Benemérita Universidad Autónoma de Puebla

Facultad de Ciencias de la Computación

Sistema web para el conteo eficiente de Conjuntos Independientes en estructuras jerárquicas (Árboles)

Tesis

Para obtener título de:

Licenciado en Ingeniería en Ciencias de la Computación

Presenta:

Luis David Morales Alcántara

Asesor:

M. C. Pedro Bello López

Puebla, Pue. Diciembre 2019

Agradecimientos

Primero y antes que nada, dar gracias a Dios, por estar conmigo en cada paso que doy, por fortalecer mi corazón e iluminar mi mente y por haber puesto en mi camino aquellas personas que han sido mi soporte y compañía durante todo el periodo de estudio.

Agradecer hoy y siempre a mi familia que siempre han procurado mi bienestar y que, si no fuese por el esfuerzo realizado por ellos, mis estudios no hubiesen sido posibles; a mi padre, que con sus consejos y experiencias ha ayudado a que se cumpla cada uno de mis objetivos; y a mi madre, por su apoyo y cariño.

De igual manera mi más sincero agradecimiento al profesor Pedro Bello López por haberme sugerido el tema de esta tesis, por su apoyo y consejos durante la realización del tema.

Al Dr. Guillermo de Ita Luna responsable del proyecto "*Proyecto del cuerpo académico (PRODEP): Algoritmos Combinatorios y aprendizaje por enriquecer con sus conocimientos";* el cual fue tema de apoyo para la realización de esta tesis.

A los miembros del jurado a la profesora Meliza Contreras González, el profesor Miguel Rodríguez Hernández y el profesor Pedro Bello López por la revisión del trabajo, una mejor retroalimentación y entendimiento, y sus aportaciones para el mejoramiento de esta investigación.

Índice

I. Introducción

Los problemas de conteo son estudiados en varias áreas del conocimiento, por ejemplo, en combinatoria enumerativa se estudian los problemas de enumerar y contar objetos combinatorios. En la física estadística se investiga, por ejemplo, el comportamiento de modelos usando funciones de partición, donde calcular éstas es un problema de conteo.

Aunque son muy interesantes matemáticamente hablando, existe relación con importantes problemas prácticos. Por ejemplo, los problemas de confiabilidad en una red son finalmente traducidos a problemas de conteo [11,12,14,15,17]. Ya que, calcular la probabilidad de que un grafo permanezca conectado dada una probabilidad de falla en cada línea de la red es esencialmente equivalente a contar el número de formas en que las líneas puedan fallar sin perder la conectividad en la red.

También en la Inteligencia artificial (IA), se usan problemas de conteo, por ejemplo, estimar las probabilidades en una red neuronal, es un problema esencial en IA [5]. Existen dos tipos de conteo, el conteo exacto y el conteo aproximado. En las Ciencias de la Computación interesa saber si existe un algoritmo eficiente para un problema de conteo, es decir, que pueda ser resuelto en tiempo polinomial [1].

Por ejemplo, el problema de contar los diferentes árboles de expansión de un grafo se puede resolver en tiempo polinomial mediante el teorema del árbol matricial de Kirchhoff [3], sin embargo, existen muchos otros problemas de conteo, donde no se han construido algoritmos eficientes que los resuelvan.

Un problema importante de conteo en grafos es el que versa sobre conjuntos independientes [4]. Un conjunto independiente de un grafo $G=(V, E)$ es un subconjunto de vértices $S \subseteq V$, tal que cada arista en E tiene a lo más un punto final en los elementos de S, esto es, $\forall\, x, y \in S, \{x, y\} \notin E$. Encontrar un conjunto independiente de tamaño máximo en un grafo dado, es un problema NP-completo.

Los problemas de conteo en el área de teoría de grafos, tal como el conteo de conjuntos independientes, son también utilizados en las aplicaciones de la física estadística. Por ejemplo, en la modelación del comportamiento de gases, en donde un vértice es un sitio posible para una partícula, y por tal, los sitios adyacentes a este no pueden ser simultáneamente ocupados por la misma partícula. La probabilidad $\pi(X)$ de que una configuración particular X del gas se presente es proporcional a $\lambda|X|$ para un parámetro positivo λ [14].

Por ejemplo: supongamos que se tienen los siguientes datos sobre las personas a, b, c, d, e, f y g [18]:

- La persona a habla inglés.
- La persona b habla inglés y español.
- La persona c habla inglés, italiano y ruso.
- La persona d habla japonés y español.
- La persona e habla alemán e italiano.
- La persona f habla francés, japonés y ruso.
- La persona g habla francés y alemán.

¿Cuál es el conjunto más grande de personas que no se puedan comunicar?

Para resolver esta pregunta se puede modelar utilizando un grafo y obteniendo el conjunto máximo de nodos que no se conecten entre sí.

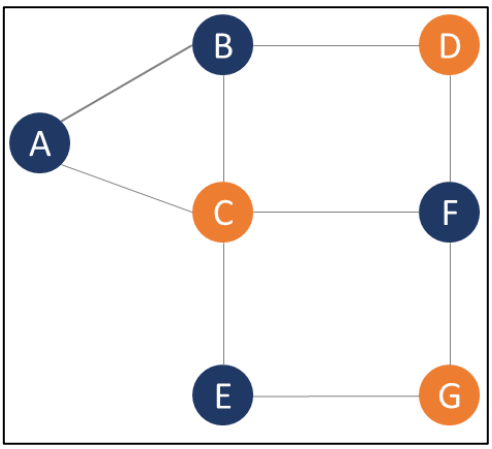


Figura 1.1. Grafo de personas con diferentes idiomas

La figura 1.1 muestra el grafo del ejemplo anterior donde se marca con color naranja el conjunto independiente máximo de personas que no se pueden comunicar entre si

Los conjuntos independientes son fundamentales en combinatoria, y tienen aplicaciones en: visión por computadora, reconocimiento de patrones, planificación, entre otras. Los conjuntos independientes se utilizan también en física estadística, donde el problema es un caso especial del modelo núcleo-duro (hard-core).

Es por esta razón que se propone en este trabajo, un sistema que permita realizar el cálculo de conjuntos independientes (CI) sobre árboles, por ser las estructuras de datos más empleadas en procesos de conteo de objetos.

La técnica que aquí es presentada, está basada en la serie de Fibonacci. El algoritmo desarrollado permite calcular de forma dinámica el número de conjuntos independientes conforme un grafo se reduce o se extiende al agregar nuevos vértices y aristas, mientras la topología del grafo esté conformada por la unión de cadenas y ciclos conectados por un vértice en común.

1.1 Objetivos generales y específicos

El objetivo general del proyecto de tesis consiste en diseñar e implementar un algoritmo que, pueda obtener el número de conjuntos independientes de una estructura jerárquica (árbol), estudiando, además, su complejidad y sus límites computacionales (tiempo y espacio). De Forma específica se tienen los siguientes objetivos:

- Diseñar el algoritmo propuesto y realizar un análisis de su complejidad.
- Implementar el análisis de conjuntos independientes mediante carga de un archivo o insertando los datos manualmente desde teclado.
- Realizar un conjunto de pruebas con árboles de diferente estructura.

II. Conceptos preliminares

Los grafos representan una forma simple de plantear y modelar diversos problemas que pueden resolverse computacionalmente, de esta forma resolvemos el problema del cálculo de conjuntos independientes en grafos. Las definiciones básicas necesarias para dicho problema se describen a continuación.

2.1 Grafos

Un grafo consta de un conjunto de nodos (o vértices) y un conjunto de arcos (o aristas). Cada arco de un grafo se especifica mediante un par de nodos. La figura 2.1 ilustra un grafo. El conjunto de nodos es {A, B, C, D, F} y el conjunto de arcos {(A, B), (A, D), (A, C), (C, D), (C, F)} para el siguiente grafo:

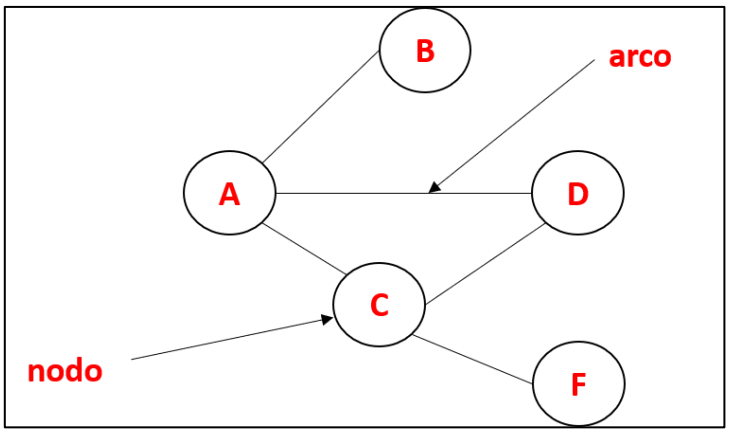


Figura 2.1: Grafo con 5 nodos y 5 arcos.

Si los pares de nodos que constituyen los arcos son pares ordenados, se dice que el grafo es un grafo dirigido (o un dígrafo). Las figuras 2.2 y 2.3 ilustran grafos dirigidos. Las flechas entre los nodos representan arcos. La punta de cada flecha representa el segundo nodo del par ordenado de nodos que constituyen un arco, y la cola de la flecha representa el primer nodo del par. El conjunto de arcos para el grafo de la figura 2.1 es {(A, B), (A, C), (A, D), (C, D), (F, C)]. Se hace uso de

paréntesis para representar los pares ordenados. Obsérvese que un grafo no es necesario un árbol (figura 2,1), pero un árbol debe ser un grafo 2.3. [2].

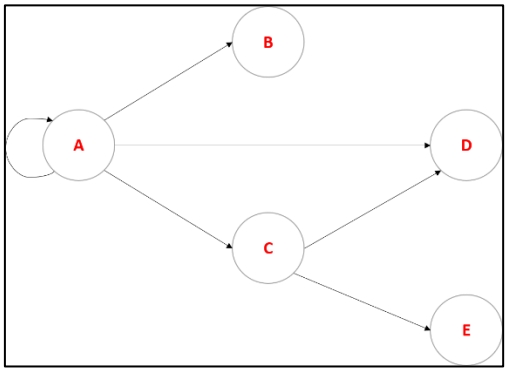


FIGURA 2.2: Grafo dirigido

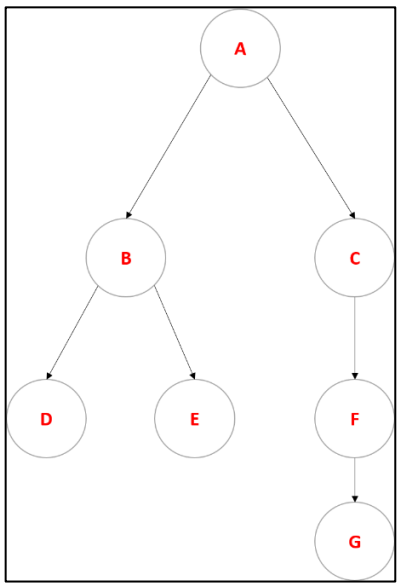


Figura 2.3: Árbol

2.1.1 Terminologia sobre grafos

- Al número de nodos del grafo se le llama orden del grafo.
- Un grafo nulo es un grafo de orden 0 (cero).
- Dos nodos son adyacentes si hay un arco que los une.
- En un grafo dirigido, si A es adyacente de B, no necesariamente B es adyacente de A.
- Camino es una secuencia de uno o más arcos que conectan con dos nodos.
- Un grafo se denomina conectado cuando existe siempre un camino que une dos nodos cualesquiera y desconectado en caso contrario.
- Un grafo es completo cuando cada nodo está conectado con todos y cada uno de los nodos restantes.
- El camino de un nodo así mismo se llama ciclo.
- Un grafo sin ciclos es un árbol.
- El entregado de un nodo indica, el número de arcos que llegan a ese nodo.
- El fuera de grado de un nodo, indica el número de arcos que salen de él.

- Un grafo de N vértices o nodos es un árbol si cumple las siguientes condiciones.
    1) Tiene N-1 arcos
    2) Existe una trayectoria entre cada par de nodos.
    3) Esta mínimamente conectado.

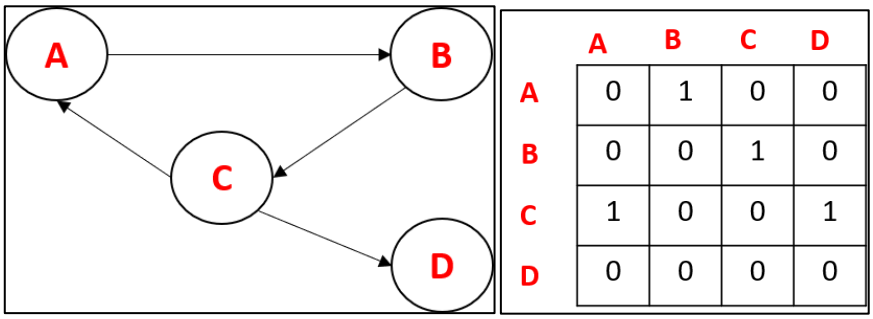


Figura 2.4: Grafo con matriz de adyacencia

La figura 2.4 representa la forma más común de representar un grafo, a través de una matriz denominada de adyacencias, donde cada columna y cada renglón se etiquetan con los nodos del grafo 8 (o dígrafo) y en la intersección de cada renglón y columna se indica con un 1 si hay una conexión y 0 en caso contrario.

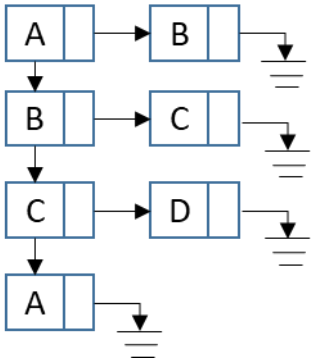


Figura 2.5: Representación de un grafo (dígrafo de la figura 2.4) mediante lista de adyacencia

La figura 2.5. muestra la representación de un grafo, un dígrafo en este caso mediante una lista ligada, este tipo de representación es dinámica y permite mayor flexibilidad a la hora de agregar o quitar nodos conservando la eficiencia en la complejidad espacial.

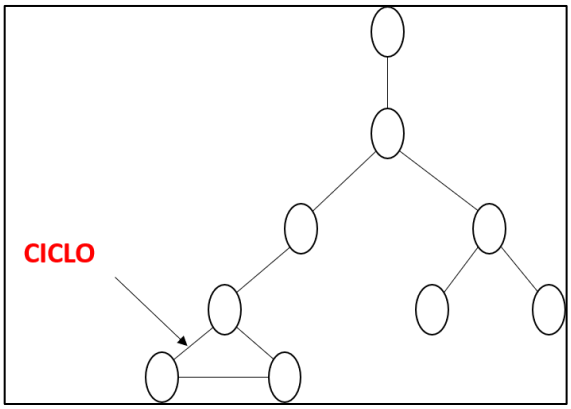


Figura 2.6. Grafo

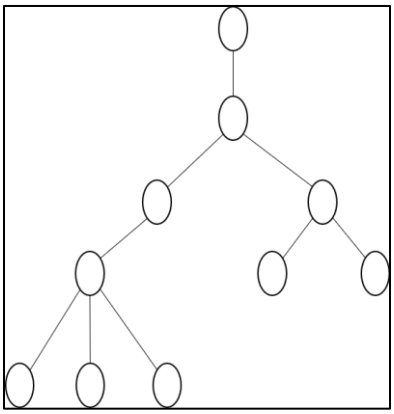


Figura 2.7. Árbol

Las figuras 2.6 y 2.7 representa la diferencia entre un grafo y un árbol, note que la diferencia la establece un ciclo.

2.2 Árboles

Un árbol es un conjunto de elementos que o está vacío o está dividido en 3 subconjuntos desarticulados. El primer subconjunto contiene un solo elemento llamado raíz del árbol. Los otros dos son en sí mismos árboles, llamados subárboles izquierdo y derecho del árbol original. Un subárbol izquierdo o derecho puede estar vacío. Cada elemento de un árbol se llama un nodo del árbol.

En la figura 2.8 se muestra un método convencional de dibujar un árbol. Este árbol consiste de nueve nodos y tiene a *A* como raíz. Su subárbol izquierdo tiene a *B* como raíz y subárbol derecho a *C*. Esto queda señalado por las dos ramas que salen en *A*: hacia *B* a la izquierda y hacia *C* a la derecha. La ausencia de ramas indica que es un subárbol vacío.

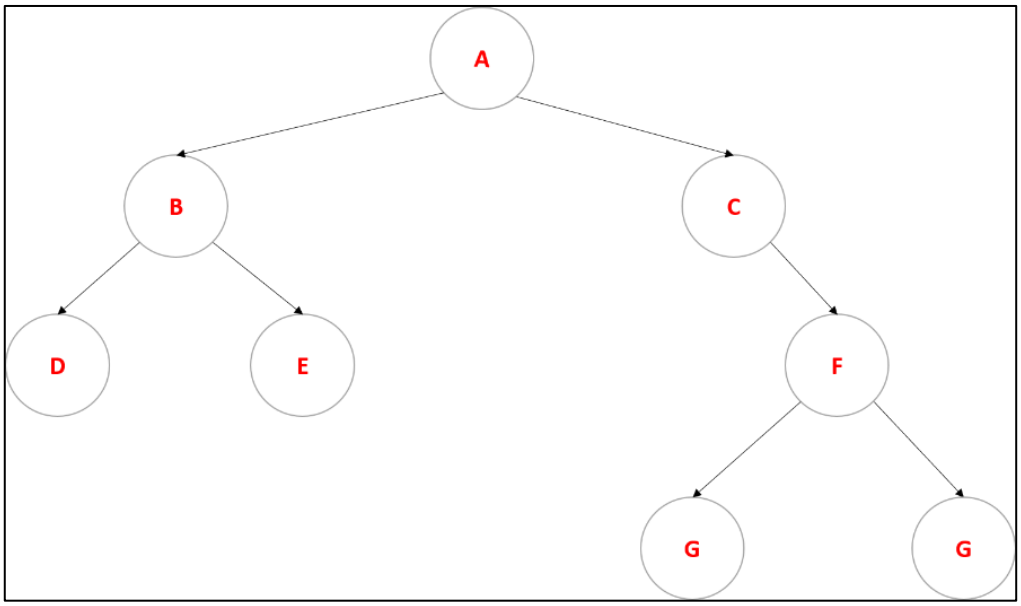


Figura 2.8. Árbol

Aunque los árboles naturales crecen con sus raíces en la tierra y sus hojas en el aire, los científicos de la computación representan, casi a nivel universal, las estructuras de datos como arboles con la raíz en la cúspide y las hojas en el fondo. La dirección de la raíz hacia las hojas se llama "*hacia abajo*", y de las hojas a la raíz "*hacia arriba*".

Ir de las hojas a la raíz se llama "*trepar*" a un árbol; e ir de la raíz hacia las hojas, "*descender*" de un árbol [2].

Una de las propiedades más llamativas de los árboles es la capacidad de acceder a muchísimos objetos desde un punto de partida o raíz en unos pocos pasos. Un árbol es una estructura de datos útil cuando cada punto de un proceso hay que tomar una decisión de dos o más opciones.

De esta forma, un árbol es un grafo en el que cualesquiera dos vértices están conectados por exactamente un camino.

Un árbol es un grafo simple no dirigido *G* que satisface:

- *G* es conexo y no tiene ciclos.

2.2.1 Recorrido sobre árboles

Los recorridos son algoritmos que nos permiten recorrer un árbol en un orden especifico, los recorridos nos pueden ayudar encontrar un nodo en el árbol, o buscar una posición determinada para insertar o eliminar un nodo.

Básicamente podemos catalogar las búsquedas en dos tipos, las búsquedas en profundidad y las búsquedas en amplitud.

2.3 Búsqueda en profundidad

- **Pre orden: (raíz, izquierdo, derecho).**

Para recorrer un árbol no vacío en pre-orden, hay que realizar las siguientes operaciones recursivamente en cada nodo, comenzando con el nodo de raíz:

1. Visite la raíz
2. Atraviese el sub-árbol izquierdo
3. Atraviese el sub-árbol derecho

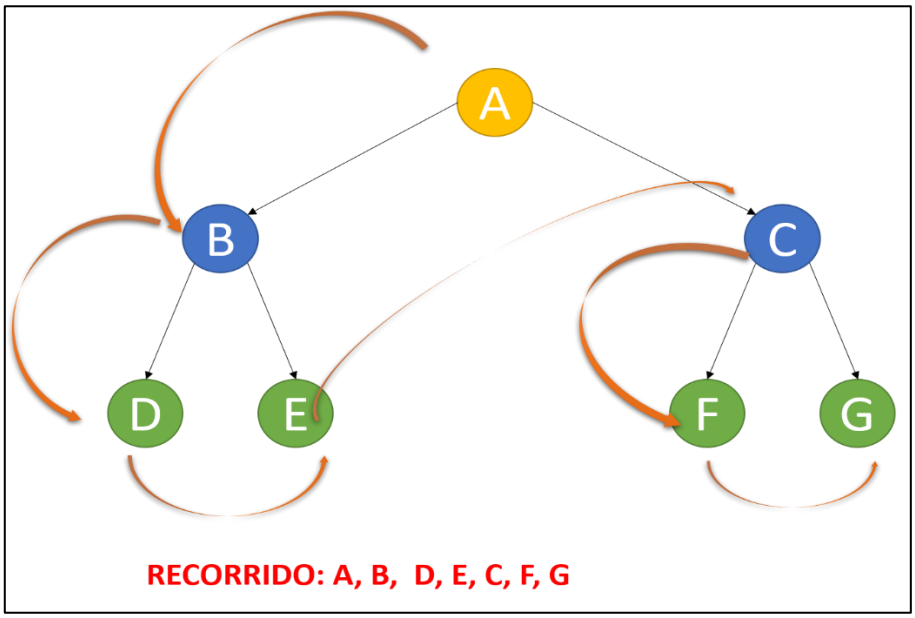


Figura 2.9. Orden del recorrido del árbol en Pre-Orden

- **In-orden: (izquierdo, raíz, derecho).**

Para recorrer un árbol no vacío en in-orden (simétrico), hay que realizar las siguientes operaciones recursivamente en cada nodo:

1. Atraviese el sub-árbol izquierdo

2. Visite la raíz

3. Atraviese el sub-árbol derecho

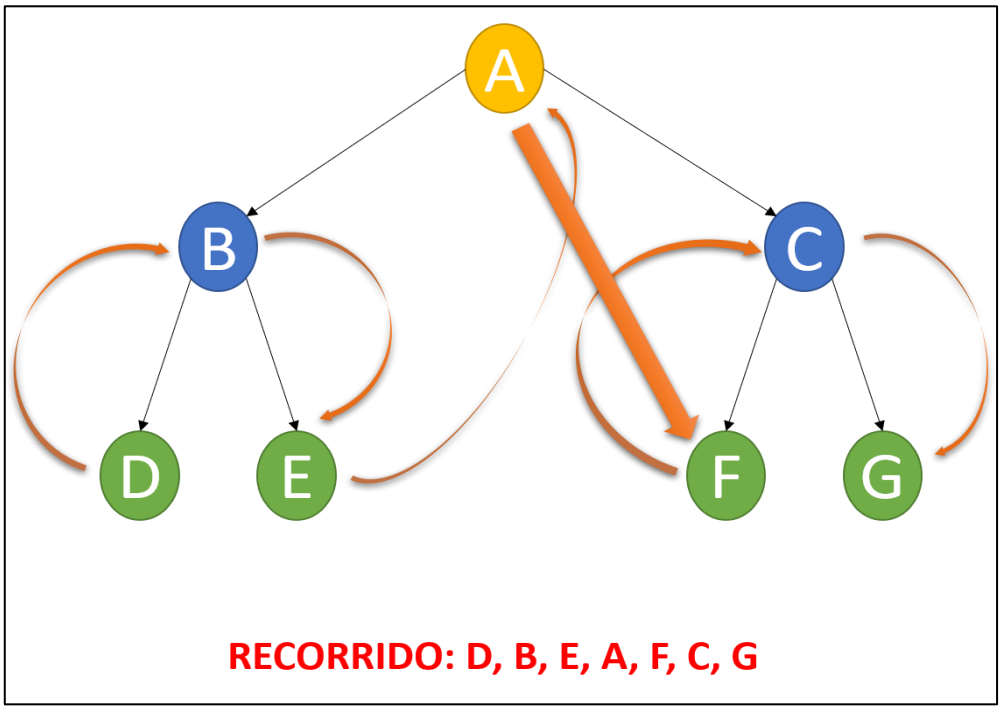


Figura 2.10. Orden del recorrido del árbol en In-Orden

- **Post-orden: (izquierdo, derecho, raíz).**

Para recorrer un árbol no vacío en post-orden, hay que realizar las siguientes operaciones recursivamente en cada nodo:

1. Atraviese el sub-árbol izquierdo
2. Atraviese el sub-árbol derecho
3. Visite la raíz

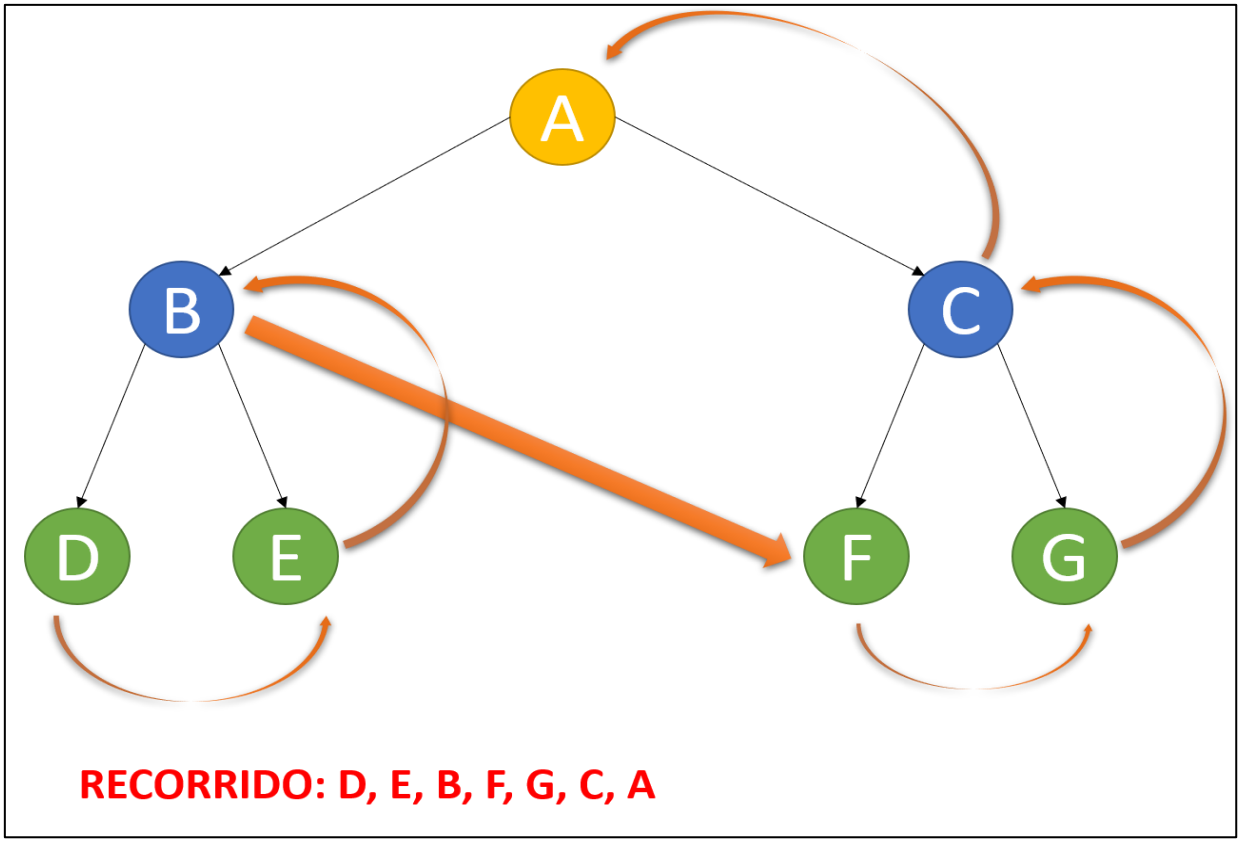


Figura 2.11. Orden del recorrido del árbol en Post-Orden

En general, la diferencia entre pre-orden, in-orden y post-orden es cuándo se recorre la raíz.

• En pre-orden, la raíz se recorre antes que los recorridos de los subárboles izquierdo y derecho.

• En in-orden, la raíz se recorre entre los recorridos de los árboles izquierdo y derecho.

• En post-orden, la raíz se recorre después de los recorridos por el subárbol izquierdo y el derecho.

2.4 Búsqueda en amplitud

En este caso el recorrido se realiza en orden por los distintos niveles del árbol. Así, se comenzaría tratando el nivel 1, que solo contiene el nodo raíz, seguidamente el nivel 2, el 3 y así sucesivamente. Al contrario que en los métodos de recorrido en profundidad, el recorrido por niveles no es de naturaleza recursiva.

Por ello, se debe utilizar una cola para recordar los subárboles izquierdos y derecho de cada nodo. El esquema algoritmo para implementar un recorrido por niveles es exactamente el mismo que el utilizado en la versión iterativa del recorrido en pre-orden pero cambiando la estructura de datos que almacena los nodos por una cola.

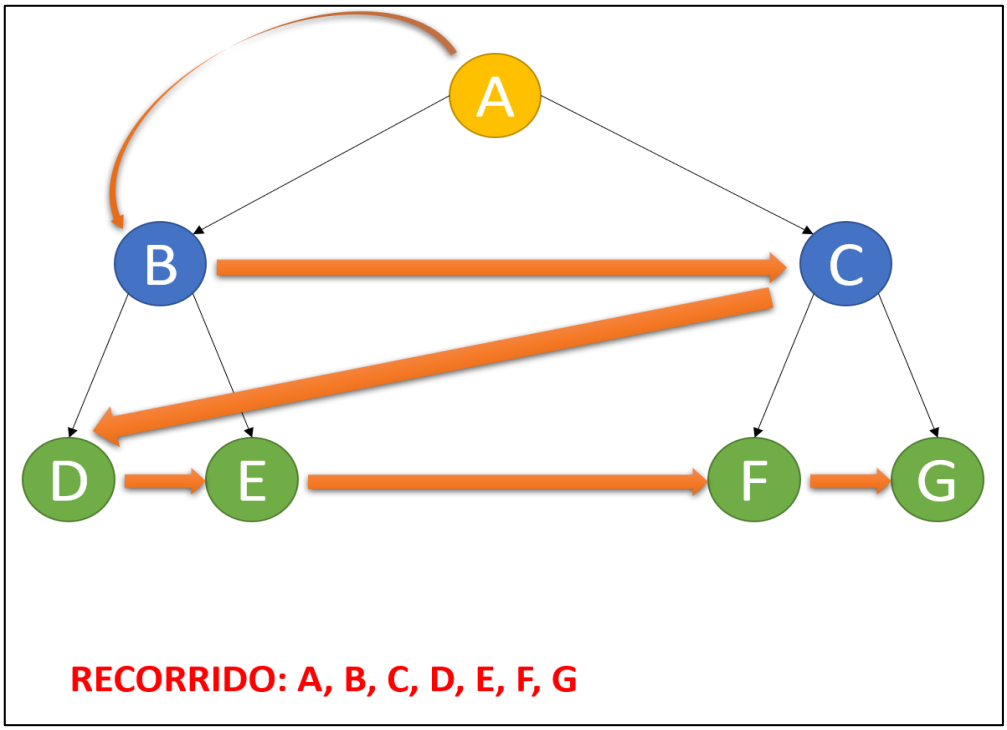


Figura 2.12. Orden del recorrido del árbol por niveles

2.5 Conjuntos independientes

En teoría de grafos, un conjunto independiente o estable es un conjunto de vértices en un grafo tal que ninguno de sus vértices es adyacente a otro.

Es decir, un conjunto V de vértices tal que para ningún par de ellos existe una arista que los conecten. En otras palabras, cada arista en el grafo contiene a lo más un vértice en V.

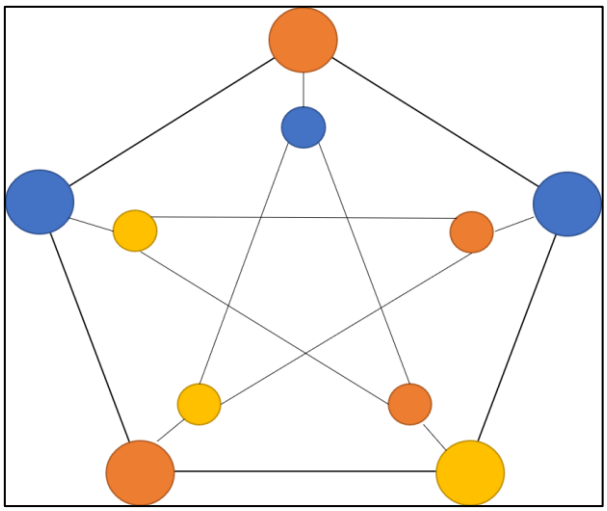


Figura 2.13. Representación de conjuntos independientes

Al conjunto de todos los conjuntos independientes de *G* (figura 2.13), se le denota por *I(G)*, mientras que a la cardinalidad de este conjunto se le denotará por *NI(G).*

Considerando que las líneas en *G* tienen igual probabilidad de falla, tomamos una de estas líneas $c \in E$, cuando *c* falla se forma una nueva red de comunicaciones *G*'= (*V, E*'), con $E = E' \cup \{c\}$. Se define la redundancia de la línea de comunicación *c* en *G* (la red original que contiene a *c*) como la probabilidad condicional: P*c*/*G* = *NI(G)*/ *NI(G')*, donde *Prob.* denota la función de probabilidad.

Calcular el conjunto independiente máximo (el que tenga el mayor número de elementos) es un problema NP-completo a partir de grafos planares de grado mayor o igual a 3, mientras que contar el número de conjuntos independientes de un grafo es un problema #P-completo a partir de grafos de grado mayor o igual a 3 [11].

2.6 Complejidad computacional

Dado un grafo *G*, un subconjunto de vértices $H \subset V(G)$ se llama conjunto independiente si no hay dos vertices en *H* que sean adyacentes.

El número de independencia de *G* es el tamaño de un conjunto independiente más grande, y se denota por $\alpha(G)$[6].

El conjunto independiente máximo corresponde al mayor conjunto independiente definible sobre un grafo (árbol) dado. El problema de encontrar un conjunto con estas características se llama problema del máximo conjunto independiente y es NP-completo, por lo cual es poco probable que exista un algoritmo que lo resuelva eficientemente.

El problema es decidir si un grafo (árbol) dado tiene un conjunto independiente de un tamaño particular; ese es el Problema de conjunto independiente, el cual es computacionalmente equivalente a decidir si un grafo (árbol) tiene un “clique” de un tamaño en particular. Esto porque, si un grafo (árbol) tiene un conjunto independiente de tamaño *k*.

Un algoritmo se puede modificar fácilmente para encontrar los conjuntos independientes de interés. En este proyecto se hará uso del algoritmo de Fibonacci (figura 2.14) para realizar el cálculo de conjuntos independientes de un árbol de tamaño n.

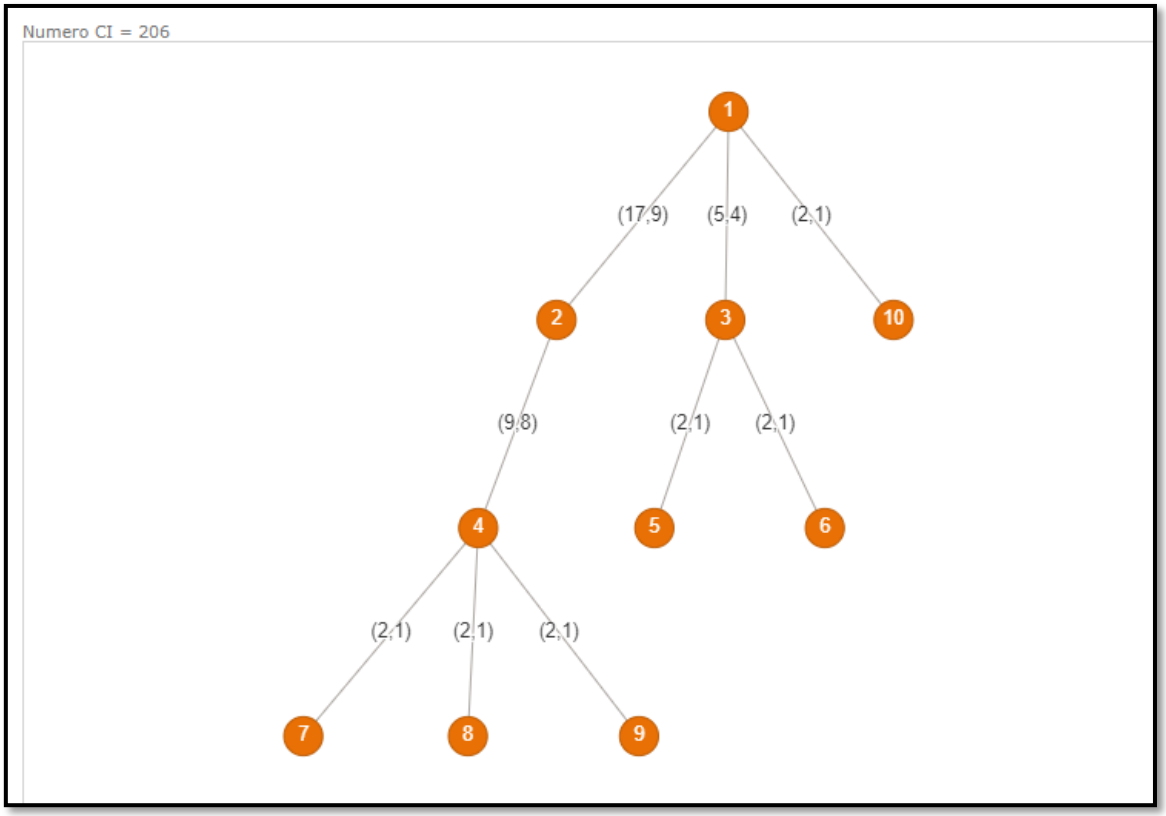


Figura 2.14. Representación de cálculo de conjuntos independientes

Primeramente, estos conjuntos se computan para las hojas de *T*, luego para los predecesores inmediatos de las hojas y se continua de esa manera hasta alcanzar la raíz.

Para guiar este proceso, definimos una reducción de un árbol, este es un ordenamiento de los nodos del árbol donde las hojas aparecen primero comenzando con valor (1,1) en coordenadas de la forma (*x,y*), luego los predecesores inmediatos de las hojas, y así sucesivamente.

Si la rama no tiene intersección con otra rama, la operación para llegar al nodo predecesor es la siguiente:

$$(x, y) = ((x + y), x)$$

De lo contrario si se encuentra una intersección con otra rama la operación que se tiene que realizar es la siguiente:

$$(x, y), (m, n) = (((x * m) + (y * n)), (x * m))$$

Estas operaciones se realizarán hasta llegar al nodo raíz y el total de los conjuntos independientes que tendrá el árbol se obtendrá sumando los valores de la coordenada final.

$$(x, y) = x + y$$

Una simple variación de recorrido por amplitud del árbol se puede usar para producir su reducción en tiempo lineal y tiene complejidad de tiempo en *O*(n).

Si se quiere clasificar los problemas, se realizará un estudio de complejidad de lenguajes decidibles de acuerdo a la cantidad de recursos necesarios para su computación.

Las clases de lenguajes son conjuntos arbitrarios de lenguajes que no necesariamente comparten propiedades computación. Para que las clases de lenguajes sean clases de complejidad se deben definir los predicados π (*) en base a las medidas de complejidad dinámicas.

Para que estas clases de complejidad tengan sentido, es decir, agrupen problemas con características computacionales parecidas; se restringen a funciones t(*x*) denominadas tiempos construibles:

- Funciones logarítmicas
- Funciones polinomiales
- Funciones exponenciales
- Funciones factoriales

En general se está de acuerdo que la clase P representan a los problemas tratables. La mayoría de los problemas de esta clase tienen algoritmos con implementación razonable, a pesar de el algoritmo polinomial para ese problema no sea el más conveniente de usar.

Sin embargo, existen ciertos problemas en P que no son tratables, debido a que:

- El grado *k* del polinomio es un número grande
- Las constantes ocultan en la notación *O*(*) son altas

- La demostración de pertenencia a *P* no es constructiva, es decir; no presenta un algoritmo.

Pero estos son casos extremos, y se considera a *P* una razonable aproximación al concepto de tratabilidad.

Otra característica de la clase *P* es el hecho de que es robusta con respecto al modelo de computación. Esto significa que la mayoría de los cambios de los modelos y cambios de lenguajes conservan inalterable la clase.

Es posible demostrar que un problema pertenece a *P* mediante dos formas:

- Mostrando un algoritmo polinomial
- Usando una reducción a otro problema que ya se sabe que está en P

Identificando la clase *P* con el concepto de tratabilidad, se estaría en las siguientes condiciones, con algunos problemas probadamente tratables y otros probadamente intratables. En forma análoga a *P*, se puede definir su contraparte no determinística *NP*.

Puede ser que todos los problemas *NP* tengan solución determinística polinomial, o puede ser que no sea equivalente el tiempo determinístico con el tiempo no determinístico. También se tiene un conjunto de problemas cuya tratabilidad o no se desconoce:

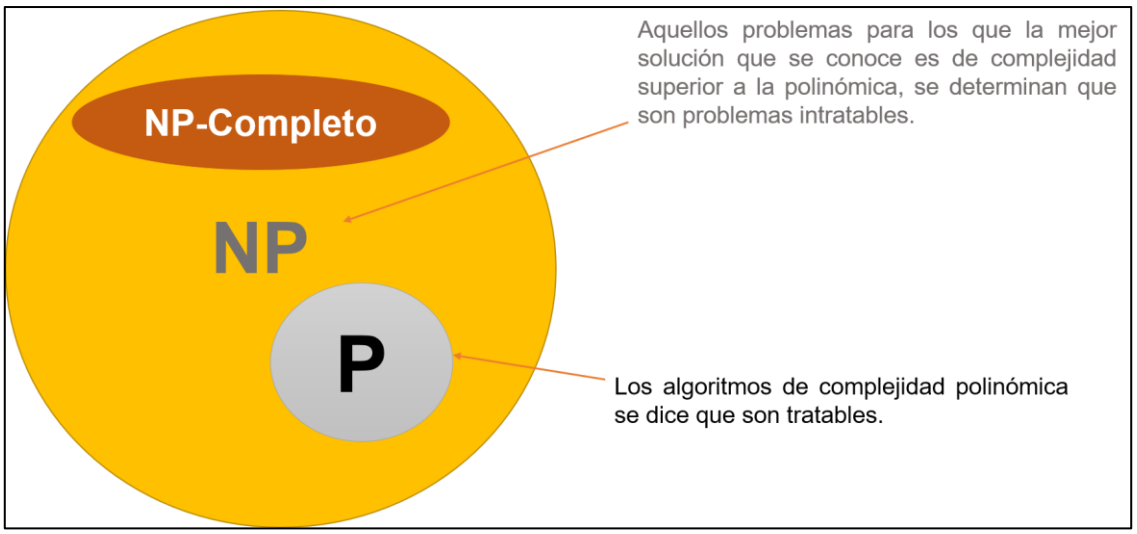


Figura 2.15. Representación de complejidad de algoritmos

Esta brecha es inaceptablemente muy grande, e impide clasificar a importantes problemas como tratable e intratable.

2.7 Herramientas de desarrollo

2.7.1 Servidor

Un servidor web o Servidor *HTTP* (figura 3.16) es una pieza de software de comunicaciones que intermedia entre el servidor en el que están alojados los datos solicitados y el computador del cliente.

Permitiendo conexiones bidireccionales o unidireccionales, síncronas o asíncronas, con cualquier aplicación del cliente, incluso con los navegadores que traducen un código traducible (renderizable) a una página web determinada.

O sea, se trata de programas que median entre el usuario de Internet y el servidor en donde está la información que solicita.

Figura 2.16. Servidor http

Los servidores web son programas de uso cotidiano en Internet, que emplean para comunicarse diversos protocolos de datos, siendo el más común y de alguna manera estándar el *HTTP* (*HyperText Transfer Protocol*).

Es posible también usar el término para referirse al computador en el que están guardados los archivos que componen un sitio web, junto al software necesario para cumplir con la conexión de datos web.

Se necesita de un servidor para publicar un sitio web, y también para tener acceso a los datos que componen una página web cualquiera.

Este proceso ocurre de la siguiente manera:

- El usuario introduce una dirección web (*URL*) en su navegador y éste envía una solicitud al servidor web.
- El servidor web (*software*) busca los archivos pertinentes ya sea en el propio servidor (*hardware*) o en un servicio de hosting en el que están siempre disponibles y en línea.
- Los archivos entonces son procesados según lo solicitado y enviados de acuerdo al protocolo de transferencia, es decir, conforme a un conjunto de reglas que regulan la comunicación entre los computadores.
- El navegador recibe los archivos y ensambla el contenido de la página web que se muestra al usuario.

Algunos de los servidores web más empleados son los siguientes:

- **Nginx (2004).** Un servidor web y Proxy desarrollado por la empresa homónima.
- **Apache (1995)**. Es un servidor web *HTTP* de código abierto, que sirve para computadores Unix, Windows y Macintosh, desarrollado y mantenido por una comunidad de usuarios que conforman la Apache Software Foundation.
- **Internet Information Services o IIS (1993**). Servidor web y conjunto de servicios diseñados para Microsoft Windows que fue originalmente incluido en su versión NT.

- **Cherokee (2001).** Es un servidor web multiplataforma escrito en lenguaje C, disponible bajo Licencia Pública General de GNU, de software libre.

- **Tomcat (1999).** Una distribución de Apache conocida también como Jakarta Tomcat, opera bajo el principio de los servlets (Java).

2.7.2 PHP

La Programación del lado del servidor es una tecnología que consiste en el procesamiento de una petición de un usuario mediante la interpretación de un script en el servidor web para generar páginas HTML dinámicamente como respuesta.

Todo lo que suceda dentro del servidor es llamado procesamiento del lado del servidor. Cuando la aplicación necesita interactuar con el servidor (por ejemplo, para cargar o guardar datos), ésta realiza una petición del lado del cliente (*client-side request*) desde el navegador, a través de la red usando invocaciones remotas a métodos (*remote procedure call, RPC*).

Mientras se está procesando una llamada RPC, tu servidor está ejecutando código del lado del servidor. La utilización de las diferentes aplicaciones o servicios de Internet se lleva a cabo respondiendo al llamado modelo cliente-servidor.

PHP es el acrónimo de *Hipertext Preprocesor*. Es un lenguaje de programación del lado del servidor gratuito e independiente de plataforma, rápido, con una gran librería de funciones y mucha documentación.

Fue creado originalmente en 1994 por Rasmus Lerdorf, pero como PHP está desarrollado en política de código abierto, a lo largo de su historia ha tenido muchas contribuciones de otros desarrolladores.

El cliente solamente recibe una página con el código HTML resultante de la ejecución de la PHP. Como la página resultante contiene únicamente código HTML, es compatible con todos los navegadores.

2.7.3 Vis.js

Existe hoy en día una revolución con las librerías de JavaScript que nos facilitan el desarrollo de nuestros proyectos.

Vis.js (figura 2.17) es una librería de visualización para proyectos Web que ha sido diseñada para que sea fácil de utilizar y gestionar grandes cantidades de datos dinámicamente, además de posibilitar la interacción del usuario con dichos datos.

Figura 3.17: Visualizador de grafos

Vis.js nos permite realizar distintos tipos de gráficos:

- **Redes**: conexiones y relaciones entre diferentes nodos.
- **Líneas temporales**: ver qué cambios ha habido a lo largo del tiempo en un determinado contexto.
- **Gráficas 2D**: gráficas en dos dimensiones, con ejes '*x*' e '*y*'.
- **Gráficas 3D**: gráficas tridimensionales para representar 3 coordenadas.

Se ha realizado el uso de la herramienta en estos meses y obtenidos resultados satisfactorios.

III. Análisis y diseño del algoritmo de C.I.

El algoritmo desarrollado se basa en la sucesión de Fibonacci lo que permite realizar el cálculo del número de conjuntos independientes de forma eficiente considerando estructuras de datos jerárquicas (Árboles).

3.1 Fibonacci

La sucesión o serie de Fibonacci hace referencia a la secuencia ordenada de los números descrita por Leonardo de Pisa, matemático italiano del siglo XIII:

0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, 987, 1597...

A cada uno de los elementos de la serie se le conoce con el nombre de número de Fibonacci.

Se proporcionan los dos primeros términos: $a_0 = 0\ y\ a_1 = 1$. Los siguientes se calculan con la siguiente formula:

$$a_{n+1} = a_{n-1} + a_n\ , n \geq 1$$

La sucesión de Fibonacci es creciente y no esta acotada (superiormente). Esto implica que la sucesión es divergente (no convergente), es decir, no tiene límite. Por lo tanto, la sucesión crece indefinidamente.

La sucesión de Fibonacci está presente prácticamente en todas las cosas del universo, tiene toda clase de aplicaciones en matemáticas, computación y juegos, y que aparece en los más diversos elementos biológicos.

Algunos ejemplos podrían ser como la disposición de las ramas de los árboles, las semillas de las flores, otros más sorprendentes puede ser que también se cumple en los huracanes e incluso hasta en los brazos en espiral de las galaxias, desde donde obtenemos la idea de la espiral de Fibonacci.

Un espiral de Fibonacci es una serie de cuartos de círculo conectados que se pueden dibujar dentro de una serie de cuadros regulados por números de Fibonacci para todas las dimensiones.

Entre sí, los cuadrados encajan a la perfección (Figura 4.1) como consecuencia de la naturaleza misma de la sucesión, en donde cualquier cifra es igual a la suma de las dos anteriores. El espiral o rectángulo resultante es conocido como el espiral dorado y el rectángulo de oro.

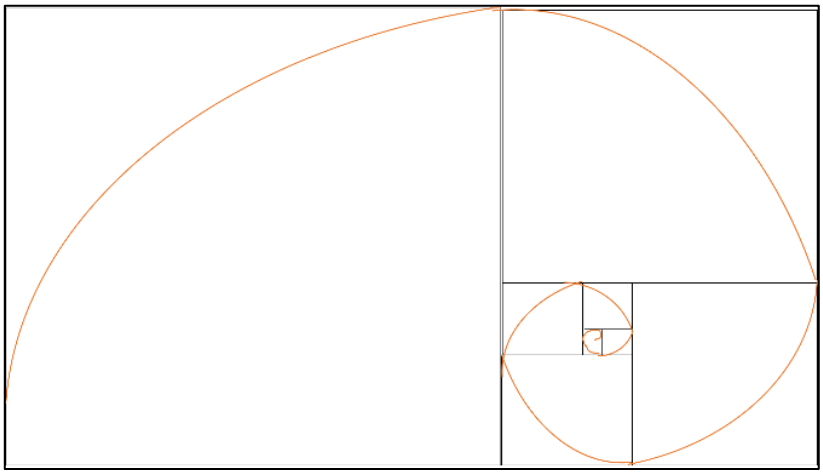


Figura 3.1. Espiral de Fibonacci

Existen diferentes formas para calcular los números de Fibonacci:

1. Partiendo de los números 0 y 1, los números de Fibonacci quedan definidos por la función:

   $f_n = f_{n-1} + f_{n-2}$

   $f_0 = 0$

   $f_1 = 1$

   $f_2 = f_1 + f_0 = 1$

   $f_3 = f_2 + f_1 = 2$

   . . .

2. Función generadora: Una función generadora para una sucesión cualquiera $a_0, a_1, a_2, \ldots$ es la función $f(x) = a_0, a_1 x, a_2 x^2 + \cdots$, es decir, una serie formal de potencias donde cada coeficiente es un elemento de la sucesión. Los números de Fibonacci tienen la función generadora:

$$f(x) = \frac{x}{1 - x - x^2}$$

3. Formula explícita: Esta manera de calcular los números de Fibonacci utiliza la expresión del número áureo:

$$f(x) = \frac{\left(\frac{1+\sqrt{5}}{2}\right)^n - \left(1 - \frac{1+\sqrt{5}}{2}\right)^n}{\sqrt{5}} = \frac{\left(\frac{1+\sqrt{5}}{2}\right)^n - \left(\frac{1-\sqrt{5}}{2}\right)^n}{\sqrt{5}}$$

Cada uno de los números de Fibonacci se acerca mucho a la llamada proporción áurea, proporción dorada o número de oro (aproximadamente 1.618034). Cuanto mayor es el par de números de Fibonacci, más cerca de la proporción dorada.

3.2 Número áureo

El número de oro es un concepto matemático y estético, sobre el cual, en 1909, Mark Barr propuso representar con la letra griega ϕ, o “phi”, este número se le conoce como proporción dorada, divina proporción, número áureo, etc.

El número áureo (figura 4.2), es el valor numérico que guardan entre si dos segmentos de recta *a* y *b* (*a* más largo que *b*): la longitud total es al segmento *a*, como a es al segmento *b*.

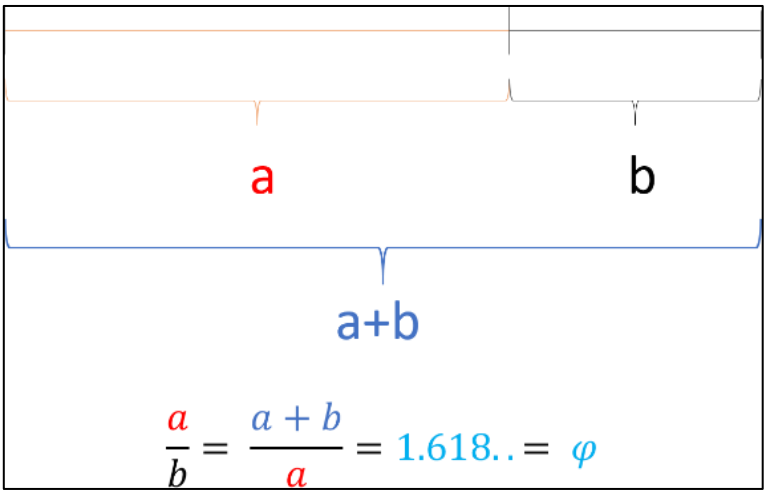


Figura 3.2. Número áureo

La razón o cociente entre un término de Fibonacci y el inmediatamente anterior varía continuamente, pero se estabiliza en el número áureo:

$$\lim_{n\to\infty} \frac{f_{n+1}}{f_n} = \phi \approx 1.6180339887$$

3.3 Si el grafo es una cadena

Sea un grafo $G=(V,E)$, con $\Delta(G)=2$ y siendo G un grafo cadena con n nodos (también llamado un camino lineal simple), por tanto $|V|=n$ y $|E|=m= n-1$.

Ordenemos los nodos en G como $V=\{v_1,v_2,\ldots,v_n\}$ de tal forma que se cumpla que $E=\{c_1,\ldots,c_{n-1}\}=\{\{v_1,v_2\}, \{v_2,v_3\}, \ldots, \{v_{n-2},v_{n-1}\}, \{v_{n-1},v_n\}\}$, es decir, se cumple que $|v(c_i) \cap v(c_{i+1})|=1$, $i=1,\ldots,n-2$. Formamos a partir de G, la familia de los subgrafos inducidos G_i, $i=\{1,\ldots,n\}$ siendo G_i el subgrafo inducido de G por exactamente los primeros i nodos de G y de acuerdo al orden en los nodos y aristas mencionado anteriormente.

La técnica básica que nos permitirá ir contando el número de conjuntos independientes $NI(G_i)$ sobre los grafos inducidos G_i de G, es una "estrategia incremental" que se basa en asociar un par de valores: $(\alpha i, \beta i)$, a cada nodo $v_i \in$

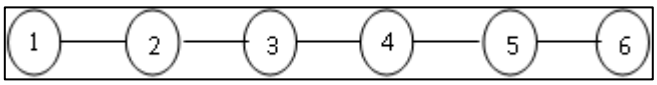
V, i={1,…,n} conforme se va visitando cada nodo del grafo en base a un recorrido a lo profundo.

El recorrido inicia en uno de los extremos de la cadena (suponiendo que este es el nodo v_1) visitando a su nodo adyacente de forma lineal y hasta llegar al otro extremo de la cadena (hasta llegar al nodo v_n).

El primer par de valores (α_1, β_1) inicia con los valores: (1,1). La relación de recurrencia a aplicar al contar el número de conjuntos independientes sobre la cadena *G*, considera que se conoce el valor (α_i, β_i) asociado al subgrafo inducido G_i de *G* y tal que $NI(G_i)=\alpha_i+\beta_i$.

El subgrafo G_i se "extiende" al adicionarle el nodo v_{i+1}, creándose el subgrafo inducido G_{i+1} de *G*, por lo que se une v_{i+1} con una sola arista $c_i \in E$ y que va de v_i a v_{i+1}. Y el nuevo par de valores $(\alpha_{i+1}, \beta_{i+1})$ se construye en base a (αi, βi) al aplicar la recurrencia que genera a los números de Fibonacci: $\alpha_{i+1} = \alpha_i + \beta_i$ y $\beta_{i+1} = \alpha_i$.

Sea el grafo *G=(V,E)* donde *V={1,2,…,6}* tiene 6 vértices y el conjunto de aristas *E={c1,…,c5}* tiene 5 aristas. El conteo de *NI(G)* se realizará aplicando la recurrencia (2) para ir formando los pares de números: $(\alpha i, \beta i)$, i=1,…,6, y de tal forma que *NI(G)* = α6+ β6 = 13 + 8 = 21, según se muestra en la figura 4.3.1.1.



(α1,β1) → (α2,β2) → (α3,β3) → (α4,β4) → (α5,β5) → (α6, β6)

(1,1) → (2,1) → (3,2) → (5,3) → (8,5) → (13,8)

Figura 3.3. Cómputo del Número de Conjuntos Independientes para una cadena.

Cada número de Fibonacci se puede acotar superior e inferiormente por $\varphi i-2 \geq Fi \geq \varphi i-1,\ i \geq 1$, donde $\varphi = 1/2 \cdot (1 + \sqrt{5})$ es conocido como 'el cociente de oro'. Y cualquier número de Fibonacci Fi puede calcularse mediante la ecuación $Fi = Entero\_mas\_Cercano\_a(\varphi i/\sqrt{5})$.

Por ejemplo, aplicando la serie Fibonacci sobre el grafo de la figura 2, se obtienen los valores: $(\alpha i, \beta i)$, $i = 1, ..., 6$: (1, 1) → (2, 1) → (3, 2) → (5, 3) → (8, 5)

→ (13, 8), y esta última serie coincide con los números de Fibonacci: (F2, F1) → (F3, F2) → (F4, F3) → (F5, F4) → (F6, F5) → (F7, F6).

Por lo tanto, se infiere que $(\alpha i, \beta i) = (Fi+1, Fi)$ y entonces $NI(Gi) = \alpha i + \beta i = Fi+1 + Fi = Fi+2, i = \{0, ...,n\}$ Es decir para n = 6, se tiene que $NI(G)= NI(G6)= F7 + F6 = F8 = 21.$

Dado un grafo cadena con n vértices, la complejidad en tiempo para calcular el número de Fibonacci, dado el índice n+2, puede realizarse por un proceso determinista en tiempo logarítmico. O bien, si se van construyendo las parejas $(\alpha i, \beta i), i = 1, ...,n$ conforme se visitan los nodos del grafo, se calculará todo la serie con una complejidad lineal en tiempo, es decir, de orden O(n).

3.4 Si el grafo es un cicló simple

Como es conocido, calcular $NI(G)$ cuando G es un grafo de grado 3 se considera que es un problema #P-completo [2], para tal clase de grafos, si se descartan los grafos acíclicos, ya que para tal subclase $NI(G)$ puede calcularse en tiempo lineal, quedarán entonces solo grafos con ciclos. Se presenta en esta sección, que aún para grafos con ciclos, existe una subclase tratable para calcular $NI(G)$.

Sea $G = (V,E)$ un ciclo simple, esto es, todos los nodos en V tienen grado 2, $|V| = |\{v1, \ldots, vn\}| = n = m = |E|$. Ordenemos las aristas como: $E = \{c1,\ldots,cm\}$ de tal forma que $|u(ci) \cap u(ci+1)| = 1$, y además $v1 = vm$.

Con el fin de calcular $NI(G)$, el ciclo G se descompone de la forma: $G = G' \cup \{cm\}$ donde G' será entonces una cadena con m-1 aristas y la arista $cm = (vm, v1)$ es el arco que si es agregado a G' nos forma el ciclo simple: $v1, c1, v2, c2, ..., vm-1, cm-1, vm, cm, v1$.

Se puede observar que $v(G')=v(G)=V$ y que cada conjunto independiente de G es un conjunto independiente de G', esto es, $I(G) \subseteq I(G')$ dado que G tiene una restricción más (una arista más) que G' por lo que cada conjunto independiente de G' el cual tiene como elementos los nodos $v1$ y vm no puede ser un conjunto independiente para G.

Por lo que al construir $I(G)$ a partir de $I(G')$ se deberán eliminar aquellos conjuntos independientes que contienen a los nodos $v1$ y vm o, en otras palabras, un conjunto independiente S de G' será un conjunto independiente de G si y solo si ($v1 \notin s$ o $vm \notin s$).

Aplicando el procedimiento de complejidad lineal basado en la ecuación de recurrencia (2), podemos calcular $NI(G')$, dado que G' es una cadena con m nodos y de acuerdo al teorema 1, $NI(G') = Fm+2$. Por otro lado, $NI(G') \geq NI(G) = NI(G' \cup \{cm\})$, de hecho, se tiene que: $NI(G) = NI(G') - |\{s \in I(G') : v1 \in s \wedge vm \in s\}|$.

Ahora, con el fin de calcular $|\{s \in I(G') : v1 \in s \wedge vm \in s\}|$, se puede fijar sobre el conjunto $I(G')$ aquellos conjuntos independientes donde el nodo $v1$ aparece, contar este último conjunto se haría con iniciar la serie $(\alpha i, \beta i),\ i=2,\ldots,m$ con el par: $(\alpha 1, \beta 1) = (0,1)$.

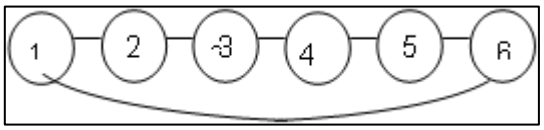
Aplicando (2) obtenemos los demás pares de la secuencia: $(\alpha i, \beta i),\ i=2,\ldots,m$ y además, con el fin de considerar solo los conjuntos independientes los cuales tienen a vm como uno de sus elementos, se deberá modificar al último par de la secuencia $(\alpha m, \beta m)$ con el valor $(0, \beta m)$, de tal forma serie tiene la forma: $(\alpha 1, \beta 1) = (0,1) \rightarrow (\alpha 2, \beta 2) = (1,0) \rightarrow (\alpha 3, \beta 3) = (1,1) \ldots \rightarrow (\alpha m, \beta m) \rightarrow (0, \beta m)$ y donde el valor βm nos proporciona el valor de $|\{s \in I(G') : v1 \in s \wedge vm \in s\}|$.

La serie anterior se corresponde con la siguiente serie que considera a los números de Fibonacci: $(F0,F1) \rightarrow (F1,F0) \rightarrow (F2,F1) \rightarrow \ldots \rightarrow (Fm-1,Fm-2)$. Y por tanto, $|\{s \in I(G') : v1 \in s \wedge vm \in s\}| = Fm-2$.

Entonces, contar el número de conjuntos independientes *NI(G)* para un ciclo *G* de *m* vértices, se calcula en base a la cadena *G'* de *m* vértices que comprende, quedando la fórmula, como:

$$NI(G)=NI(G') - |\{s \in I(G') : v1 \in s \wedge vm \in s) | = Fm+2 - Fm-2$$

Se considera al conjunto de aristas $E = \{ci\}6i=1=\{(v1, v2), (v2, v3), (v3,,v4), (v4, v5), (v5, v6), (v6, v1)\}$ que nos conforma un ciclo simple $Gc=(V,E)$. Nótese que $|u(ci) \cap u(ci+1)| = 1, i=1,\ldots,6$. Sea $G'=(V,E')$ tal que $E=E' \cup \{c6\}$ esto es, el nuevo grafo *G'* es *Gc* pero sin considerar la arista *c6*.



$(\alpha1,\beta1) \rightarrow (\alpha2,\beta2) \rightarrow (\alpha3,\beta3) \rightarrow (\alpha4,\beta4) \rightarrow (\alpha5,\beta5) \rightarrow (\alpha6, \beta6)$

(1,1) --- (2,1) --- (3,2) ----- (5,3) ----- (8,5) --- (13,8)

(0,1) --- (1,0) --- (1,1) ----- (2,1) ----- (3,2) --- (5,3)

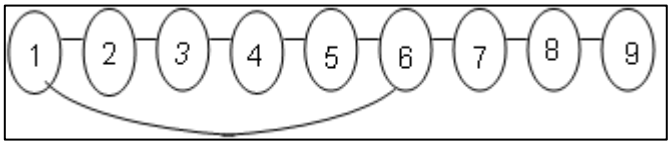
Figura 3.4. Cómputo del Número de Conjuntos Independientes en un ciclo.

Como *G´* es una cadena con 6 nodos, al aplicar (2) se obtiene la serie: *(αi, βi) i=1,…,6*, que inicia con el par (1,1) y termina con el par *(α6, β6)=(13,8)*, siendo *NI(G') = 13+8 = 21=Fm+2 = F8*. Mientras que calcular $|\{S \in I(G') : v1 \in S \wedge v6 \in S)|$ se obtiene al aplicar la recurrencia (2) a través de los 6 nodos pero iniciando con el par: $(0,1) \rightarrow (1,0) \rightarrow (1,1) \rightarrow (2,1) \rightarrow (3,2) \rightarrow (5,3)$, obteniéndose que sólo 3 conjuntos independientes de I(G') contienen al nodo 1 y al nodo 6, es decir, $|\{S \in I(G') : v1 \in S \wedge v6 \in S)| = 3 = F4$.

Finalmente, el par asociado al nodo 6 del ciclo original *Gc* corresponde a *(α6, β6)=(13,8) –(0,3)=(13,5)* y por tanto *NI(Gc) = 13 + 5 = 18*, que a su vez podría calcularse de acuerdo al teorema 2, como: *NI(Gc) = F8 - F4 = 21 – 3 =18*.

Podemos ahora considerar grafos que resultan de componer los dos casos anteriores, por ejemplo, considerar la unión de dos cadenas mediante un vértice común o la unión de dos ciclos, también mediante un vértice común, así como la unión de un ciclo y una cadena mediante un vértice común.

Si al ciclo del ejemplo 3.4, denotado por *Gc* se le adiciona una cadena de 3 vértices, denotada por *Gl*, ligada a partir de uno cualquiera de los nodos de *Gc*, entonces podemos reordenar el grafo total G y presentarlo como un ciclo unido a través de su último nodo con la cadena *Gl* (ver figura 3.5).



$(\alpha1,\beta1)\rightarrow(\alpha2,\beta2)\rightarrow(\alpha3,\beta3)\rightarrow(\alpha4,\beta4)\rightarrow(\alpha5,\beta5)\rightarrow(\alpha6,\beta6)$

$(1,1) \rightarrow (2,1) \rightarrow (3,2) \rightarrow (5,3) \rightarrow (8,5) \rightarrow (13,8)$

$(0,1) \rightarrow (1,0) \rightarrow (1,1) \rightarrow (2,1) \rightarrow (3,2) \rightarrow (5,3)$

$(\alpha6,\beta6) \rightarrow (\alpha7,\beta7) \rightarrow (\alpha8,\beta8) \rightarrow (\alpha9,\beta9)$

$(13,5) \rightarrow (18,13) \rightarrow (31,18) \rightarrow (49,31)$

3.5. Cómputo del Número de Conjuntos Independientes de un ciclo y una cadena

En este caso, se tiene que al último para asociado al vértice común de *Gc* con *Gl* (el grafo cadena) sería: *(α6, β6)Gc = (13,8) - (0,3) = (13,5)* que será el par asociado al nodo común v6 del ciclo y la cadena, por lo que a partir de este par se iniciaría el conteo para la serie asociada a la cadena *Gl*, generándose los pares: $(13,5) \rightarrow (18,13) \rightarrow (31,18) \rightarrow (49,31)$. Por lo que *NI(G) = 49 + 31 = 80*.

Este proceso de contar conjuntos independientes puede realizarse en tiempo lineal por procesos deterministas según se vaya visitando en base a una búsqueda a lo profundo, cada nodo del grafo. Con lo que estamos determinando nuevas clases polinomiales para el problema del conteo de conjuntos

independientes, caracterizados por la unión de subgrafos básicos (cadenas y ciclos) vía un vértice común entre cada subgrafo básico.

3.5 Pseudocódigo del algoritmo de CI

Para el diseño del algoritmo primero fue necesario entender cómo es que cotidianamente se cuentan los CI de un grafo. Esto se hace incrementalmente primero contando todos los CI de tamaño = 1, después los de tamaño = 2 y así consecutivamente hasta terminar.

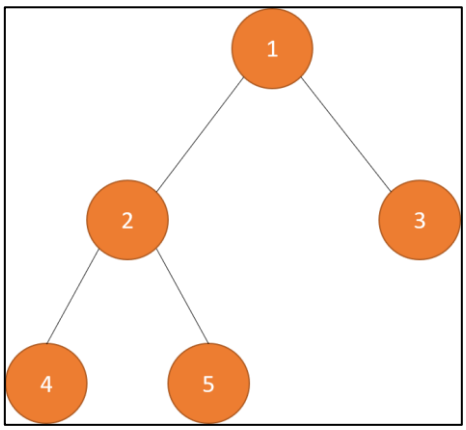


Figura 3.6. Árbol con 5 nodos y 4 aristas

Un ejemplo de los conjuntos independientes se puede mostrar usando el grafo de la figura 3.6 con el árbol presentado se pueden obtener los conjuntos independientes por tamaño o por número de elementos como se muestra:

Tamaño 1 = {1}, {2}, {3}, {4}, {5}, {Ø} cantidad = 6

Tamaño 2 = {1,4}, {1,5}, {2,3}, {3,4}, {3,5}, {4,5} cantidad = 6

Tamaño 3 = {1,4,5}, {3,4,5} cantidad = 2

En total se obtienen 13 más conjunto vacío se tienen en total 14 conjuntos independientes.

El código se diseñó mediante módulos en un archivo para solo realizar llamados en la página principal y hacer uso de las funciones al invocarlos. La función fib(n) calcula el total de conjuntos independientes sobre una estructura de tipo jerárquica (árboles) mediante un proceso iterativo partiendo de las ramas del árbol.

---

función fib(n)

si n <= 0 **entonces**

devuelve 0

i ← n -1

auxOne ← 0

auxTwo ← 1

(a,b) ← (auxTwo, auxOne)

(c,d) ← (auxOne, auxTwo)

mientras i > 0 **hacer**

si i es impar **entonces**

auxOne ← $(c^2 + d^2)$

auxTwo ← (d(b+a) + cb)

(a,b) ← (auxOne, auxTwo)

auxOne ← $(c^2 + d^2)$

auxTwo ← (d(2c+d))

(c,d) ← (auxOne, auxTwo)

i ← $\frac{i}{2}$

devuelve a+b

Aplicando el algoritmo anterior en el árbol de la figura 3.7. obtenemos el total de conjuntos independientes en tiempo polinomial.

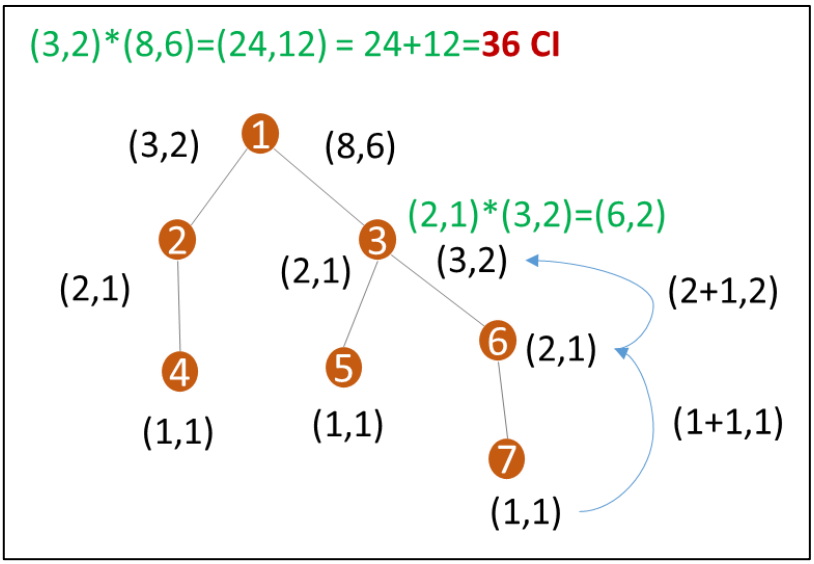


Figura 3.7. Árbol con 5 nodos y 4 aristas

La eficiencia espacial del algoritmo propuesto se obtiene gracias a la representación utilizando listas ligadas simples en forma invertida como se muestra en la figura 3.8. Cada rama es visitada almacenado cada valor (α,β) en el nodo, cuando un nodo llegan dos o más ramas, entonces se multiplican y se actualiza dicho valor (α,β).

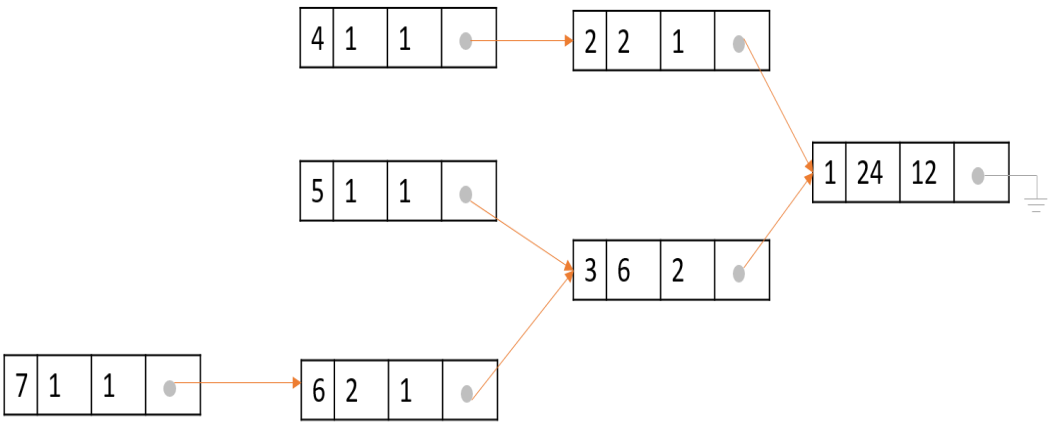


Figura 3.8. Estructura de datos utilizada para representar el árbol de la figura 3.7

3.6 Complejidad del algoritmo

Para el cálculo de la complejidad del algoritmo hay que analizar su comportamiento, un algoritmo se deduce utilizando las leyes de los exponentes como:

$$x^n = \begin{cases} x & si\ n = 1 \\ \left(x^{\frac{n}{2}}\right)^2 & si\ n\ es\ par \\ x * x^{n-1} & si\ n\ es\ impar \end{cases}$$

De esta manera se divisa el algoritmo de tipo “divide y vencerás” donde solo se requeriría hacer, aproximadamente, $\log_2(n)$ multiplicaciones matriciales. Sin embargo, no es necesario almacenar los 4 valores de cada matriz dado que cada una tiene la forma:

$$\begin{bmatrix} a & b \\ b & a+b \end{bmatrix}$$

De esta manera, cada matriz queda representada por los valores a y b, y su cuadrado se puede calcular como:

$$\begin{bmatrix} a & b \\ b & a+b \end{bmatrix}^2 = \begin{bmatrix} a^2 + b^2 & b(2a+b) \\ b(2a+b) & (a+b)^2 + b^2 \end{bmatrix}$$

Por lo que las características del algoritmo son de complejidad $\boldsymbol{O(n)}$, haciendo uso de un condicional if y un while.

A pesar de lo engorroso que parezca, este algoritmo permite reducir enormemente el número de operaciones que se necesitan para calcular números de Fibonacci muy grandes.

Debido a que el algoritmo propuesto se basa en el cálculo de la sucesión de Fibonacci, y al ser polinomial, el algoritmo propuesto también se mantiene de forma polinomial si la estructura a evaluar es un árbol, recordando además que cada rama se considera una cadena. Si cada cadena se calcula en tiempo lineal, entonces *n* cadenas (ramas de un árbol) se calculan en tiempo polinomial.

IV. Implementación y pruebas del sistema

La implementación del algoritmo se realizó en el lenguaje de programación PHP bajo un entorno web con el fin de ponerlo a disposición de cualquier persona interesada en el cálculo de conjuntos independientes en árboles.

4.1 Pantalla inicial del sistema

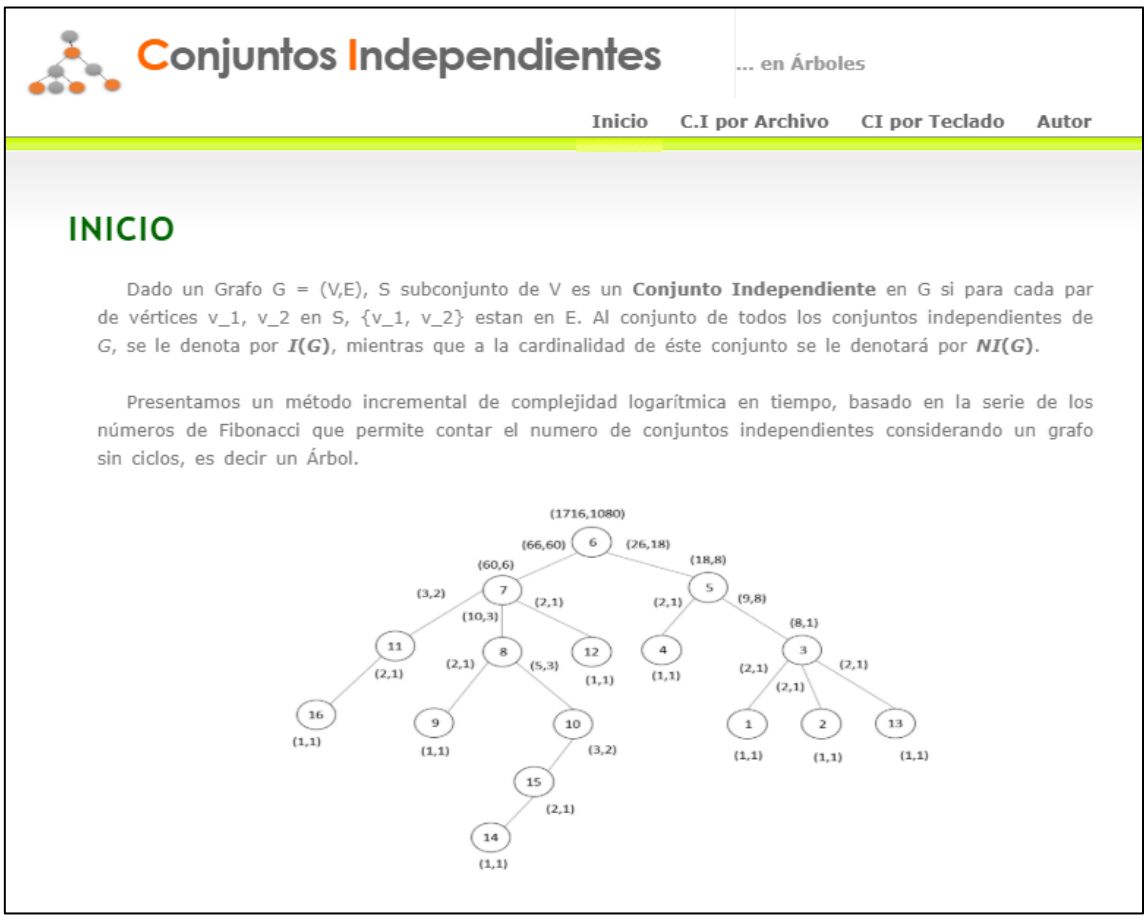


Figura 4.1: Pantalla principal del sistema Web.

En esta sección presentamos el funcionamiento del sistema, en la Figura 4.1 se muestra la pantalla inicial del sistema donde se definen los conjuntos independientes y se muestra un ejemplo de su conteo en la estructura de árboles, así como su implementación interna que se realiza por medio de listas ligadas.

4.2 Entrada de datos

En la Figura 4.2, se muestra la interfaz para dar de alta la configuración del árbol, para ello se le solicita al usuario que proporcione el archivo de texto que requiere contener el total de nodos y las aristas correspondientes. Se debe considerar que el archivo de texto tiene una estructura simple pero que debe coincidir con lo especificado en el sistema.

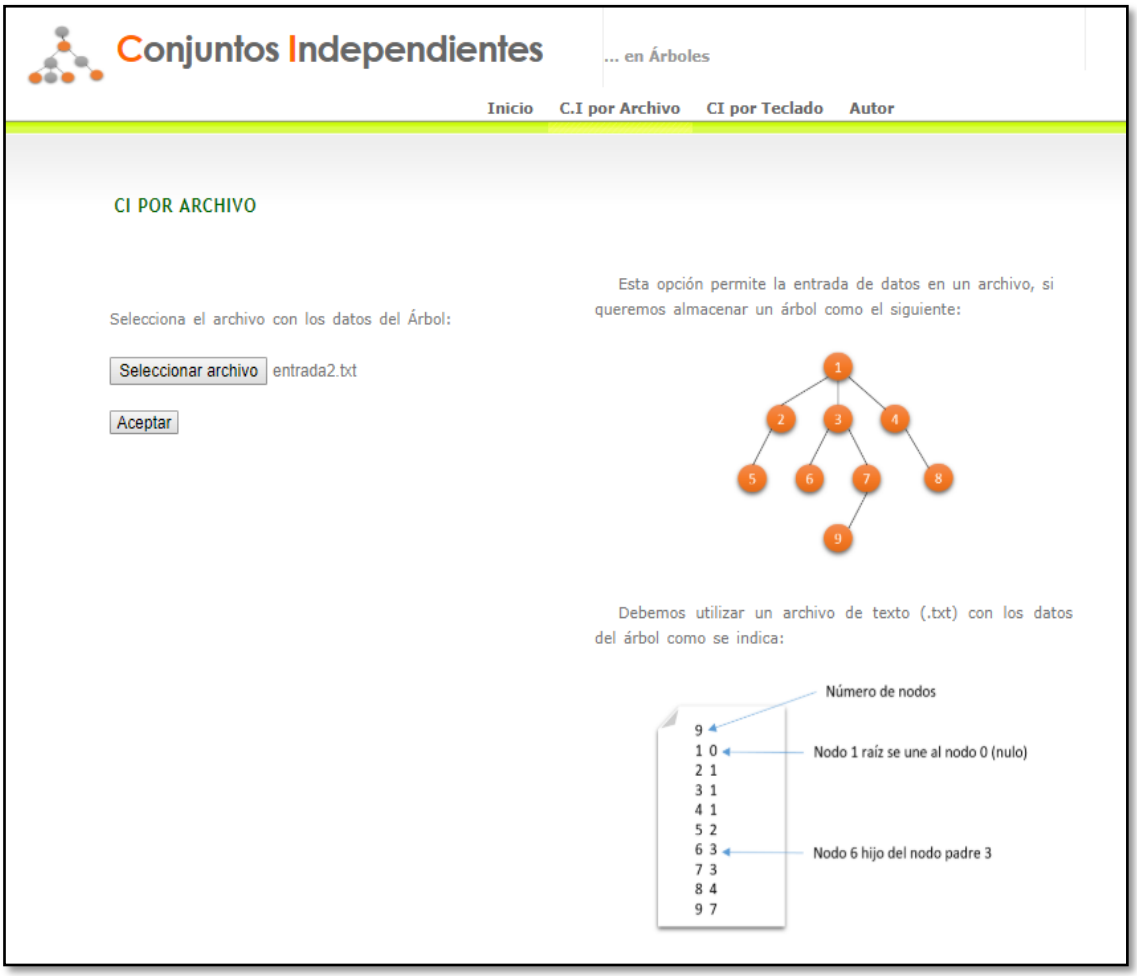


Figura 4.2: Sección para cargar un archivo.

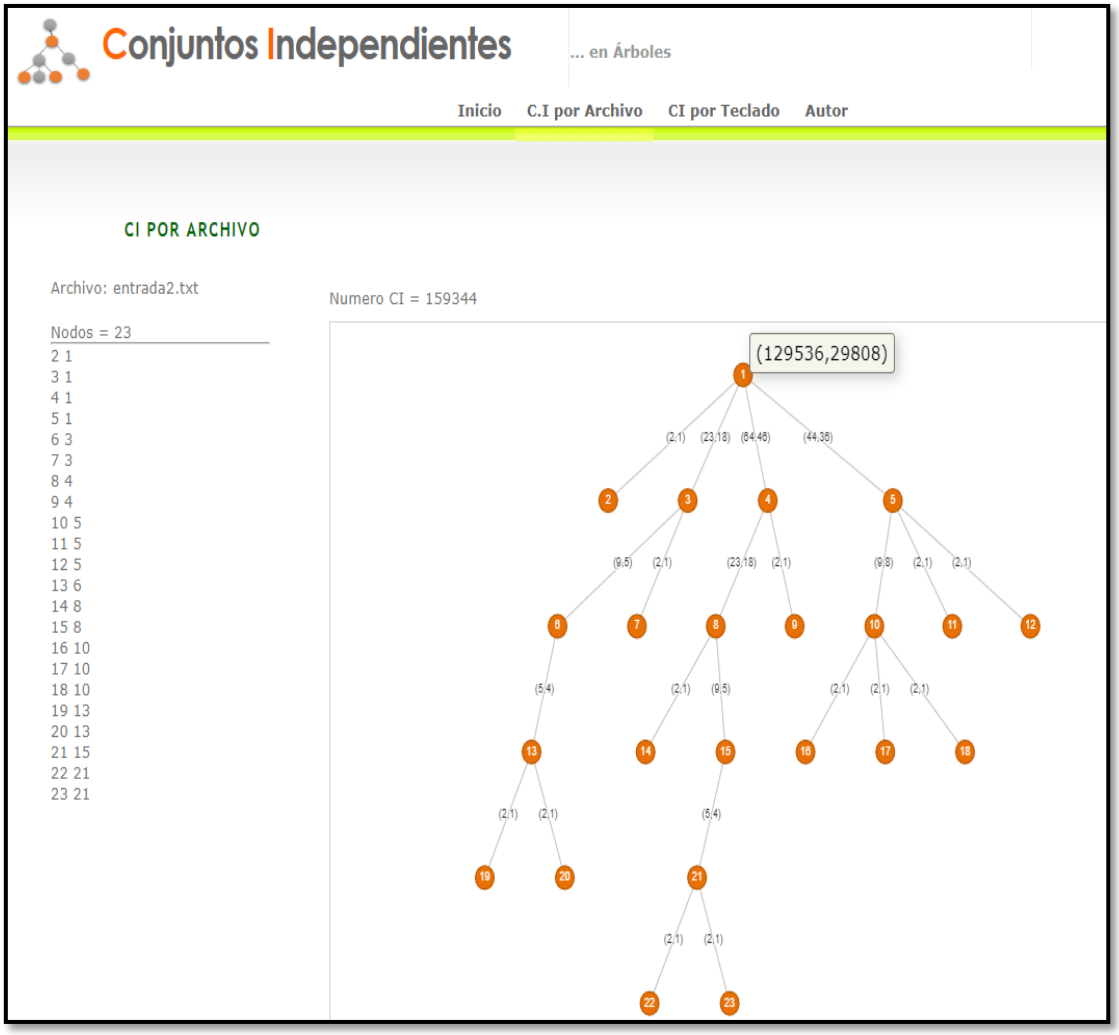


Figura 4.3: Ejemplo del conteo en el árbol considerando CI (α, β) por cada nodo.

En la Figura 4.3 se muestra la entrada proporcionada por el usuario en el panel izquierdo del frameset; en el panel derecho se visualiza el árbol generado mediante la herramienta Vis.js. Este árbol proporciona información importante paso a paso.

Para el cálculo de los conjuntos independientes de acuerdo a [7], considera el par *(αi, βi)*, que en el caso de los nodos tipo hoja inician el conteo con los valores *(α0, β0) = (1,1)*, esa información cuando sube al ancestro mediante el recorrido a lo profundo se recalcula mediante la recurrencia (1).

En la Figura 4.3 se observa; en las aristas de conexión entre los nodos tipo hoja y sus ancestros aparece el par *(α1, β1) = (2,1)*, que resulta de aplicar la

recurrencia (1), como se observa el cálculo en los nodos 2, 9, 11, 12, 16, 17, 18, 19, 20, 22 y 23.

$$\alpha i = \alpha i\text{-}1 + \beta i\text{-}1 \text{ y } \beta i = \alpha i\text{-}1 \qquad (1)$$

En el caso de que los ancestros actuales sólo tengan un descendiente, se aplica la recurrencia (1) hacia sus ancestros. Si, por el contrario, hacia el ancestro convergen varios nodos hijos, antes de retornar el valor de aplicar la recurrencia (1) a su ancestro, requiere aplicarse la recurrencia de convergencia de nodos hijos, recurrencia (2), y una vez hecho este cálculo, se aplica la recurrencia (1) hacia el nodo padre, como en el caso de los nodos 1, 3, 4, 5, 8, 10, 13 y 21.

$$\alpha i = \alpha i\text{-}1Hijo1 * \alpha i\text{-}1Hijo2 * \ldots * \alpha i\text{-}1HijoN \text{ y } \beta i = \beta i\text{-}1Hijo1 * \beta i\text{-}1Hijo2 * \ldots * \beta i\text{-}1HijoN \qquad (2)$$

También se visualiza en el nodo raíz del árbol una etiqueta amarilla, esta opción está disponible cada vez que se haga clic sobre el nodo y muestra el valor actual en el mismo, en este caso en la raíz se visualiza el par (129536, 29808) que es el cálculo resultante de aplicar la recurrencia (2) a los hijos de este nodo, Hijo1 *(αi, βi) = (2,1), Hijo1 (αi, βi) = (23,18), Hijo3 (αi, βi) = (64,46), Hijo4 (αi, βi) = (44,36)*, es decir *α=2*23*64*44=129536* y *β=1*18*46*36=29808*. El total de conjuntos independientes del árbol aparecen en la sección superior del papel derecho, que en este caso es *CI = 159344* que se calcula sumando el último par *(α, β) = (129536,29808)*.

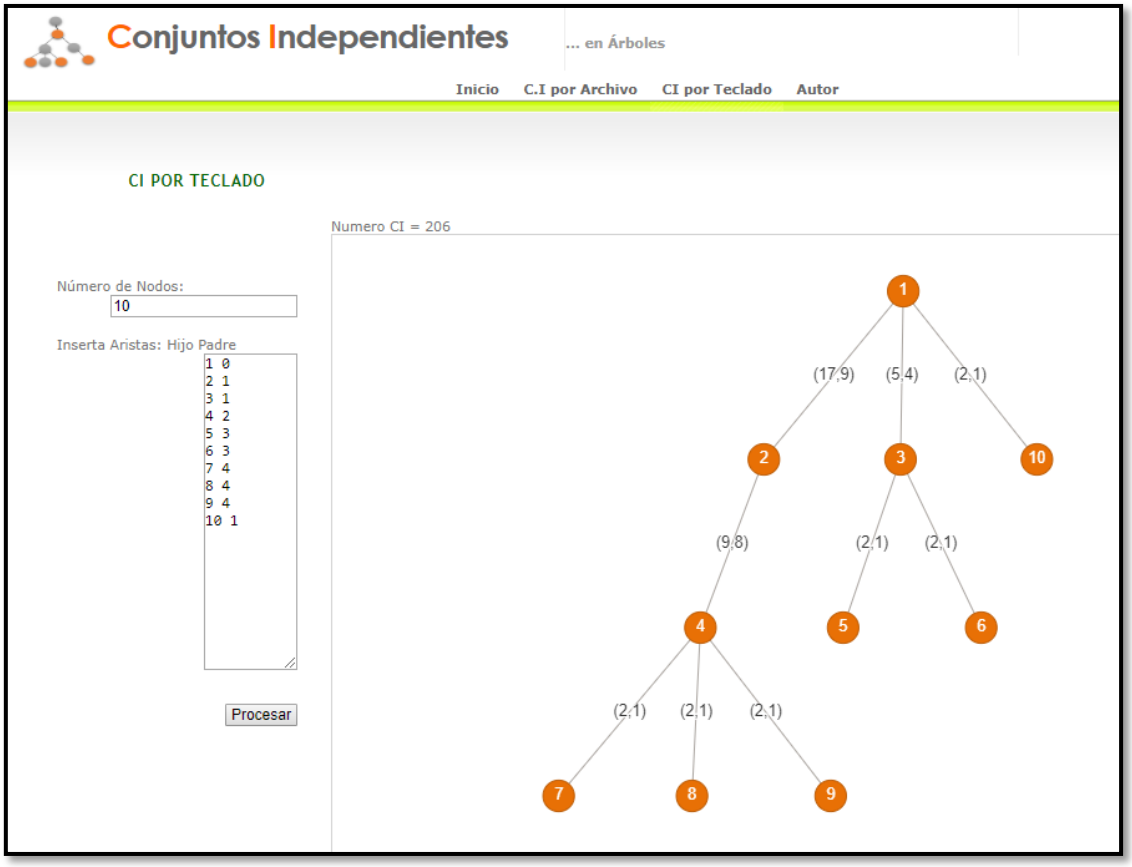


Figura 4.4: Sección para crear el árbol manualmente por teclado, se permite agregar nuevos nodos y realizar el cálculo en tiempo real.

Si lo que se prefiere es capturar la información del árbol directamente en el sistema, en la Figura 4.4 se muestra un ejemplo de ejecución de esa funcionalidad. Primero se proporciona el número de nodos y las aristas y se da clic sobre el botón procesar, de forma automática y en tiempo real generará el árbol resultante y calculará el CI de la estructura.

La ventaja de esta opción es que se pueden realizar cambios como agregar más nodos o cambiar las conexiones y así facilita el análisis de resultados. Si el usuario ya está satisfecho con la visualización de la información tiene también la opción de dar clic derecho sobre la figura y exportarla a un formato de imagen.

Conclusiones

Contar con sistemas que permitan realizar pruebas sobre conteo de estructuras siempre resulta de gran utilidad considerando que el cálculo manual es complejo y tedioso.

En este caso, este sistema brinda facilidades para generar cualquier tipo de árbol, despliega el cálculo de los CI acumulados por nodo, modificar la estructura original cuantas veces sea necesario ya sea modificando el archivo de texto o en tiempo real desde el teclado, así como conocer el CI total del árbol e incluso si se requiere exportar el árbol con los resultados se puede realizar.

Este sistema es una herramienta útil para la enseñanza de estructuras de datos sobre todo en materias de algoritmia como de combinatoria.

Desde la perspectiva de la investigación permite conocer el comportamiento de los conjuntos independientes por lo que esta área puede resultar más atractiva para futuros egresados o especialistas de otras áreas que requieren conocer el total de conjuntos independientes de una estructura para sus proyectos.

Como trabajo a futuro se plantea generar instancias de prueba para caracterizar las topologías de árboles que minimicen o maximicen el número de CI. El sistema al ser web puede ser montada a través de un servidor web para ser utilizado desde cualquier lugar con fines de investigación.

Bibliografía

[1] A. Pedersen and P. Vestergaard. Bounds On The Number Of Vertex Independent Sets In A Graph. Taiwanese Journal of Mathematics, Volume 10,2006.

[2] Aaron M. Tenenbaum, Yedidyah Langsam, Moshe J. Augenstein. Estructuras de datos en C. brooklyn college .México : Pearson Educación/Prentice-Hall, c1997.

[3] G. Valiente, Algorithms on Trees and Graphs. Springer- Verlag Berlin Heidelberg, 2002.

[4] H. Law, On the number of independent sets in a tree. Mathematical Institute, Oxford University,2010.

[5] H. Prodinger and R. Tichy. Fibonacci numbers of graphs. The Fibonacci Quaterly, volumen 9,1982.

[6] Kolman, Bernard; Hill, David R. Álgebra Lineal. México: PEARSON EDUCACIÓN. 2006.

[7] G. De Ita, and P. Bello. Recognizing Structural Patterns on Graphs for the Efficient Computation of #2SAT, Pattern Recognition, LNCS 7914, pp 274-283, Springer Verlag, 2013.

[8] W. Samotij. Counting independent sets in graphs. European Journal of Combinatorics, 48, pp.5-18, 2015.

[9] G. De Ita, and P. Bello. Efficient Counting of the Number of Independent Sets on Polygonal Trees, Pattern Recognition, LNCS 9703, pp 167-176, Springer Verlag, 2016.

[10] H. Prodinger and R. Tichy. Fibonacci numbers of graphs. The Fibonacci Quaterly, 9, pp. 16-21,1982.

[11] C. Greenhill, The Complexity of Counting Colourings and Independent Sets in Sparse Graphs and Hypergraphs, Computational Complexity, To appear.

[12] S.P. Vadhan, The Complexity of Counting, Undergraduate thesis, Hardvard University, 1995.

[13] D. Roth, On the Hardness of approximate reasoning, Artificial Intelligence, 82 (1996), pp. 273-302.

[14] M. Dyer, C. Greenhill, On Markov chains for independent sets, 1999

[15] Dyer M., Greenhill C., Some \#P-completeness Proofs for Coulorings and Independent Sets, Research Report Series, University of Leeds, 1997.

[16] Darwiche Adnan, On the Tractability of Counting Theory Models and its Application to Belief Revision and Truth Maintenance, Jour. of Applied Non-classical Logics, 11(1-2), (2001), 11-34.

[17] Valiant L.G., The complexity of enumeration and reliability problems, SIAM Journ. On Computing 8 (3), 1979, 410-421.

[18] R. Martínez, E. Quiroga. Estructuras de datos : referencia práctica con orientación a objetos. Alfaomega, México, 2002