Universidad de Buenos Aires
Facultad de Ciencias Exactas y Naturales
Departamento de Computación

Sobre la thinness de árboles

Tesis de Licenciatura en Ciencias de la Computación

Lucía Rabinowicz

Directora: Flavia Bonomo

Buenos Aires, 2019

SOBRE LA THINNESS DE ÁRBOLES

El concepto de thinness fue introducido por [Man+07]. Grafos con thinness acotada son una generalización de los grafos de intervalo, que son aquellos con thinness 1.

Para entender el concepto de thinness de un grafo vamos a imaginar a los nodos del mismo como fichitas de un juego donde cada fichita tendrá un conjunto de fichas vecinas. Sea una ficha f que representa el nodo x en el grafo, las fichas vecinas de f serán los nodos adyacentes a x en el grafo. Para calcular la thinness del grafo, asignaremos a cada fichita un número. Luego iremos armando pilas con las fichas de forma tal que siempre se cumpla que al momento de agregar una ficha nueva a una pila cualquiera, si esta ficha tiene fichas vecinas en alguna pila, todas las fichas que se encuentran por encima de la vecina, también son vecinas. El orden en el que se apilan las fichas esta dado por el número asociado a la misma. El valor de la thinness del grafo es la mínima cantidad de pilas posibles, para cualquier orden de las fichas.

En esta tesis vamos a calcular la thinness para grafo con un orden prefijado ($thin_<(G)$), es decir, la mínima cantidad de pilas para un orden fijo de las fichas.

En grafos de thinness acotada, muchos problemas NP-completos pueden ser resueltos en tiempo polinomial, como por ejemplo el problema del conjunto independiente ponderado máximo [Man+07] o el problema de coloreo capacitado con aplicación al problema de combinación de recorrido de recogida y entrega [BMO11].

Dado un orden para un grafo, la complejidad de hallar la thinness, para ese orden en particular, es $O(n^3)$ con n la cantidad de nodos del grafo. Esto es así ya que hallar la thinness del grafo es equivalente a encontrar el coloreo en un grafo auxiliar (asociado al orden) también de n vértices pero que resulta de co-comparabilidad [BE18]. Es un problema abierto la complejidad computacional de calcular la thinness de un grafo sin un orden prefijado.

En esta tesis se realiza un estudio sobre la thinness para orden fijo en grafos sin ciclos (árboles o bosques). Dado un orden con estructura recursiva para numerar los nodos del árbol (como inorder, preorder, postorder o BFS), calcularemos la thinness en tiempo lineal

en la cantidad de nodos introduciendo el nuevo concepto de Grupos de Conflicto.

Palabras claves: Grafos árbol, Thinness, Grupos de Conflicto.

A mis papás y a mi hermana.

Índice general

1. INTRODUCCIÓN

El concepto de thinness fue introducido por [Man+07]. Grafos con thinness acotada son una generalización de los grafos de intervalo, que son aquellos con thinness 1.

Para entender el concepto de thinness de un grafo vamos a imaginar a los nodos del mismo como fichitas de un juego donde cada fichita tendrá un conjunto de fichas vecinas. Sea una ficha f que representa el nodo x en el grafo, las fichas vecinas de f serán los nodos adyacentes a x en el grafo. Para calcular la thinness del grafo, asignaremos a cada fichita un número. Luego iremos armando pilas con las fichas de forma tal que siempre se cumpla que al momento de agregar una ficha nueva a una pila cualquiera, si esta ficha tiene fichas vecinas en alguna pila, todas las fichas que se encuentran por encima de la vecina, también son vecinas. El orden en el que se apilan las fichas esta dado por el número asociado a la misma. El valor de la thinness del grafo es la mínima cantidad de pilas posibles, para cualquier orden de las fichas.

En esta tesis vamos a calcular la thinness para grafo con un orden prefijado ($thin_<(G)$), es decir, la mínima cantidad de pilas para un orden fijo de las fichas.

En grafos de thinness acotada, muchos problemas NP-completos pueden ser resueltos en tiempo polinomial, como por ejemplo el problema del conjunto independiente ponderado máximo [Man+07] o el problema de coloreo capacitado con aplicación al problema de combinación de recorrido de recogida y entrega [BMO11].

Dado un orden para un grafo, la complejidad de hallar la thinness, para ese orden en particular, es $O(n^3)$ con n la cantidad de nodos del grafo. Esto es así ya que hallar la thinness del grafo es equivalente a encontrar el coloreo en un grafo auxiliar (asociado al orden) también de n vértices pero que resulta de co-comparabilidad [BE18]. Es un problema abierto la complejidad computacional de calcular la thinness de un grafo sin un orden prefijado.

En esta tesis se realiza un estudio sobre la thinness para orden fijo en grafos sin ciclos (árboles o bosques). Dado un orden con estructura recursiva para numerar los nodos del árbol (como inorder, preorder, postorder o BFS), calcularemos la thinness en tiempo lineal

en la cantidad de nodos introduciendo el nuevo concepto de Grupos de Conflicto.

Cabe destacar que la thinness para un orden fijo en el grafo es una cota superior para la thinness en el mismo.

1.1. Organización de la tesis

Describimos ahora la forma en que está organizado este trabajo.
En este Capítulo, además de definir los temas principales de la tesis, se presentan una serie de definiciones básicas de teoría de grafos y la notación utilizada, un resumen sobre clases de grafos haciendo énfasis en grafos de tipo árbol y un resumen sobre la thinness de los grafos.
En el Capítulo 2 presentamos los órdenes para los nodos del grafo con los que trabajaremos en esta tesis. Dado que la thinness de un grafo está asociada a un orden para los nodos, en esta tesis trabajaremos con 4 órdenes: inorder, preorder, postorder y BFS.
El Capítulo 3 está destinado a explicar el concepto de Grupos de Conflicto. Este es un nuevo concepto que utilizaremos para calcular la thinness de un grafo árbol en tiempo lineal.
En los Capítulos 4, 5, 6 y 7 se explica para inorder, preorder, postorder y BFS respectivamente cómo hallar la thinness utilizando el concepto de grupos de conflicto y se presenta el algoritmo correspondiente especificando la complejidad. Además comparamos la cota ya conocida para el valor de la thinness en grafos de tipo m-arios completos con la cota hallada para la thinness para cada uno de los órdenes. También se intenta encontrar la forma de mejorar la thinness modificando la representación del árbol de entrada como árbol enraizado con hijos izquierdo, centros y derecho, de forma tal que se altere la numeración pero no las adyacencias. Por último, comparamos cada orden con el algoritmo ya conocido, experimentando sobre diferentes tipos de grafos árbol y analizando complejidad temporal.
En el Capítulo 8 se compara la thinness relativa a los diferentes órdenes.
En el Capítulo 9 se presentan algunas conclusiones que surgen de este trabajo y las líneas futuras de investigación.

1.2. Grafos y clases de grafos

Denotaremos un grafo G por un par $(V(G), E(G))$, donde $V(G)$ es un conjunto finito, el conjunto de vértices de G, y $E(G)$ es un conjunto de pares no ordenados de vértices de

G, llamados aristas. En un grafo dirigido el conjunto $D(G)$ de aristas es un conjunto de pares ordenados de vértices. Un grafo se dice *trivial* si tiene un solo vértice.

Un vértice v es adyacente a otro vértice w en G si $(v, w) \in E(G)$. Decimos que v y w son los extremos de la arista. El vecindario de un vértice v es el conjunto $N(v)$ que consiste de todos los vértices adyacentes a v. El grado de un vértice v es la cardinalidad del conjunto $N(v)$ y se nota $d(v)$.

El complemento de un grafo G, denotado por $\overline{G}$, es el grafo que tiene el mismo conjunto de vértices de G y tal que dos vértices distintos son adyacentes en $\overline{G}$ si y sólo si no son adyacentes en G.

Un grafo H es un subgrafo de un grafo G si $V(H) \subseteq V(G)$ y $E(H) \subseteq E(G)$. El grafo H es un subgrafo inducido de G si $E(H) = (V(H) \times V(H)) \cap E(G)$.

Un camino en un grafo G es una secuencia de vértices distintos $P = v_1, v_2, ..., v_k$, donde $(v_i, v_{i+1}) \in E(G)$, $i \in \{1, ..., k-1\}$.

Un ciclo en un grafo G es una secuencia de vértices $C = v_1, v_2, ..., v_k$, donde $v_1, ..., v_k$ es un camino, v_1 es adyacente a v_k y $k \geq 3$.

Un grafo G es completo si cualquier par de vértices distintos de G son adyacentes. Una clique es un subgrafo completo en un grafo. Llamamos K_n al grafo completo con n vértices. Dado un grafo G se denota por $\omega(G)$ el tamaño de una clique con máxima cantidad de vértices en G.

Dada una familia de conjuntos, se define su grafo de intersección como el grafo obtenido al representar cada conjunto por un vértice de modo que dos vértices sean adyacentes si y solo si los conjuntos que representan tienen intersección no vacía.

Un grafo de intervalos es el grafo intersección de intervalos en una recta.

Un grafo $G = (V(G), E(G))$ es de comparabilidad si es posible direccionar sus aristas de modo de que el grafo dirigido resultante $G' = (V(G), D(G))$ satisfaga: $(u, v) \in D(G), (v, w) \in D(G) \Rightarrow (u, w) \in D(G)$ (transitividad). Un grafo es de co-comparabilidad si su complemento es de comparabilidad.

Un conjunto independiente es un conjunto de vértices entre los cuales no existen aristas.

Un coloreo de un grafo G es una partición de $V(G)$, donde cada clase de la partición es un conjunto independiente al que identificamos con un color. Un k-coloreo es una partición de $V(G)$ en k conjuntos independientes. Si G admite un k-coloreo, decimos que G es k-cromático. El número cromático de G es el menor k para el cual existe un k-coloreo de G y se nota $\chi(G)$.

Un grafo G es perfecto si para todo subgrafo inducido H de G vale $\omega(H) = \chi(H)$. Tanto los grafos de comparabilidad como los de co-comparabilidad son grafos perfectos [Gol77].

La forma habitual de dibujar un grafo en el plano es representando los vértices por puntos y las aristas por curvas entre sus extremos, que no pasan por ningún otro vértice.
Un grafo G es planar si existe una representación de G en el plano de modo que las aristas no se crucen, excepto en los vértices.

1.2.1. Grafos árbol

Definición 1.2.1. Árbol: Un árbol es un grafo no dirigido en el que cualesquier dos vértices están conectados por exactamente un camino. Todo grafo conexo acíclico es un árbol, y viceversa.

Definición 1.2.2. Rooted Tree (árbol con raíz): Un Rooted Tree es un grafo árbol en el cual un vértice fue designado como raíz.

Definición 1.2.3. Relaciones entre los nodos:

- El padre de un nodo es el nodo conectado a él en el camino hacia la raíz. Vale la propiedad de que todo nodo excepto la raíz tiene un único padre.
- Un hijo de un nodo v es un nodo del cual v es el padre.
- Un descendiente de un nodo v es cualquier nodo que sea hijo de v o (recursivamente) el descendiente de cualquiera de los hijos de v.
- Un hermano de un nodo v es cualquier otro nodo en el árbol que tiene el mismo padre que v.
- Llamaremos hoja a todo nodo del grafo que no tenga hijos.

Definición 1.2.4. Árboles m-arios: Un árbol m-ario es un rooted tree en el cual cada nodo tiene a lo sumo m hijos. Llamaremos árboles binarios a los árboles 2-arios y árboles ternarios a los árboles 3-arios.

Definición 1.2.5. Nivel de un nodo: El nivel de un nodo se define como 1 + el tamaño del camino entre la raíz y el nodo.

A partir de ahora, la representación planar del árbol será de la siguiente manera: Colocaremos la raíz r en la parte superior. Luego dibujaremos en una línea horizontal debajo de r a todos los hijos de la raíz siendo el primer hijo el de más a la izquierda. Repetiremos recursivamente este proceso para cada hijo de r.

Definición 1.2.6. Hijo izquierdo, centro y derecho:
Dada una representación planar del rooted tree, llamaremos:

- Hijo izquierdo de v al nodo adyacente a v, distinto de su padre, que se encuentra más a la izquierda.
- Hijo derecho de v al nodo adyacente a v, distinto de su padre, que se encuentra más a la derecha.
- Hijo(s) centro(s) de v a el/los nodos adyacentes a v, distintos de su padre, que no son el hijo izquierdo ni el derecho.

Si un vértice v tiene un único hijo, el mismo será izquierdo si lo dibujamos abajo y a la izquierda de v o derecho si lo dibujamos abajo a la derecha de v. En esta tesis no consideraremos la opción de tener solo hijos centros.

Definición 1.2.7. Subárboles:
Sea un rooted tree G y v un nodo en G. Llamaremos:

- Subárbol izquierdo de v al subárbol donde el hijo izquierdo de v es raíz y todo nodo descendiente del hijo izquierdo de v pertenece al subárbol.
- Subárboles centros de v a los subárboles donde un hijo centro de v es raíz y todo nodo descendiente del hijo centro de v pertenece al subárbol.
- Subárbol derecho de v al subárbol donde el hijo derecho de v es raíz y todo nodo descendiente del hijo derecho de v pertenece al subárbol.

Definición 1.2.8. Hermanos:
Sea un rooted tree G. Sea x un nodo en G cuyos hijos son $\{x_0, x_1, ..., x_k\}$.

- Llamaremos hermanos izquierdos de x_i al conjunto de nodos $x_j \ \forall \, 0 \leq j < i$
- Llamaremos hermanos derechos de x_i al conjunto de nodos $x_j \ \forall \, i < j \leq k$

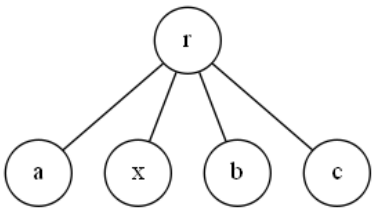


Fig. 1.1: Ejemplo de un rooted tree

En la figura 1.1 se puede observar un ejemplo de un rooted tree donde el nodo r es la raíz. Además, r es padre de a, x, b y c. a es hijo izquierdo de r y hermano izquierdo de x. x es hijo centro de r, al igual que b. b y c son hermanos derechos de x. c es el hijo derecho de r.

Definición 1.2.9. G_x: Sea G un grafo árbol y x un nodo en G. llamaremos G_x al subárbol que tiene como raíz a x y todo nodo descendiente de x pertenece al subárbol.

Definición 1.2.10. Árboles Zig-Zag: Un árbol Zig-Zag es un rooted tree en el cual cada nodo tiene exactamente un hijo excepto el nodo hoja. Además se cumple que para cada nodo x, si x es hijo derecho y no es hoja, luego x tiene solamente hijo izquierdo. Si x es hijo izquierdo y no es hoja, luego x tiene solamente hijo derecho. Notar que el grafo árbol Zig-Zag es un árbol binario.

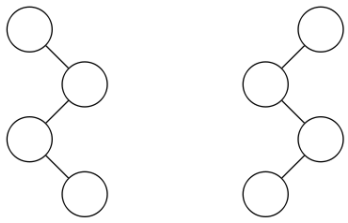


Fig. 1.2: Árboles Zig-Zag

Definición 1.2.11. Árboles Caterpillar: Un árbol Caterpillar es un árbol en el cual todo vértice está a lo sumo a distancia uno de un camino central. En este trabajo usaremos a los árboles Caterpillar como rooted tree en donde la raíz es un nodo extremo del camino central.

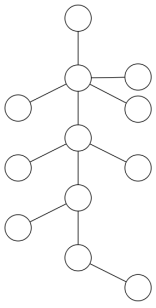


Fig. 1.3: Árbol caterpillar

Definición 1.2.12. Árboles balanceados a izquierda: Sea G un árbol m-ario, diremos que G es balanceado a izquierda si la raíz tiene m hijos y luego cada hijo izquierdo y centro tiene m hijos pero el hijo derecho ninguno.

Definición 1.2.13. Árboles balanceados a derecha: Sea G un árbol m-ario, diremos que G es balanceado a derecha si la raíz tiene m hijos y luego el hijo derecho tiene m hijos pero cada hijo izquierdo y centro ninguno.

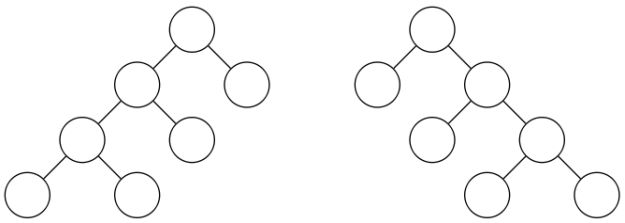


Fig. 1.4: Árboles balanceados a izquierda y a derecha respectivamente

Definición 1.2.14. Bosque: Un bosque es un grafo no dirigido, donde todas sus componentes conexas son árboles; en otras palabras, un bosque es una unión disjunta de árboles. De manera equivalente, un bosque es un grafo acíclico no dirigido.

Definición 1.2.15. Movimiento: Partiendo de un nodo, definimos movimiento $\in$ {izquierda, centro, derecha}, representando el acto de pasar de un nodo a su hijo izquierdo, a cualquiera de sus hijos centros o derecho, respectivamente.

Definición 1.2.16. Secuencia de Movimientos: Llamaremos secuencia de movimientos a una lista de movimientos, por ejemplo $(izquierda, derecha)$.

Definición 1.2.17. Lista de Secuencias de Movimientos: Llamaremos lista de secuencias de movimientos a, como lo indica su nombre, una lista de secuencias de movimientos, por ejemplo $[(izquierda, derecha), (centro, izquierda)]$.

A menos que se especifique lo contrario, en esta tesis vamos a trabajar siempre con grafos de tipo rooted tree.

1.3. Thinness

1.3.1. Definición

Un grafo $G = (V, E)$ es k-thin si existe un orden $v_1, ..., v_n$ de V y una partición de V en k clases (V^1, ..., V^k) tal que, para cada tripla (r, s, t) donde $r < s < t$, si v_r, v_s pertenecen a la misma clase y $v_t v_r \in E$, luego $v_t v_s \in E$. En este caso se dice que el orden y la partición son consistentes. El mínimo k tal que G es k-thin se llama la thinness de G y lo denotaremos como $thin(G)$.
La thinness no está acotada en la clase de todos los grafos y grafos con thinness acotada fueron introducidos en [Man+07] como una generalización de los grafos de intervalo, que son exactamente los grafos 1-thin [Ola91], [RR88].
Cuando se da una representación de un grafo como un grafo k-thin, para un valor constante

k, pueden ser resueltos en tiempo polinomial algunos problemas NP-completos como el conjunto independiente ponderado máximo (maximum weighted independent set) [Man+07] y el problema de coloreo acotado con una cantidad fija de colores (bounded coloring with fixed number of colors) [BMO11]. Estos algoritmos se aplicaron para mejorar la heurística de dos problemas del mundo real: el problema de asignación de frecuencia en redes GSM (Frequency Assignment Problem in GSM networks) [Man+07] y el problema del vendedor ambulante doble con múltiples pilas (Double Traveling Salesman Problem with Multiple Stacks) [BMO11] respectivamente.

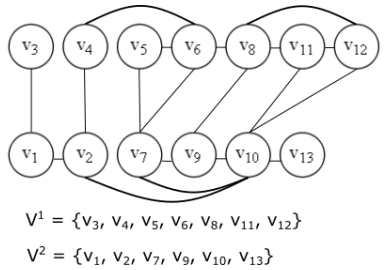


Fig. 1.5: Grafo con un orden y una partición consistente de los vértices

1.3.2. Cómo calcular la thinness de un grafo para un orden dado

Definición 1.3.1. $thin_<(G)$: Definiremos la thinness de un grafo para un orden $<$ dado ($thin_<(G)$) como la mínima cantidad de clases en una partición consistente con dicho orden.

Teorema 1.3.1. *[BE18] Dado un grafo G y un orden $<$ para los nodos, se puede encontrar en tiempo polinomial un grafo $G_<$ con las siguientes propiedades:*

- $V(G) = V(G_<)$
- *El número cromático de $G_<$ es igual al mínimo k tal que haya una partición de $V(G)$ en k clases consistente con el orden $<$, y si dividimos los nodos por color formamos una partición consistente con $<$.*
- *$G_<$ es un grafo de co-comparabilidad.*

Demostración. Sea G un grafo y $<$ un orden de los nodos. Vamos a construir un grafo $G_<$ tal que $V(G) = V(G_<)$ y $v < w$ son adyacentes en $G_<$ si y solo si no pueden pertenecer a la misma clase en una partición consistente con $<$. Por definición de consistencia, esto sucede si y solo si existe vértice z en G tal que $v < w < z$, z es adyacente a v y no es adyacente a w. Entonces definimos $E(G_<)$ de forma tal que $v < w$, $vw \in E(G_<)$ si y solo

si existe un vértice z en G tal que $v < w < z$, $zv \in E(G)$ y $zw \notin E(G)$.
Veamos ahora que $<$ es un orden de comparabilidad en $\overline{G}_<$. Supongamos que, al contrario, existe una tripla $r < s < t$ en $V(G)$ tal que rs, st son aristas de $\overline{G}_<$ y rt no es una arista en $\overline{G}_<$. Por definición de $G_<$ existe entonces un nodo z tal que $r < s < t < z$, $zr \in E(G)$ y $zt \notin E(G)$. Si $zs \notin E(G)$, entonces rs es una arista de $G_<$, contradicción. Si $zs \in E(G)$, entonces st es una arista de $G_<$, también una contradicción. Entonces $G_<$ es un grafo de co-comparabilidad, siendo $<$ un orden de comparabilidad en $\overline{G}_<$.
Como definimos $G_<$ tal que $V(G_<) = V(G)$ y $v < w$ son adyacentes en $G_<$ si y solo si no pueden pertenecer a la misma clase para una partición consistente con $<$, se deduce que hay una correspondencia uno a uno entre particiones de $V(G)$ consistentes con $<$ y la partición que da el coloreo de $G_<$. En particular, el mínimo k tal que existe una partición de $V(G)$ en k clases consistente con $<$ es el numero cromático de $G_<$. El coloreo de $G_<$ puede ser computado en $O(n^3)$ (donde n es la cantidad de vértices) ya que $G_<$ es un grafo de co-comparabilidad [Gol77]. □

Se puede observar fácilmente que $G_<$ puede obtenerse partiendo de G en tiempo $O(n^3)$, ya que alcanza con verificar si para todo conjunto de tres nodos u, v, w, donde $u < v < w$, sucede que $uw \in E(G)$ y $wv \notin E(G)$. Si se cumple, entonces agregamos uv a $G_<$. Luego de generar $G_<$, hay que colorearlo, que toma $O(n^3)$ por ser un grafo de co-comparabilidad. Por lo tanto, la complejidad del algoritmo es $O(2 * n^3)$, lo que queda en $O(n^3)$.
Si por el contrario, fijamos la partición de los vértices de un grafo podría no existir un orden consistente y es NP-Completo decidir si existe o no [BE18].

Un ejemplo de $G_<$ para un grafo G con un orden dado se encuentra en la figura 1.6.

(a) G *(b) $G_<$*

Fig. 1.6: G y $G_<$ para un orden dado.

1.3.3. Thinness en grafos no conexos

Teorema 1.3.2. *[Man+07] Sea G un grafo no conexo con $\{G_0, G_1, ..., G_m\}$ sus componentes conexas, la thinness de G es $\max_{0 \le i \le m} thin(G_i)$.*

Teorema 1.3.3. *Sea G un grafo no conexo con $\{G_0, G_1, ..., G_m\}$ sus componentes conexas, $thin_<(G)$ es $\max_{0 \le i \le m} thin_<(G_i)$ para un orden $<$ dado donde los vértices de cada componente*

conexa son consecutivos.

Demostración. Veamos que no puede ser menor.
Sea $k = \max_{0 \leq i \leq m} thin_{<}(G_i)$. Esto implica que para algún i $\chi(G_{i<}) = k$. Como $G_{i<}$ es un grafo perfecto por ser de co-comparabilidad [Gol77], entonces $\chi(G_{i<}) = \omega(G_{i<})$. Notemos que $G_{i<}$ es subgrafo inducido de $G_<$. Por lo tanto en $G_<$ se tiene una clique de k nodos. Por lo tanto, la thinness no puede ser menor que el máximo valor de las thinness de las diferentes componentes conexas.
Veamos que no es mayor.
Supongamos por el absurdo que es mayor. Luego en la máxima clique en $G_<$ hay nodos pertenecientes a dos componentes conexas distintas. Esto implica que existen tres nodos $v_r < v_s < v_t$ tal que v_r y v_s pertenecen a dos componentes diferentes y sucede que $v_t v_r \in E$, pero $v_t v_s \notin E$. Como $v_t v_r \in E$, ambos deben pertenecer a la misma componente conexa, pero $v_r < v_s < v_t$. Absurdo. Los vértices de cada componente conexa son consecutivos. □

Este teorema solo vale cuando los vértices de cada componente conexa son consecutivos. Ese es el caso de casi cualquier algoritmo de numeración de vértices de un grafo en general y bosques en particular. Por lo tanto, en esta tesis vamos a trabajar con grafos árboles (conexos). En caso de tener un grafo bosque hay que aplicar el algoritmo en cada una de las componentes conexas y luego la thinness del grafo será el máximo valor de thinness hallado.

1.3.4. Cota para árboles m-arios completos

Fue demostrado que la cota de la thinness para un árbol m-ario completo con m un valor fijo, es $\Theta(log(n))$ [BE18]

1.3.5. Incentivo para el cálculo de la thinness de un grafo

La creciente complejidad de las estrategias de planificación en empresas de transporte y logística ha inducido recientemente la aparición de nuevos problemas de ruteo de vehículos. Uno de estos problemas es el problema del vendedor ambulante doble con varias pilas, (Double Traveling Salesman Problem with Multiple Stacks, DTSPMS) introducido por Petersen y Madsen [PM09].
Es un problema de ruteo en el que algunas recolecciones y entregas deben realizarse en dos redes independientes, verificando algunas restricciones de carga y descarga impuestas en el vehículo.

Se consideran dos regiones independientes que se supone que están muy alejadas, y n elementos deben recogerse en diferentes ubicaciones de la primera región y entregarse a las ubicaciones correspondientes de la segunda región. Se utiliza un único vehículo y una vez que se carga un paquete no puede ser reacomodado.
Los paquetes se acomodan dentro del vehículo en diferentes compartimentos que se rigen bajo el criterio de Last-In-First-Out (LIFO) y no hay restricciones entre compartimentos. De ahora en adelante, llamaremos a los compartimentos "Stacks"(pilas).
En la aplicación de la vida real que motivó originalmente el problema, los artículos que se entregarían eran Euro Pallets estandarizados, que podrían utilizarse para cargar diferentes tipos de productos. Los vehículos utilizados para transportar las mercancías tenían un contenedor de Pallets de 40 pies en el cual los Euro Pallets estandarizados cabían 3 por 11, lo que sugiere el uso de 3 pilas disponibles con una capacidad máxima de 11 artículos. El conductor no tenía permitido tocar los productos por motivos de seguridad y de seguros y, como consecuencia, no se le permitió reacomodar paquetes durante el proceso.
Por lo tanto, una solución viable para una instancia del DTSPMS consiste en un recorrido de recolección, un recorrido de entrega y una asignación de pila, es decir, qué pila se asigna a cada elemento. El objetivo es minimizar la suma de las longitudes de los recorridos de recogida y entrega.
El subproblema de verificar si, para un recorrido de recogida determinado y un recorrido de entrega determinado, existe una asignación de pila factible se conoce como el problema de combinación de recorrido de recogida y entrega (pick up and delivery tour combination, PDTC). Este problema puede ser resulto en tiempo polinomial cuando se tiene un número fijo de stacks [BMO11]. Como mencionamos antes, en el caso real que originalmente motivó el problema, $s = 3$ y $h = 11$. Entonces, para el punto de vista práctico, es razonable pensar a s como una constante y h como un parámetro.
Para alcanzar este resultado, se sigue el enfoque del coloreo de grafo en [CCN12], pero teniendo en cuenta las restricciones en la capacidad. En [BMO11] se muestra que el problema de PDTC es equivalente al problema de coloreo acotado (bounded coloring, BC) en grafos de permutación. Luego demuestra también que el problema de coloración capacitada (capacitated coloring, CC), un problema que generaliza el problema de coloración equitativa, puede ser resuelto en tiempo polinomial en grafos de co-comparabilidad, superclase de los grafos de permutación, cuando la cantidad de colores s es fija.
En [Man+07] se muestra que el problema de conjunto independiente ponderado máximo puede ser resuelto en tiempo polinomial en grafos con thinness acotada cuando se de una representación adecuada del grafo. En [BMO11] se muestra que, bajo la misma hipótesis,

el problema de CC puede ser resuelto en tiempo polinomial si la cantidad de colores s es fija. Luego, dada una instancia del problema de CC en un grafo de co-comparabilidad, si la cantidad de colores s es fija, entonces en tiempo polinomial podemos mostrar que la instancia no es factible o mostrar que G tiene thinness a lo sumo s y encontrar la correspondiente representación.

1.4. Divide and conquer

En la cultura popular, divide y vencerás (o Divide and conquer en inglés) hace referencia a un refrán que implica resolver un problema difícil, dividiéndolo en partes más simples tantas veces como sea necesario, hasta que la resolución de las partes se torna obvia. La solución del problema principal se construye con las soluciones de los subproblemas.
En las ciencias de la computación, el término divide y vencerás (DYV) hace referencia a uno de los más importantes paradigmas de diseño algorítmico. El método está basado en la resolución recursiva de un problema dividiéndolo en dos o más subproblemas de igual tipo o similar. El proceso continúa hasta que éstos llegan a ser lo suficientemente sencillos como para que se resuelvan directamente. Al final, las soluciones a cada uno de los subproblemas se combinan para dar una solución al problema original.
Esta técnica es la base de los algoritmos eficientes para casi cualquier tipo de problema como, por ejemplo, algoritmos de ordenamiento (quicksort, mergesort, entre muchos otros), multiplicar números grandes (Karatsuba), análisis sintácticos (análisis sintáctico top-down) y la transformada discreta de Fourier.
En esta tesis usaremos la técnica de divide and conquer para resolver el problema de hallar la thinness en grafos de tipo árbol dado un orden fijo de tipo recursivo:
Sea G el grafo rooted tree con r la raíz. Sean $\{x_0, ..., x_k\}$ los hijos de r. Llamaremos G_{x_i} al subárbol de G que tiene como raíz al nodo x_i para $0 \leq i \leq k$. Vamos a dividir el problema en problemas más chicos, pasando ahora a pedir la thinness para cada uno de los subgrafos $\{G_{x_0}, ..., G_{x_k}\}$ y así recursivamente para cada nodo hasta llegar al caso base en donde quiero calcular la thinness de un grafo trivial. En este caso sabemos que la thinness es 1.
El proceso de combinar será pasar determinada información desde cada nodo hijo hacia el padre (como por ejemplo la thinness del subárbol que se corresponde con ese hijo) para poder ir calculando la thinness a medida que subimos desde las hojas hacia la raíz. Más adelante mostraremos qué información será necesario pasar al nodo r desde cada uno de los hijos para poder determinar la thinness del árbol.

2. ÓRDENES

En esta tesis vamos a hallar el valor de la thinness de un grafo árbol para un orden específico de los nodos. Vamos a trabajar con 4 órdenes distintos:

- Inorder
- Preorder
- Postorder
- Búsqueda a lo ancho (BFS - Breadth-first search)

Éstos son métodos para recorrer todos los nodos de un grafo árbol. El número asignado a cada nodo será el numero de orden en el que fueron recorridos. Es decir, el primer nodo visitado, será el nodo con numeración 1, el segundo será el nodo 2, y así sucesivamente hasta recorrer todos los nodos del árbol.
Los tres primeros órdenes están definidos solo para árboles binarios, por lo que vamos a definirlos también para árboles m-arios, especificando en qué orden se deberán recorrer los hijos centros.

2.1. Inorder

Sea G un grafo árbol binario. En Inorder el grafo se recorre de la siguiente manera:

1. Revisar si el grafo actual es vacío.
2. Recorrer el subárbol izquierdo llamando recursivamente a la función Inorder.
3. Visitar la raíz del grafo actual. En este caso, numerarla.
4. Recorrer el subárbol derecho llamando recursivamente a la función Inorder.

2.1.1. Inorder m-ario

Sea G un grafo árbol m-ario. En Inorder el grafo se recorre de la siguiente manera:

1. Revisar si el grafo actual es vacío.
2. Recorrer el subárbol izquierdo llamando recursivamente a la función Inorder.
3. Recorrer cada uno de los subárboles centro, en orden de izquierda a derecha, llamando recursivamente a la función Inorder.
4. Visitar la raíz del grafo actual. En este caso, numerarla.
5. Recorrer el subárbol derecho llamando recursivamente a la función Inorder.

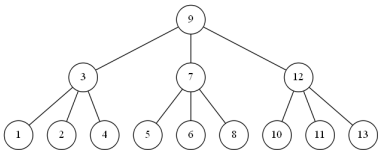


Fig. 2.1: Árbol numerado con inorder

Teorema 2.1.1. *Es indistinto para un nodo tomar el hijo de más a la izquierda como centro o como izquierdo.*

Demostración. Sea $c > 0$, $d \in \{0, 1\}$, entonces decir que un nodo x tiene c hijos centros, d hijos derechos y no tiene hijo izquierdo es equivalente a decir que x tiene hijo izquierdo, $c - 1$ hijos centros y d hijos derechos.
Sea el árbol con raíz r, donde r tiene como hijos los nodos $\{x_0, x_1, ..., x_k\}$ con x_k hijo derecho.
Sea $\#(x_i)$ con $0 \leq i < k$ la cantidad de nodos que tiene el subárbol que tiene como raíz al nodo x_i. Por definición de inorder:

- El subárbol que tiene como raíz a x_0 será numerado con los números del 0 al $\#(x_0)-1$.
- Sea $1 \leq j < k$. El subárbol que tiene como raíz a x_j será numerado con los números del
$$\sum_{i=0}^{j-1} \#(x_i) \quad al \quad \sum_{i=0}^{j-1} \#(x_i) + \#(x_j)$$
- La raíz será numerada con el número resultante de
$$\sum_{i=0}^{k-1} \#(x_i)$$
- El subárbol que tiene como raíz a x_k, es decir, el subárbol derecho, será numerado con los números del
$$\sum_{i=0}^{k-1} \#(x_i) + 1 \quad al \quad \sum_{i=0}^{k-1} \#(x_i) + 1 + \#(x_k)$$

En caso de no haber hijo derecho, la numeración es igual que lo anterior eliminando el cuarto ítem. Esto sucede sin importar si el hijo que se encuentra más a la izquierda es centro o izquierdo.
Luego, si x tiene c hijos, $c > 0$, sin perdida de generalidad, x tiene hijo izquierdo. □

2.2. Preorder

Sea G un grafo árbol binario. En Preorder el grafo se recorre de la siguiente manera:

1. Revisar si el grafo actual actual es vacío.
2. Visitar la raíz del grafo actual. En este caso, numerarlo.
3. Recorrer el subárbol izquierdo llamando recursivamente a la función Preorder.
4. Recorrer el subárbol derecho llamando recursivamente a la función Preorder.

2.2.1. Preorder m-ario

Sea G un grafo árbol m-ario. En Preorder el grafo se recorre de la siguiente manera:

1. Revisar si el grafo actual es vacío.
2. Visitar la raíz del grafo actual. En este caso, numerarlo.
3. Recorrer el subárbol izquierdo llamando recursivamente a la función Preorder.
4. Recorrer cada uno de los subárboles centro, en orden de izquierda a derecha, llamando recursivamente a la función Preorder.
5. Recorrer el subárbol derecho llamando recursivamente a la función Preorder.

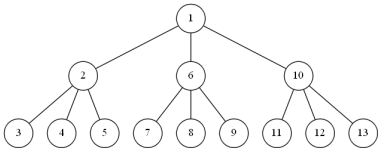


Fig. 2.2: Árbol numerado con preorder

2.3. Postorder

Sea G un grafo árbol binario. En Postorder el grafo se recorre de la siguiente manera:

1. Revisar si el grafo actual es vacío.
2. Recorrer el subárbol izquierdo llamando recursivamente a la función Postorder.
3. Recorrer el subárbol derecho llamando recursivamente a la función Postorder.
4. Visitar la raíz del grafo actual. En este caso, numerarlo.

2.3.1. Postorder m-ario

Sea G un grafo árbol m-ario. En Postorder el grafo se recorre de la siguiente manera:

1. Revisar si el grafo actual es vacío.
2. Recorrer el subárbol izquierdo llamando recursivamente a la función Postorder.
3. Recorrer cada uno de los subárboles centro, en orden de izquierda a derecha, lla-

mando recursivamente a la función Postorder.

4. Recorrer el subárbol derecho llamando recursivamente a la función Postorder.
5. Visitar la raíz del grafo actual. En este caso, numerarlo.

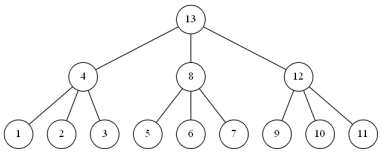


Fig. 2.3: Árbol numerado con postorder

Observación 2.3.1. Por el mismo argumento por el cuál es indistinto tomar al hijo centro como centro o como izquierdo en el caso de no haber izquierdo en inorder, al ser consecutivas las numeraciones de los subárboles en el caso de preorder y postorder es indistinto para un nodo tomar el hijo de más a la izquierda como centro o como izquierdo y el hijo de más a la derecha como centro o como derecho.

2.4. Búsqueda a lo ancho (BFS - Breadth first search)

BFS es un algoritmo para recorrer los nodos de un grafo con o sin ciclos [Koz92]. En este trabajo vamos a usar solamente grafos de tipo árbol, por lo que vamos a mostrar el funcionamiento de BFS solo para grafos conexos acíclicos.
El recorrido comienza en la raíz del árbol y explora todos los nodos adyacentes a la profundidad actual antes de pasar a los nodos del siguiente nivel de profundidad. Para ello, utiliza una cola (FIFO) en la que encola la raíz y luego, mientras que la cola no esté vacía, desencola un nodo y encola sus hijos de forma ordenada de izquierda a derecha. Cuando la cola se vacía se puede asegurar que ya se recorrieron todos los nodos del árbol.

Algoritmo 1 BFS - Búsqueda a lo ancho

```
1: S = Cola
2: S.encolar(raíz)
3: while S no es vacía do
4:     v ← S.desencolar()
5:     for h ∈ hijos(v) do
6:         S.encolar(h)
7:     end for
8: end while
```

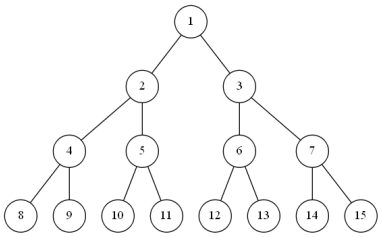


Fig. 2.4: Árbol numerado con BFS

2.5. Búsqueda en profundidad (DFS - Depth first search)

DFS es un algoritmo para recorrer los nodos de un grafo con o sin ciclos [Koz92]. En este trabajo vamos a usar solamente grafos de tipo árbol, por lo que vamos a mostrar el funcionamiento de DFS solo para grafos conexos acíclicos.
El recorrido comienza en la raíz del árbol y explora a lo largo de cada rama antes de retroceder. Para ello, utiliza una pila o "stack"(LIFO) en la que apila la raíz y luego, mientras que la pila no esté vacía, desapila un nodo y apila sus hijos de forma ordenada de izquierda a derecha. Cuando la cola se vacía se puede asegurar que ya se recorrieron todos los nodos del árbol.

Algoritmo 2 DFS - Búsqueda en profundidad

```
1: S = Pila
2: S.apilar(raíz)
3: while S no es vacía do
4:     v ← S.desapilar()
5:     for h ∈ hijos(v) do
6:         S.apilar(h)
7:     end for
8: end while
```

Notar que cuando se trata de grafos árbol, el recorrido de DFS es igual que Preorder. Dado que trabajaremos solo con grafos árbol, en esta tesis DFS y Preorder son equivalentes.

3. GRUPOS DE CONFLICTO

Como vimos, la thinness de un grafo con nodos numerados con un orden $<$ se calcula generando un grafo $G_<$, donde $V(G) = V(G_<)$ y dos nodos son adyacentes si y solo si son conflictivos, es decir, no pueden pertenecer a la misma clase en una partición consistente con $<$. Y luego calculando el número cromático de $G_<$, ya que $\chi(G_<) = thin_<(G)$. Recordemos que como el grafo $G_<$ es un grafo de co-comparabilidad, en particular es perfecto, y entonces $\chi(G_<) = \omega(G_<)$.
En esta tesis, para hallar la thinness en el grafo árbol para un orden $<$, vamos a introducir el concepto de grupos de conflicto y su relación con las cliques y adyacencias de los nodos en $G_<$. Esta relación es puramente a fines teóricos para demostrar la correctitud del algoritmo ya que la ventaja de nuestro algoritmo es que no necesita construir el grafo $G_<$.

Definición 3.0.1. Conflictos de un nodo:
Diremos que un nodo x tiene conflicto con un nodo y si existe z en G tal que $x < y < z$, x es adyacente a z e y no es adyacente a z, es decir, el conjunto $\{y \in N_{G_<}(x) | y > x\}$.

Definición 3.0.2. Grupo de conflicto:
Sea x_i un nodo del grafo, sean los nodos $n_0, ..., n_k$ los nodos con los que x_i tiene conflicto. Llamamos grupo de conflicto de x_i o C_{x_i} al conjunto formado por $\{n_0, ..., n_k\}$. Nos referiremos a x_i como el nodo conflictivo.

Definición 3.0.3. Grupos de conflicto anidados:
Sean g grupos de conflicto, $C_{x_1}, C_{x_2}, ..., C_{x_g}$, se dicen anidados si
$\forall\ 1 \leq i < j \leq g,\ x_i \in C_{x_j}$ y $\exists$ un nodo a tal que $\forall\ 1 \leq k \leq g, a \in C_{x_k}$ y $a \neq x_k$. Notar que a pesar de que el nombre es "anidados", no necesariamente $C_{x_i} \cup \{x_i\} \subseteq C_{x_j}$ para $i < j$.

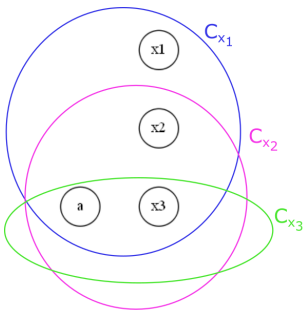


Fig. 3.1: Grupos de conflicto anidados.

Definición 3.0.4. Grado de anidación de un nodo: Sea x un nodo en el grafo, llamaremos grado de anidación de x ($\alpha(x)$) a la cantidad máxima de grupos de conflictos anidados que contienen a x.

Definición 3.0.5. Entrar en un grupo de conflicto:
Llamaremos entrar a un grupo de conflicto cuando partimos de un nodo x en G, nos desplazamos hacia algún nodo v descendiente de x, y v pertenece a un grupo de conflicto al que x no pertenecía.

Definición 3.0.6. Nodo dentro de grupo de conflicto:
Diremos que un nodo x se encuentra **dentro** de un grupo de conflicto si $x \in C_i$ para algún nodo i y $x \neq i$.

Notación 3.0.1. Llamaremos $\#GDCA$ a la máxima cantidad de grupos de conflictos anidados.

3.1. Thinness y grupos de conflicto

Teorema 3.1.1. $thin_<(G) = \#GDCA + 1$

Demostración. Para demostrar este teorema vamos a ver que $thin_<(G) \geq \#GDCA + 1$ y que $thin_<(G) \leq \#GDCA+1$. Luego si ambas condiciones se cumplen entonces $thin_<(G) = \#GDCA + 1$.

Veamos que $thin_<(G) \geq \#GDCA + 1$:
Sea $t = \#GDCA$ en G. Sean $x_1, ..., x_t$ con $C_{x_1}, ..., C_{x_t}$ sus grupos de conflicto respectivamente tal que $\forall 1 \leq i < j \leq t, x_i \in C_{x_j}$. Sea además un nodo a tal que $\forall 1 \leq k \leq t, a \in C_{x_k}$ y $a \neq x_k$.
Por definición de grupo de conflicto, a es adyacente a $x_1, ..., x_t$ en $G_<$ y $\forall i, j$ x_i es adyacente a x_j en $G_<$. Por lo tanto $x_1, ..., x_t, a$ forman una clique en $G_<$, entonces $\chi(G_<) \geq t+1$. Luego $thin_<(G) = \chi(G_<) \geq t+1$ entonces $thin_<(G) \geq t+1$.

Veamos que $thin_<(G) \leq \#GDCA + 1$:
Sea $k = thin_<(G)$. Sabemos que entonces $k = \chi(G_<)$. Como $G_<$ es un grafo perfecto, entonces $k = \omega(G_<)$.
Sean $x_0, ..., x_{k-1}$ una clique máxima de $G_<$ tal que $x_0 > x_1 > ... > x_{k-1}$. Como x_0 es mayor que todos y adyacente en $G_<$, luego $\forall\ 1 \leq i \leq k-1$, $x_0 \in C_{x_i}$ y $x_0 \neq x_i$. x_0 se corresponde con a de la definición de grupos de conflicto anidados. Sea $1 \leq i < j \leq k-1$, por definición

$x_i > x_j$ y son adyacentes en $G_<$, entonces $x_i \in C_{x_j}$. Por lo tanto $C_{x_1}, ..., C_{x_{k-1}}$ son grupos de conflicto anidados. Entonces $thin_<(G) - 1 \leq \#GDCA$. □

4. INORDER

4.1. Thinness para árboles Inorder

Por el Teorema 3.1.1 vimos que la thinness en un grafo G es uno más que la máxima cantidad de grupos de conflicto anidados que hay en G. Por lo tanto, para hallar la thinness en un grafo árbol, vamos a hallar la máxima cantidad de grupos de conflicto anidados que hay. En este capítulo vamos a analizar cómo hacer esto en un grafo G numerado con el método Inorder, es decir, donde $<$ es inorder. Primero lo haremos para árboles binarios y luego para árboles m-arios.

4.1.1. Árboles binarios

4.1.1.1 Cómo se forman los grupos de conflicto

Sabemos por definición de inorder que para cualquier nodo x vale que:

- Todo nodo en el subárbol izquierdo de x es menor que x. Esto implica que x no va a tener conflicto con ningún nodo de ese subárbol.
- Todo nodo en el subárbol derecho de x es mayor que x.

Sea x un nodo en el árbol, hijo izquierdo de un nodo p. Entonces sabemos que vale que:

- Por definición de inorder p es mayor que x y p es el máximo adyacente a x.
- Por ser un grafo árbol ningún nodo en un subárbol de x va a ser adyacente a p.
- Por definición de inorder todos los nodos en el subárbol derecho son mayores que x.
- Por definición de inorder todos los nodos en el subárbol izquierdo son menores que x.
- Por definición de inorder los únicos nodos mayores y adyacentes a x son p y el hijo derecho de x, llamémoslo dx.
- Por definición de inorder, los nodos que se encuentran entre x y p (en el orden dado por el inorder) y los nodos que se encuentran entre x y dx están en el subárbol derecho de x.

Por lo tanto, como no puede tener conflicto con ningún nodo fuera de su subárbol derecho y tiene conflicto con todo su subárbol derecho por ser adyacente a p, el grupo de conflicto de x (C_x) será todo nodo perteneciente a su subárbol derecho.

Sea x un nodo en el árbol, hijo derecho de un nodo p. Entonces sabemos que vale que:

- Por definición de inorder p es menor que x.
- Por definición de inorder todos los nodos en su subárbol izquierdo son menores que x.
- Por definición de inorder todos los nodos en su subárbol derecho son mayores que x.
- Por definición de inorder el único adyacente a x mayor que x es su hijo derecho, llamémoslo dx.
- Por definición de inorder los nodos que se encuentran entre x y dx se encuentran en el subárbol izquierdo de dx.

Por lo tanto los nodos pertenecientes al grupo de conflicto de x (C_x) será el conjunto de nodos perteneciente al subárbol izquierdo de dx sacando el hijo izquierdo de dx ya que este es adyacente a dx.

Sea x el nodo raíz en el árbol. Entonces sabemos que vale que:

- Por definición de inorder todos los nodos en el subárbol izquierdo son menores que x.
- Por definición de inorder todos los nodos en el subárbol derecho son mayores que x.
- Por definición de inorder el único adyacente a x mayor que x es su hijo derecho, llamémoslo dx.
- Por definición de inorder los nodos que se encuentran entre x y dx se encuentran en el subárbol izquierdo de dx.

Por lo tanto los nodos pertenecientes al grupo de conflicto de x (C_x) van a ser los nodos del subárbol izquierdo de dx sacando el hijo izquierdo de dx ya que este es adyacente a dx.

Observación 4.1.1. La raíz se comporta igual que un nodo hijo derecho. Esto lo usaremos más adelante.

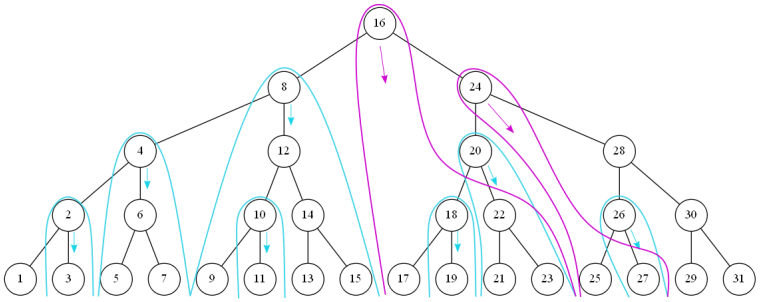


Fig. 4.1: Grupos de conflicto en un árbol binario.

4.1.2. Árboles m-arios

Por Teorema 2.1.1 vamos a considerar, sin perdida de generalidad, que si el conjunto de los hijos centros de x es distinto de vacío, luego existe un hijo izquierdo de x.

4.1.2.1 Cómo se forman los grupos de conflicto

Sabemos por definición de inorder que para cualquier nodo x vale que:

- Todo nodo en el subárbol izquierdo de x es menor que x. Esto implica que x no va a tener conflicto con ningún nodo de ese subárbol.
- Todo nodo en un subárbol centro de x es menor que x. Esto implica que x no va a tener conflicto con ningún nodo de ese subárbol.
- Todo nodo en el subárbol derecho de x es mayor que x. Por lo tanto, x puede tener conflicto con nodos en ese subárbol.

Sea x un nodo en el árbol, hijo izquierdo o centro de un nodo p. Entonces sabemos que vale que:

- Por definición de inorder p es mayor que x y p es el máximo adyacente a x.
- Por ser un grafo árbol ningún nodo en un subárbol de x va a ser adyacente a p.
- Por definición de inorder todos los nodos en el subárbol derecho son mayores que x.
- Por definición de inorder todos los nodos en el subárbol izquierdo o centro son menores que x.
- Por definición de inorder los únicos nodos mayores y adyacentes a x son p y el hijo derecho de x, llamémoslo dx.

- Por definición de inorder, los nodos que se encuentran entre x y p y los nodos que se encuentran entre x y dx están en el subárbol derecho de x y en todo subárbol que tiene como raíz un hijo centro de p pero que se encuentra más a la derecha que x.

Entonces sea $0 \leq i < k$. Sean $\{x_0, x_1, ..., x_i, ...x_k\}$ los hijos de p.

Si x_k es el hijo derecho de p. Entonces el grupo de conflicto de x_i (C_{x_i}) serán los nodos que pertenezcan al subárbol derecho de x_i unión con todos los subárboles donde la raíz es un hijo de x_j $\forall i \leq j < k$.

Si p no tiene hijo derecho. Entonces el grupo de conflicto de x_i (C_{x_i}) serán los nodos que pertenezcan al subárbol derecho de x_i unión con todos los subárboles donde la raíz es un hijo de x_j $\forall i \leq j \leq k$.

Observación 4.1.2. Sea i un nodo en G hijo izquierdo de un nodo p. Sea además un nodo c, hijo centro de p. Sea $\{i_0, ..., i_k\}$ el conjunto de hijos de i. Sea $\{c_0, ..., c_k\}$ el conjunto de hijos de c con i_k y c_k hijos derechos de i y de c respectivamente. La única diferencia entre i y c es que $\forall\ 0 \leq j < k$, i_j no pertenece a ningún grupo de conflicto y c_j sí.

Sea x un nodo en el árbol, hijo derecho de un nodo p. Entonces sabemos que vale que:

- Por definición de inorder p es menor que x.
- Por definición de inorder todos los nodos en el subárbol izquierdo o centro son menores que x.
- Por definición de inorder todos los nodos en el subárbol derecho son mayores que x.
- Por definición de inorder el único adyacente a x mayor que x es su hijo derecho, llamémoslo dx.
- Por definición de inorder los nodos que se encuentran entre x y dx se encuentran en el subárbol izquierdo o centro de dx.

Por lo tanto los nodos pertenecientes al grupo de conflicto de x (C_x) van a ser los nodos del subárbol izquierdo y centro de dx sacando el hijo izquierdo y los hijos centros de dx ya que estos son adyacentes a dx.

Sea x el nodo raíz en el árbol. Entonces sabemos que vale que:

- Por definición de inorder todos los nodos en el subárbol izquierdo o centro son menores que x.
- Por definición de inorder todos los nodos en el subárbol derecho son mayores que x.
- Por definición de inorder el único adyacente a x mayor que x es su hijo derecho, llamémoslo dx.
- Por definición de inorder los nodos que se encuentran entre x y dx se encuentran en el subárbol izquierdo o centro de dx.

Por lo tanto, los nodos pertenecientes al grupo de conflicto de x (C_x) van a ser los nodos del subárbol izquierdo y centro de dx sacando el hijo izquierdo y los hijos centros de dx ya que estos son adyacentes a dx.

Observación 4.1.3. La raíz se comporta igual que un nodo hijo derecho. Utilizaremos esto más adelante.

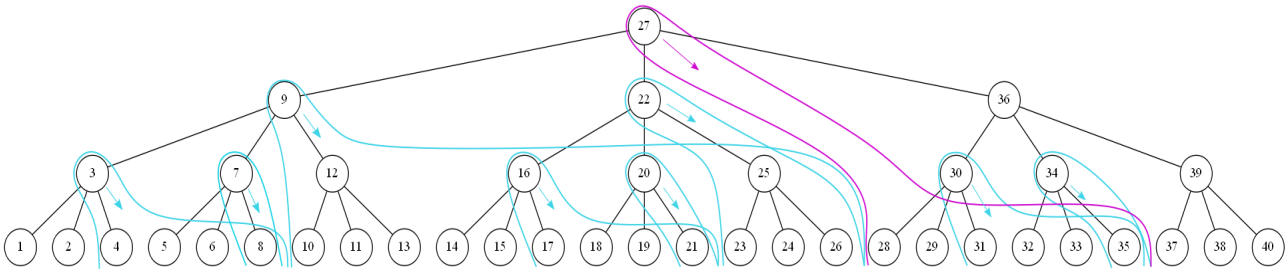


Fig. 4.2: Grupos de conflicto en un árbol ternario.

4.1.3. Grupos de conflicto y movimientos en el árbol

Ahora vamos a hablar de árboles m-arios. Si es binario, no hay hijos centro pero vale lo mismo que para los m-arios sabiendo que el conjunto de hijos centros es vacío.

4.1.3.1 Formas de entrar a un grupo de conflicto

Veamos que entonces se tienen las siguientes formas de entrar en un grupo de conflicto:

1. Dado un nodo x, sea ix su hijo izquierdo y di el hijo derecho de ix, luego $di \in C_{ix}$ y $\forall v$ con v perteneciente a la descendencia de di, $v \in C_{ix}$. Es decir, si realizo los movimientos:
   Izquierda - Derecha llego a un nodo perteneciente a un nuevo grupo de conflicto.

2. Dado un nodo x, sea cx un hijo centro e ix su hijo izquierdo. Sea dc el hijo derecho de cx, luego $dc \in C_{cx}$ y $\forall v$ con v perteneciente a la descendencia de dc, $v \in C_{cx}$. Sea hc un hijo centro, izquierdo o derecho de cx, luego $hc \in C_{ix}$ y $\forall v$ con v perteneciente a la descendencia de hc, $v \in C_{ix}$. Es decir, si realizo los movimientos:
Centro - Izquierda/Centro/Derecha llego a un nodo perteneciente a un nuevo grupo de conflicto.

3. Dado un nodo x, sea dx su hijo derecho, dd el hijo derecho de dx y hd un hijo centro o el hijo izquierdo de dd. Sea hh un hijo de hd, luego $hh \in C_{dx}$ y $\forall v$ con v perteneciente a la descendencia de hh, $v \in C_{dx}$. Es decir, si realizo los movimientos:
Derecha - Derecha - Izquierda/Centro - Izquierda/Centro/Derecha llego a un nodo perteneciente a un nuevo grupo de conflicto.

4. Dado un nodo r raíz del grafo. Sea dr su hijo derecho, hd el hijo izquierdo o centro de dr. Sea hh un hijo de hd, luego $hh \in C_r$ y $\forall v$ con v perteneciente a la descendencia de hh, $v \in C_r$. Es decir, si realizo los movimientos partiendo de la raíz:
Derecha - Izquierda/Centro - Izquierda/Centro/Derecha llego a un nodo perteneciente a un nuevo grupo de conflicto.

Notar que la única diferencia entre los ítems 3 y 4 es que hay un movimiento a la derecha más en el ítem 3. Utilizaremos esta observación más adelante.

En la figura 4.3 podemos observar ejemplos de algunos de los movimientos. Estos se encuentran marcados con diferentes colores.

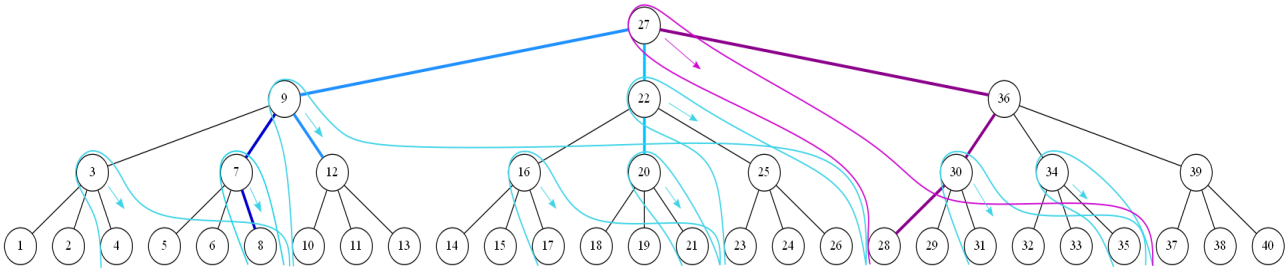


Fig. 4.3: Formas de entrar a los grupos de conflicto.

Definición 4.1.1. Movimientos de anidación: Llamaremos movimientos de anidación a las secuencias de movimientos mencionadas arriba.

4.1.3.2 Anidando grupos de conflicto

Teorema 4.1.1. *Sean dos nodos x e y, con $y \in C_x$. Sea d un nodo perteneciente a la descendencia de y, entonces $d \in C_x$.*

Demostración. Sean dos nodos x e y, con $y \in C_x$. Sea d un nodo perteneciente a la descendencia de y.
Sea x hijo izquierdo o centro de un nodo p_x, con C_x grupo de conflicto de x. Por lo visto en la sección 4.1.2.1, x tendrá conflicto exactamente con todo nodo k no adyacente a p_x y tal que $x < k < p_x$. Supongamos $d \notin C_x$ y descendiente de y. Como $y \in C_x$ vale que $x < y < p_x$. Luego d puede ser:

- O descendiente del hijo derecho de x
- O descendiente de un hermano derecho de x distinto del hijo derecho de p_x.

En ambos casos por definición de inorder $x < d < p_x$, entonces x tiene conflicto con d, es decir $d \in C_x$.

Sea x hijo derecho de un nodo p_x, con C_x grupo de conflicto de x. Por lo visto en la sección 4.1.2.1, x tendrá conflicto con toda la descendencia de sus nietos izquierdos. Como el conflicto es con toda la descendencia de los nietos, luego para todo nodo d perteneciente a la descendencia de los nietos izquierdos vale que $d \in C_x$. □

Por lo tanto, si recorremos el árbol desde la raíz hacia las hojas, una vez que llego a un nodo que está en un grupo de conflicto, luego todos los nodos por los que siga pasando también estarán en ese grupo.

Observación 4.1.4. La única forma de anidar grupos de conflicto es realizando movimientos hacia abajo en el árbol. Es decir, sea x un nodo conflictivo en G. Sea C_x su grupo de conflicto. El nivel de todo nodo perteneciente a C_x es menor estricto que el nivel de x.

Recordemos que $\alpha(i)$ es la cantidad máxima de grupos de conflicto anidados a los que pertenece un nodo i del grafo.

Teorema 4.1.2. *Sean x e y dos nodos en el grafo. Sea x alcanzable desde y haciendo solo uno de los movimientos de anidación. Luego $\alpha(x) = \alpha(y) + 1$.*

Demostración. Veamos que no es menor. Demostración por absurdo:
Sea x un nodo conflictivo, C_x su grupo de conflicto. Sea $k \in C_x$. Sea t un nodo al que llego haciendo alguno de los movimientos ya mencionados partiendo de k. Luego $t \in C_l$

con l un nodo conflictivo. Para que C_x y C_l no sean anidados entonces debería suceder que $l \notin C_x$. Absurdo. No puede suceder ya que una vez que estoy en un nodo que está en un grupo de conflicto, todos los nodos por los que siga pasando estarán en ese grupo de conflicto.

Veamos que no es mayor. Demostración por absurdo:
Supongamos que es mayor entonces eso quiere decir que haciendo un solo movimiento de anidación anidamos, por lo menos, dos grupos de conflicto más.
Supongamos que partimos de un nodo x en G. Como solo estamos haciendo un movimiento de anidación, vamos a tomar solamente el subárbol cuya raíz es x ya que suponer que existe otro nodo que genera un grupo de conflicto anidado fuera de este subárbol implica que hay más movimientos posibles.

- Sea el movimiento de anidación x - Izquierda - a -Derecha - b:
  Luego, veamos que b no pertenece a dos grupos de conflicto más que x:
  Sabemos que solo anidamos si nos movemos para abajo. Por lo tanto, tiene que haber dos nodos conflictivos entre x y b. Absurdo entre x y b solo esta a.

- Sea el movimiento de anidación x - Centro - a - Izquierda/Centro/Derecha - b
  Luego, b no pertenece a dos grupos de conflicto más que x por igual razonamiento que en el ítem anterior.

- Sea el movimiento de anidación x - Derecha - a - Derecha - b - Izquierda/Centro - c - Izquierda/Centro/Derecha - d:
  Luego, veamos que d no pertenece a dos grupos de conflicto más que x:
  Sabemos que solo anidamos si nos movemos para abajo. Por lo tanto, tiene que haber dos nodos conflictivos entre x y d. Absurdo $d \in C_x$ pero $a, b, c \notin C_x$.

Por lo tanto no anidamos más de un grupo de conflicto al realizar un movimiento de anidación.
Luego podemos asegurar que cada vez que entro a un grupo de conflicto y hago un movimiento de anidación, entonces anido un y solo un grupo de conflicto más. □

Teorema 4.1.3. *Los movimientos de anidación no se pueden superponer en más de 1 movimiento.*

Demostración. Veamos que si lo superponemos en más de uno entonces el nodo conflictivo no queda dentro de algún C_i.

Los movimientos de anidación son:

1. a - Izquierda - b - Derecha - c
2. a - Centro - b - Izquierda/Centro/Derecha - c
3. a - Derecha - b - Derecha - c - Izquierda/Centro - d - Izquierda/Centro/Derecha - e

Entre los primeros dos ítems no hay forma de superponer en más de uno. Entonces veamos las combinaciones entre los primeros dos con el tercero.

- Ítem 1 con ítem 3: El único movimiento en el cuál los puedo superponer en más de uno es si hacemos:
  a - Derecha - b - Derecha - c - Izquierda- d - Derecha - e
  En este caso puedo tener dos nodos conflictivos: b o d, pero $b \notin C_d$ y $d \notin C_b$, por lo tanto, no son anidados. Por lo tanto, no anido cuando se superponen en más de uno.
- Ítem 2 con ítem 3: Los movimientos en los cuales los puedo superponer en más de uno es si hacemos:
  1) a - Derecha - b - Derecha - c - Centro - d - Izquierda - e
  Sea f un hermano izquierdo de d. En este caso puedo tener dos nodos conflictivos: b o f, pero $f \notin C_b$ y $b \notin C_f$, por lo tanto, no son anidados. Por lo tanto, no anido cuando se superponen en más de uno.

  2) a - Derecha - b - Derecha - c - Centro - d - Centro - e
  Sea f un hermano izquierdo de d. En este caso puedo tener dos nodos conflictivos: b o f, pero $f \notin C_b$ y $b \notin C_f$, por lo tanto, no son anidados. Por lo tanto, no anido cuando se superponen en más de uno.

  3) a - Derecha - b - Derecha - c - Centro - d - Derecha - e
  En este caso puedo tener dos nodos conflictivos: b o d, pero $b \notin C_d$ y $d \notin C_b$, por lo tanto no son anidados. Entonces no anido cuando se superponen en más de uno.

Entonces si hacemos un movimiento de anidación que son dos superpuestos, es decir, alguno de los 4 movimientos anteriores sólo sumamos un grupo de conflicto anidado, no dos. □

4.1.4. InThinness

Definición 4.1.2. InThinness: Llamaremos InThinness a la thinness de un árbol cuyos nodos se ordenan con el método de Inorder.

A partir de ahora:
Sea un grafo G al que se le quiere calcular la InThinness, definimos el grafo G' como sigue:

- Por lo que observamos en 4.1.3, la raíz se comporta como un hijo derecho. Entonces vamos a agregar un nodo r' tal que el hijo derecho de r' será la raíz de G y no consideraremos la secuencia de movimientos Derecha - Izquierda/Centro - Izquierda/Centro/Derecha como una secuencia que aumente en uno la cantidad de grupos de conflicto anidados en el grafo.

- En G' si un nodo tiene al menos un hijo centro y no tiene hijo izquierdo, entonces lo modificaremos de forma tal que tenga como hijo izquierdo el hijo centro que se encuentre más a la izquierda. Si tenía otros centros estos quedan como centro. Si tenía uno solo ahora no tiene hijo centro.

Teorema 4.1.4. *InThinness(G) = InThinness(G')*

Demostración. Ya vimos que el segundo ítem no la altera. Veamos entonces que haciendo las modificaciones del primer ítem mantenemos igualdad entre la InThinness de G y de G'.

Veamos que no es mayor:
Supongamos, por el absurdo, que aumentó la InThinness agregando éste nodo. Lo que sucede es que entonces, este nodo genero un nuevo grupo de conflicto anidado. Entonces tiene que haber ahora una secuencia de movimientos que genere que haya un nuevo grupo de conflicto anidado. La única secuencia de movimientos que inicia con derecha es:
Derecha - Derecha - Izquierda/Centro - Izquierda/Centro/Derecha
Luego esta tiene que ser un secuencia nueva. Además esta secuencia debe empezar en el nodo r', el cual tiene como hijo derecho a r. Pero antes estábamos considerando la secuencia Derecha - Izquierda/Centro - Izquierda/Centro/Derecha que parte de r. Por lo tanto, no estamos agregando nuevas secuencias.

Veamos que no es menor:
Para que disminuya la InThinness tiene que suceder que haya al menos una secuencia de movimientos que generan que se agregue un nuevo grupo de conflicto anidado que ya no se pueda hacer. Esto no puede suceder ya que solo se agregó una arista, no se sacó ninguna. Por lo tanto, la InThinness no puede disminuir. □

4.1.4.1 Nodos que forman un completo en $G_<$

Vimos que hay algunas secuencias de movimientos que hacen que anide un grupo de conflicto más. Veamos para cada secuencia cuál es el nodo que se agrega como nodo conflictivo que luego será parte del completo en $G_<$

- a - Izquierda - b - Derecha - c: Parto del nodo a, me muevo a la izquierda llegando al nodo b. Luego desde b me muevo a la derecha llegando así al nodo c. En este caso el nodo conflictivo es b. C_b es su grupo de conflicto y sabemos que $C_b \neq \emptyset$ ya que $c \in C_b$. Luego b será un nodo que formará parte del completo en $G_<$. Además $a, b \notin C_b$.

- a - Centro - b - Izquierda/Centro - c: Parto del nodo a, me muevo a algún centro llegando al nodo b. Luego desde b me muevo hacia la izquierda o a cualquier centro llegando así al nodo c. Sea d un hermano izquierdo de b, luego d será el nodo conflictivo, C_d es su grupo de conflicto y sabemos que $C_d \neq \emptyset$ ya que $c \in C_d$. Luego d será un nodo que formará parte del completo en $G_<$. Además $a, b \notin C_d$.

- a - Centro - b - Derecha - c: Parto del nodo a, me muevo a algún centro llegando al nodo b. Luego desde b me muevo hacia la derecha llegando así al nodo c. Luego b será el nodo conflictivo, C_b es su grupo de conflicto y sabemos que $C_b \neq \emptyset$ ya que $c \in C_b$. Luego b será un nodo que formará parte del completo en $G_<$. Además $a, b \notin C_b$.

- a - Derecha - b - Derecha - c - Izquierda/Centro - d - Izquierda/Centro/Derecha - e: Parto del nodo a, me muevo a la derecha llegando al nodo b. Luego desde b me muevo nuevamente hacia la derecha llegando así al nodo c. Luego me muevo a la izquierda o a algún centro y llegamos al nodo d. Por último realizo cualquier movimiento (es decir, me muevo hacia la derecha, hacia la izquierda o hacia cualquier centro). Luego b será el nodo conflictivo, C_b es su grupo de conflicto y sabemos que $C_b \neq \emptyset$ ya que $e \in C_b$. Luego b será un nodo que formará parte del completo en $G_<$. Además $a, b, c, d \notin C_b$.

Hasta acá solo tengo tantos nodos en $G_<$ pertenecientes a la clique máxima como grupos de conflicto anidados tenga. Luego el último nodo perteneciente al completo máximo es el último al que haya llegado. Este nodo pertenecerá a todos los C_i ya que sabemos que una vez que entro a uno no puedo salir y sabemos que existe porque todos los movimientos terminan en un nodo perteneciente al grupo de conflicto y además ése nodo no fue agregado a $G_<$ anteriormente (ningún movimiento termina en un nodo que es el conflictivo).

4.1.4.2 Calculando la InThinness

Teorema 4.1.5. *Sea $\#h$ la cantidad de hojas del grafo árbol G. Sea r la raíz en G. Sea h_i la i-ésima hoja, con $0 \leq i < \#h$. Sea $GDCA_i$ la cantidad de grupos de conflicto anidados en el camino de la raíz a la i-ésima hoja. Entonces la cantidad máxima de grupos de conflicto anidados en el grafo árbol inorder es $1 + \max_{0 \leq i < \#h} GDCA_i$*

Demostración. Sea G el grafo al que le agregamos un nodo raíz cuyo hijo derecho es la raíz del grafo árbol al que le queremos calcular la InThinness.
Sean h grupos de conflicto anidados. Cuando anidamos un grupo más, aumentamos en uno el completo máximo en $G_<$. Por lo tanto la InThinness aumenta en uno.
Sea 1 grupo de conflicto, la InThinness es 2, ya que tenemos un K_2 en $G_<$.
Luego, si cada vez que entramos a un grupo de conflicto sumamos uno a la InThinness y después incrementamos ese valor en uno (ya que un grupo de conflicto genera InThinness 2), entonces tenemos la InThinness de G.
Como la única forma de entrar en un grupo de conflicto es realizando alguno de los movimientos ya mencionados, y solo anidamos si hacemos el recorrido de la raíz a una hoja entonces podemos decir que si sumamos uno cada vez que realizamos esos movimientos (a lo sumo superpuestos en uno) en el recorrido de la raíz a una hoja y luego incrementamos ese valor en uno tendremos el valor de la InThinness de G.
Entonces si recorro todos los caminos de raíz a hoja, alguno(s) de ellos va(n) a ser el/los que tenga(n) mayor cantidad de grupos de conflicto anidados, y estos serán los que tengan la suma máxima. Luego tenemos que
$\text{InThinness}(G) = 1 + \max_{0 \leq i < \#h} GDCA_i$ □

Más adelante veremos una forma de calcular este valor sin necesidad de recorrer todos los caminos de raíz a hoja utilizando la técnica de divide and conquer.

4.2. Cota para la InThinness

Teorema 4.2.1. *La thinness de un árbol G está acotada por la altura.*

Demostración. La thinness es la cantidad máxima de grupos de conflicto anidados incrementado en uno. Sabemos que solo anidamos cuando nos movemos hacia abajo en el árbol y solo sumo de a uno, es decir, nunca puedo sumar más de los que recorro. Entonces la altura es mayor o igual a la cantidad de grupos de conflictos anidados. Sea h la altura del árbol,

$h \geq$ cantidad de grupos de conflicto anidados = $thin(G)$ - 1, entonces

$h \geq thin(G) - 1$, equivalentemente $h + 1 \geq thin(G)$.

Además sabemos que el movimiento más corto que anida grupos de conflicto tiene tamaño dos, es decir, requiere dos desplazamientos en el árbol, entonces

$h \geq thin(G)$.

□

Como la InThinness es una cota superior para la thinness, tenemos el siguiente resultado.

Corolario 4.2.1. *La thinness de un árbol está acotada por su altura.*

4.2.1. Comparando con la cota existente

Fue demostrado que la cota de la thinness para un árbol m-ario completo con m un valor fijo, es $\Theta(log(n))$ [BE18].

Ahora veamos que la cota para grafos árboles m-arios completos, con m un valor fijo, numerado con el método inorder cumple con $\Theta(log(n))$.

Como se trata de grafos árbol completo, $h = log(n)$, con n la cantidad de nodos. Por lo tanto cumple con la cota $\Theta(log(n))$.

4.3. Algoritmo para encontrar la InThinness

4.3.1. En árboles binarios

Sabemos que solo sumo uno cuando me muevo en un movimiento de anidación, que los movimientos de anidación son 2 (son los mismos que para árboles m-arios pero sin centros) y no se pueden superponer en más de uno.

El algoritmo utilizará la técnica de Divide and Conquer. Por lo tanto, lo que debemos pensar es, dado un nodo x, qué información necesitamos de cada hijo para determinar la InThinness del subárbol con raíz x.

Sea G el grafo árbol binario. Sea x un nodo en G con k_0, k_1 sus hijos. Sea $i \in \{0, 1\}$, x va a recibir por cada uno de sus hijos:

- El valor de la InThinness para el subárbol que tiene como raíz al nodo k_i, $thin(G_{k_i})$.
- Una lista de secuencias de movimientos. Cada una de estas secuencias son posibles movimientos desde k_i hasta una hoja. Además se cumple que si recorremos el subárbol empezando en k_i y realizando los movimientos en orden indicados en alguna secuencia, luego habremos anidado exactamente $thin(G_{k_i}) - 1$ grupos de conflicto. Por ejemplo, sea el subárbol con raíz en k_i un árbol completo de 7 nodos. Luego k_i recibirá de cada uno de sus hijos la lista $[(izquierda), (derecha)]$.

Luego con esa información, el nodo x agrega para toda secuencia proveniente del hijo derecho, un movimiento "derecho"y para cada secuencia que proviene del hijo izquierdo agrega "izquierdo".
Ahora nos fijamos para cada hijo, para cada una de las secuencias de movimientos, si se generó un movimiento de anidación. Es decir, si el movimiento de anidación es prefijo en la secuencia. Si lo es, entonces incrementamos la $thin(G_{k_i})$ en uno y dejamos solo las secuencias de movimientos conflictivos. Luego nos quedamos solo con las secuencias de conflicto que generaron la mayor InThinness.

4.3.1.1 Cota para la lista de secuencias

Sabemos que para inorder:

- Existen únicamente 2 posibles movimientos de anidación.
- El movimiento de anidación más grande tiene 4 movimientos.
- Solo se pueden superponer movimientos de anidación en un movimiento.

Sea x un nodo en G con k_0, k_1 sus hijos. Sea $i \in \{0, 1\}$. x solo necesitará saber como máximo los últimos 3 últimos movimientos hasta llegar al k_i.
Esto hace que cada secuencia en la lista tenga a lo sumo 3 movimientos. Como no necesitamos repetidos en la lista de secuencias, el máximo tamaño que puede tener esa lista es 2^3 ya que existen dos movimientos y cada lista tiene tamaño máximo 3.

4.3.1.2 Complejidad del algoritmo

Vimos que pasamos una vez por cada nodo por lo que partimos de una complejidad de $O(n)$, con n la cantidad de nodos del grafo.
Para cada nodo debemos:

- Agregar un movimiento a cada secuencia de movimientos.
- Chequear si alguna de las secuencias de movimientos es conflictiva.
- Juntar todas las que generan la mayor InThinness eliminando repetidos.

Los tres ítems trabajan sobre la lista de secuencias de movimientos que tiene un tamaño fijo, por lo tanto la complejidad de procesamiento por nodo es $O(1)$.
Luego, la complejidad final del algoritmo es $O(n)$.

4.3.2. En árboles m-arios

Sabemos que:

- Solo sumo uno cuando me muevo en un movimiento de anidación.
- Los movimientos de anidación son 3.
- Los movimientos de anidación no se pueden superponer en más de un movimiento.

Sea G el árbol m-ario. Sea x un nodo en G con k_0, k_1, ..., k_l hijos.
x va a recibir por cada uno de sus hijos:

- El valor de la InThinness para el subárbol que tiene como raíz al nodo k_i $thin(G_{k_i})$.
- Una lista de secuencias de movimientos. Cada una de estas secuencias son posibles movimientos desde k_i hasta una hoja. Además se cumple que si recorremos el subárbol empezando en k_i y realizando los movimientos en orden indicados en alguna secuencia, luego habremos anidado exactamente $thin(G_{k_i}) - 1$ grupos de conflicto. Por ejemplo, sea G_{k_i} un árbol completo de 7 nodos. Luego k_i recibirá de cada uno de sus hijos la lista $[(izquierda), (derecha)]$.

Luego con esa información, el nodo x agrega para toda secuencia proveniente del hijo derecho, un movimiento "derecho". Para cada secuencia que proviene de cada hijo centro agrega "centro"y para cada secuencia que proviene del hijo izquierdo agrega "izquierdo". Ahora nos fijamos para cada hijo, para cada una de las secuencias de movimientos, si se trata de una secuencia de movimientos de conflicto. Si lo es, entonces incrementamos la $thin(G_{k_i})$ en uno dejando solo las secuencias de movimientos conflictivos. Luego nos quedamos solo con las secuencias de conflicto que generaron la mayor InThinness.

4.3.2.1 Cota para la lista de secuencias

Sabemos que para inorder:

- Existen únicamente 3 posibles movimientos de anidación.
- El movimiento de anidación más grande tiene 4 movimientos.
- Solo se pueden superponer movimientos en uno solo.

Por lo tanto, dado un nodo x en G, solo necesito saber como máximo los últimos 3 últimos movimientos hasta llegar al k_i.
Esto hace que la lista de secuencias tenga a lo sumo 3^3.

4.3.2.2 Complejidad del algoritmo

Vimos que pasamos una vez por cada nodo por lo que partimos de una complejidad de $O(n)$, con n la cantidad de nodos del grafo.
Veamos ahora cual es la complejidad de procesamiento de cada nodo sabiendo que:

- Las secuencias no pueden tener más de 3 movimientos.
- La lista de secuencias no puede tener más de 3^3 secuencias, ya que no deben haber repetidos.
- Cada nodo recibe m de estas listas, una por cada hijo.

Luego lo que deberá hacer es, para cada hijo:

- Agregar un movimiento al inicio de cada secuencia de cada lista. El movimiento que agrega es el movimiento necesario para pasar del nodo al hijo asociado a esa lista.
- Para cada secuencia en la lista ver si se generó un movimiento de anidación. Es decir, ver si el inicio de la lista coincide con algún movimiento de anidación.
- Si existe secuencia que genere un nuevo movimiento de anidación, entonces remover las secuencias que no generan uno y para cada secuencia que generó un nuevo movimiento de anidación, dejar solo el primer movimiento, es decir, el último agregado ya que solo se pueden superponer en uno. Además, aumentar en uno el valor de la InThinness y eliminar secuencias repetidas.
- Si no existe secuencia que genere un nuevo movimiento de anidación, entonces dejar todas las secuencias recortándolas de manera que ninguna tenga más de 3 elementos. Si alguna tiene más de 3 elementos, entonces los últimos serán eliminados, es decir, dejando solo los movimientos más recientemente agregados.

Luego se compararán los valores de la InThinness para cada subárbol correspondiente a cada hijo. Para los que tengan la mayor de las InThinness se concatenarán las listas de secuencias eliminando repetidos y se dejará el valor de dicha InThinness.
Como todas las operaciones se realizan sobre la lista de secuencias de movimientos que tiene un tamaño acotado la complejidad de procesamiento de cada nodo será $O(m)$.
Como esto se hace por cada hijo, la complejidad del algoritmo será $O(m * n)$.

4.4. Optimizaciones para la InThinness

Dado un árbol podemos elegir como designar hijos derechos, centros e izquierdos para minimizar la InThinness sin alterar las adyacencias.
Sea G el grafo árbol m-ario sobre el cual se quiere calcular la InThinness. Sea x un nodo en el grafo G con h hijos, entonces

- Si $h = m$ entonces no podemos modificar nada, ya que no hay posibilidad de alteración entre hijos.
- Si $h = 1$, $m > 1$ entonces podemos decidir si G tendrá hijo derecho o hijo izquierdo.
- Si $m > 2$, $h < m$ entonces podemos decidir si G tendrá hijo derecho o hijo centro. Ya vimos que no vamos a tener hijo centro si no hay hijo izquierdo.

Ya vimos que la InThinness de un grafo es uno más que la cantidad de grupos de conflictos anidados. Por lo tanto el objetivo será realizar las modificaciones necesarias para minimizar la cantidad de grupos de conflictos anidados.

Como vimos anteriormente, para árboles con numeración inorder existen diferentes secuencias de movimientos que generan que se anide un nuevo grupo de conflicto. Nuestro objetivo será entonces desarmar estas secuencias de movimientos.
Sea G el grafo árbol m-ario sobre el cual se quieren realizar estas modificaciones para minimizar la InThinness. Sean x y p_x dos nodos del grafo, donde p_x es padre de x. Tenemos tres casos para analizar:

- Cuando x es hijo izquierdo
- Cuando x es hijo centro
- Cuando x es hijo derecho

Y se quiere decidir para cada nodo hijo de x si será derecho, centro o izquierdo.
Sea x un nodo del grafo G con h hijos, donde $h < m$. Sea x:

- Hijo izquierdo: Como ya vimos, x va a tener conflicto con todo su subárbol derecho. Por lo tanto sabemos que derecho no conviene. Sabemos además que no va a haber hijo centro sin que haya izquierdo. Entonces alcanza con decir que x no va a tener hijo derecho.

- Hijo centro: Como ya vimos, x va a tener conflicto con todo su subárbol derecho. Por lo tanto sabemos que derecho no conviene. Sabemos además que no va a haber hijo centro sin que haya izquierdo. Entonces alcanza con decir que x no va a tener hijo derecho. Además, si x tiene un hermano izquierdo, llamémoslo i, luego i tiene conflicto con todo hijo de x. Por lo tanto no hay forma de mejorar el grupo de conflicto generado con i como nodo conflictivo.

- Hijo derecho:
  Sea l uno de los hijos de x. Podemos observar que:

  - Si asignamos a l como hijo izquierdo, luego si l tiene un hijo derecho, llamémoslo dl entonces generamos un nuevo grupo de conflicto con l nodo conflictivo y $dl \in C_l$.
  - Si asignamos a l como hijo centro, luego si l tiene un hijo derecho, llamémoslo dl entonces generamos un nuevo grupo de conflicto con l nodo conflictivo y $dl \in C_l$.
  - Si asignamos a l como hijo derecho, luego no importa qué hijos tenga l, no generamos conflicto alguno.

  Luego podemos decir que si x es hijo derecho, nos conviene que tenga hijo derecho. Sabemos además que no va a haber hijo centro sin que haya izquierdo. Entonces alcanza con decir que x va a tener hijo derecho.

4.5. Experimentación

A continuación, se presentan las pruebas experimentales que se realizaron con el fin de evaluar y comparar los diferentes métodos ya presentados, junto con los resultados obtenidos y una discusión de los mismos.

4.5.1. Experimento 1: Pruebas de rendimiento

Este experimento se realizó con el objetivo de comparar el rendimiento entre el algoritmo presentado en este trabajo en la sección 4.3 y el algoritmo propuesto en [BE18] teniendo en cuenta el tiempo insumido para la ejecución de cada uno de ellos. Se realizaron tres tipos de pruebas:

- Rendimiento en función de la altura del árbol, utilizando árboles completos.

- Rendimiento en función de la cantidad máxima de hijos que puede tener un nodo, utilizando árboles completos.
- Rendimiento en función de la altura del árbol, utilizando árboles Zig-Zag.

Para ello primero se variará la altura del árbol manteniendo constante el tipo de árbol y la cantidad máxima de hijos que puede tener un nodo. Para el segundo ítem se dejará constante la altura y el tipo de árbol y se variará la máxima cantidad de hijos que puede tener un nodo. Luego tomaremos árboles de tipo Zig-Zag y variaremos la altura del mismo. Para cada una de las variaciones se medirá el tiempo de ejecución de cada uno de los métodos.
Con el fin de mejorar la precisión de los resultados y evitar posibles interferencias (por ejemplo, las generadas por otras tareas que pueda estar ejecutando la computadora) cada medición se repitió 5 veces, tomando luego el desviación estándar de los datos obtenidos. En todos los casos, las medidas se eligieron de forma arbitraria, procurando que los casos de prueba no fueran excesivamente grandes para no prolongar innecesariamente la duración de las pruebas. Para la cantidad máxima de hijos por nodo, cuando la misma debe ser constante, se eligió 3 y para la altura de árbol, cuando ésta debe ser constante se eligió 5, números elegidos arbitrariamente, ya que no se consideró que este parámetro fuera de relevancia para el experimento.

4.5.1.1 Resultados

En cada gráfico se muestran las curvas representando el tiempo insumido por el algoritmo propuesto en la sección 4.3 ($GDCA$) y el algoritmo exhibido en [BE18] ($G_<$) con respecto a la cantidad de nodos del grafo. Además, se muestran las curvas $f(x) = x$ y $f(x) = x^3$ representando las complejidades del algoritmo introducido en la sección 4.3 y el algoritmo propuesto en [BE18] respectivamente.
En el título de cada gráfico indica el tipo de árbol, el orden utilizado y la variable que se mantiene constante seguida del valor fijo utilizado. Notar que en un grafo árbol Zig-Zag la cantidad de hijos máxima es 2. m representa la máxima cantidad de hijos por nodo y h la altura del árbol.
En todos los gráficos el eje Y está en escala logarítmica.

4.5.1.2 Discusión

Como se puede observar en los resultados, pudo verificarse que el algoritmo propuesto en la sección 4.3 es más eficiente. Esto es así ya que tanto al comparar el tiempo de ejecución

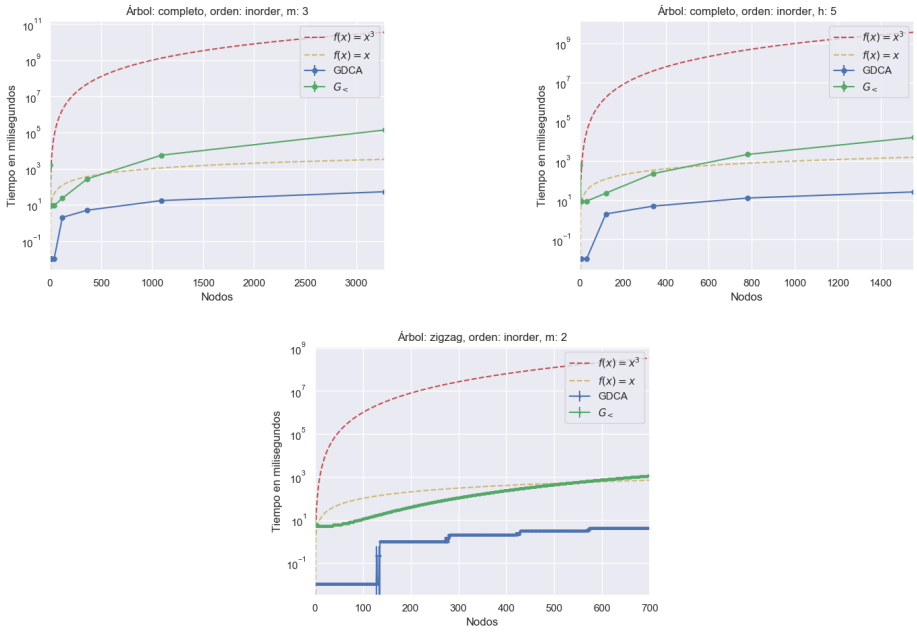


Fig. 4.4: Comparando tiempos entre diferentes algoritmos

variando la altura y variando la cantidad máxima de hijos por nodo, como cuando se comparó entre árboles de tipo Zig-Zag con diferentes alturas, el tiempo de ejecución del algoritmo introducido en la sección 4.3 es menor.
También se puede observar que el método propuesto en la sección 4.3 tiene complejidad lineal en la cantidad de nodos; no se considera necesario agregar más pruebas con diferente cantidad de nodos, otros tipos de árboles o variaciones en la máxima cantidad de hijos por nodos.

4.5.2. Experimento 2: InThinness optimizada

El objetivo de este experimento fue contrastar los resultados arrojados por el algoritmo presentado en la sección 4.3 cuando se realizan las optimizaciones propuestas en la sección 4.4 antes de calcular la InThinness y cuando no se hacen optimizaciones.
Como vimos anteriormente, sea x un nodo en el grafo, las optimizaciones posibles a realizar son:

- Si x es hijo derecho, entonces conviene que tenga hijo derecho.
- Si x es hijo izquierdo, entonces no conviene que tenga hijo derecho.

Si tenemos un grafo árbol Zig-Zag, podremos ver de forma clara como funcionan las optimizaciones, ya que para todo nodo es realizable una optimización. Por lo tanto tomaremos

como parámetros de entrada, grafos árbol de tipo Zig-Zag con diferentes alturas.

4.5.2.1 Resultados

En el gráfico se muestran las curvas representando el valor de la InThinness obtenida con el algoritmo propuesto en la sección 4.3 sin realizar las optimizaciones (*Sin optimizaciones*) y el valor de la InThinness realizando las optimizaciones propuestas en la sección 4.4 (*Con optimizaciones*).

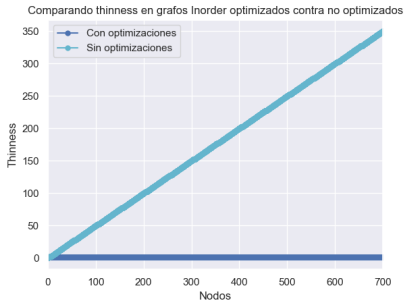


Fig. 4.5: Comparando InThinness de grafos optimizados y no optimizados.

4.5.2.2 Discusión

Del análisis de los valores obtenidos para la InThinness, pudo desprenderse que, al realizar las optimizaciones que se muestran en la sección 4.4 el valor de la InThinness es menor que cuando no se corren.
Dado que el valor de la InThinness mejoró, a continuación vamos a analizar los tiempos de ejecución entre el método propuesto por [BE18], el método introducido en la sección 4.3 y el tiempo total de correr las optimizaciones sumado con el tiempo de obtener la InThinness utilizando el algoritmo propuesto en esta tesis.

4.5.3. Experimento 3: Tiempos con InThinness optimizada

En este experimento se buscó comparar los tiempos de ejecución entre el método propuesto por [BE18], el método presentado en la sección 4.3 y el tiempo total de correr las optimizaciones sumado con el tiempo de obtener la InThinness utilizando el algoritmo propuesto en esta tesis. Para ello tomaremos los mismos grafos de entrada que para el Experimento 1.

4.5.3.1 Resultados

En cada gráfico se muestran las curvas representando el tiempo insumido por el algoritmo propuesto en la sección 4.3 con optimizaciones (*GDCA Optimizado*) y sin optimizaciones (*GDCA*) y el algoritmo exhibido en [BE18] ($G_<$) con respecto a la cantidad de nodos del grafo. Además, se muestran las curvas $f(x) = x$ y $f(x) = x^3$ representando las complejidades del algoritmo introducido en la sección 4.3 y el algoritmo propuesto en [BE18] respectivamente.

En el título de cada gráfico indica el tipo de árbol, el orden utilizado y la variable que se mantiene constante seguida del valor fijo utilizado. Notar que en un grafo árbol Zig-Zag la cantidad de hijos máxima es 2. m representa la máxima cantidad de hijos por nodo y h la altura del árbol.

En todos los gráficos el eje Y está en escala logarítmica.

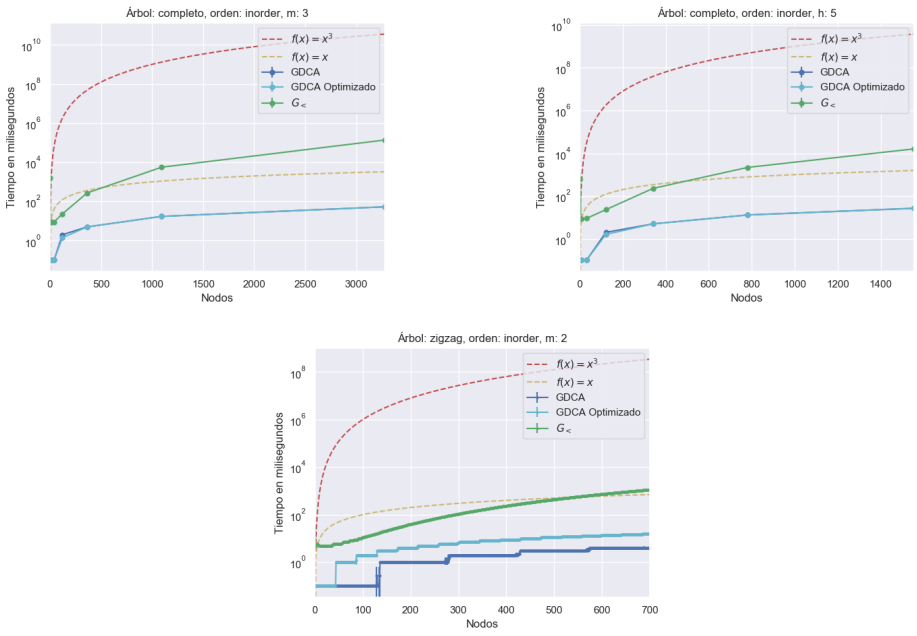


Fig. 4.6: Comparando tiempos entre diferentes algoritmos

4.5.3.2 Discusión

Como se puede observar en los resultado, tanto cuando se corren las optimizaciones como cuando no se corren, los tiempos de ejecución son menores que el algoritmo cúbico.

Dado que cuando se corren las optimizaciones aumenta el tiempo de ejecución pero disminuye la InThinness, la elección de uno u otro deberá ser evaluada cuidadosamente a partir de las preferencias particulares del caso en el que se esté utilizando.

4.6. Conclusiones

Como vimos, el algoritmo propuesto en la sección 4.3 es más rápido que el propuesto por [BE18], por lo tanto entre estos dos métodos es clara la conveniencia de uno sobre el otro. No es así en el caso de las optimizaciones, que a pesar de que suceda que optimizando el gráfico antes de calcular la InThinness es más eficiente que el algoritmo exhibido en [BE18], correr antes las optimizaciones hacen que tome más tiempo el cálculo de la InThinness. Por lo tanto correr o no las optimizaciones dependerá del caso de uso en particular ya que con optimizaciones o sin, ambos resultados son cotas superiores para la InThinness. De todas formas, tanto las complejidades de realizar las optimizaciones como calcular la InThinness con el algoritmo presentado en esta tesis son $O(n)$. Por lo tanto optimizar el grafo y luego calcular la InThinness no aumenta el grado de complejidad.

5. PREORDER

5.1. Thinness para árboles Preorder

Por el Teorema 3.1.1 vimos que la thinness en un grafo G es uno más que la máxima cantidad de grupos de conflicto anidados que hay en G. Por lo tanto, para hallar la thinness en un grafo árbol, vamos a hallar máxima cantidad de grupos de conflicto anidados que hay en G. En este capítulo vamos a analizar como hacer esto en un grafo G numerado con el método Preorder, es decir, donde $<$ es Preorder. Primero lo haremos para árboles binarios y luego lo mostraremos para árboles m-arios.

5.1.1. Árboles binarios

5.1.1.1 Cómo se forman los grupos de conflicto

Sabemos por definición de preorder que para cualquier nodo x vale que:

- Todo nodo en el subárbol izquierdo o derecho es mayor que x
- El nodo más grande adyacente a x es su hijo derecho. Llamémoslo d.
- Todo nodo en el subárbol izquierdo de x es menor que d.
- Ningún nodo en el subárbol izquierdo de x es adyacente a d por definición de árbol.
- El nodo más chico del subárbol derecho de x es d. Por lo tanto no puede tener conflicto con ningún nodo de éste subárbol.

Por lo tanto, como no puede tener conflicto con ningún nodo del subárbol derecho, y tiene conflicto con todo nodo del subárbol izquierdo, el grupo de conflicto de x (C_x) será todo nodo perteneciente a su subárbol izquierdo si x tiene hijo derecho. En caso contrario x no tiene conflicto con ningún nodo.

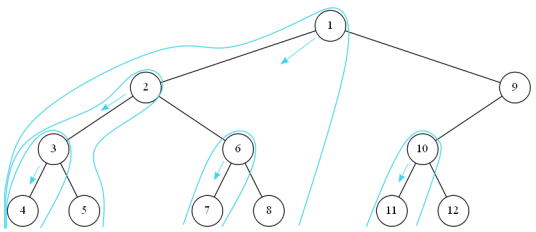


Fig. 5.1: Grupos de conflicto en un árbol binario.

5.1.2. Árboles m-arios

Por Observación 2.3.1 vamos a considerar, sin perdida de generalidad, que solo puede haber hijos centro si ya existe uno izquierdo y uno derecho.

5.1.2.1 Cómo se forman los grupos de conflicto

Sabemos por definición de preorder que para cualquier nodo x vale que:

- Todo nodo en el subárbol izquierdo, derecho o cualquiera de los subárboles centros es mayor que x.
- El nodo más grande adyacente a x es su hijo derecho. Llamémoslo d.
- Todo nodo en el subárbol izquierdo o en cualquiera de los subárboles centros de x es menor que d.
- Ningún nodo en el subárbol izquierdo o un cualquiera de los subárboles centros de x es adyacente a d por definición de árbol.
- El nodo más chico del subárbol derecho de x es d. Por lo tanto no puede tener conflicto con ningún nodo de éste subárbol.

Por lo tanto, como no puede tener conflicto con ningún nodo del subárbol derecho, y tiene conflicto con todo nodo del subárbol izquierdo y de todo subárbol centro, el grupo de conflicto de x (C_x) será todo nodo perteneciente a su subárbol izquierdo y a cualquier subárbol centro si x tiene al menos dos hijos. Si x tiene un único hijo no tiene conflicto con ningún nodo.

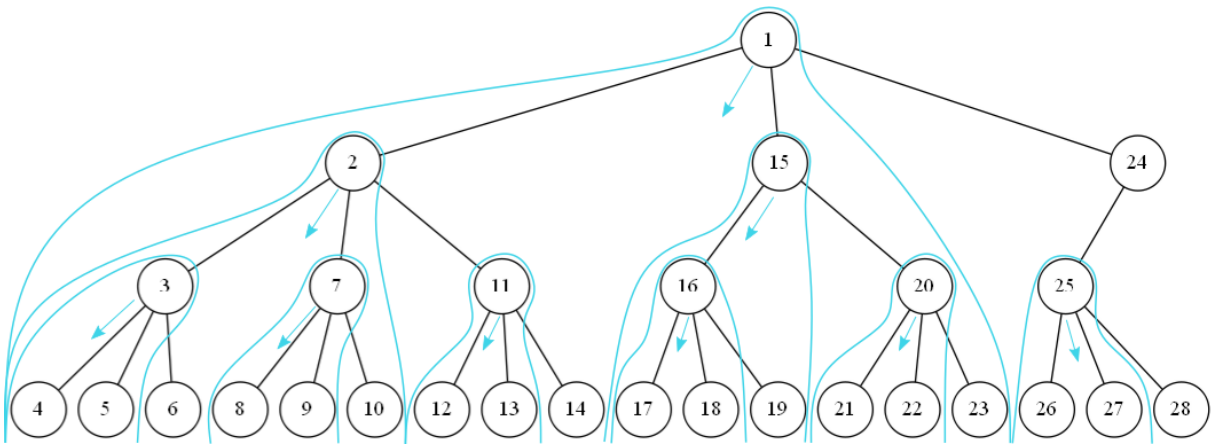


Fig. 5.2: Grupos de conflicto en un árbol ternario.

5.1.3. Grupos de conflicto y movimientos en el árbol

Ahora vamos a hablar de árboles m-arios. Si es binario, no hay hijos centro pero vale lo mismo que para los m-arios sabiendo que el conjunto de hijos centros es vacío.

5.1.3.1 Forma de entrar a un grupo de conflicto

Veamos que entonces se tiene la siguiente forma de entrar en un grupo de conflicto:

- Dado un nodo x en el grafo con por lo menos dos hijos (esto asegura que hay un hijo derecho), si me muevo al hijo izquierdo o a algún centro entonces llego a un nodo perteneciente a un nuevo grupo de conflicto.

En la figura 5.3 podemos observar ejemplos de algunos de los movimientos. Estos se encuentran marcados con color.

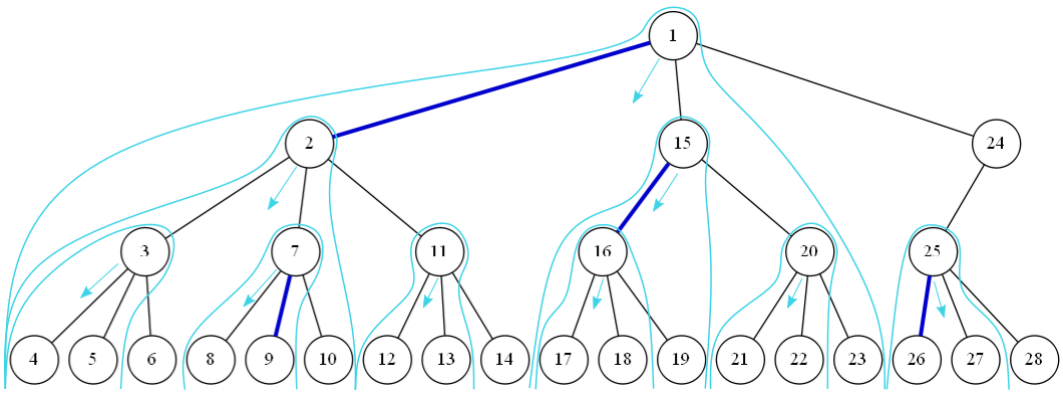


Fig. 5.3: Formas de entrar a los grupos de conflicto.

Definición 5.1.1. Movimiento de anidación: Llamaremos movimiento de anidación al movimiento mencionado arriba.

5.1.3.2 Anidando grupos de conflicto

Teorema 5.1.1. *La única forma de anidar grupos de conflicto es realizando movimientos hacia abajo en el árbol.*

Demostración. Sean x e y dos nodos en el grafo. Sea C_x el grupo de conflicto de x. Si $y \in C_x$, entonces y es descendiente de x. Demostración por absurdo:
Supongamos que y no es descendiente de x pero sí pertenece a C_x. Entonces quiere decir que x tiene conflictos con un nodo fuera de su descendencia. Pero por definición de preorder sabemos que x solo tiene conflictos con todo nodo perteneciente a sus subárboles derecho y centros. Absurdo. y no pertenece a la descendencia de x. □

Teorema 5.1.2. *Sean dos nodos x e y, con $y \in C_x$. Sea d un nodo perteneciente a la descendencia de y, entonces $d \in C_x$.*

Demostración. Sean dos nodos x e y, con $y \in C_x$. Sea d un nodo perteneciente a la descendencia de y. Por lo visto en la sección 5.1.2.1, x tendrá conflicto con todo nodo perteneciente al subárbol izquierdo y centro. Luego para todo nodo d perteneciente a la descendencia del hijo izquierdo o de cualquier hijo centro vale que $d \in C_x$. □

Por lo tanto si recorremos el árbol desde la raíz hacia las hojas, una vez que llego a un nodo que está en un grupo de conflicto, luego todos los nodos por los que siga pasando también estarán en ese grupo.

Teorema 5.1.3. *Sean x e y dos nodos en el grafo. Sea x alcanzable desde y haciendo el movimiento de anidación. Luego $\alpha(x) = \alpha(y) + 1$.*

Demostración. Veamos que no es menor. Demostración por absurdo:
Sea k un nodo conflictivo, C_k su grupo de conflicto. Sea $y \in C_k$. Sea x un nodo al que llego haciendo el movimiento conflictivo desde y. Luego $x \in C_l$ con l un nodo conflictivo. Para que C_k y C_l no sean anidados entonces debería suceder que $l \notin C_x$. Absurdo. No puede suceder ya que una vez que estoy en un nodo que esta en un grupo de conflicto, todos los nodos por los que siga pasando estarán en ese grupo de conflicto.

Veamos que no es mayor. Demostración por absurdo:
Supongamos que es mayor entonces eso quiere decir que haciendo el movimiento de anidación anidamos dos grupos de conflicto o más.
Supongamos que partimos de un nodo y que tiene por lo menos dos hijos. Como solo estamos haciendo un movimiento de anidación, vamos a tomar solamente el subárbol cuya raíz es y ya que suponer que existe otro nodo que genera un grupo de conflicto anidado fuera de este subárbol implica que hay más movimientos posibles.

- Sea el movimiento de anidación y - Izquierda/Centro - x:
  Luego, veamos que x no pertenece a dos grupos de conflicto más que y:
  Sabemos que solo anidamos si nos movemos para abajo. Por lo tanto tiene que haber dos nodos conflictivos entre y y x. Absurdo entre y y x no hay nodos.

Por lo tanto no anidamos más de un grupo de conflicto al realizar un movimiento de anidación.
Luego podemos asegurar que cada vez que entro a un grupo de conflicto y hago el movimiento de anidación, entonces anido un y solo un grupo de conflicto más. □

5.1.4. PreThinness

Definición 5.1.2. PreThinness: Llamaremos PreThinness a la thinness de un árbol cuyos nodos se ordenan con el método de preorder.

A partir de ahora:
Dado un grafo G al que se le quiere calcular la PreThinness, lo alteraremos de las siguientes formas:

- En G, si un nodo tiene al menos un hijo centro y no tiene hijo izquierdo, entonces lo modificaremos de forma tal que tenga como hijo izquierdo el hijo centro que se encuentre más a la izquierda. Si tenia otros centros estos quedan como centro. Si tenia uno solo ahora no tiene hijo centro.
- En G, si un nodo tiene al menos un hijo centro y no tiene hijo derecho, entonces lo modificaremos de forma tal que tenga como hijo derecho el hijo centro que se encuentre más a la derecha. Si tenia otros centros estos quedan como centro. Si tenia uno solo ahora no tiene hijo centro.
- En G, si un nodo tiene un hijo, entonces éste será hijo izquierdo.

Llamaremos G' al grafo resultante de realizar estas modificaciones sobre G.

Observación 5.1.1. PreThinness(G) = PreThinness(G').

5.1.4.1 Nodos que forman un completo en $G_<$

Vimos que existe una secuencia de movimientos que hacen que anide un grupo de conflicto más. Veamos cuál es el nodo que se agrega como nodo conflictivo que luego será parte del completo en $G_<$:

- a - Izquierda/Centro - b, donde a tiene hijo derecho: Parto del nodo a, me muevo al hijo izquierdo o a algún nodo que sea hijo centro, llegando al nodo b. En este caso el nodo conflictivo es a. C_a es su grupo de conflicto y sabemos que $C_a \neq \emptyset$ ya que $b \in C_a$. Luego a será un nodo que formará parte del completo en $G_<$.

Dado que solo hicimos un movimiento, solo tenemos un nodo en $G_<$. El otro nodo perteneciente a la clique máxima es el último al que haya llegado en el recorrido de la raíz a una hoja. Este nodo pertenecerá a todos los C_i ya que sabemos que una vez que entro a uno no puedo salir. Sabemos que existe porque todos los movimientos terminan en un nodo perteneciente al grupo de conflicto y además a ése nodo no lo agregué al $G_<$ anteriormente.

Teorema 5.1.4. *No se pueden superponer los movimiento de anidación.*

Demostración. Dado que el movimiento de anidación tiene un solo movimiento, entonces superponer significaría sumar dos haciendo un solo movimiento. Absurdo. Ya vimos que solo sumamos un grupo de conflicto anidado cada vez que realizamos el movimiento de anidación. □

5.1.4.2 Calculando la PreThinness

Teorema 5.1.5. *Sea $\#h$ la cantidad de hojas del grafo árbol G. Sea r la raíz en G. Sea h_i la i-ésima hoja, con $0 \leq i < \#h$. Sea $GDCA_i$ la cantidad de grupos de conflicto anidados en el camino de la raíz a la i-ésima hoja. Entonces la cantidad máxima de grupos de conflicto anidados en el grafo árbol preorder es $1 + \max_{0 \leq i < \#h} GDCA_i$*

Demostración. Sea G el grafo árbol al que le hicimos las modificaciones mencionadas para que siempre haya hijo izquierdo, después derecho y después centros.
Sean h grupos de conflicto anidados. Cuando anidamos un grupo más, aumentamos en uno el tamaño de la clique máxima en $G_<$. Por lo tanto la PreThinness aumenta en uno.
Sea 1 grupo de conflicto, la PreThinness es 2, ya que tenemos un K_2 en $G_<$.
Luego, si cada vez que entramos a un grupo de conflicto sumamos uno a la PreThinness y después incrementamos ese valor en uno (ya que un grupo de conflicto genera PreThinness 2), entonces tenemos la PreThinness de G.
Como la única forma de entrar en un grupo de conflicto es realizando el movimiento de anidación ya mencionado, y solo anidamos si hacemos el recorrido de la raíz a una hoja entonces podemos decir que si sumamos uno cada vez que realizamos ese movimiento en el recorrido de la raíz a una hoja y luego incrementamos ese valor en uno tendremos el valor de la PreThinness de G.
Entonces si recorro todos los caminos de raíz a hoja, alguno(s) de ellos va(n) a ser el/los que tenga(n) mayor cantidad de grupos de conflicto anidados, y estos serán los que tengan la suma máxima. Luego tenemos que
$\text{PreThinness}(G) = 1 + \max_{0 \leq i < \#h} GDCA_i$ □

Más adelante veremos una forma de calcular este valor sin necesidad de recorrer todos los caminos de raíz a hoja utilizando la técnica de divide and conquer.

5.2. Cota para la PreThinness

Teorema 5.2.1. *La thinness de un árbol G está acotada por la altura.*

Demostración. Como la thinness es la cantidad máxima de grupos de conflicto anidados incrementado en uno y solo anidamos cuando nos movemos hacia abajo en el árbol, luego la altura es mayor o igual a la cantidad de grupos de conflictos anidados. Luego, sea h la altura del árbol,

$h \geq$ cantidad de grupos de conflicto anidados $= thin(G) - 1$, entonces

$h \geq thin(G) - 1$, equivalentemente $h + 1 \geq thin(G)$.

Además sabemos que es requisito para anidar grupo de conflicto que exista hijo derecho, por lo tanto

$h \geq thin(G)$. □

5.2.0.1 Comparando con la cota existente

Fue demostrado que la cota de la thinness para un árbol m-ario completo con m un valor fijo, es $\Theta(log(n))$ [BE18].

Ahora veamos que la cota para grafos árboles m-arios completos, con m un valor fijo, numerado con el método preorder cumple con $\Theta(log(n))$.
Como se trata de grafos árbol completo, $h = log(n)$, con n la cantidad de nodos. Por lo tanto cumple con la cota $\Theta(log(n))$.

5.3. Algoritmo para encontrar la PreThinness

Como vimos, en preorder aumento la cantidad de grupos de conflicto anidados cuando desde un nodo x que tiene hijo derecho, me muevo a un hijo izquierdo o centro.
Notar que solo se puede tener hijos centro si tiene hijo izquierdo y derecho por la modificación que hacemos en el grafo.

En esta sección vamos a describir el algoritmo para encontrar la PreThinness en un grafo árbol G numerado con el método preorder. El algoritmo tiene complejidad lineal en la cantidad de nodos. Para ello, utilizaremos la técnica de divide and conquer.

Sea x un nodo en G con $\{x_0, ..., x_k\}$ sus hijos. Sea $0 \leq i \leq k$, llamaremos G_{x_i} al grafo árbol, subgrafo de G, que tiene como raíz a x_i. Supongamos que conocemos la PreThinness de G_{x_i} para todo $i \in \{0, ..., k\}$. Llamemos $thin(G_{x_i})$ a la PreThinness del grafo G_{x_i}. Veamos ahora como podemos calcular la PreThinness de G_x, el subgrafo de G cuya raíz es x.

Si x tiene hijo derecho, G_{x_k}, entonces la PreThinness de G_x es:

$$\text{máximo} \left(\left(\max_{0 \leq j < k} thin(G_{x_j}) + 1 \right), thin(G_{x_k}) \right)$$

Esto es así porque x tendrá conflicto con todo nodo perteneciente a G_{x_j}, por lo que si el completo en $G_<$ era de t nodos, ahora será de $t+1$ ya que x también pertenece por tener conflicto con los t nodos. Además, x no tiene conflicto con ningún nodo perteneciente a G_{x_k}, por lo que la PreThinness no incrementa de este lado.

Si x no tiene hijo derecho entonces tampoco tiene hijos centros. Por lo que x tiene un único hijo. Llamémoslo j. Luego la PreThinness de G_x es igual a la PreThinness de G_j. Esto es así ya que x para anidar un grupo de conflicto es requisito que x tenga hijo derecho.

Con la técnica de divide and conquer, recorremos una única vez el grafo árbol, llamando recursivamente al algoritmo que calcula la PreThinness para cada hijo. El costo de procesamiento en cada nodo es $O(1)$ ya que alcanza con ver el resultado que va devolviendo cada hijo y compararlo con el valor de la PreThinness máxima hasta el momento.
Por lo tanto la complejidad final es $O(n)$.

5.4. Optimizaciones para la PreThinness

Como vimos anteriormente, en árboles con numeración preorder todo nodo con más de dos hijos tiene conflicto con todo subárbol cuya raíz es cualquiera de sus hijos excepto el de más a la derecha. Es decir, sean $\{x_0, ..., x_k\}$ los hijos de x, luego x tiene conflicto con todo subárbol cuya raíz es x_j con $0 \leq j < k$. Esto es así sin importar si x_0 es hijo izquierdo o centro o si x_k es hijo centro o derecho. Luego, dado un nodo, es indistinto si su hijo de más a la izquierda es considerado hijo izquierdo o centro, o si su hijo de más a la derecha es considerado centro o derecho. Por lo tanto no es posible realizar optimizaciones en el grafo numerado con preorder.

5.5. Experimentación

A continuación, se presentan las pruebas experimentales que se realizaron con el fin de evaluar y comparar los diferentes métodos ya presentados, junto con los resultados obtenidos y una discusión de los mismos.

5.5.1. Experimento 1: Pruebas de rendimiento

Este experimento se realizó con el objetivo de comparar el rendimiento entre el algoritmo presentado en este trabajo en la sección 5.3 y el algoritmo propuesto en [BE18] teniendo en cuenta el tiempo insumido para la ejecución de cada uno de ellos. Se realizaron tres tipos de pruebas:

- Rendimiento en función de la altura del árbol, utilizando árboles completos.
- Rendimiento en función de la cantidad máxima de hijos que puede tener un nodo, utilizando árboles completos.
- Rendimiento en función de la altura del árbol, utilizando árboles Zig-Zag.

Para ello primero se variará la altura del árbol manteniendo constante el tipo de árbol y la cantidad máxima de hijos que puede tener un nodo. Para el segundo ítem se dejará constante la altura y el tipo de árbol y se variará la máxima cantidad de hijos que puede tener un nodo. Luego tomaremos árboles de tipo Zig-Zag y variaremos la altura del mismo. Para cada una de las variaciones se medirá el tiempo de ejecución de cada uno de los métodos.
Con el fin de mejorar la precisión de los resultados y evitar posibles interferencias (por ejemplo, las generadas por otras tareas que puedan estar ejecutando la computadora) cada medición se repitió 5 veces, tomando luego el desviación estándar de los datos obtenidos. En todos los casos, las medidas se eligieron de forma arbitraria, procurando que los casos de prueba no fueran excesivamente grandes para no prolongar innecesariamente la duración de las pruebas. Para la cantidad máxima de hijos por nodo, cuando la misma debe ser constante, se eligió 3 y para la altura de árbol, cuando ésta debe ser constante se eligió 5, números elegidos arbitrariamente, ya que no se consideró que este parámetro fuera de relevancia para el experimento.

5.5.1.1 Resultados

En cada gráfico se muestran las curvas representando el tiempo insumido por el algoritmo propuesto en la sección 5.3 ($GDCA$) y el algoritmo exhibido en [BE18] ($G_<$) con respecto a la cantidad de nodos del grafo. Además, se muestran las curvas $f(x) = x$ y $f(x) = x^3$ representando las complejidades del algoritmo introducido en la sección 5.3 y el algoritmo propuesto en [BE18] respectivamente.
En el título de cada gráfico indica el tipo de árbol, el orden utilizado y la variable que se

mantiene constante seguida del valor fijo utilizado. Notar que en un grafo árbol Zig-Zag la cantidad de hijos máxima es 2. m representa la máxima cantidad de hijos por nodo y h la altura del árbol.
En todos los gráficos el eje Y está en escala logarítmica.

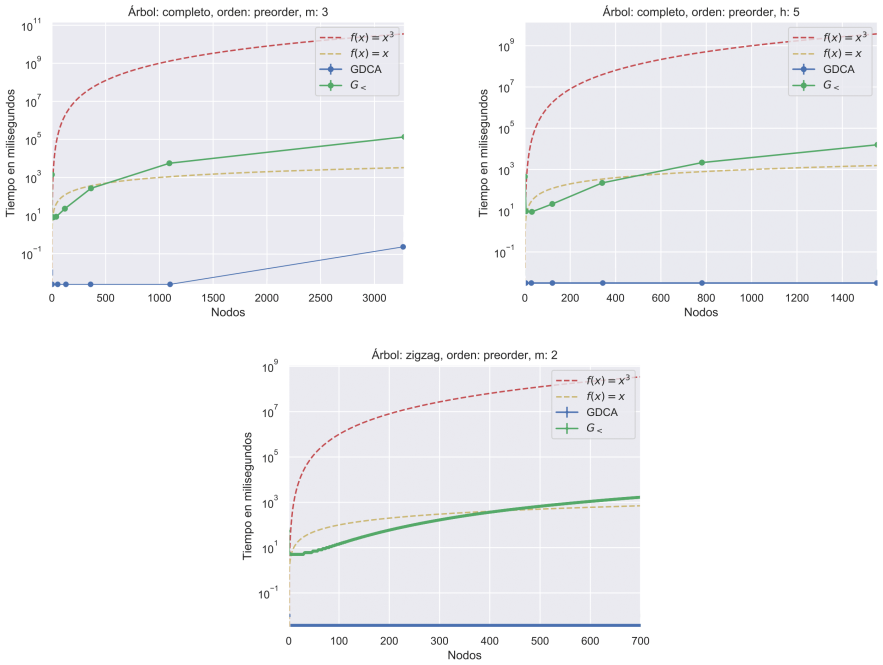


Fig. 5.4: Comparando tiempos entre diferentes algoritmos

5.5.1.2 Discusión

Como se puede observar en los resultados, pudo verificarse que el algoritmo propuesto en la sección 5.3 es más eficiente. Esto es así ya que tanto al comparar el tiempo de ejecución variando la altura y variando la cantidad máxima de hijos por nodo, como cuando se comparó entre árboles de tipo Zig-Zag con diferentes alturas, el tiempo de ejecución del algoritmo introducido en la sección 5.3 es menor.
También se puede observar que el método propuesto en la sección 5.3 tiene complejidad lineal en la cantidad de nodos; no se considera necesario agregar más pruebas con diferente cantidad de nodos, otros tipos de árboles o variaciones en la máxima cantidad de hijos por nodos.

5.6. Conclusiones

Como vimos, el algoritmo propuesto en la sección 5.3 es más rápido que el propuesto por [BE18], por lo tanto entre estos dos métodos es clara la conveniencia de uno sobre el otro.

6. POSTORDER

6.1. Thinness para árboles Postorder

Por el Teorema 3.1.1 vimos que la thinness en un grafo G es uno más que la máxima cantidad de grupos de conflicto anidados que hay en G. Por lo tanto, para hallar la thinness en un grafo árbol, vamos a hallar máxima cantidad de grupos de conflicto anidados que hay en G. En este capítulo vamos a analizar como hacer esto en un grafo G numerado con el método postorder, es decir, donde $<$ es postorder. Primero lo haremos para árboles binarios y luego lo mostraremos para árboles m-arios.

6.1.1. Árboles binarios

6.1.1.1 Cómo se forman los grupos de conflicto

Sabemos por definición de postorder que para cualquier nodo x vale que:

- Todo nodo en el subárbol izquierdo o derecho es menor que x
- El nodo más grande adyacente a x es su padre.

Sea x un nodo en el grafo árbol, hijo izquierdo de r. Sea d hijo derecho de r (hermano derecho de x).

- Por definición de postorder sabemos que todo nodo perteneciente al subárbol que tiene como raíz a d será mayor que x y menor que r.
- Por ser un grafo de tipo árbol, el único nodo en el subárbol que tiene como raíz a d y es adyacente a r es d.

Por lo tanto, como no puede tener conflicto con ningún nodo de su subárbol derecho o izquierdo, y tiene conflicto con todo nodo del subárbol que tiene como raíz a su hermano derecho excluyendo a su hermano derecho, el grupo de conflicto de x (C_x) será todo nodo perteneciente a la descendencia de su hermano derecho.

Sea x un nodo en el grafo árbol, hijo derecho de r.

- Por definición de postorder, sea k el número asignado por postorder a x, luego r será numerado con $k + 1$.

Por lo tanto, como el nodo adyacente más grande a x es su padre que tiene como numeración el número siguiente al número asignado a x, entonces no tiene conflicto con ningún nodo. Es decir $C_x = \emptyset$.

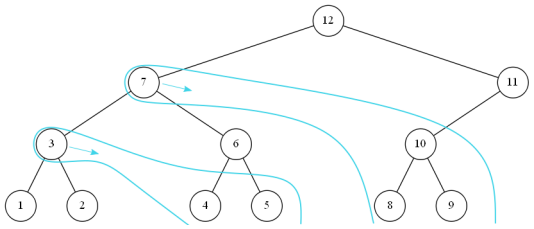


Fig. 6.1: Grupos de conflicto en un árbol binario.

6.1.2. Árboles m-arios

Por Observación 2.3.1 vamos a considerar, sin perdida de generalidad, que solo puede haber hijos centro si ya existe uno izquierdo y uno derecho.

6.1.2.1 Cómo se forman los grupos de conflicto

Sabemos por definición de postorder que para cualquier nodo x vale que:

- Todo nodo en el subárbol izquierdo, centro o derecho es menor que x
- El nodo más grande adyacente a x es su padre.

Sea x un nodo en el grafo árbol, hijo izquierdo de r. Sean $\{x, x_0, ..., x_k\}$ los hijos de r.

- Por definición de postorder sabemos que todo nodo perteneciente a los subárboles que tienen como raíz a $x_0, ..., x_k$ serán mayores que x y menores que r.
- Por ser un grafo de tipo árbol, los únicos nodos en los subárboles que tienen como raíz a $x_0, ..., x_k$ adyacentes a r son exactamente $x_0, ..., x_k$.

Por lo tanto, como no puede tener conflicto con ningún nodo de su subárbol izquierdo, centro o derecho, y tiene conflicto con todo nodo perteneciente a los subárboles que tienen como raíz a $x_0, ..., x_k$ excluyendo a $x_0, ..., x_k$, el grupo de conflicto de x (C_x) será todo nodo perteneciente a la descendencia de sus hermanos.

Sea x_j un nodo en el grafo árbol, con $0 \leq j < k$, hijo centro de r. Sean $\{x_0, ..., x_j, ..., x_k\}$ los hijos de r.

- Por definición de postorder sabemos que todo nodo perteneciente a los subárboles que tienen como raíz a $x_{j+1}, ..., x_k$ serán mayores que x y menores que r.
- Por ser un grafo de tipo árbol, los únicos nodos en los subárboles que tienen como raíz a $x_0, ..., x_k$ adyacentes a r son exactamente $x_0, ..., x_k$.

Por lo tanto, como no puede tener conflicto con ningún nodo de sus subárboles izquierdo, centro(s) o derecho, y tiene conflicto con todo nodo perteneciente a los subárboles que tienen como raíz a $x_{j+1}, ..., x_k$ excluyendo a $x_{j+1}, ..., x_k$, el grupo de conflicto de x (C_x) será todo nodo perteneciente a la descendencia $x_{j+1}, ..., x_k$.

Sea x un nodo en el grafo árbol, hijo derecho de r.

- Por definición de postorder, sea k el número asignado por postorder a x, luego r será numerado con $k + 1$.

Por lo tanto, como el nodo adyacente más grande a x es su padre que tiene como numeración el número siguiente al número asignado a x, entonces no tiene conflicto con ningún nodo. Es decir $C_x = \emptyset$.

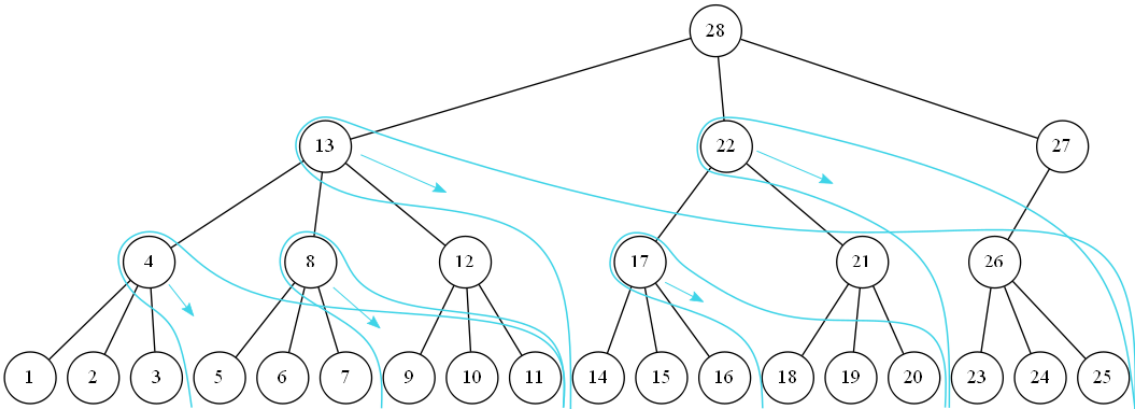


Fig. 6.2: Grupos de conflicto en un árbol ternario.

6.1.3. Grupos de conflicto y movimientos en el árbol

Ahora vamos a hablar de árboles m-arios. Si es binario, no hay hijos centro pero vale lo mismo que para los m-arios sabiendo que el conjunto de hijos centros es vacío.

6.1.3.1 Forma de entrar a un grupo de conflicto

Veamos que entonces se tiene la siguiente forma de entrar en un grupo de conflicto:

- Dado un nodo x en el grafo con por lo menos un hermano izquierdo, si me muevo a algún hijo de x, llego a un nodo perteneciente a un nuevo grupo de conflicto.

En la figura 6.3 podemos observar ejemplos de algunos de los movimientos. Estos se encuentran marcados con color.

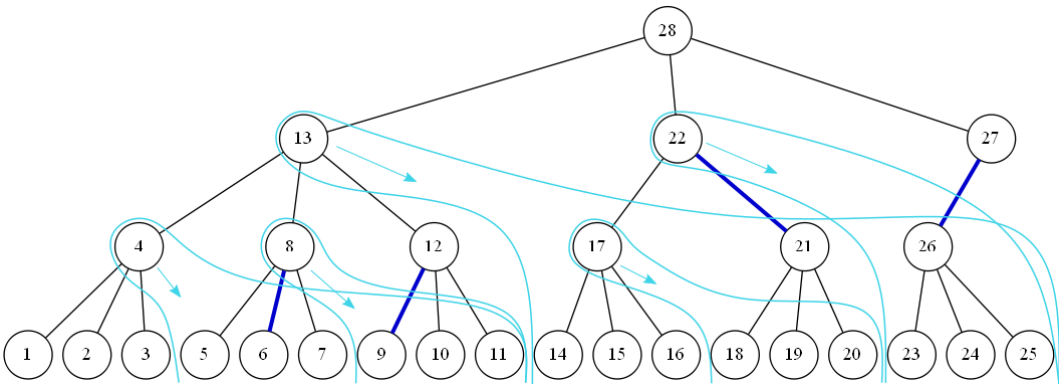


Fig. 6.3: Formas de entrar a los grupos de conflicto.

Definición 6.1.1. Movimiento de anidación: Llamaremos movimiento de anidación al movimiento mencionado arriba.

6.1.3.2 Anidando grupos de conflicto

Teorema 6.1.1. *Sean dos nodos x e y, con $y \in C_x$. Sea d un nodo perteneciente a la descendencia de y, entonces $d \in C_x$.*

Demostración. Por absurdo: Sean dos nodos x e y, con $y \in C_x$. Sea d un nodo perteneciente a la descendencia de y con $d \notin C_x$.
Sabemos por lo visto en la Sección 6.1.2.1:

- x no puede ser hijo derecho ya que $C_x \neq \emptyset$ porque $y \in C_x$.
- x tiene conflicto con todo nodo perteneciente a la descendencia de sus hermanos.

Por lo tanto y tiene que ser un nodo perteneciente a la descendencia de los hermanos de x. Por definición, d pertenece a la descendencia de y. Por lo tanto d es parte de la descendencia de los hermanos de x. Absurdo. Si esto es así, $d \in C_x$. El absurdo vino de suponer que $d \notin C_x$. □

Por lo tanto si recorremos el árbol desde la raíz hacia las hojas, una vez que llego a un nodo que está en un grupo de conflicto, luego todos los nodos por los que siga pasando también estarán en ese grupo.

Teorema 6.1.2. *Sea x un nodo en G. Todo nodo en el grupo de conflicto de x pertenece a un nivel inferior que x. Además, sea l el nivel de x, si $C_x \neq \emptyset$, entonces existe t nodo en G tal que $t \in C_x$ y el nivel de t sea $l + 1$.*

Demostración. Sale trivialmente por definición. Sabemos que x tiene conflicto con todo nodo descendiente de sus hermanos derechos. □

Teorema 6.1.3. *Sean x e y dos nodos en el grafo. Sea x alcanzable desde y haciendo el movimiento de anidación. Luego $\alpha(x) = \alpha(y) + 1$*

Demostración. Veamos que no es menor. Demostración por absurdo:
Sea y un nodo en el grafo. Sea x el nodo del árbol al que llego haciendo el movimiento conflictivo desde y. Sabemos que por ser x descendiente de y luego $\alpha(x) \geq \alpha(y)$ por Teorema 6.1.2.1. Además, llegamos a x desde y moviéndonos a través de un movimiento de anidación, por lo tanto sabemos que y tiene un hermano izquierdo, llamémoslo iy y x es hijo de y. Luego iy es un nodo conflictivo y $x \in C_{iy}$. Luego $\alpha(x) \geq \alpha(y) + 1$.

Veamos que no es mayor. Demostración por absurdo:
Sea iy el nodo conflictivo tal que $x \in C_{iy}$ pero $y \notin C_{iy}$. Ahora queremos un nodo k tal que $x, iy \in C_k$ pero $y \notin C_k$. Como tiene que tener conflicto con x y no con y, k debe pertenecer al mismo nivel de y. Esto es así ya que todo nodo que se encuentre en un nivel superior que y que tenga conflicto con x, también tendrá conflicto con y por sección 6.1.2.1. A su vez, $iy \in C_k$, por lo tanto por teorema 6.1.2 el nivel de k debe ser mayor que el de iy. Como iy e y pertenecen al mismo nivel, entonces el nivel de k debe ser mayor y menor que el nivel de y. Absurdo. Por lo tanto no anidamos más de un grupo de conflicto al realizar un movimiento de anidación.
Luego podemos asegurar que cada vez que entro a un grupo de conflicto y hago el movimiento de anidación, entonces anido un y solo un grupo de conflicto más. □

6.1.4. PostThinness

Definición 6.1.2. PostThinness: Llamaremos PostThinness a la thinness de un árbol cuyos nodos se ordenan con el método de postorder.

A partir de ahora:
Dado un grafo G al que se le quiere calcular la PostThinness, lo alteraremos de las siguientes formas:

- En G, si un nodo tiene al menos un hijo centro y no tiene hijo izquierdo, entonces lo modificaremos de forma tal que tenga como hijo izquierdo el hijo centro que se encuentre más a la izquierda. Si tenia otros centros estos quedan como centro. Si tenia uno solo ahora no tiene hijo centro.
- En G, si un nodo tiene al menos un hijo centro y no tiene hijo derecho, entonces lo modificaremos de forma tal que tenga como hijo derecho el hijo centro que se encuentre más a la derecha. Si tenia otros centros estos quedan como centro. Si tenia uno solo ahora no tiene hijo centro.
- En G, si un nodo tiene un hijo, entonces éste será hijo izquierdo.

Llamaremos G' al grafo resultante de realizar estas modificaciones sobre G.

Observación 6.1.1. PostThinness(G) = PostThinness(G').

6.1.4.1 Nodos que forman un completo en $G_<$

Vimos que existe un movimiento que hace que se anide un grupo de conflicto más. Veamos cuál es el nodo que se agrega como nodo conflictivo que luego será parte del completo en $G_<$:

- a - Izquierda/Centro/Derecha - b, donde a tiene hijo izquierdo: Parto del nodo a, me muevo a algún hijo, llegando al nodo b. En este caso el nodo conflictivo es cualquier hermano izquierdo de a, Llamémoslo ia. C_{ia} es su grupo de conflicto y sabemos que $C_{ia} \neq \emptyset$ ya que $b \in C_{ia}$. Luego ia será un nodo que formará parte del completo en $G_<$.

Dado que solo hicimos un movimiento, solo tenemos un nodo en $G_<$. El otro nodo perteneciente a la clique máxima es el último al que haya llegado en el camino de la raíz a una hoja. Este nodo pertenecerá a todos los C_i ya que sabemos que una vez que entro a uno no puedo salir. Sabemos que existe porque todos los movimientos terminan en un nodo perteneciente al grupo de conflicto y además a ése nodo no lo agregué al $G_<$ anteriormente.

6.1.4.2 Calculando la PostThinness

Teorema 6.1.4. *Sea $\#h$ la cantidad de hojas del grafo árbol G. Sea r la raíz en G. Sea h_i la i-ésima hoja, con $0 \leq i < \#h$. Sea $GDCA_i$ la cantidad de grupos de conflicto anidados en el camino de la raíz a la i-ésima hoja. Entonces la cantidad máxima de grupos de conflicto anidados en el grafo árbol postorder es $1 + \max_{0 \leq i < \#h} GDCA_i$.*

Demostración. Sea G el grafo árbol al que le hicimos las modificaciones mencionadas para que siempre haya hijo izquierdo, después derecho y después centros.
Sean h grupos de conflicto anidados. Cuando anidamos un grupo más, aumentamos en uno el tamaño de la clique máxima en $G_<$. Por lo tanto la PostThinness aumenta en uno.
Sea 1 grupo de conflicto, la PostThinness es 2, ya que tenemos un K_2 en $G_<$.
Luego, si cada vez que entramos a un grupo de conflicto sumamos uno a la PostThinness y después incrementamos ese valor en uno (ya que un grupo de conflicto genera PostThinness 2), entonces tenemos la PostThinness de G.
Como la única forma de entrar en un grupo de conflicto es realizando el movimiento de anidación ya mencionado, y solo anidamos si hacemos el recorrido de la raíz a una hoja entonces podemos decir que si sumamos uno cada vez que realizamos ese movimiento en el recorrido de la raíz a una hoja y luego incrementamos ese valor en uno tendremos el valor de la PostThinness de G.
Entonces si recorro todos los caminos de raíz a hoja, alguno(s) de ellos va(n) a ser el/los que tenga(n) mayor cantidad de grupos de conflicto anidados, y estos serán los que tengan la suma máxima. Luego tenemos que
$\text{PostThinness}(G) = 1 + \max_{0 \leq i < \#h} GDCA_i$ □

Más adelante veremos una forma de calcular este valor sin necesidad de recorrer todos los caminos de raíz a hoja utilizando la técnica de divide and conquer.

6.2. Cota para la PostThinness

Teorema 6.2.1. *La thinness de un árbol G está acotada por la altura.*

Demostración. Como la PostThinness es la cantidad máxima de grupos de conflicto anidados incrementado en uno y solo anidamos cuando nos movemos hacia abajo en el árbol, luego la altura es mayor o igual a la cantidad de grupos de conflictos anidados. Luego, sea h la altura del árbol,

$h \geq$ cantidad de grupos de conflicto anidados $= thin(G) - 1$, entonces

$h \geq thin(G) - 1$, equivalentemente $h + 1 \geq thin(G)$.

Además sabemos que es requisito para anidar grupo de conflicto que exista al menos un hermano izquierdo del nodo del que parto para hacer el movimiento de anidación, por lo tanto

$h \geq thin(G)$. □

6.2.1. Comprando con la cota existente

Fue demostrado que la cota de la thinness para un árbol m-ario completo con m un valor fijo, es $\Theta(log(n))$ [BE18].
Ahora veamos que la cota para grafos árboles m-arios completos, con m un valor fijo, numerado con el método postorder cumple con $\Theta(log(n))$.
Como se trata de grafos árbol completo, $h = log(n)$, con n la cantidad de nodos. Por lo tanto cumple con la cota $\Theta(log(n))$.

6.3. Algoritmo para encontrar la PostThinness

Como vimos, en postorder aumento la cantidad de grupos de conflicto anidados cuando me muevo desde un nodo x a cualquiera de sus hijos si x tiene al menos un hermano izquierdo.
Equivalentemente, sea k un nodo en G. Si k tiene hijo izquierdo (llamémoslo k_i), me muevo a cualquier hijo de k distinto de k_i, llamémoslo x y luego me muevo a cualquier hijo de x, entonces aumente en uno la cantidad de grupos de conflictos anidados.
Notar que solo se puede tener hijos centro si tiene hijo izquierdo y derecho por la modificación que hacemos en el grafo.

En esta sección vamos a describir el algoritmo para encontrar la PostThinness en un grafo árbol G numerado con el método postorder con complejidad lineal en la cantidad de nodos. Para ello, utilizaremos la técnica de divide and conquer.

Sea x un nodo en G con $\{x_0, ..., x_k\}$ sus hijos. Sea $0 \leq i \leq k$, llamaremos G_{x_i} al grafo árbol, subgrafo de G, que tiene como raíz a x_i. Supongamos que conocemos la PostThinness de G_{x_i} para todo $i \in \{0, ..., k\}$. Llamemos $thin(G_{x_i})$ a la PostThinness del grafo G_{x_i}. Veamos ahora como podemos calcular la PostThinness de G_x, el subgrafo de G cuya raíz es x.

Sea x_0 hijo izquierdo de x, x_0. Sean $\{x_{u_0}, ..., x_{u_z}\}$ los hijos de x que tienen al menos un hijo. Sean $\{x_{v_0}, ..., x_{v_z}\}$ los hijos de x sin hijos. Entonces la PostThinness de G_x es:

$$\text{máximo} \left(thin(G_{x_0}), \left(\max_{j \in \{x_{u_0}, ..., x_{u_z}\}} thin(G_{x_j}) + 1 \right), \left(\max_{j \in \{x_{v_0}, ..., x_{v_z}\}} thin(G_{x_j}) \right) \right)$$

Esto es así porque x tendrá conflicto con todo nodo perteneciente a $G_{x_{u_i}}$, $0 \leq i \leq z$, por lo que si el completo en $G_<$ era de t nodos, ahora será de $t+1$ ya que x también pertenece

por tener conflicto con los t nodos. Además, x no tiene conflicto con ningún nodo perteneciente a $G_{x_{v_i}}$ ni a G_{x_0}, por lo que la PostThinness no incrementa si usamos estos subárboles.

Si x no tiene hijo izquierdo entonces tampoco tiene hijos centros. Por lo que x tiene un único hijo. Llamémoslo k. Luego la PostThinness de G_x es igual a la PostThinness de G_k. Esto es así ya que partiendo desde x para anidar un grupo de conflicto es requisito que k tenga hermano izquierdo.

Con la técnica de divide and conquer, recorremos una única vez el grafo árbol, llamando recursivamente al algoritmo que calcula la PostThinness para cada hijo. El costo de procesamiento en cada nodo es $O(1)$ ya que alcanza con ver el resultado que va devolviendo cada hijo y compararlo con el valor de la PostThinness máxima hasta el momento.
Por lo tanto la complejidad final es $O(n)$.

6.4. Optimizaciones para la PostThinness

Como vimos anteriormente, en árboles con numeración Postorder, el conflicto se genera cuando hay un nodo que tiene al menos un hermano a la izquierda. Llamemos $\{x_0, ..., x_k\}$ a los hermanos izquierdos de x, un nodo en G. Luego $\{x_0, ..., x_k\}$ tendrán conflicto con todo nodo perteneciente a la descendencia de x. Llamemos $\{x_{k+1}, ..., x_l\}$ a los hermanos derechos de x. Esto es así sin importar si x_0 es hijo izquierdo o centro o si x_l es hijo centro o derecho. Luego, dado un nodo, es indistinto si su hijo de más a la izquierda es considerado hijo izquierdo o centro, o si su hijo de más a la derecha es considerado centro o derecho. Por lo tanto no es posible realizar optimizaciones en el grafo numerado con postorder.

6.5. Experimentación

A continuación, se presentan las pruebas experimentales que se realizaron con el fin de evaluar y comparar los diferentes métodos ya presentados, junto con los resultados obtenidos y una discusión de los mismos.

6.5.1. Experimento 1: Pruebas de rendimiento

Este experimento se realizó con el objetivo de comparar el rendimiento entre el algoritmo presentado en este trabajo en la sección 6.3 y el algoritmo propuesto en [BE18] teniendo en cuenta el tiempo insumido para la ejecución de cada uno de ellos. Se realizaron tres tipos de pruebas:

- Rendimiento en función de la altura del árbol, utilizando árboles completos.
- Rendimiento en función de la cantidad máxima de hijos que puede tener un nodo, utilizando árboles completos.
- Rendimiento en función de la altura del árbol, utilizando árboles Zig-Zag.

Para ello primero se variará la altura del árbol manteniendo constante el tipo de árbol y la cantidad máxima de hijos que puede tener un nodo. Para el segundo ítem se dejará constante la altura y el tipo de árbol y se variará la máxima cantidad de hijos que puede tener un nodo. Luego tomaremos árboles de tipo Zig-Zag y variaremos la altura del mismo. Para cada una de las variaciones se medirá el tiempo de ejecución de cada uno de los métodos.
Con el fin de mejorar la precisión de los resultados y evitar posibles interferencias (por ejemplo, las generadas por otras tareas que puedan estar ejecutando la computadora) cada medición se repitió 5 veces, tomando luego el desviación estándar de los datos obtenidos. En todos los casos, las medidas se eligieron de forma arbitraria, procurando que los casos de prueba no fueran excesivamente grandes para no prolongar innecesariamente la duración de las pruebas. Para la cantidad máxima de hijos por nodo, cuando la misma debe ser constante, se eligió 3 y para la altura de árbol, cuando la misma debe ser constante se eligió 5, números elegidos arbitrariamente, ya que no se consideró que este parámetro fuera de relevancia para el experimento.

6.5.1.1 Resultados

En cada gráfico se muestran las curvas representando el tiempo insumido por el algoritmo propuesto en la sección 6.3 ($GDCA$) y el algoritmo exhibido en [BE18] ($G_<$) con respecto a la cantidad de nodos del grafo. Además, se muestran las curvas $f(x) = x$ y $f(x) = x^3$ representando las complejidades del algoritmo introducido en la sección 6.3 y el algoritmo propuesto en [BE18] respectivamente.
En el título de cada gráfico indica el tipo de árbol, el orden utilizado y la variable que se mantiene constante seguida del valor fijo utilizado. Notar que en un grafo árbol Zig-Zag la cantidad de hijos máxima es 2. m representa la máxima cantidad de hijos por nodo y h la altura del árbol.
En todos los gráficos el eje Y está en escala logarítmica.

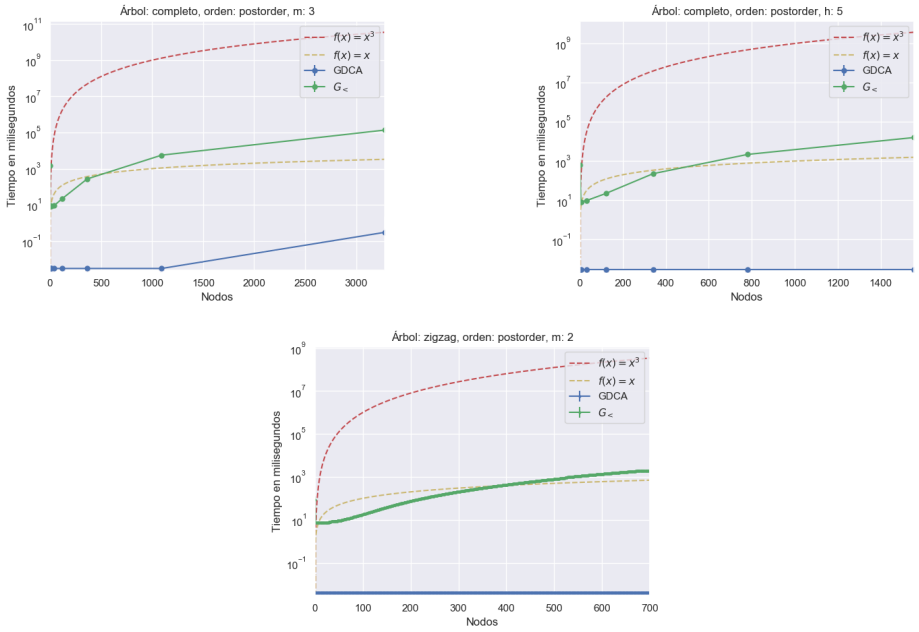


Fig. 6.4: Comparando tiempos entre diferentes algoritmos

6.5.1.2 Discusión

Como se puede observar en los resultados, pudo verificarse que el algoritmo propuesto en la sección 6.3 es más eficiente. Esto es así ya que tanto al comparar el tiempo de ejecución variando la altura y variando la cantidad máxima de hijos por nodo, como cuando se comparó entre árboles de tipo Zig-Zag con diferentes alturas, el tiempo de ejecución del algoritmo introducido en la sección 6.3 es menor.
También se puede observar que el método propuesto en la sección 6.3 tiene complejidad lineal en la cantidad de nodos; no se considera necesario agregar más pruebas con diferente cantidad de nodos, otros tipos de árboles o variaciones en la máxima cantidad de hijos por nodos.

6.6. Conclusiones

Como vimos, el algoritmo propuesto en la sección 6.3 es más rápido que el propuesto por [BE18], por lo tanto entre estos dos métodos es clara la conveniencia de uno sobre el otro.

7. BÚSQUEDA A LO ANCHO (BFS - BREADTH FIRST SEARCH)

7.1. Thinness para árboles BFS

Por el Teorema 3.1.1 vimos que la thinness en un grafo G es uno más que la máxima cantidad de grupos de conflicto anidados que hay en G. Por lo tanto, para hallar la thinness en un grafo árbol, vamos a hallar máxima cantidad de grupos de conflicto anidados que hay en G. En este capítulo vamos a analizar como hacer esto en un grafo G numerado con el método BFS, es decir, donde $<$ es BFS.

7.1.1. Cómo se forman los grupos de conflicto

Sabemos por definición de BFS que para cualquier nodo x vale que:

- Todo nodo perteneciente a niveles menores que el nivel de x tiene numeración menor que x. Por lo tanto el padre de x es menor que x.
- Todo nodo en el mismo nivel que x que se encuentre a la izquierda de x es más chico que x.
- Todo nodo en el mismo nivel que x que se encuentre a la derecha de x es más grande que x.
- Todo nodo perteneciente a niveles mayores que el nivel de x es más grande que x. Entonces todo hijo de x es mayor que x.
- El nodo más grande adyacente a x es el hijo de x que se encuentre más a la derecha.

Sea x un nodo en el árbol sin hijos. Entonces como sabemos que su padre es menor y no hay otro nodo adyacente a x, entonces x no tiene conflictos.

Sea x un nodo en el árbol con al menos un hijo. x va a tener conflicto con:

- todo nodo de su mismo nivel que se encuentre a la derecha y
- todo nodo del siguiente nivel que se encuentre a la izquierda del hijo de más a la derecha de x.

Es importante notar que no hay grupo de conflicto que incluya nodos de más de dos niveles. Por lo tanto solo se puede anidar con nodos cuya diferencia de nivel no es mayor que uno.
Además, sea x un nodo en el grafo con al menos un hijo, entonces x tiene conflicto con todos los nodos de su nivel que están a la derecha. Esto hace que, dado un nivel, todo nodo con al menos un hijo forma un completo en $G_<$.
Existen dos escenarios en donde podríamos sumar un nodo más al completo en $G_<$:

Sea el nivel l de árbol, sean $\{x_0, x_1, ..., x_s\}$ los nodos del nivel l en orden (es decir, $x_0 < x_1 < ... < x_s$ y sean $\{v_0, v_1, ..., v_k\}$ los nodos del nivel l que forman el completo en $G_<$, también ordenados (notar que $\{v_0, v_1, ..., v_k\}$ es un subconjunto de $\{x_0, x_1, ..., x_s\}$).

- Si existe un nodo p tal que $p \in \{x_0, x_1, ..., x_s\}$ y $p > v_k$, entonces este nodo también pertenece al completo en $G_<$ por más que no tenga hijos, ya que $\forall v \in \{v_0, v_1, ..., v_k\}$, v tiene conflicto con p.

- Si v_0 tiene por lo menos dos hijos, luego el hijo de más a la izquierda de v_0 también pertenece al completo en $G_<$ ya que $\forall v \in \{v_0, v_1, ..., v_k\}$, v tiene conflicto con ese nodo.

Notar que solo podemos sumar uno de los dos nodos antes mencionados ya que entre estos dos no hay conflicto.

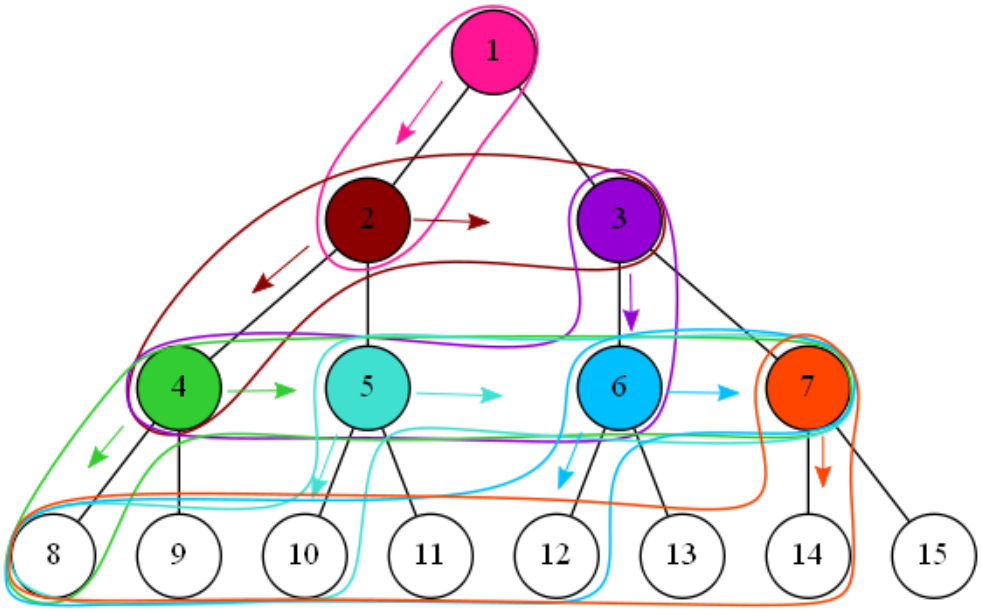


Fig. 7.1: Grupos de conflicto.

7.1.2. BFSThinness

Definición 7.1.1. BFSThinness: Llamaremos BFSThinness a la thinness de un árbol cuyos nodos se ordenan con el método BFS.

7.1.2.1 Nodos que forman un completo en $G_<$

Los nodos que forman la clique máxima en $G_<$ entonces, son los nodos del nivel que contenga la mayor cantidad de nodos en el grupo de conflicto.

7.1.2.2 Calculando la BFSThinness

Para calcular la BFSThinness alcanza con hacer una recorrida por nivel en el árbol contando la cantidad de nodos en el nivel que tienen al menos un hijo y luego sumando uno si se cumple alguno de los dos escenarios en donde se podría sumar un nodo más al completo en $G_<$. Por lo tanto se desprende la siguiente cota.

Corolario 7.1.1. *La BFSThinness está acotada por la máxima cantidad de nodos en un nivel + 1.*

Como la BFSThinness es una cota superior para la thinness, tenemos el siguiente resultado.

Corolario 7.1.2. *La thinness de un árbol está acotada por la máxima cantidad de nodos en un nivel + 1.*

7.2. Cota para la BFSThinness

Fue demostrado que la cota de la thinness para un árbol m-ario completo con m un valor fijo, es $\Theta(log(n))$ [BE18].
Sabemos que la cota para BFS es uno más que la máxima cantidad de nodos en un nivel. En un árbol completo esto es mayor que $O(log(n))$. Esto nos da una intuición de que cuando se trata de un árbol con pocos nodos por nivel pero con mucha altura, BFS será un buen orden. En caso contrario conviene utilizar inorder, preorder o postorder.

7.3. Algoritmo para encontrar la BFSThinness

Como vimos, para calcular la BFSThinness alcanza con saber la cantidad de nodos que tiene al menos un hijo en un nivel y verificar si:

- Existe un nodo sin hijos después del último nodo en el nivel que tiene al menos un hijo.

- El primer nodo en el nivel que tiene al menos un hijo, tiene al menos dos.

El algoritmo recorre el árbol con el método de BFS (por nivel), agregando la información del nivel en el que está cada nodo a medida que se lo agrega a la cola. A medida que recorremos el árbol, contamos la cantidad de nodos con al menos un hijo. Luego, cuando hay un cambio de nivel, verificamos si se cumple alguna de las dos condiciones (esto es $O(1)$ ya que podemos mantener un booleano representando cada una de las condiciones). Si se cumple alguna de las condiciones, luego la BFSThinness asociada a ese nivel es la cantidad de nodos con al menos un hijo + 1. Si no se cumple ninguna de las condiciones, la BFSThinness asociada a ese nivel es la cantidad de nodos con al menos un hijo.
A medida que avanzamos en el árbol, vamos manteniendo el valor de la BFSThinness y lo actualizamos cuando la BFSThinness asociada a un nivel es mayor que el valor guardado. Luego la complejidad del algoritmo es $O(n)$, con n la cantidad de nodos del árbol.

7.4. Optimizaciones para la BFSThinness

Como vimos anteriormente, en árboles con numeración BFS, la numeración no depende de qué hijo sea el nodo. Luego, dado un nodo, es indistinto si su hijo de más a la izquierda es considerado hijo izquierdo o centro, o si su hijo de más a la derecha es considerado centro o derecho. Por lo tanto no es posible realizar optimizaciones en el grafo numerado con BFS.

7.5. Experimentación

A continuación, se presentan las pruebas experimentales que se realizaron con el fin de evaluar y comparar los diferentes métodos ya presentados, junto con los resultados obtenidos y una discusión de los mismos.

7.5.1. Experimento 1: Pruebas de rendimiento

Este experimento se realizó con el objetivo de comparar el rendimiento entre el algoritmo presentado en este trabajo en la sección 7.3 y el algoritmo propuesto en [BE18] teniendo en cuenta el tiempo insumido para la ejecución de cada uno de ellos. Se realizaron tres tipos de pruebas:

- Rendimiento en función de la altura del árbol, utilizando árboles completos.

- Rendimiento en función de la cantidad máxima de hijos que puede tener un nodo, utilizando árboles completos.

- Rendimiento en función de la altura del árbol, utilizando árboles Zig-Zag.

Para ello primero se variará la altura del árbol manteniendo constante el tipo de árbol y la cantidad máxima de hijos que puede tener un nodo. Para el segundo ítem se dejará constante la altura y el tipo de árbol y se variará la máxima cantidad de hijos que puede tener un nodo. Luego tomaremos árboles de tipo Zig-Zag y variaremos la altura del mismo. Para cada una de las variaciones se medirá el tiempo de ejecución de cada uno de los métodos.
Con el fin de mejorar la precisión de los resultados y evitar posibles interferencias (por ejemplo, las generadas por otras tareas que puedan estar ejecutando la computadora) cada medición se repitió 5 veces, tomando luego el desviación estándar de los datos obtenidos. En todos los casos, las medidas se eligieron de forma arbitraria, procurando que los casos de prueba no fueran excesivamente grandes para no prolongar innecesariamente la duración de las pruebas. Para la cantidad máxima de hijos por nodo, cuando la misma debe ser constante, se eligió 3 y para la altura de árbol, cuando la misma debe ser constante se eligió 5, números elegidos arbitrariamente, ya que no se consideró que este parámetro fuera de relevancia para el experimento.

7.5.1.1 Resultados

En cada gráfico se muestran las curvas representando el tiempo insumido por el algoritmo propuesto en la sección 7.3 ($GDCA$) y el algoritmo exhibido en [BE18] ($G_<$) con respecto a la cantidad de nodos del grafo. Además, se muestran las curvas $f(x) = x$ y $f(x) = x^3$ representando las complejidades del algoritmo introducido en la sección 7.3 y el algoritmo propuesto en [BE18] respectivamente.
En el título de cada gráfico indica el tipo de árbol, el orden utilizado y la variable que se mantiene constante seguida del valor fijo utilizado. Notar que en un grafo árbol Zig-Zag la cantidad de hijos máxima es 2. m representa la máxima cantidad de hijos por nodo y h la altura del árbol.
En todos los gráficos el eje Y está en escala logarítmica.

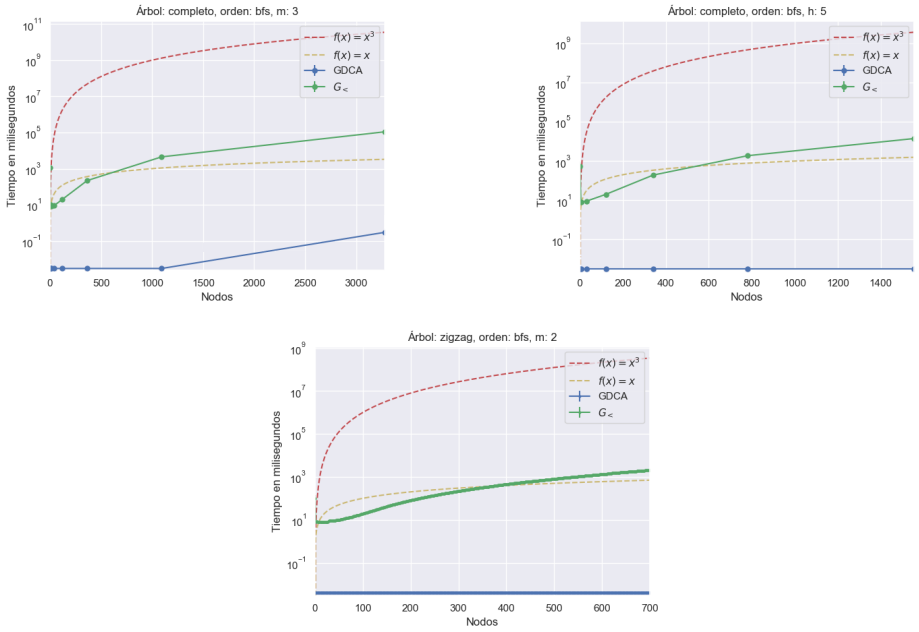


Fig. 7.2: Comparando tiempos entre diferentes algoritmos

7.5.1.2 Discusión

Como se puede observar en los resultados, pudo verificarse que el algoritmo propuesto en la sección 7.3 es más eficiente ya que tanto al comparar el tiempo de ejecución con respecto a la altura, a la cantidad máxima de hijos por nodo y al tipo de árbol, el tiempo de ejecución es menor.

También se puede observar que el método propuesto en la sección 7.3 tiene complejidad lineal en la cantidad de nodos; no se considera necesario agregar más pruebas con diferente cantidad de nodos, otros tipos de árboles o variaciones en la máxima cantidad de hijos por nodos.

7.6. Conclusiones

Como vimos, el algoritmo propuesto en la sección 7.3 es más rápido que el propuesto por [BE18], por lo tanto entre estos dos métodos es clara la conveniencia de uno sobre el otro.

8. EXPERIMENTACIÓN ENTRE DIFERENTES ÓRDENES

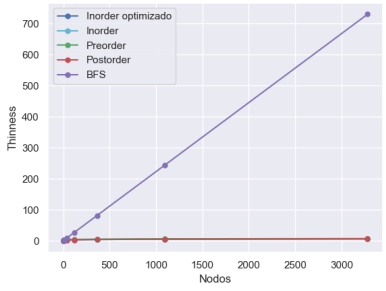
8.1. Experimentación

El objetivo de este experimento fue contrastar los valores de la thinness para árboles con los nodos numerados con los métodos de inorder, preorder, postorder y BFS. También se comparan los valores de la thinness para para los grafos numerados con inorder a los cuales se les realizan las optimizaciones propuestas en la sección 4.4 antes de numerarlos. La metodología consistió en tomar diferentes árboles numerarlos con cada uno de los órdenes y luego correr el algoritmo para calcular el valor de la thinness correspondiente a cada orden para así observar con cuál se obtiene la menor thinness.
Para ello se realizaron diferentes pruebas:

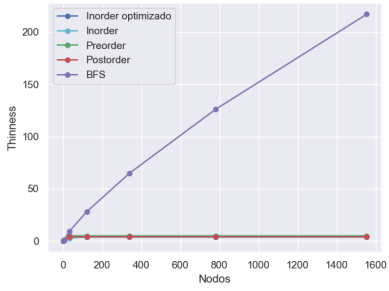
- Con árboles completos variando primero la máxima cantidad de hijos por nodo y luego la altura.
- Con árboles Zig-Zag, variando la altura.
- Con árboles balanceados a izquierda con 2 hijos máximo por nodo.
- Con árboles balanceados a derecha con 2 hijos máximo por nodo.
- Con árboles caterpillar dónde la cantidad de hijos por nodo es 3.

8.1.0.1 Resultados

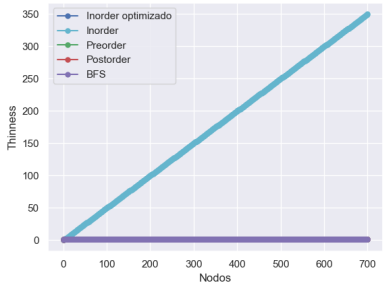
En cada gráfico se muestran las curvas representando cada una un orden distinto para el grafo. En el eje X se muestra la cantidad de nodos del grafo y en el eje Y el valor de la thinness. Al pie de cada gráfico se indica el tipo de árbol y la variable que se mantiene constante seguida del valor fijo utilizado, donde m representa la máxima cantidad de hijos por nodo y h la altura del árbol.



(a) Árbol completo, $m = 3$



(b) Árbol completo, $h = 5$



(c) Árbol Zig-Zag

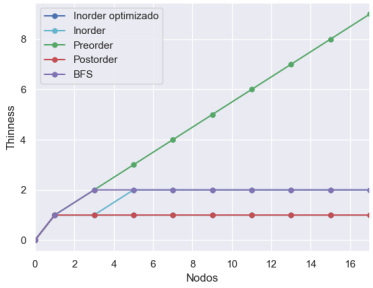
Fig. 8.1: Comparando thinness entre diferentes órdenes

En la figura 8.1 hay curvas que se superponen por lo que no se pueden apreciar. Para completar la información, a continuación aclararemos cuáles son los valores para esas curvas.

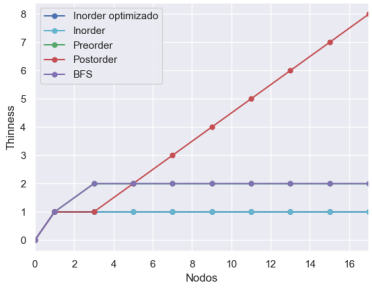
En la Figura 8.1a, para cada grafo utilizado, los valores de la thinness cuando se los numera con inorder y con postorder son idénticos. También coinciden con éstos los valores de la thinness para preorder hasta los grafos con 2 nodos. Para mayor cantidad de nodos, preorder es exactamente uno más que inorder y postorder. La thinness del grafo numerado con inorder en el cual se le hacen las optimizaciones vistas en la sección 4.4 antes de numerarlo y la thinness del grafo sin optimizaciones también numerado con inorder son exactamente iguales.

En la Figura 8.1b, para grafos con hasta 5 nodos los valores de la thinness cuando se los numera con inorder, preorder y postorder coinciden. En el grafo con 31 nodos la thinness del grafo numerado con inorder es 3, con postorder es 4 y con preorder 5. Para grafos con más de 31 nodos el valor de la thinness para inorder y postorder es 4 y para preorder es 5. La thinness del grafo numerado con inorder en el cual se le hacen las optimizaciones vistas en la sección 4.4 antes de numerarlo y la thinness del grafo sin optimizaciones también numerado con inorder son exactamente iguales.

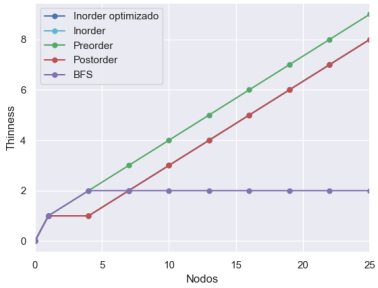
En la Figura 8.1c, para grafos numerados con BFS, preorder, postorder e inorder con optimizaciones el valor de la thinness es 1.



(a) Árbol balanceado a derecha



(b) Árbol balanceado a izquierda



(c) Árbol Caterpillar

Fig. 8.2: Comparando thinness entre diferentes órdenes

En la figura 8.2 hay curvas que se superponen por lo que no se pueden apreciar. Para completar la información, a continuación aclararemos cuáles son los valores para esas curvas.

En la Figura 8.2a, para cada grafo utilizado, los valores de la thinness cuando se los numera con inorder optimizado y con postorder son idénticos. Los grafos numerados con BFS e inorder también coinciden excepto cuando el mismo tiene exactamente 3 nodos, donde inorder tiene thinness 1 y BFS 2.

En la Figura 8.2b, para cada grafo utilizado, los valores de la thinness cuando se los numera con inorder optimizado y con inorder sin optimizar son idénticos. Los grafos numerados con BFS y preorder también coinciden entre ellos.

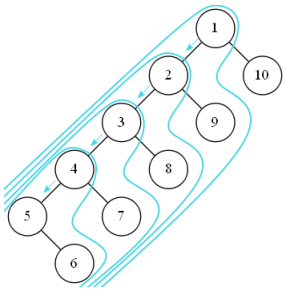
En la Figura 8.2c, para cada grafo utilizado, los valores de la thinness cuando se los numera con inorder con y sin optimizaciones y con postorder son idénticos. El valor de la thinness para los grafos numerados con preorder es exactamente uno más que la thinness de los valores para los otros órdenes.

8.1.0.2 Discusión

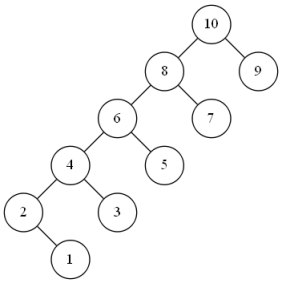
Como se puede observar, los valores de la thinness de grafos completos numerados con inorder realizando antes las optimizaciones vistas en la sección 4.4 antes de numerarlo y sin realizar las optimizaciones son idénticas. Esto es así ya que cuando se trata de grafos completos no hay forma de optimizar.

En los grafos completos también podemos ver que los valores de la thinness para los grafos numerados con BFS es mucho mayor que para los otros órdenes. Esto es así ya que en el caso de BFS la cota está dada por la cantidad máxima de nodos que puede haber en un nivel. En el caso de los grafos completos, este valor es mucho mayor que el de la altura, cota de los órdenes inorder, preorder y postorder. En cambio, en el caso de los grafos Zig-Zag, la cantidad de nodos máxima por nivel es uno sin importar que tan alto sea el árbol. Por lo tanto en este caso el valor de la thinness cuando se numera el grafo con BFS es 1. Similar a lo que sucede con los árboles Zig-Zag sucede con los árboles de tipo caterpillar, pero en éstos la thinness del grafo numerado con BFS es menor que la de los otros órdenes.

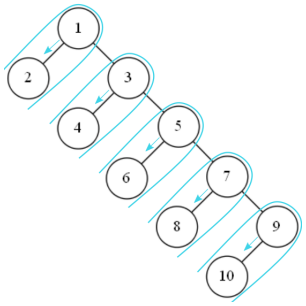
En los árboles balanceados a izquierda podemos ver que postorder es mucho mejor que preorder, al contrario de árboles balanceados a derecha. Esto es así ya que lo que sucede es lo que se muestra en la figura 8.3.



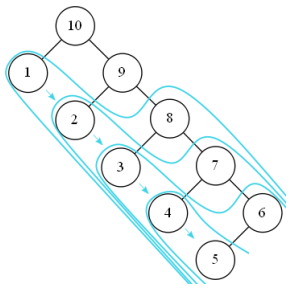
(a) Balanceado a izquierda - Preorder



(b) Balanceado a izquierda - Postorder



(c) Balanceado a derecha - Preorder



(d) Balanceado a derecha - Postorder

Fig. 8.3: Árboles balanceados a izquierda y derecha numerados con preorder y postorder.

8.2. Conclusiones

Por lo que observamos con la experimentación comparando la thinness de diferentes grafos numerados con los distintos órdenes, podemos ver que no hay un caso en el que uno sea mejor que el resto siempre. Dependiendo de la forma del grafo, cuál de los métodos de numeración parece más conveniente. Cuando se trata de árboles más largos que anchos, es decir, árboles en los que la cantidad máxima de nodos en un nivel es menor que la altura del mismo, parece más conveniente usar el método BFS que cualquiera de los otros. En caso contrario, la thinness es menor con inorder, preorder o postorder.
Cuándo el árbol es balanceado a derecha, es mejor usar postorder. En cambio si es balanceado a izquierda, el grafo numerado con preorder tiene menor thinness.

9. CONCLUSIONES Y TRABAJO A FUTURO

A lo largo de esta tesis hemos trabajado con cuatro órdenes para numerar los nodos de un grafo árbol: inorder, preorder, postorder y BFS. Para cada uno de ellos, presentamos un algoritmo para hallar el valor de la thinness del grafo numerado en tiempo lineal utilizando el concepto de grupos de conflicto. Realizamos experimentación comparando, para cada orden, tiempos de ejecución del algoritmo presentado en [BE18] y el algoritmo presentado en esta tesis. En todos los órdenes vimos que el algoritmo propuesto en este trabajo es más eficiente con respecto al tiempo que toma en ejecutarse.

Dado un árbol podemos elegir cómo designar hijos derechos, centros e izquierdos para minimizar la thinness sin alterar las adyacencias. En el caso de los grafos con numeración inorder, presentamos un algoritmo para realizar estas alteraciones al grafo, también de complejidad lineal en la cantidad de nodos del grafo. Mostramos luego, que realizar esas alteraciones antes de numerar el grafo puede reducir el valor de la thinness del mismo. Realizar estas optimizaciones no aumenta el orden de complejidad pero sí el tiempo de ejecución, por lo tanto la elección de optimizar o no deberá ser evaluada cuidadosamente a partir de las preferencias particulares del caso en el que se esté utilizando. Demostramos también que para los órdenes preorder, postorder y BFS no es posible realizar optimizaciones.

Además comparamos el valor de la thinness entre los diferentes órdenes para diferentes tipos de grafos árbol. Como resultado de este experimento observamos que dependiendo de la forma del grafo árbol, cuál de los métodos de numeración es más conveniente. Cuando se trata de árboles más largos que anchos, es decir, árboles en los que la cantidad máxima de nodos en un nivel es menor que la altura del mismo, es más conveniente usar el método BFS que cualquiera de los otros. En caso contrario, la thinness es menor con inorder, preorder o postorder. Por otro lado, definimos árbol balanceado a derecha y árbol balanceado a izquierda y obtuvimos como resultado que cuando el árbol es balanceado a derecha, obtenemos menor thinness al usar postorder. En cambio, si es balanceado a izquierda, el grafo numerado con preorder tiene menor thinness.

Como trabajo futuro se podrían analizar otros órdenes como el pre-post-order o los inversos de todos los ya analizados en este trabajo y pre-post-order.
Sigue quedando como problema abierto estudiar el valor de la thinness para árboles sin tener un orden predeterminado. También queda por investigar qué sucede cuando se tiene

un grafo árbol en el cual hay distintas partes que siguen diferentes patrones. Por ejemplo, un árbol en el que se tiene una parte en la que es un Zig-Zag, otra un completo, otra balanceado a derecha o izquierda. En estos casos podríamos analizar la posibilidad de numerar con diferentes órdenes cada sector de acuerdo con el orden más conveniente.

BIBLIOGRAFíA

[BE18] Flavia Bonomo y Diego de Estrada. «On the thinness and proper thinness of a graph». En: *Discrete Applied Mathematics* (2018). ISSN: 0166-218X. DOI: https://doi.org/10.1016/j.dam.2018.03.072. URL: http://www.sciencedirect.com/science/article/pii/S0166218X1830180X.

[BMO11] Flavia Bonomo, Sara Mattia y Gianpaolo Oriolo. «Bounded coloring of co-comparability graphs and the pickup and delivery tour combination problem». En: *Theoretical Computer Science* 412.45 (2011), págs. 6261-6268. ISSN: 0304-3975. DOI: https://doi.org/10.1016/j.tcs.2011.07.012. URL: http://www.sciencedirect.com/science/article/pii/S0304397511006220.

[CCN12] Marco Casazza, Alberto Ceselli y Marc Nunkesser. «Efficient algorithms for the double traveling salesman problem with multiple stacks». En: *Computers and Operations Research* 39.5 (2012), págs. 1044-1053. ISSN: 0305-0548. DOI: https://doi.org/10.1016/j.cor.2011.06.008. URL: http://www.sciencedirect.com/science/article/pii/S0305054811001705.

[Gol77] Martin Golumbic. «The complexity of comparability graph recognition and coloring». En: *Computing* 18 (sep. de 1977), págs. 199-208. DOI: 10.1007/BF02253207.

[Koz92] Dexter C. Kozen. «Depth-First and Breadth-First Search». En: *The Design and Analysis of Algorithms*. New York, NY: Springer New York, 1992, págs. 19-24. ISBN: 978-1-4612-4400-4. DOI: 10.1007/978-1-4612-4400-4_4. URL: https://doi.org/10.1007/978-1-4612-4400-4_4.

[Man+07] Carlo Mannino, Gianpaolo Oriolo, Federico Ricci-Tersenghi y L. Chandran. «The stable set problem and the thinness of a graph». En: *Oper. Res. Lett.* 35 (ene. de 2007), págs. 1-9. DOI: 10.1016/j.orl.2006.01.009.

[Ola91] Stephan Olariu. «An optimal greedy heuristic to color interval graphs». En: *Information Processing Letters* 37.1 (1991), págs. 21-25. ISSN: 0020-0190. DOI: https://doi.org/10.1016/0020-0190(91)90245-D. URL: http://www.sciencedirect.com/science/article/pii/002001909190245D.

[PM09] Hanne Løhmann Petersen y Oli B.G. Madsen. «The double travelling salesman problem with multiple stacks - Formulation and heuristic solution approaches». English. En: *European Journal of Operational Research* 198.1 (2009), págs. 139-147. ISSN: 0377-2217. DOI: 10.1016/j.ejor.2008.08.009.

[RR88] G. Ramalingam y C.Pandu Rangan. «A unified approach to domination problems on interval graphs». En: *Information Processing Letters* 27.5 (1988), págs. 271-274. ISSN: 0020-0190. DOI: https://doi.org/10.1016/0020-0190(88)90091-9. URL: http://www.sciencedirect.com/science/article/pii/0020019088900919.