemente conectados.

Un subgrafo es un **componente fuerte** de *G* si es fuertemente conectado y no puede ser agrandado a otro subgrafo mediante la adición de nodos extras y sus correspondientes arcos. Cada nodo puede pertenecer solamente a un componente fuerte (que puede estar constituido por un nodo simple) y todos los nodos de cualquier ciclo de un grafo deben estar en el mismo componente fuerte. En consecuencia, los componentes fuertes definen una partición del grafo. Al menos debe haber un componente fuerte en un grafo de modo tal que no haya un camino de cualquiera de sus nodos a cualquier nodo de otro componente fuerte.

APENDICE B
RESUMEN DE LOS PROGRAMAS EN COMPUTADORA PARA EL RASGADO Y ORDENAMIENTO DE PROCESOS QUIMICOS

La simulación de procesos químicos, con la ayuda de computadoras, tuvo sus primeras manifestaciones en el ámbito universitario y poco a poco fue penetrando en la industria química. Esta simulación tuvo sus orígenes en el auge mismo de la investigación operacional, por ser una novedosa herramienta para realizar confiable y velozmente diversos cálculos numéricos.

La realización de una simulación comprende dos etapas:

1. Sistematización de la información
2. Resolución

La etapa 1 es el llamado preprocesamiento en donde se realizan las siguientes operaciones:

- ✓ Particionado: detección de los subsistemas o bloques independientes del DFI que contienen reciclos.
- ✓ Rasgado: selección de las corrientes iteradoras de cada bloque.
- ✓ Ordenamiento: determinación de la secuencia de resolución u orden de precedencia.

En la Figura B.1 se muestran claramente las tres operaciones que se deben de cumplir para resolver un DFI.

La implementación de algoritmos de particionado, rasgado y ordenamiento de procesos químicos en computadora se convierte en una herramienta que ayuda a una selección más rápida y eficiente de las corrientes de corte, así como a la determinación de un orden de precedencia que permite reducir el tiempo de simulación de un proceso químico para su análisis.

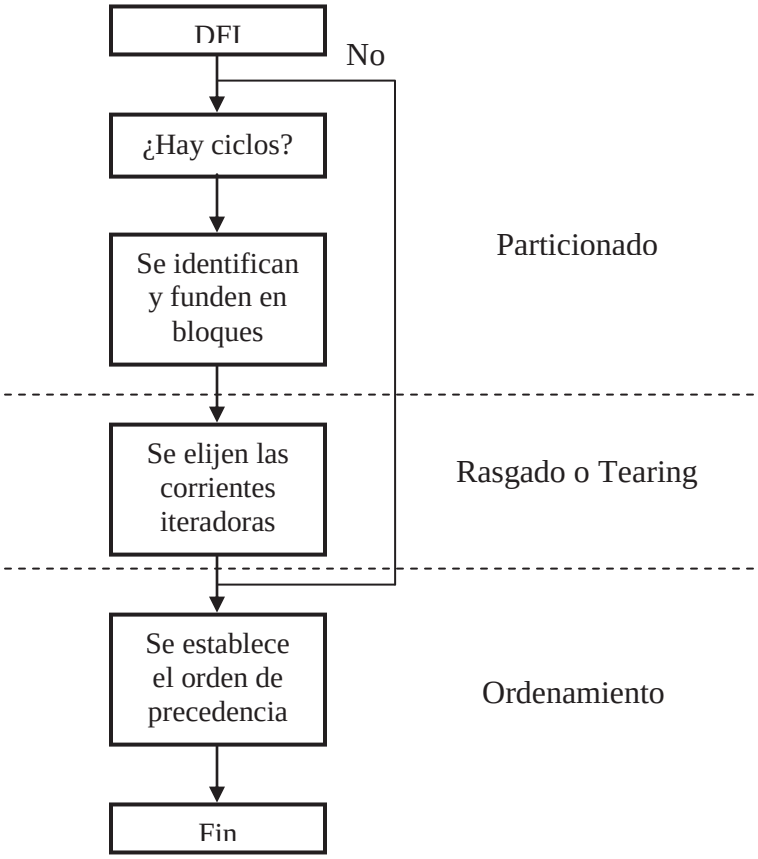


Figura B.1 Operaciones del preprocesamiento

Los programas de computadora para el rasgado y ordenamiento de procesos químicos reportados en el presente trabajo fueron desarrollados utilizando el lenguaje JAVA y el paquete GAMS. Para la identificación de ciclos simples, así como para el rasgado y ordenamiento se utilizó el lenguaje JAVA. El programa para la selección de corrientes de corte implementado en JAVA se basa en un método heurístico y su funcionamiento es comparado y respaldado por el funcionamiento del programa desarrollado con el paquete GAMS, el cual usa una formulación de programación entera para el mismo propósito.

B.1 Programa en lenguaje JAVA para la identificación de ciclos simples

B.1.1 Instalación de las herramientas necesarias para correr el programa

Java es un lenguaje de programación desarrollado por Sun Microsystems a principios de los años 90. Los objetivos de diseño que persiguieron y lograron los creadores de JAVA fueron desarrollar un lenguaje familiar, orientado a objetos, robusto, de alto rendimiento, portable, lo más simple posible y seguro. Todas estas características permiten que la creación de programas en este lenguaje se vuelva más sencilla.

Para correr los programas en JAVA se necesita instalar el paquete JDK de Sun, que es de libre distribución. La dirección del sitio electrónico es: www.sun.com. También es necesario instalar jGRASP, que es un ambiente desarrollado para proporcionar la generación automática de visualizaciones del software que ayudan a mejorar la comprensión del mismo y que hace más sencilla la compilación de los programas y el entendimiento de los resultados arrojados. El programa para la identificación de ciclos simples, que lleva por nombre "IdentificaciondeCiclosSimples", se podrá descargar de la página web:
http://www.moreliahosting.com/posgradofiq/index.php?option=com_content&view=article&id=19&Itemid=41

B.1.2 Especificación e ingreso de los datos de entrada

Teniendo todas las herramientas necesarias instaladas, para comenzar a utilizar el programa es necesario abrirlo con jGRASP, lo que se puede hacer de dos formas básicamente: ya sea dando doble clic sobre el programa IdentificaciondeCiclosSimples o abriendo jGRASP y buscándolo, a través de la opción "open file", en la carpeta o dirección en donde se haya guardado.

Los datos necesarios para poner en marcha el programa, considerando que ya se cuenta con el digrafo reducido sin nodos fuente o sumidero del caso a analizar, son:

a) *Nombre del caso bajo estudio*. Se recomienda asignarlo para poder guardar el programa con otro nombre y, en caso de ocuparlo en un futuro, tenerlo disponible e identificarlo rápidamente. Si el nombre que se le quiere asignar consta de dos o más palabras deberán escribirse juntas, tal como el nombre original del programa "IdentificaciondeCiclosSimples".

b) *Matriz de adyacencia de nodos del caso bajo estudio*. Es importante resaltar que en esta matriz la adyacencia entre un nodo y otro tiene que ser indicada con el número de la corriente que los une, por lo que los ceros indicarán la no conectividad entre nodos y los números diferentes de cero corresponden al número del arco de conexión. Hay que recordar también que las filas representan a los nodos emisores y las columnas a los nodos receptores.

c) *Número aproximado de ciclos esperados*. Este dato no es necesario conocerlo con mucha exactitud, por lo que recomienda suponer un número muy grande o usar el número ya asignado por default en el programa, el cual ya es suficientemente grande. Esta opción no generará ningún conflicto o error en el programa al momento de correrlo, pero si lo habrá cuando se suponga un número más pequeño de ciclos esperados que el real.

d) *Ruta de almacenamiento de resultados*. Es la ubicación que se le dará al programa con el nombre arriba asignado.

e) Cantidad de corrientes o arcos que componen el digrafo del caso bajo estudio.

La Tabla B.1 y la Figura B.3 muestran cada uno de los datos de entrada arriba descritos e indica cuál es la ubicación de los mismos dentro del programa.

Tabla B.1 Datos de entrada para la identificación de ciclos simples en JAVA y su ubicación de ingreso

Dato de entrada	Dónde ingresarlo
a) Nombre del caso bajo estudio	Después de la leyenda “class”, en lugar del nombre “IdentificaciondeCiclosSimples” y sin quitar la “{“que aparece al final. Ver figura B.3 (a)
b) Matriz de adyacencia de nodos	Luego de la leyenda “//red de entrada”, sustituyendo la matriz que ya existe, agregando las filas, columnas y llaves que hagan falta, cuidando de mantener el formato como el que se muestra en la Figura B.3 (b)
c) Número aproximado de ciclos esperados	Posterior a la leyenda “static int numerociclosaprox =”, sustituyendo el número “15000" en caso de que se considere que el caso bajo estudio tenga una mayor cantidad de ciclos. Ver Figura B.3 (c)
d) Ruta de almacenamiento de resultados	Enseguida de la leyenda “out = new PrintStream(new FileOutputStream”, sustituyendo la dirección escrita entre paréntesis por la deseada, conservando el sentido de las diagonales “/” y recordando escribir “.txt” al final. Ver figura B.3 (d)

e) Cantidad de corrientes o arcos que componen el digrafo bajo estudio	A continuación de la leyenda “int c =”, reemplazando el número grabado por el correspondiente al caso de estudio. Ver Figura B.3 (e)

NOTAS:

1. Los datos deberán ser ingresados tal como se muestran en la Figura B.3, teniendo cuidado de no quitar o modificar algún signo de puntuación (coma, punto, punto y coma, paréntesis, etc.); en caso de hacer modificaciones, el programa mostrará errores y no arrojará resultados.
2. El digrafo que aparece en la Figura B.2 es el caso de estudio que servirá de apoyo para ejemplificar el uso de los programas.

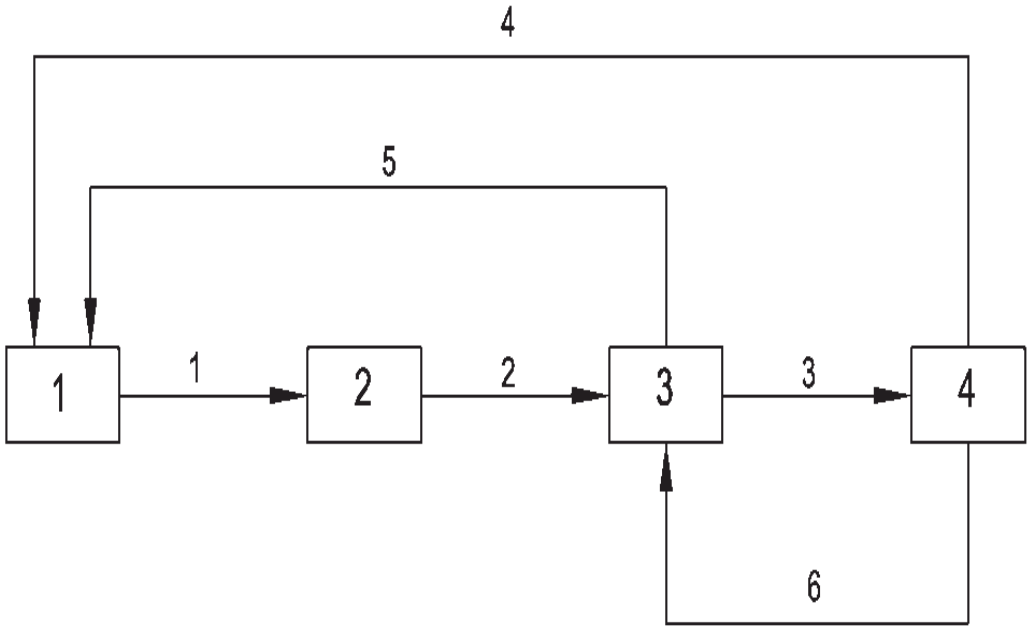


Figura B.2 Ejemplo. Digrafo de Upadhye y Grens

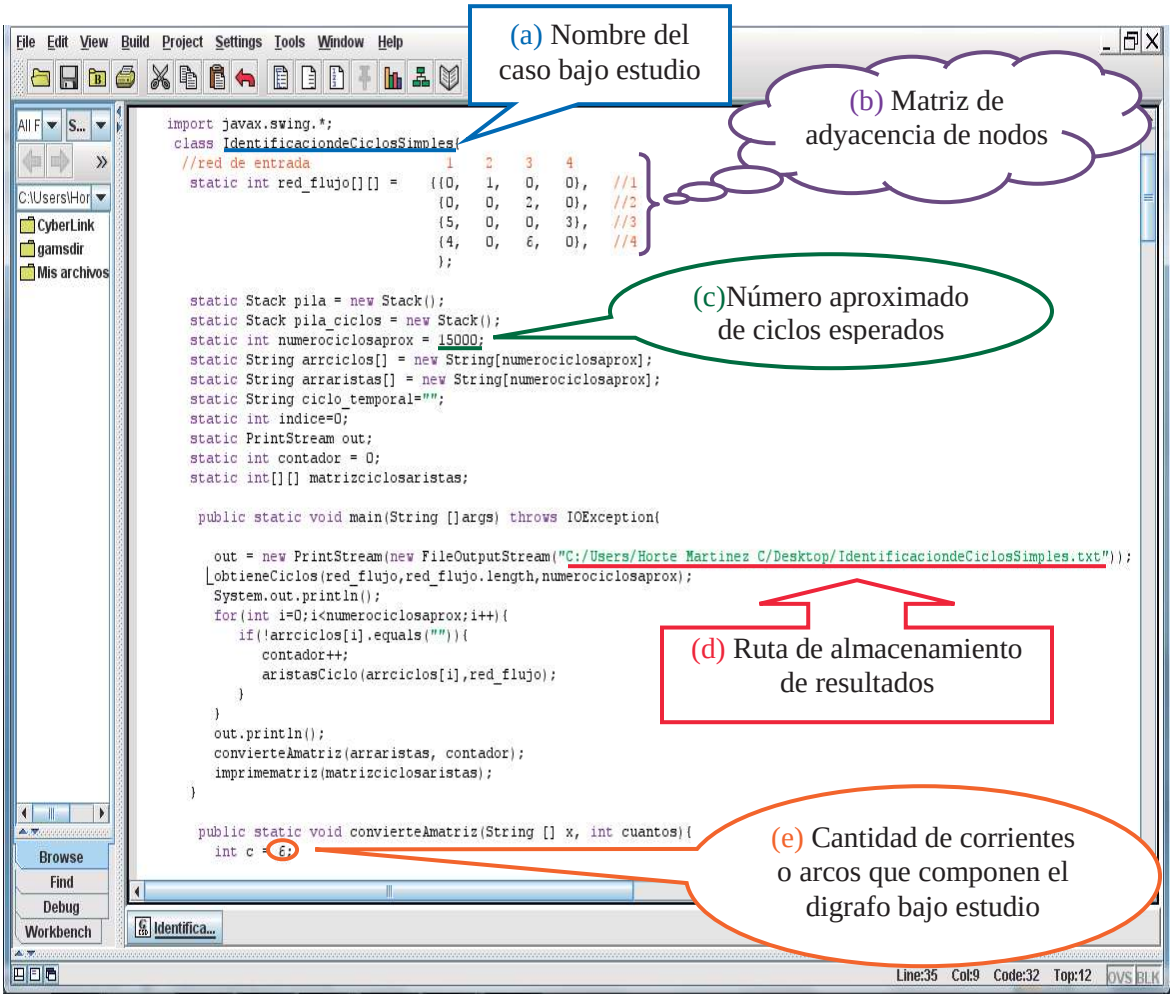


Figura B.3 Datos de entrada y ubicación de ingreso

B.1.3 Cómo correr el programa

Antes de correr el programa se deben de llevar a cabo dos pasos.

1. ***Guardar***. Existen dos modalidades para realizar esta acción. La primera consiste en dar un clic sobre el icono , lo que hará que el programa se guarde con el mismo nombre "IdentificaciondeCiclosSimples" pero con diferentes datos. Para ello no es necesario hacer lo que se indicó en el inciso a) del subtema anterior, no habría necesidad de modificar el nombre. La segunda modalidad consiste en guardar el programa con otro nombre para un futuro requerimiento del mismo, lo cual implica llevar a cabo el inciso a), antes descrito, y dar clic en la opción File, después en "Save as", en donde se va a seleccionar la ubicación deseada para este programa y a escribir, en el recuadro "nombre de archivo", el mismo

nombre escrito en a); posteriormente se dará clic en "Save" y el programa se habrá guardado. De no escribir el mismo nombre en ambos lugares, jGRASP arrojará una ventana que da tres opciones: "Save Anyway", "Save with Expected Name" y "Cancel", como se muestra en la Figura B.4. Si se da clic sobre la opción "Save Anyway", el programa se guardará con el nombre escrito en el recuadro "Nombre de archivo" pero más tarde, al intentar correr el programa, en lugar de que éste arroje los resultados en la ventana inferior del programa arrojará un error como el que aparece en la Figura B.5 y no generará ningún archivo de resultados. Si se selecciona la opción "Save with Expected name", el archivo se guardará con el nombre escrito en a) y correrá sin problema alguno; lo único que se deberá de rectificar es que el nombre con el que se guardará sea el nombre deseado. La opción "Cancel" nos permite ver nuevamente la ventana "Save As" para corregir el error cometido.

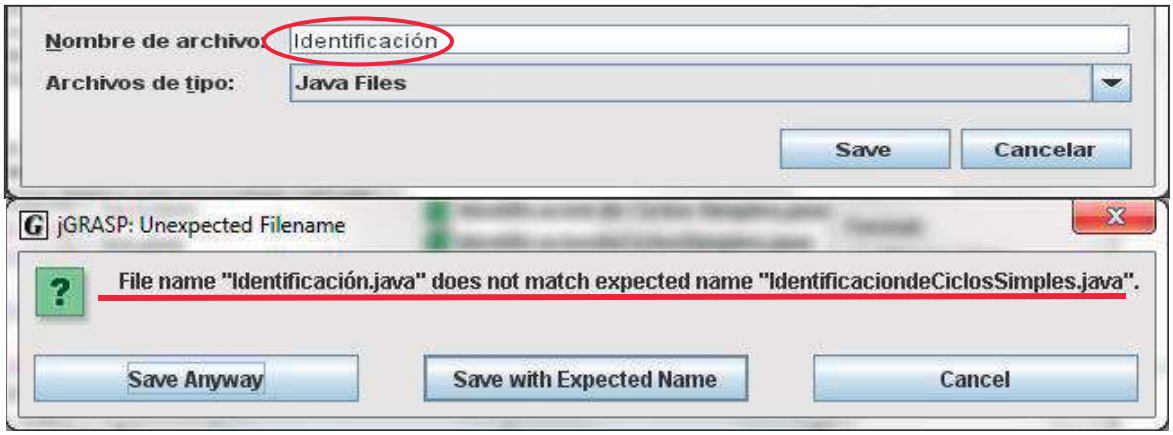


Figura B.4 Ejemplo de incompatibilidad entre el nombre asignado y el nombre esperado

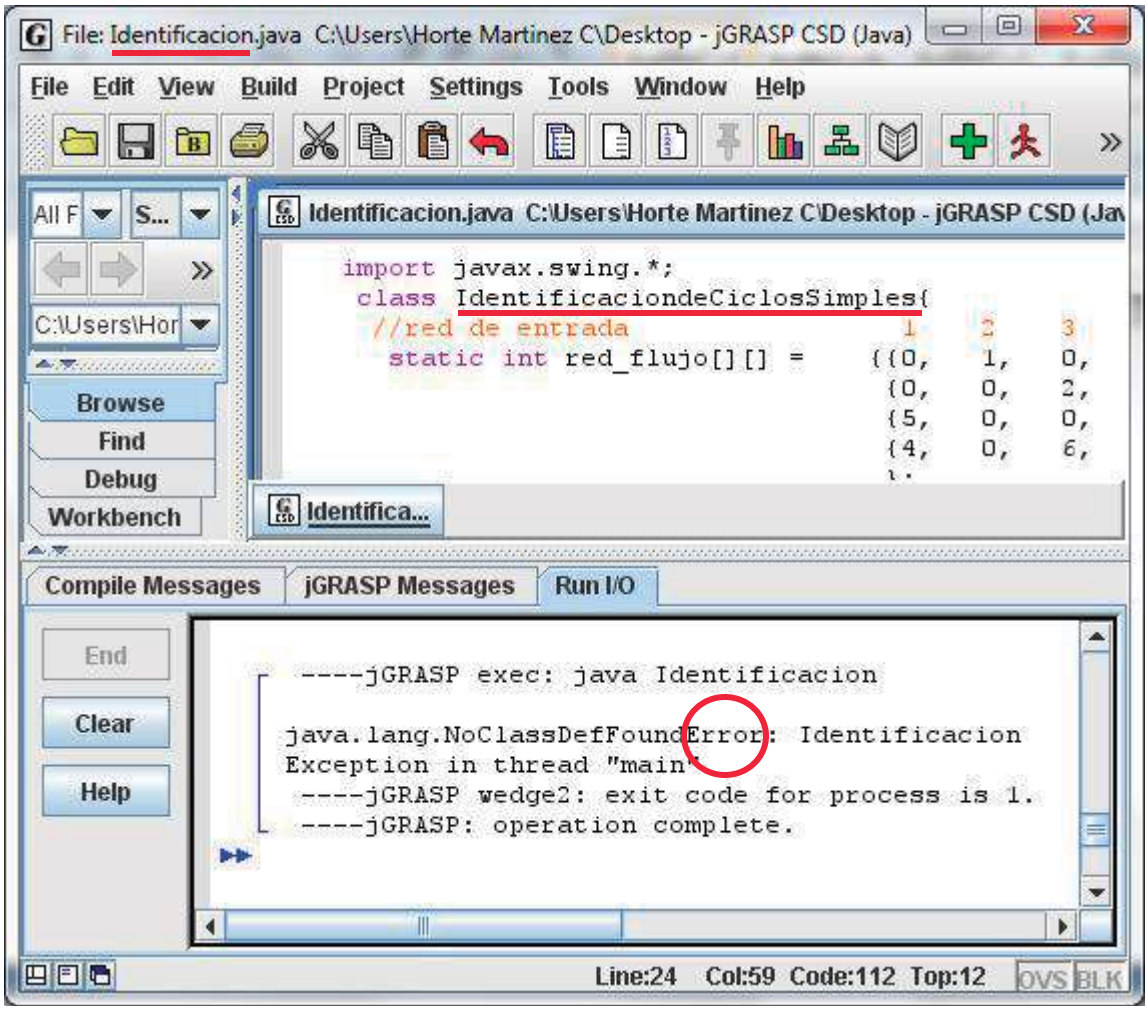


Figura B.5 Error por incompatibilidad de nombres

2. ***Compilar.*** Este paso permite traducir el lenguaje en el que fue creado el programa a un lenguaje de máquina, lo que se puede interpretar también como un lenguaje más familiarizado con el ser humano que permite un entendimiento más claro de los resultados arrojados. Este paso es muy sencillo y consiste solamente en dar un clic sobre el ícono . Cuando el programa termina de compilar arroja el texto "----jGRASP: operation complete", en la ventana inferior del programa, lo que indica que el programa fue compilado con éxito y que no hubo errores en esta etapa.

Realizados los pasos anteriores, ahora sí es posible correr el programa sin problema alguno. Esto es posible haciendo clic sobre el ícono . Cuando el programa termina de correr arroja la leyenda "----jGRASP: operation complete."

La Figura B.6 muestra a detalle los pasos que se fueron realizando para lograr los resultados deseados.

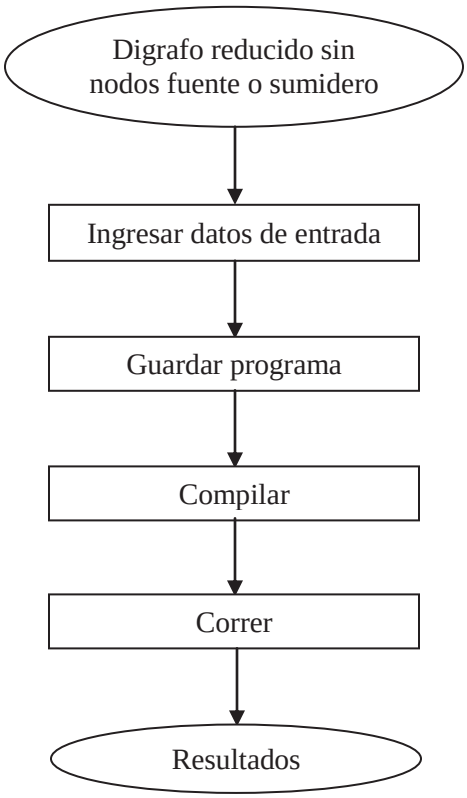


Figura B.6 Diagrama de Flujo de los pasos realizados para llegar a los resultados

B.1.4 Interpretación de los resultados

Después de la ejecución del programa, se imprimen los resultados en un archivo de texto en la dirección que fue asignada en el inciso d), descrito en la sección B.1.2, con el formato que se muestra en la Figura B.7.

Al inicio de la hoja aparecen enlistados todos los ciclos simples; cada ciclo contiene los arcos que lo conforman. Más adelante aparece una matriz de unos y ceros, en la cual las columnas son los arcos del digrafo y las filas son los ciclos simples del mismo. Esta matriz es la matriz de ciclos simples, donde los unos indican la aparición de un arco en un ciclo mientras que los ceros indican lo contrario.

La matriz de ciclos es la parte más importante de estos resultados, ya que minimiza el trabajo en el ingreso de datos del programa desarrollados para la identificación del conjunto mínimo de corte.

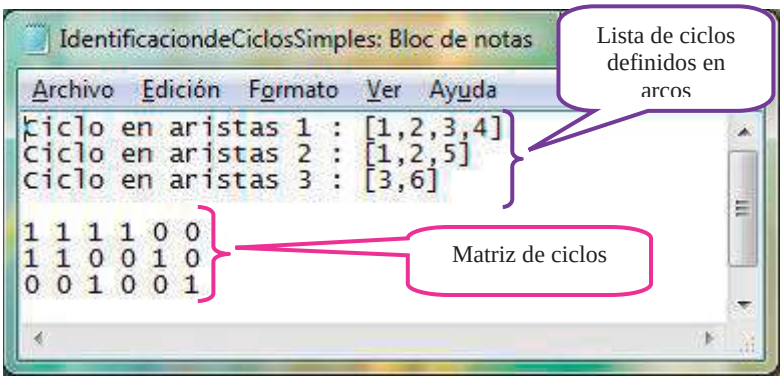


Figura B.7 Resultados en formato .txt generados en Bloc de notas

Los resultados también pueden ser visibles en la ventana inferior del programa como se observa en la Figura B.8.

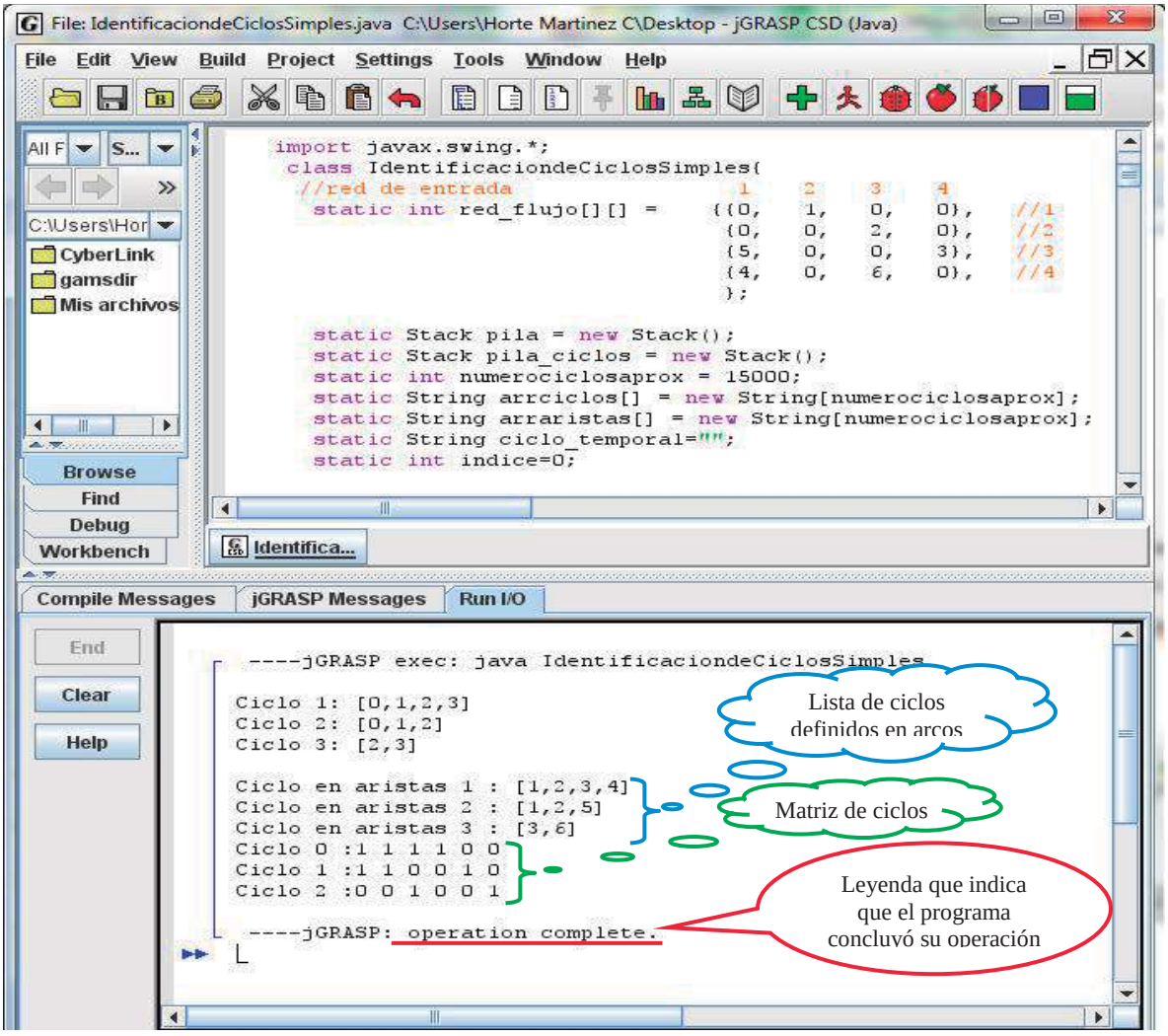


Figura B.8 Resultados y confirmación de operación terminada en la ventana inferior de jGRASP

B.2 Programa en lenguaje JAVA para la selección del conjunto mínimo de corrientes de corte

B.2.1 Instalación de las herramientas necesarias para correr el programa

Teniendo instaladas las herramientas de JAVA y jGRASP, lo único que falta por instalarse es el programa para la selección del conjunto mínimo de corrientes de corte, el cual lleva por nombre "CorrientesdeCorte". Este programa se podrá descargar de la página. web:

http://www.moreliahosting.com/posgradofiq/index.php?option=com_content&view=article&id=19&Itemid=41

B.2.2 Especificación e ingreso de los datos de entrada

Al igual que para el programa de identificación de ciclos simples, éste se puede abrir con jGRASP de dos diferentes formas: ya sea dando doble clic sobre el programa "CorrientesdeCorte" o abriendo jGRASP y buscándolo, a través de la opción "open file", en la carpeta o dirección en donde se haya guardado.

El dato más importante para correr el programa es la matriz de ciclos simples, que se obtiene del archivo creado por el programa anterior (ver Figura B.9).

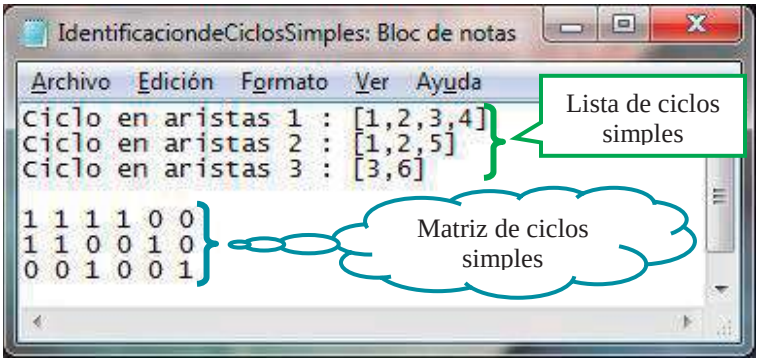


Figura B.9 Archivo generado por el programa "IdentificaciondeCiclosSimples"

Dicha matriz se deberá de guardar en un archivo de texto (bloc de notas), como se muestra en la Figura B.10, con el nombre deseado. El formato de la matriz de ciclos simples no deberá de ser alterado para que el programa funcione correctamente.

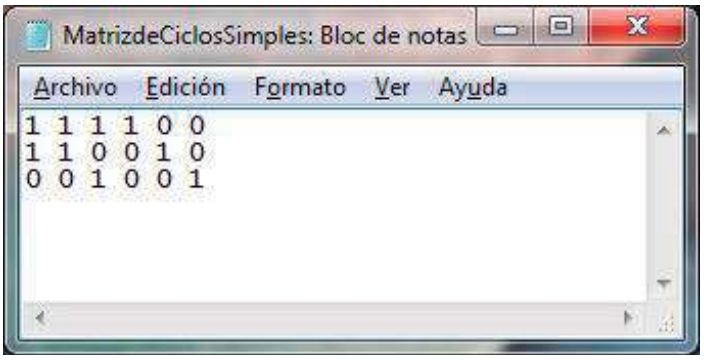


Figura B.10 Matriz de Ciclos Simples guardada como archivo de texto

Después de haber generado el archivo con la matriz de ciclos simples, se ingresa su ruta de almacenamiento en la línea 165 del programa. Cada vez que se avanza o retrocede un renglón (línea), en la parte inferior del programa se puede observar el número del renglón sobre el cual se encuentra posicionado el cursor, tal como se muestra en la Figura B.11. En caso de que el archivo se encuentre guardado dentro de la misma carpeta en donde está el programa, no será necesario escribir la ruta completa, sólo bastará con escribir el nombre del archivo como se observa en la Figura B.12.

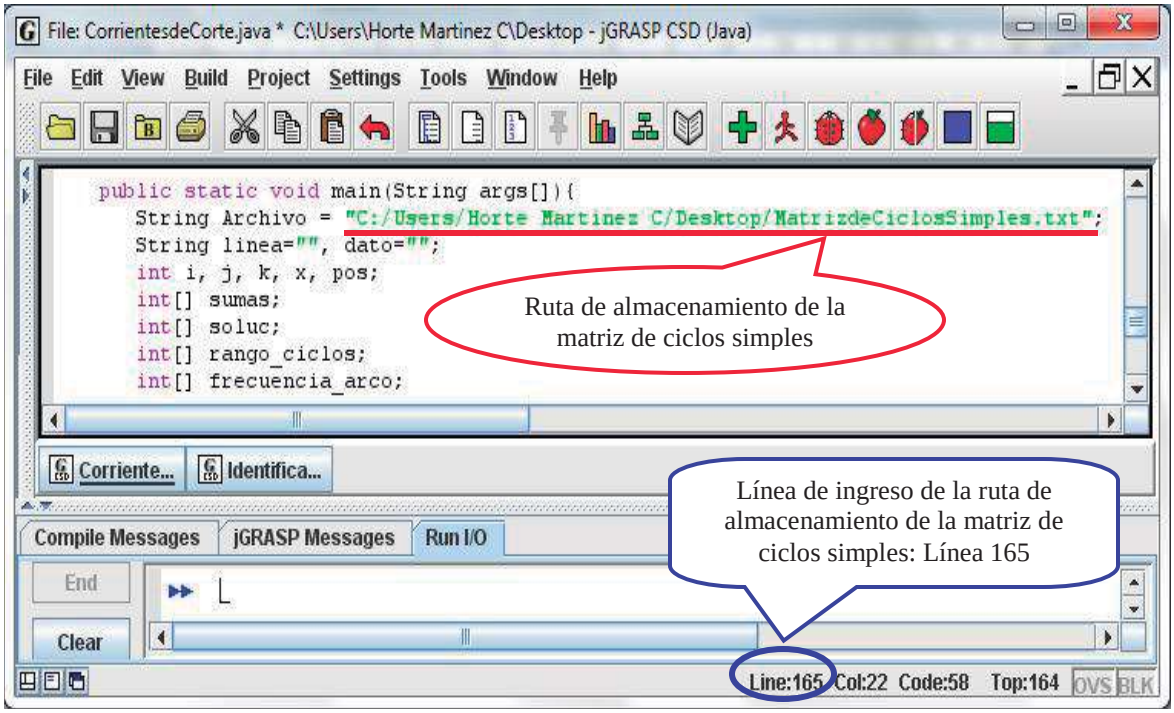


Figura B.11 Ingreso de la ruta de almacenamiento de la matriz de ciclos simples en el programa para la selección del conjunto mínimo de corrientes de corte

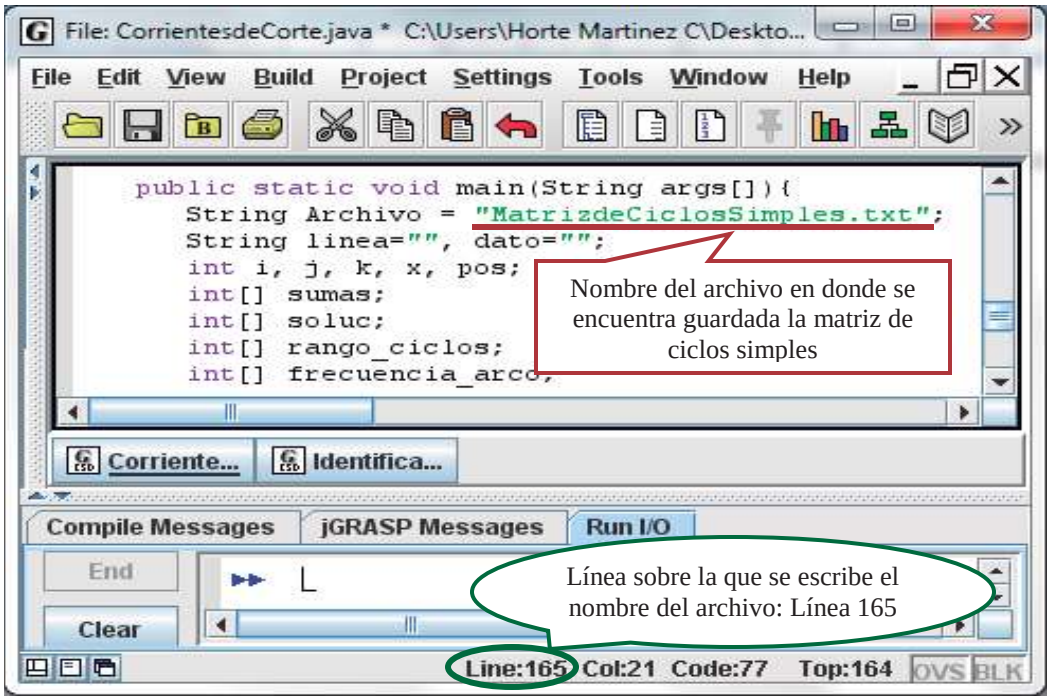


Figura B.12 Ingreso del nombre del archivo de la matriz de ciclos simples en el programa para la selección del conjunto mínimo de corrientes de corte

Cabe mencionar que los datos de entrada deberán escribirse manteniendo los signos de puntuación, tal como se ejemplifica en las Figuras B.11 y B.12, para que el programa corra adecuadamente.

B.2.3 Cómo correr el programa

Antes de correr el programa y después de haber modificado la ruta de almacenamiento de la matriz de ciclos existente por la del caso bajo estudio, se debe compilar el programa. La compilación se logra dando un clic sobre el ícono  . Cuando el programa termina de compilar arroja el texto "----jGRASP: operation complete", el cual aparece en el recuadro blanco que se encuentra en la parte inferior de la pantalla de jGRASP, como se aprecia en la Figura B.13. Esto indica que el programa fue compilado con éxito y que no hubo errores en esta etapa.

Después de compilar, el programa está listo para correrse y para tal efecto es necesario dar clic sobre el icono  . Cuando el programa termina de correr arroja la leyenda "----jGRASP: operation complete.".

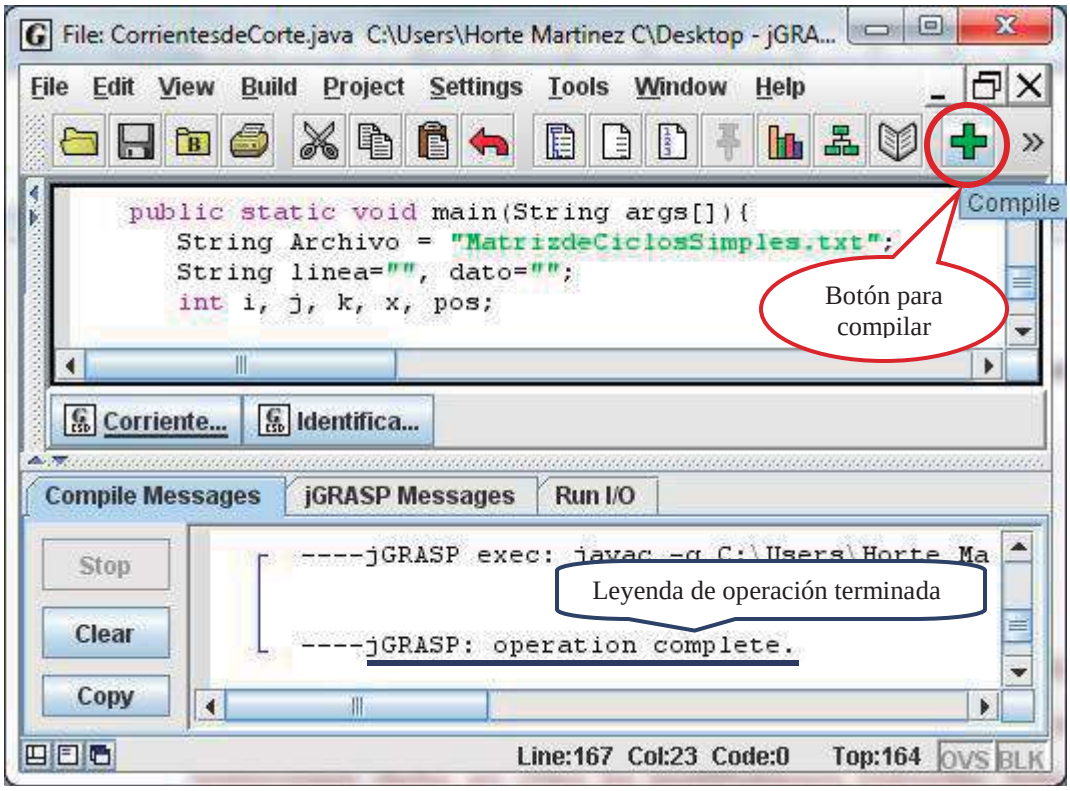


Figura B.13 Compilación

La Figura B.14 muestra los pasos a seguir para correr el programa y obtener los resultados deseados.

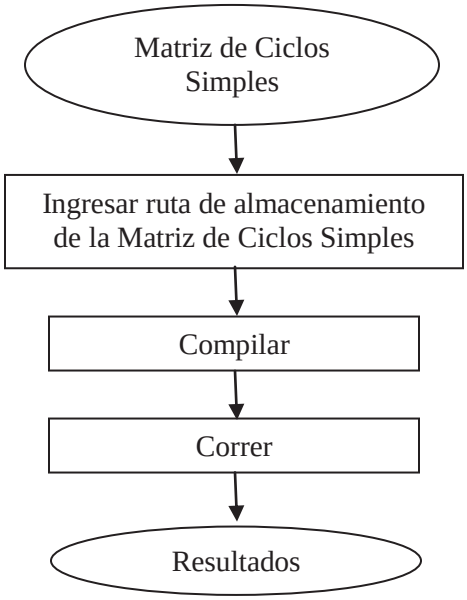


Figura B.14 Diagrama de Flujo de los pasos realizados para llegar a los resultados

B.2.4 Interpretación de los resultados

Los resultados son visibles en la ventana inferior del programa como se observa en la Figura B.15. Aquí se puede observar el conjunto de corte que permite resolver el caso bajo estudio.

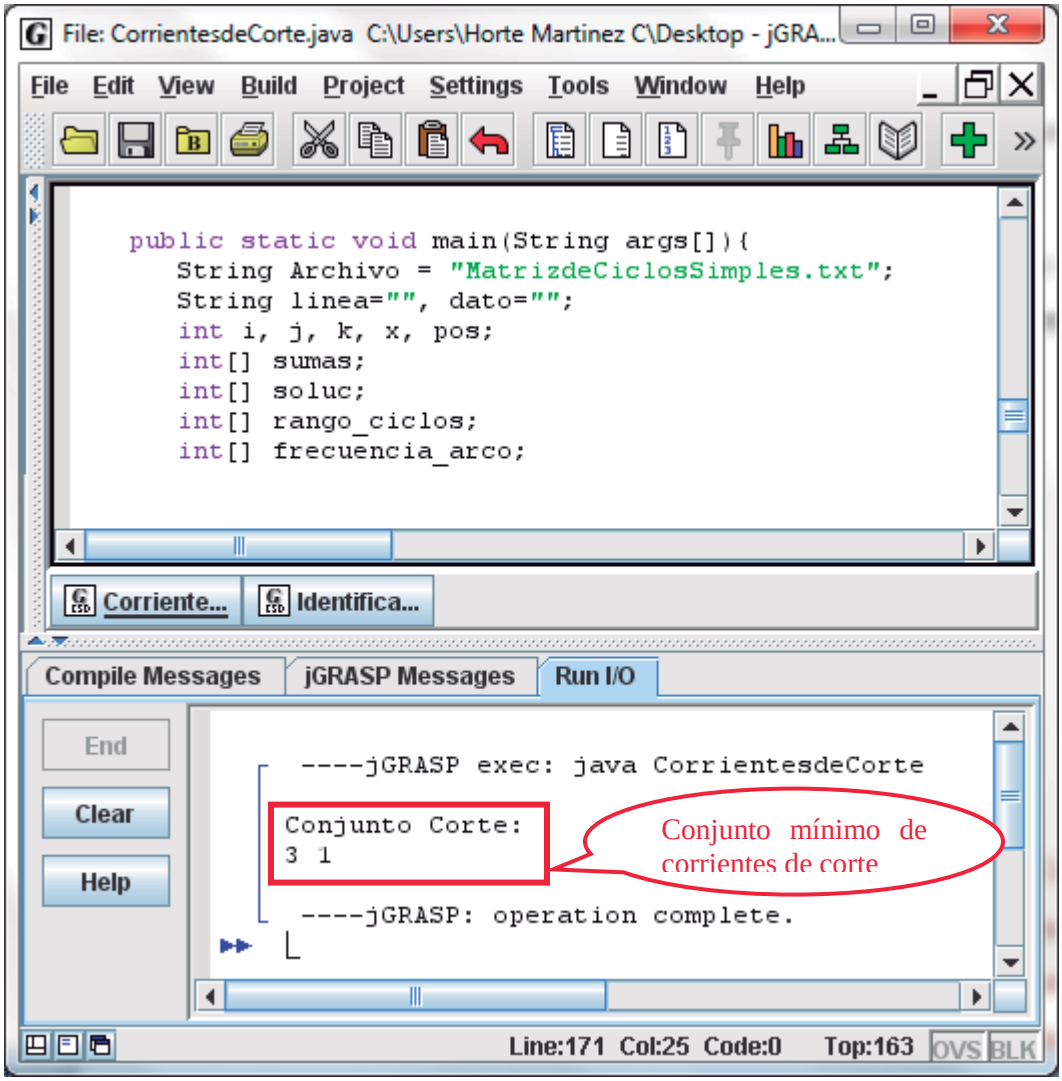


Figura B.15 Conjunto mínimo de corrientes de corte

B.3 Programa en GAMS para la selección del conjunto mínimo de corrientes de corte

B.3.1 Instalación de las herramientas necesarias para correr el programa

GAMS (General Algebraic Modeling System) es un sistema de modelado de alto nivel para la optimización matemática. Este sistema está diseñado para modelar y resolver problemas lineales, no lineales y de programación mixta. También está diseñado para aplicaciones complejas y modelado a gran escala.

GAMS inicialmente fue diseñado para aplicaciones relacionadas con la economía y la administración y actualmente es utilizado también en diversos ámbitos de la ingeniería y la ciencia.

El paquete GAMS puede ser descargado directamente de la página web http://www.gams.com. El programa para la identificación del conjunto mínimo de corrientes de corte, que lleva por nombre "IDENTIFICACION DE CORRIENTES DE CORTE", se puede descargar de la página web: http://www.moreliahosting.com/posgradofiq/index.php?option=com_content&view=article&id=19&Itemid=41

B.3.2 Especificación e ingreso de los datos de entrada

Habiendo instalado las herramientas necesarias se procederá a abrir el programa "IDENTIFICACIÓN DE CORRIENTES DE CORTE", ya sea dando doble clic sobre éste o abriendo GAMS y buscándolo, a través de la opción "open file", en la carpeta o dirección en donde se haya guardado.

La información más importante para echar a andar este programa es la matriz de ciclos simples generada en JAVA. A continuación se describen a detalle los datos que deberán ser modificados dentro del programa para ponerlo en marcha y obtener el conjunto mínimo de corrientes de corte.

a) *Cantidad de ciclos que componen la matriz de ciclos simples*. Este dato se obtiene de la matriz generada por el programa de JAVA, recordando que las filas de esta matriz representan los ciclos y las columnas los arcos.

b) *Cantidad de arcos que componen la matriz de ciclos simples*. La cantidad de arcos de esta matriz se conoce desde el momento en el que se tiene definido el digrafo sin nodos fuente y sumidero con el cual se trabajará. O bien, también se puede conocer a través de la matriz generada en JAVA al igual que el dato anterior.

c) *Matriz de ciclos simples expresada en arcos*. Esta matriz es la misma mencionada en los incisos anteriores y puede ser extraída del archivo de texto que generó el programa para la Identificación de ciclos simples de JAVA. El único detalle que hay que considerar para su ingreso en el programa es que hay que numerar las filas y las columnas como se muestra en la Figura B.16 (c).

La Tabla B.2 y la Figura B.16 muestran cada uno de los datos de entrada arriba descritos e indica cual es la ubicación de los mismos dentro del programa.

Tabla B.2 Datos de entrada para la selección del conjunto mínimo de corrientes de corte en GAMS y su ubicación de ingreso

Dato de entrada	Dónde ingresarlo
a) Cantidad de ciclos que componen la matriz de ciclos simples	Después de la leyenda "INDICE PARA LOS CICLOS (RENGLONES DE C) ", colocar el número correspondiente a los ciclos del caso bajo estudio en el espacio indicado en la Figura B.16 (a)
b) Cantidad de arcos que componen la matriz de ciclos simples	Después de la leyenda "INDICE PARA LOS ARCOS (COLUMNAS DE C) ", colocar el número correspondiente a los arcos del caso bajo estudio en el espacio indicado en la Figura B.16 (b)
c) Matriz de ciclos simples expresada en arcos	Luego de la leyenda "TABLE C(I,J) ELEMENTOS DE LA MATRIZ C", sustituyendo la matriz que ya existe, agregando las filas y columnas que hagan falta, cuidando de mantener el formato como el que se muestra en la Figura B.16 (c)

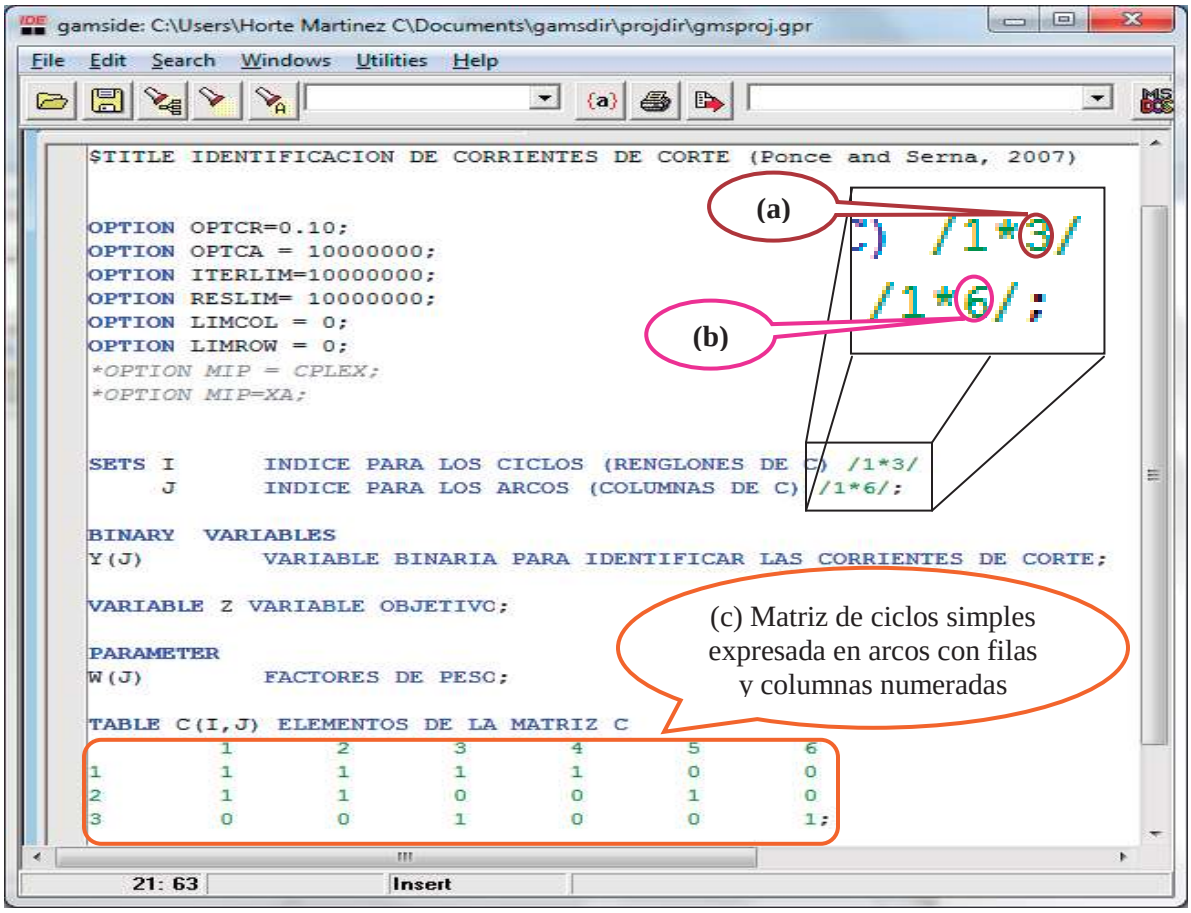


Figura B.16 Datos de entrada y ubicación de ingreso

NOTA: Los datos deberán ser ingresados tal como se muestran en la Figura B.16, teniendo cuidado de no quitar o modificar algún signo de puntuación (coma, punto, punto y coma, paréntesis, etc.) pues, en caso de hacerlo, el programa mostrará errores y no arrojará resultados.

B.3.3 Cómo correr el programa

Antes de correr el programa se pueden guardar los cambios realizados, ya sea que se guarde el programa con el nombre del caso bajo estudio o con el nombre que tenía anteriormente. La opción de cambiarle el nombre es para tenerlo en caso de ocuparlo en un futuro e identificarlo fácilmente.

El programa se ejecuta al dar clic sobre el ícono  . Al terminar de correr, el programa abrirá automáticamente 2 ventanas; en una de ellas aparecerán las leyendas

"Reading solution for Model CORTE" y "***Status: Normal completion", como se observa en la figura B.17, lo que indica que el programa ha concluido su operación satisfactoriamente.

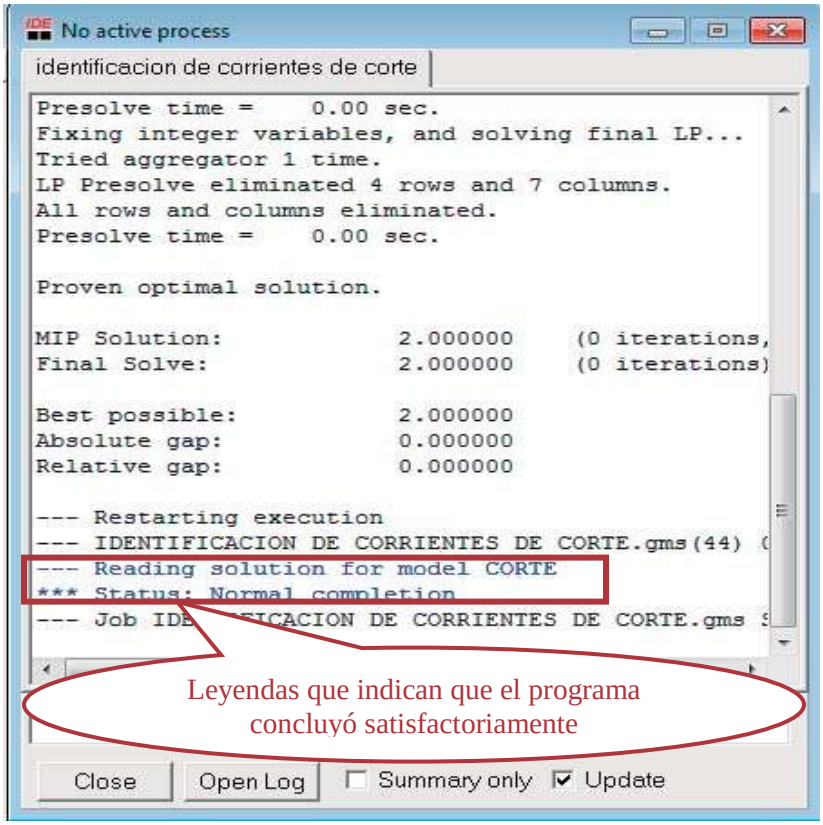


Figura B.17 Ventana de reporte operativo del programa

Los pasos que se siguieron para llegar a los resultados se pueden observar a continuación en la Figura B.18.

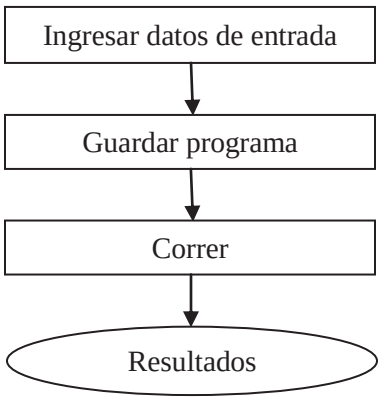


Figura B.18 Diagrama de Flujo de los pasos realizados para llegar a los resultados

B.3.4 Interpretación de los resultados

Como se mencionó anteriormente, cuando el programa termina de operar arroja dos ventanas, una que es el reporte de los resultados y la otra que es el reporte de operación del programa, como se observa en la Figura B. 19 a) y b) respectivamente.

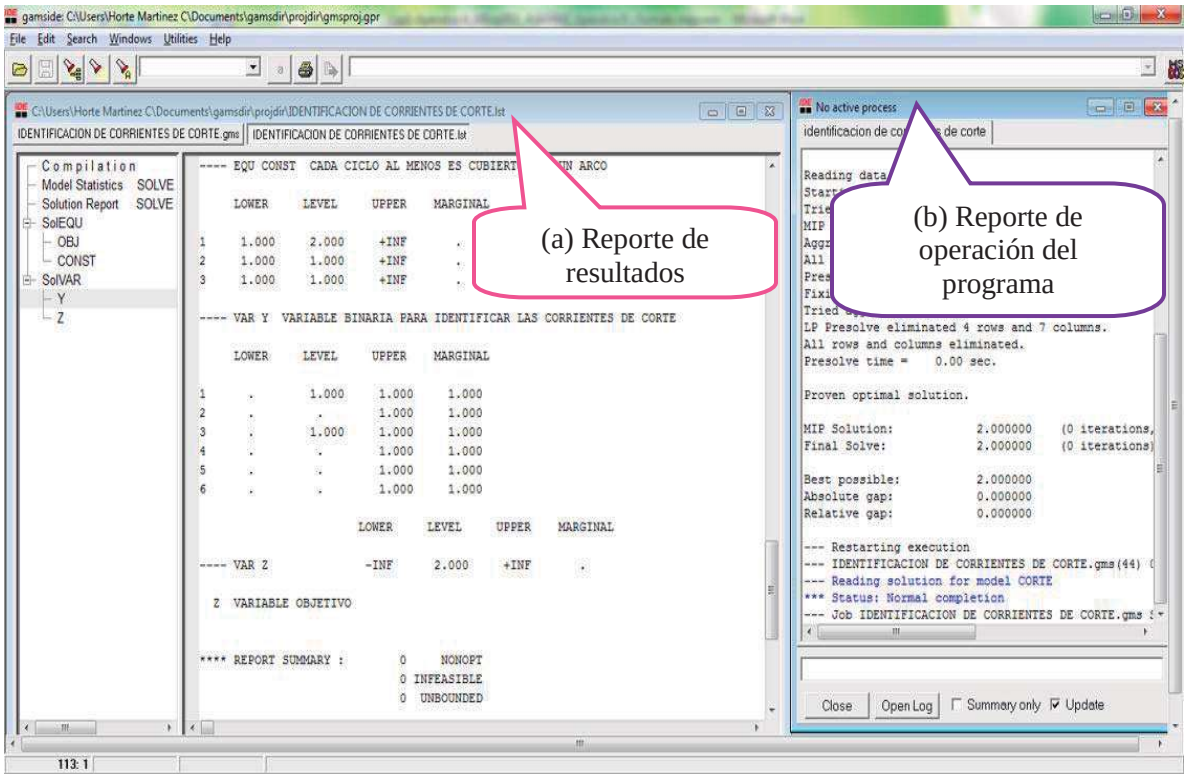


Figura B.19 Ventana de resultados y ventana de operación del programa

Los resultados acerca de las corrientes de corte seleccionadas por el programa se pueden observar casi al finalizar la hoja del reporte de resultados, después del renglón en el que aparece la leyenda "---- VAR Y VARIABLE BINARIA PARA IDENTIFICAR LAS CORRIENTES DE CORTE", como aparece en la Figura B.20. Si se da doble clic sobre la letra Y que aparece debajo de la palabra SolVAR, del lado izquierdo de la ventana mencionada, automáticamente la página muestra la zona en la que se encuentran los resultados. Los números que aparecen en la primera columna se refieren a los arcos que componen el digrafo bajo estudio, mientras que los números 1.000 posicionados debajo de la columna LEVEL indican las corrientes de corte que forman parte del conjunto mínimo de corrientes de corte.

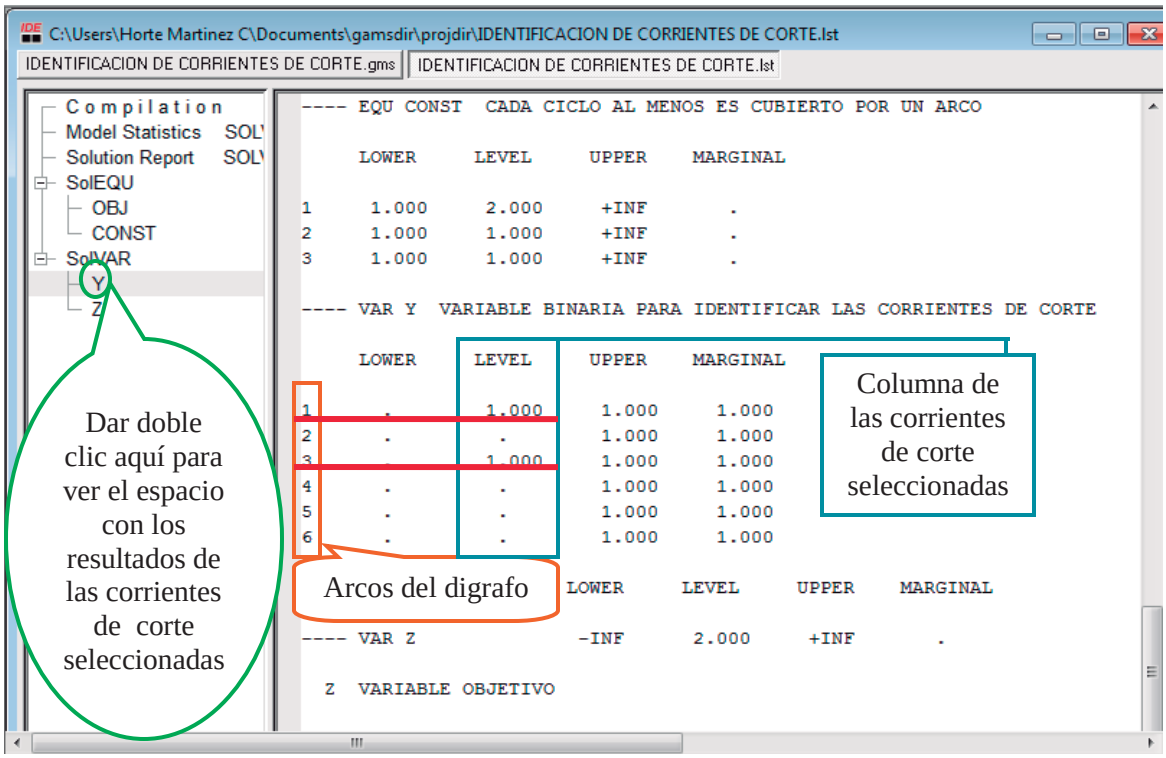


Figura B.20 Ventana de resultados e interpretación

Como se puede observar en la figura anterior, el resultado para este caso de estudio son las corrientes 1 y 3, las cuales se encuentran subrayadas en color rojo. En este caso el resultado es el mismo que el del programa desarrollado en JAVA; sin embargo, no en todos los casos se obtienen las mismas corrientes de corte por ambos métodos, por lo que esto nos da una opción más para seleccionar el conjunto mínimo de corte que más nos convenga.

B.4 Programa en lenguaje JAVA para la determinación del orden de precedencia

B.4.1 Instalación de las herramientas necesarias para correr el programa

Las herramientas necesarias para correr este programa son JAVA y jGRASP, las cuales se instalan como se explica en la sección B.1.1. A continuación, lo único que falta por instalar es el programa para determinar el orden de cálculo de los nodos del caso bajo estudio que lleva por nombre “ORDENAMIENTO”, el cual se podrá descargar u obtener de la página web:

http://www.moreliahosting.com/posgradofiq/index.php?option=com_content&view=article&id=19&Itemid=41

B.4.2 Especificación e ingreso de los datos de entrada

La matriz de adyacencia de nodos es uno de los datos más importantes para que el programa determine el orden de cálculo del caso bajo estudio. Por lo tanto, antes de ingresar los datos de entrada, es necesario crear un archivo de texto con la matriz mencionada, la cual deberá de tener el formato que se muestra en la Figura B.21. Como se puede apreciar, esta matriz es muy parecida a la que se utilizó en el programa de JAVA para la identificación de ciclos simples, con la diferencia de que ahora la matriz no se escribirá dentro del programa y no contendrá la fila ni la columna que enumeran en orden secuencial los nodos que la conforman.

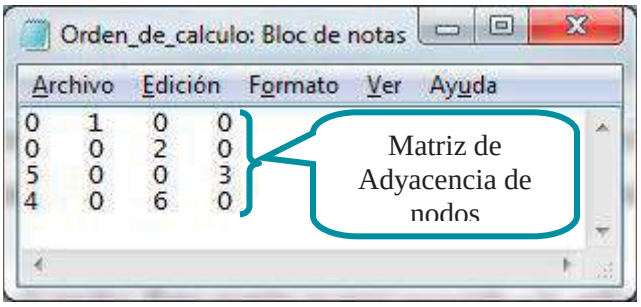


Figura B.21 Matriz de Adyacencia de Nodos guardada como archivo de texto

Con este archivo creado, las herramientas necesarias instaladas y el programa "Ordenamiento" abierto, es momento de ingresar los datos de entrada. El primer dato a ingresar es la ubicación del archivo de texto que contiene la matriz de adyacencia de nodos. Si el archivo de texto no se encuentra guardado en la misma carpeta que el programa será necesario escribir entre comillas sobre la línea 121 la ruta del mismo; en caso contrario, en la misma línea sólo se deberá de indicar el nombre del archivo entre comillas tal como se muestra en la Figura B.22. El número de línea se puede observar en la parte inferior de la ventana del programa.

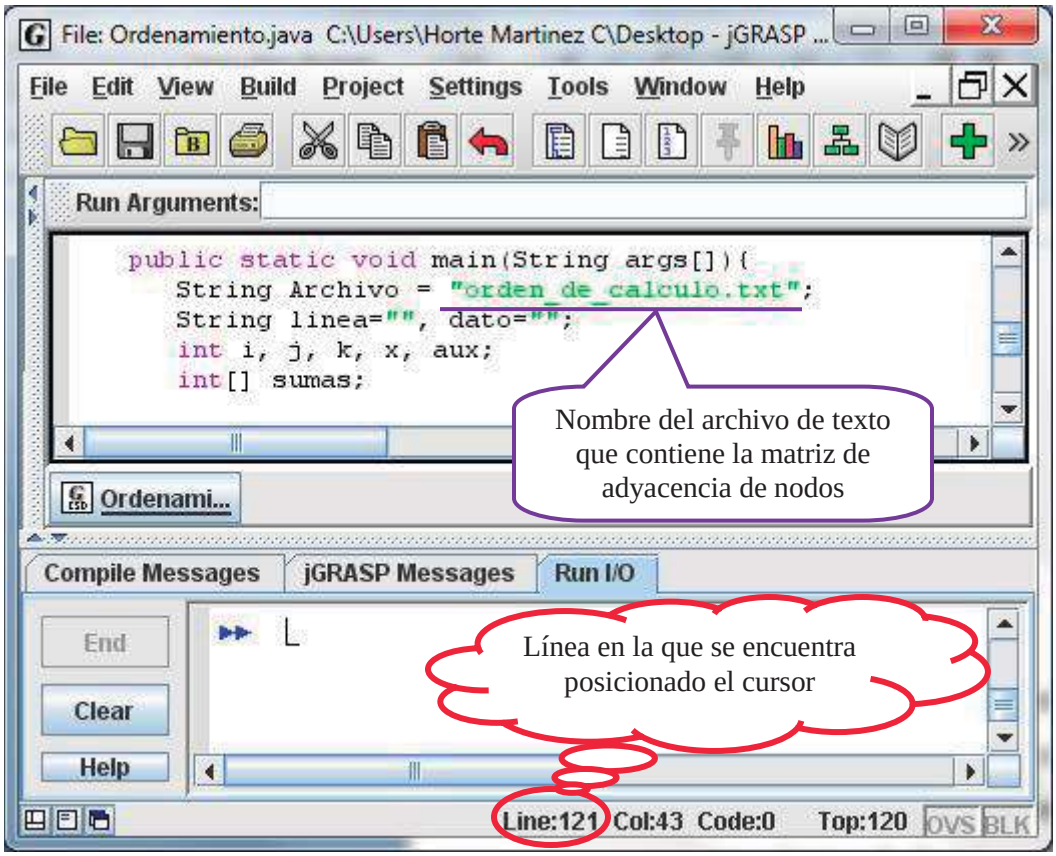


Figura B.22 Ingreso del nombre del archivo de texto que contiene la matriz de adyacencia de nodos

A continuación se deberán ingresar las corrientes de corte, ya sea las que resultaron del programa en JAVA o las arrojadas por el programa en GAMS. Dichas corrientes se deberán ingresar separadas por un espacio en el recuadro "Run Arguments", que aparece en la parte superior de la ventana como se indica en la Figura B.23. En caso de no ver este recuadro, anexarlo dando clic sobre la pestaña "Build", enseguida se desplegará una lista de la cual se debe seleccionar la opción "Run Arguments" y después aparecerá el recuadro para ingresar la información mencionada (ver la Figura B.24).

En caso de no respetar los formatos que se ejemplifican en las figuras es muy probable que el programa arroje, en lugar de los resultados esperados, notificaciones de algún error ocurrido al momento de leer los datos de entrada.

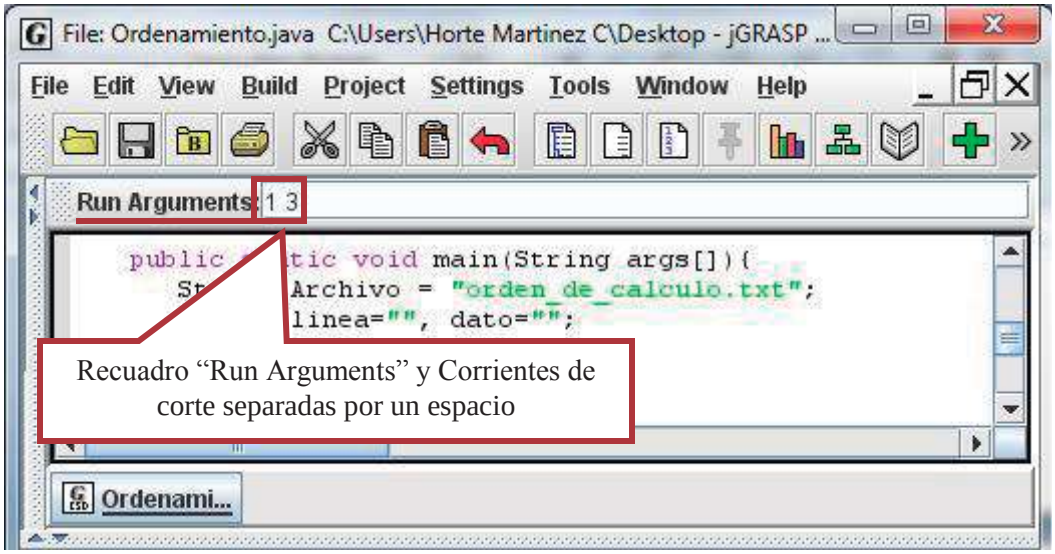


Figura B.23 Ingreso de las corrientes de corte

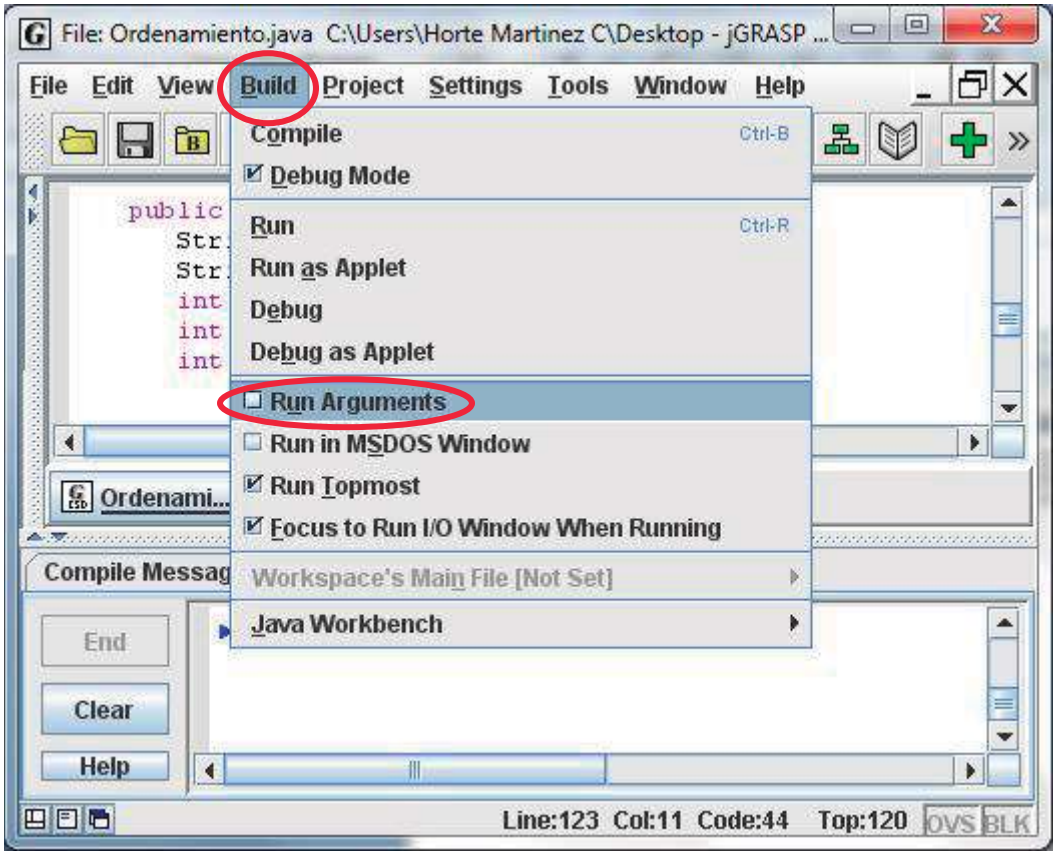


Figura B.24 Opción "Run Arguments"

B.3.3 Cómo correr el programa

El programa se debe compilar antes de ejecutarlo, lo cual se logra dando un clic sobre el ícono  . Cuando termina la compilación, aparece el texto "----jGRASP: operation complete", el cual se puede observar en el recuadro blanco de la parte inferior de la pantalla, indicando que el programa fue compilado con éxito y que no hubo ningún error.

Ahora sí, el programa está listo para entrar en marcha y esto se realiza haciendo clic sobre el ícono  . Cuando el programa termina de operar, en el recuadro inferior, se puede observar la leyenda "*----jGRASP: operation complete"*, tal como se aprecia en la Figura B.25.

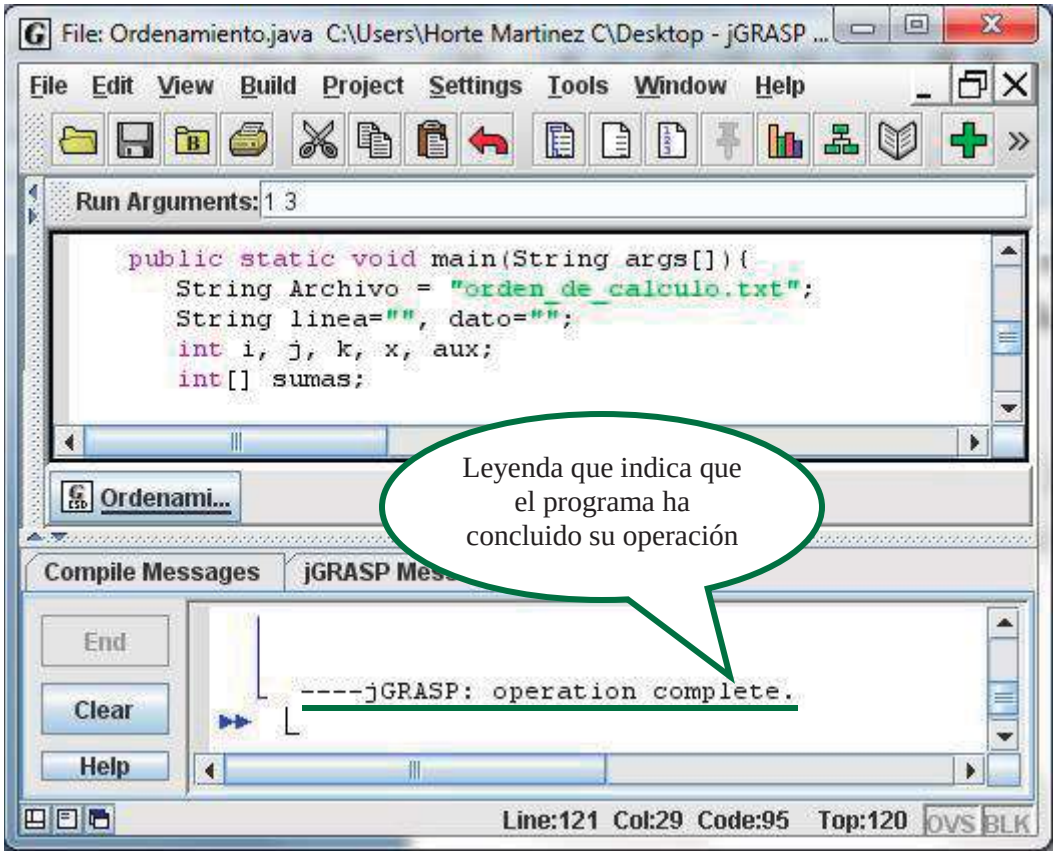


Figura B.25 Fin de operación del programa

B.3.4 Interpretación de los resultados

Antes de que aparezca la leyenda que indica que el programa ha concluido su operación satisfactoriamente, en la misma ventana inferior aparecen el conjunto de corrientes de corte ingresado y enseguida, en forma de lista y numerado, el orden de cálculo encontrado por el programa. Lo descrito anteriormente se puede apreciar detalladamente en la Figura B.26.

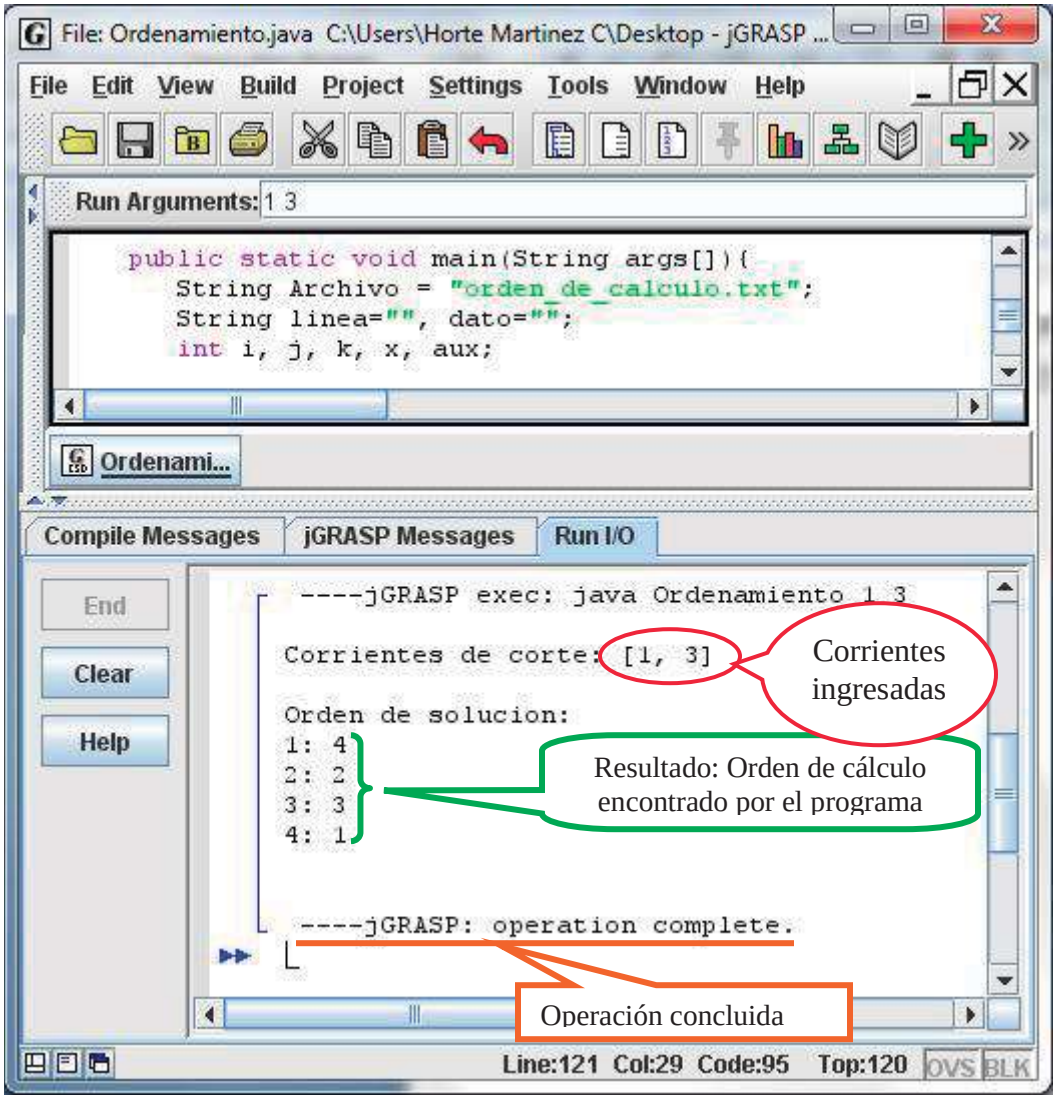


Figura B.26 Interpretación de resultados

En este caso, el orden de cálculo que determinó el programa es: 4, 2, 3 y 1 y corresponde al conjunto de corte {1, 3}. Cabe mencionar que existen varias opciones de

orden de cálculo para algunos casos y que la solución propuesta por este programa es sólo una opción.

EXENCIÓN DE RESPONSABILIDAD

Los programas incluidos en este texto son para uso educativo solamente. Son proporcionados sin ninguna garantía. Estos no deberán venderse bajo ninguna circunstancia.

REFERENCIAS BIBLIOGRAFICAS

Barkley, R.W. y R.L. Motard, "Decomposition of Nets". *Chem. Eng. J*., Vol. 3, 266-275 (1972).

Gundersen, T., y Hertzberg, T., "Partitioning and Tearing chemical process flowsheets," Proc. Int. Symp. Process Syst. Eng., PSE '82, Kyoto, Japón, Vol. 2, 9 (1982).

Gundersen, T., y Hertzberg, T., "Modeling, Identification and Control", 4**,** 139 (1983).

Husain, A., Chemical Process Simulation**,** Wiley Eastern Ltd, New Delhi (1986).

Johnson, D.B., "Finding all the Elementary Circuits of a Directed Graph", SIAM J. Comput., 4(1), pág. 77-84 (1975).

Kehat, E., and Shacham, M., "Process Technol International"**,** 18 (3), 115 (1973).

Kusiak, A. y J. Wang. "Efficient Organizing of Design Activities". *International Journal of Production Research,* Vol. 31, No. 3, pp. 753-769 (1993).

Lakshminarayanan, S., S. Subba Rao y Ch. Durgaprasada Rao, "Tearing and Precedence Ordering in Flowsheeting – An Effective Method", AIChE J., Vol. 115, pp. 53-66 (1992).

Lee, W. y D.F. Rudd, "On the Ordering of Recycle Calculations", *AIChE J*., Vol. 12, No. 6. pp 1184-1190 (1966).

Lien. K.M., and Hertzberg, T., Modeling, Identification and Control, 11(1), 13 (1990).

Mah, R.S.H., *Chemical Process Structures and Information Flows*, Butterworths Publishers, Stoneham, MA, USA (1990).

Mateti, P. y N. Deo, "On Algorithms for Enumerating all Circuits of a Graph", SIAM J. Comput., 5(1), pág. 90-99 (1976).

Motard, R.L., and Westerberg, A.W. AIChE., 27, No. 5, 725 (1981).

Murthy, C.L.N., and Husain, A., Proceedings, 34th annual session, IIChE, 3, 54 (1981).

Murthy, C.L.N., and Husain, A., Computers & Chem. Eng., 7, 2, 133 (1983).

Norman, R.L. A Matrix Method for Location of Cycles of a Directed Graph. AIChE Journal, Vol. 11, No. 3. pp 450-452 (1965).

Ollero, P. y C. Amselem, "Decomposition Algorithm for Chemical Process Simulation", *Chem. Eng. Res. Des*., Vol. 61, pp. 303-307 (1983).

Pho, T.K. and L. Lapidus, “Topics in Computer-Aided Design: Part I. “An Optimum Tearing Algorithm for Recycle Systems”, AIChE J., 19(6), pág. 1170-1181 (1973).

Read, R.C. and R. E. Tarjan, “Bounds on Backtrack Algorithms for Listing Cycles, Paths and Spanning Trees”, Networks, 5, pág. 237-252 (1975).

Reingold, E.M., “Combinatorial Algorithms”, Prentice-Hall, Englewood Cliffs, NJ (1977).

Rubin, D. I., *Chem. Eng.* Progr. *Symposium Ser.,* No. 37, 58, 54 (1962).

Sargent, R.W.H. y A.W. Westerberg. SPEEDUP in Chemical Engineering Design. *Trans. Inst. Chem. Engrs.* Vol. 42, pp. T190-197 (1964).

Sood, M.K.et al. AIChE J., Vo. 25, (No. 2), 209 (1979)

Tarjan, R. Depth-firts search and linear graph algorithms. *SIAM J. Compt.*, 1(2), 146-160. (1972).

Tarjan, R., SIAM J. Comp., 2, 211 (1973).

Tiernan, J.C., “An Efficient Search Algorithm to Find the Elementary Circuits of a Graph”, Comm. ACM, 13(12), pág. 722-726 (1970).

Ulanowicz, R.E. Identifying the structure of cycling in ecosystems. *Math. Biosci.*, 65, pp. 219-237 (1983).

Upadhye, R.S. y E.A. Grens II, “An Efficient Algorithm for Optimum Decomposition of Recycle Systems”, *AIChE J*., Vol. 18, No. 3, pp. 533-439 (1972).

Upadhye, R.S. y E.A. Grens II, “Selection of Decompositions for Chemical Process Simulation”, *AIChE J*., Vol. 21, No. 1, pp. 136-143 (1975).

Weinblatt, H., “A New Search Algorithm for Finding the Simple Cycles of a Finite Directed Graph”, J. Assoc. Comput. Mach., 19(1), pág. 43-56 (1972).

Zhou Li et al., Computers & Chem. Eng., 12, (No. 6), 581 (1988).

Zhu, J., Z. Han, M. Rao y K.T. Chuang. “Identification of Heat Load Loops and Downtream Paths in Heat Exchanger Networks”. *The Canadian Journal of Chemical Engineering*. Vol. 74, pp. 876-882, (1996).