Universidad Nacional de San Agustín
Escuela Profesional Ciencia de la Computación
Splay Tree vs Tango Tree

Aaron Tapia[1], **Romel Huamachuco**[1]

[1]Estructuras de Datos Avanzadas

{atapia,rhuamachuco}@unsa.edu.pe

Abstract. *En este artículo presentamos resultados empíricos para árboles splay y tango. Estos resultados proporcionan una mejor comprensión del tiempo de ejecución de una consulta. En base a estos resultados, ofrecemos pautas en términos de tamaño de árbol y patrón de acceso para que estos árboles serán probablemente más eficientes.*
Keywords*: splay tree, tango tree, optimalidad, rendimiento.*

1. Introducción

Un árbol de distribución es un árbol de búsqueda binaria autoajustable con la propiedad adicional de acceso rápido a los elementos accedidos recientemente. Realiza operaciones básicas tales como inserción, búsqueda y eliminación en tiempo amortizado O (log n). Para muchas secuencias de operaciones no aleatorias, los árboles de cobertura funcionan mejor que otros árboles de búsqueda, incluso cuando se desconoce el patrón específico de la secuencia. El árbol desplegable fue inventado por Daniel Sleator y Robert Tarjan en 1985.

Todas las operaciones normales en un árbol de búsqueda binaria se combinan con una operación básica, llamada splaying . Al exponer el árbol en busca de un determinado elemento, se reorganiza el árbol para que el elemento se coloque en la raíz del árbol. Una forma de hacerlo es realizar primero una búsqueda de árbol binario estándar para el elemento en cuestión, y luego usar rotaciones de árbol de una manera específica para llevar el elemento a la parte superior. Alternativamente, un algoritmo de arriba hacia abajo puede combinar la búsqueda y la reorganización de árbol en una sola fase.

Un árbol de tango es un tipo de árbol de búsqueda binaria propuesto por Erik D. Demaine , Dion Harmon , John Iacono y Mihai Pătraşcu en 2004. Lleva el nombre de Buenos Aires , cuyo tango es emblemático.

Es un árbol de búsqueda binario en línea que logra un displaystyle O (log log n) O (log log n) relación competitiva relativa al árbol de búsqueda binaria óptimo sin conexión , mientras que solo se usa displaystyle O (log log n) O (log log n) bits adicionales de memoria por nodo. Esto mejoró con respecto a la relación competitiva más conocida anterior, que era displaystyle O (log n) O (log n) .

2. Marco Teórico

2.1. Splay Tree

Cuando se accede a un nodo x , se realiza una operación de apertura en x para moverlo a la raíz. Para realizar una operación de separación, llevamos a cabo una secuencia de pasos desplegables, cada uno de los cuales mueve x más cerca de la raíz. Al realizar una operación de separación en el nodo de interés después de cada acceso, los nodos a los que se accede recientemente se mantienen cerca de la raíz y el árbol permanece aproximadamente equilibrado, de modo que se alcanzan los límites de tiempo amortizados deseados.

Cada paso en particular depende de tres factores:

- Si x es el hijo izquierdo o derecho de su nodo padre, p ,
- si p es la raíz o no, y si no
- si p es el hijo izquierdo o derecho de su padre, g (el abuelo de x).

Es importante recordar configurar gg (el bisabuelo de x) para apuntar a x después de cualquier operación de apertura. Si gg es nulo, entonces obviamente x ahora es la raíz y debe actualizarse como tal.

Hay tres tipos de pasos de separación, cada uno de los cuales tiene un caso para diestros y zurdos. En aras de la brevedad, solo se muestra uno de estos dos para cada tipo. Estos tres tipos son:

Paso Zig: este paso se realiza cuando p es la raíz. El árbol se gira en el borde entre x y p . Existen pasos en zig para tratar el problema de paridad y se hará solo como el último paso en una operación de separación y solo cuando x tenga una profundidad impar al comienzo de la operación.

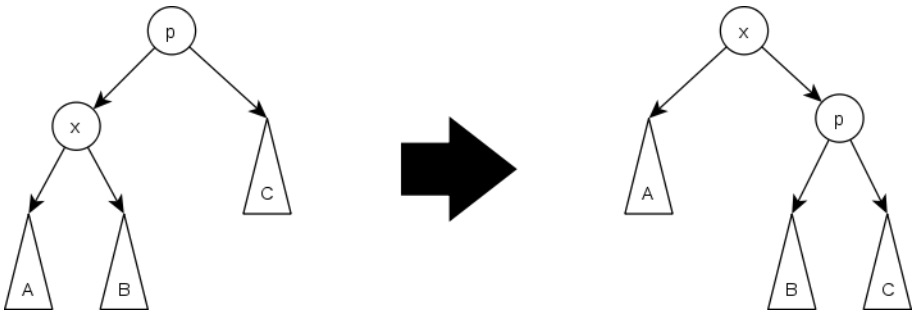

Figura 1. Paso Zig

Paso zig-zig: este paso se realiza cuando p no es la raíz yx y p son ambos hijos correctos o ambos son hijos izquierdos. La siguiente imagen muestra el caso en el que x y p son ambos hijos abandonados. El árbol se gira en el borde uniendo p con su padre g , luego se gira en el borde uniendo x con p . Tenga en cuenta que los pasos de zig-zig son lo único que diferencia a los árboles de dispersión del método de rotación a raíz introducido por Allen y Munro [4] antes de la introducción de los árboles de cobertura.

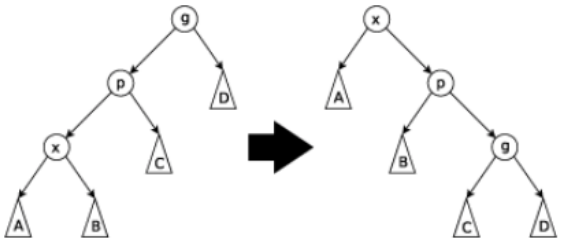

Figura 2. Paso Zig-Zig

Paso zig-zag: este paso se realiza cuando p no es la raíz y x es un niño derecho yp es un niño izquierdo o viceversa. El árbol se gira en el borde entre p y x, y luego se gira en el borde resultante entre x y g.

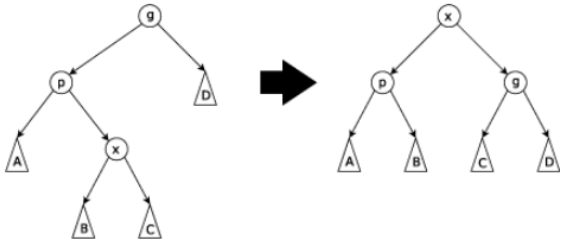

Figura 3. Paso Zig-Zag

2.1.1. Conjetura de optimalidad dinámica

Además de las garantías de rendimiento comprobadas para los árboles de cobertura, hay una conjetura no probada de gran interés del papel original de Sleator y Tarjan. Esta conjetura se conoce como la conjetura de optimalidad dinámica y básicamente afirma que los árboles de dispersión funcionan tan bien como cualquier otro algoritmo de árbol de búsqueda binaria hasta un factor constante.

Conjetura de optimalidad dinámica: Sea A cualquier algoritmo de árbol de búsqueda binario que accede a un elemento x al atravesar el camino desde la raíz hasta x al costo de d(x) +1, y que entre los accesos puede hacer cualquier rotación en el árbol a un costo de 1 por rotación. Dejar A(S) ser el costo de A para realizar la secuencia S de accesos. Entonces, el costo para que un árbol desplegable realice los mismos accesos es O[n + A(S)].

2.2. Tango tree

Los árboles de tango funcionan dividiendo un árbol de búsqueda binario en un conjunto de caminos preferidos , que a su vez se almacenan en árboles auxiliares (por lo que el árbol de tango se representa como un árbol de árboles).

2.2.1. Árbol de referencia

Para construir un árbol de tango, simulamos un árbol completo de búsqueda binaria llamado árbol de referencia , que es simplemente un árbol de búsqueda binaria tradicional que contiene todos los elementos. Este árbol nunca aparece en la implementación real, pero es la base conceptual detrás de las siguientes piezas de un árbol de tango.

2.2.2. Rutas preferidas

Primero, definimos para cada nodo su hijo preferido , que informalmente es el hijo tocado más recientemente por una búsqueda de árbol de búsqueda binaria tradicional. Más formalmente, considere un subárbol T , enraizado en p , con hijos l (izquierda) y r (derecha). Establecemos r como el hijo preferido de p si el nodo al que se ha accedido más recientemente en T está en el subárbol enraizado en r , y l como el secundario preferido en caso contrario. Tenga en cuenta que si el nodo de T más recientemente visitado es p mismo, entonces l es el hijo preferido por definición.

Una ruta preferida se define comenzando en la raíz y siguiendo los hijos preferidos hasta alcanzar un nodo hoja. La eliminación de los nodos en esta ruta divide el resto del árbol en una serie de subárboles, y recurrimos en cada subárbol (formando una ruta preferida desde su raíz, que divide el subárbol en más subárboles).

2.2.3. Árboles Auxiliares

Para representar una ruta preferida, almacenamos sus nodos en un árbol de búsqueda binaria equilibrado , específicamente un árbol rojo-negro . Para cada nodo no hoja n en una ruta preferida P , tiene un hijo c no preferido, que es la raíz de un nuevo árbol auxiliar. Adjuntamos la raíz de este otro árbol auxiliar (c) a n en P , uniendo así los árboles auxiliares. También aumentamos el árbol auxiliar almacenando en cada nodo la profundidad mínima y máxima (profundidad en el árbol de referencia, es decir) de nodos en el subárbol bajo ese nodo.

2.2.4. Algoritmo

Para buscar un elemento en el árbol de tango, simplemente simulamos buscar el árbol de referencia. Comenzamos buscando la ruta preferida conectada a la raíz, que se simula buscando en el árbol auxiliar correspondiente a esa ruta preferida. Si el árbol auxiliar no contiene el elemento deseado, la búsqueda termina en el padre de la raíz del subárbol que contiene el elemento deseado (el comienzo de otra ruta preferida), así que simplemente procedemos buscando el árbol auxiliar para esa ruta preferida , Etcétera.

Para mantener la estructura del árbol de tango (los árboles auxiliares corresponden a las rutas preferidas), debemos realizar algunas tareas de actualización cuando los hijos preferidos cambien como resultado de las búsquedas. Cuando un niño preferido cambia, la parte superior de una ruta preferida se separa de la parte inferior (que se convierte en su propia ruta preferida) y se vuelve a conectar a otra ruta preferida (que se convierte en

la nueva parte inferior). Para hacer esto de manera eficiente, definiremos las operaciones de corte y unión en nuestros árboles auxiliares.

Nuestra operación de combinación combinará dos árboles auxiliares siempre que tengan la propiedad de que el nodo superior de uno (en el árbol de referencia) es un elemento secundario del nodo inferior del otro (esencialmente, que las rutas preferidas correspondientes pueden concatenarse). Esto funcionará según la operación de concatenación de árboles rojo-negro, que combina dos árboles, siempre que tengan la propiedad de que todos los elementos de uno son menos que todos los elementos del otro, y se dividen , lo que hace lo contrario. En el árbol de referencia, tenga en cuenta que existen dos nodos en la ruta superior, de modo que un nodo está en la ruta inferior si y solo si su valor clave está entre ellos. Ahora, para unirnos a la ruta inferior a la ruta superior, simplemente dividimos la ruta superior entre esos dos nodos, luego concatenamos los dos árboles auxiliares resultantes a cada lado del árbol auxiliar de la ruta inferior, y tenemos nuestro árbol auxiliar final unido.

Nuestra operación de corte dividirá una ruta preferida en dos partes en un nodo determinado, una parte superior y una parte inferior. Más formalmente, partirá un árbol auxiliar en dos árboles auxiliares, de modo que uno contenga todos los nodos a una profundidad determinada o superior en el árbol de referencia, y el otro contendrá todos los nodos por debajo de esa profundidad. Al igual que en join , tenga en cuenta que la parte superior tiene dos nodos que delimitan la parte inferior. Por lo tanto, podemos simplemente dividir en cada uno de estos dos nodos para dividir la ruta en tres partes, luego concatenar las dos externas para que terminemos con dos partes, la parte superior e inferior, según se desee.

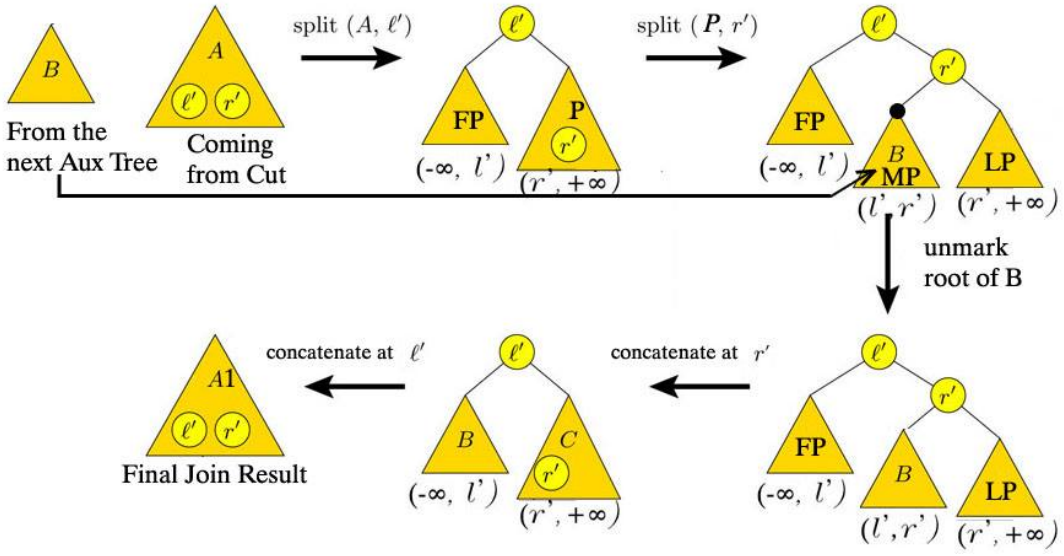

Figura 4. Algoritmo Tango Tree

2.3. Análisis

Para encontrar un límite inferior en el trabajo realizado por el árbol óptimo de búsqueda binaria fuera de línea, nuevamente utilizamos la noción de hijos preferidos. Al considerar una secuencia de acceso (una secuencia de búsquedas), llevamos un registro de cuántas veces cambia el hijo preferido de un nodo de árbol de referencia. El número total de conmutadores (sumados en todos los nodos) proporciona un límite inferior asintótico en el trabajo realizado por cualquier algoritmo de árbol de búsqueda binario en la secuencia de acceso dada. Esto se llama límite inferior de intercalación .

Para conectar esto con árboles de tango, encontraremos un límite superior en el trabajo realizado por el árbol de tango para una secuencia de acceso dada. Nuestro límite superior será (k + 1) O (log log n) , donde k es el número de entrelazamientos.

El costo total se divide en dos partes, la búsqueda del elemento y la actualización de la estructura del árbol de tango para mantener las invariantes adecuadas (cambiar los hijos preferidos y volver a organizar los caminos preferidos).

3. Metodología de Comparación

3.1. Patrones de acceso y tamaño de árboles

Nosotros nos enfocamos en patrones de secuencias de búsqueda. Estas secuencias de búsquedas son secuencias predeterminadas y ordenadas de acceso de datos, sin embargo para nuestro experimento los accesos no seran secuenciales sino aleatorios, esto porque son mas prácticos para las estructuras de datos que estamos tratando.

Estas secuencias aleatorias serán de un 10% del total de elementos del árbol, ya sea splay o tango tree.

Nuestros experimentos incluyeron árboles que varían en tamaño desde 10000 (10^5) nodos hasta 100000000 (10^8) nodos en potencias de 10. Estos tamaños se eligieron para que las búsquedas de secuencia sean mas grandes y asi probar el rendimiento de los árboles. Estos árboles son generados secuencialmente, es decir (1....n) de esta manera al realizar las búsquedas todas seran casos de éxito, esto se hace para simplificar a implementación de los árboles.

3.2. Implementación

Todas las implementaciones se escribieron en C++ y se compilaron utilizando GCC 4.9.1 sin optimización. Los resultados se obtuvieron tras la compilación de los datos explicados anteriormente. La implementación se obtuvo a partir del algoritmo del tango tree del paper de Erik D. Demaine, Dion Harmon, John Iacono, and Mihai Patrascu. Dynamic optimality. El arbol de busqueda binaria equilibrado que indica el paper, que en el caso de esta implemetacion utilizamos el Arbol Red- Black, este por tener un costo computacional de O(log n) para las operaciones de buscar, insertar y borrar.

Se implementaron las funciones cortar y unir subarboles del arbol principal para la construccion de los arboles auxiliares, junto con la funcion de hallar y marcar las rutas privilegiadas.

También se implementó las funciones de dividir y mezclar indicadas en el paper para la actualización de los arboles auxiliares tras una nueva serie de consultas. En los nodos de la estructura tango tree le dimos los atributos de marcado, profundidad, profundidad maxima y minima para poder trabajar con las funciones mencionadas anteriormente.

Finalmente se construyo la función main del programa con la intencion de obtener los resultados de tiempo de compilacion y las graficas de un numero fijo de series de busqueda. Estos tamaños de árboles se eligieron para realizar busquedas de scuencia aleatorias mucho mas grandes.

3.3. Sistema de pruebas

Todas las mediciones de tiempo se realizaron en una máquina Lenovo Y50 Intel Core i7 (4th Gen) 4700HQ / 2.4 GHz, ejecutando Linux Mint. El sistema de prueba tiene 12GB de memoria principal. Se hizo uso de la libreria time.h de c++ para obtener el tiempo de procesador consumido por el programa, haciendo uso de las funciones clock_gettime y clock_realtime.

La comparación se hace de la siguiente manera: n valores se insertan en el árbol (1.....n), y luego usando el mismo árbol ejecutamos las secuencias aleatorias de búsqueda, los tiempos de búsqueda son el promedio de cuatro secuencias aleatorias.

4. Resultados

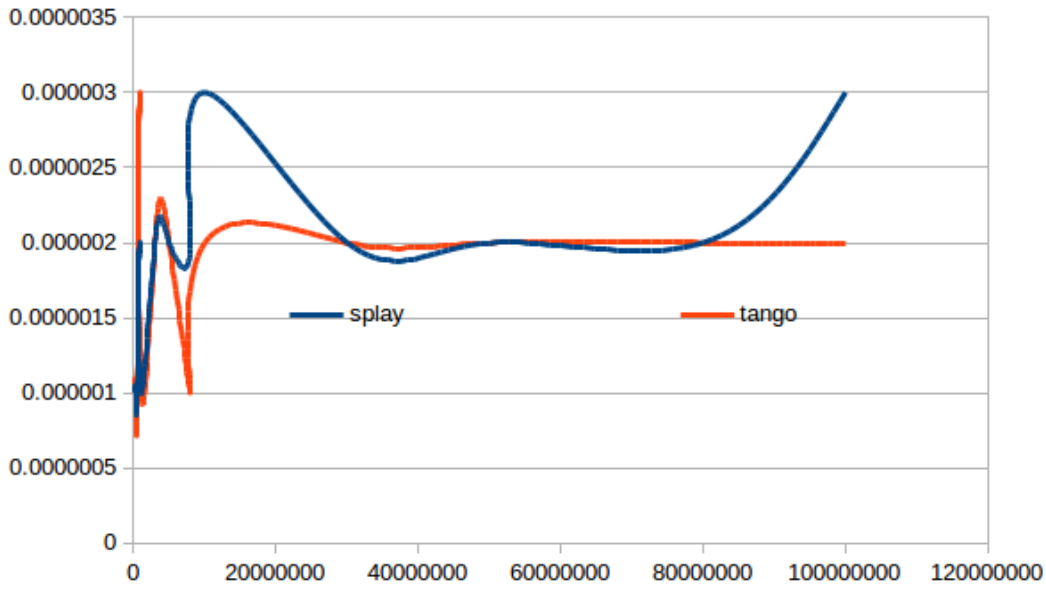

Figura 5. Gráfico de tiempo de ejecución de consulta search

En la Figura 5 tenemos número de nodos del árbol en el eje X y tiempos búsqueda promedio en nanosegundos en el eje Y. Los tiempos de búsqueda son básicamente los mismos para los 2 árboles. La diferencia es pequeña pero parece aumentar un poco a medida que se agregan más nodos al árbol. Tango Tree se comportó un poco mejor ya que mostró que va logrando mas estabilidad que el Splay Tree. En general, los 2 árboles se comportaron de manera muy similar.

Table 1: Comparación de tiempo de consulta(en nanosec)

n(nodos)	Secuencia(rand)	Splay(nsec)	Tango(nsec)
100000	10000	0.000001	0.000001
300000	30000	0.000001	0.000001
500000	50000	0.000001	0.000001
800000	80000	0.000002	0.000003
1000000	100000	0.000001	0.000001
3000000	300000	0.000002	0.000002
5000000	500000	0.000002	0.000002
8000000	800000	0.000002	0.000001
10000000	1000000	0.000003	0.000002
30000000	3000000	0.000002	0.000002
50000000	5000000	0.000002	0.000002
80000000	8000000	0.000002	0.000002
100000000	10000000	0.000003	0.000002

La Tabla 1 muestra los resultados para los diferentes tamaños de árbol y secuencias aleatorias. La primera tendencia notable es que el tango tree a mayor número de nodos se va estabilizando su tiempo de consulta, esto hace suponer que a mayor cantidad de nodos se notaria un mejor desempeño por parte del tango tree sobre el splay tree.

5. Conclusiones

En este artículo, hemos comparado el rendimiento de los árboles splay y tango. En particular, nos hemos centrado en el tiempo de consulta de los dos árboles. Nuestros resultados demuestran que los dos árboles tienen un tiempo de consulta similar, al menos con una cantidad de nodos relativamente pequeña.

El splay tree y el tango tree teóricamente hablando tienen una complejidad de O (log n) y O (log log n) respectivamente, lo cual es una gran diferencia, sin embargo en base a nuestros experimentos podriamos decir que en una aplicación práctica tanto el splay tree como el tango tree tienen un tiempo de consulta similar no reflejando la diferencia teórica.

Ahora como vimos en los experimentos el tango tree le saca una diferencia muy pequeña al splay tree al probarlos con cantidades de nodos cada vez mayores, es así que para se note una ventaja real del tango tree sobre el splay tree la cantidad de nodos tiene que ser extremadamente grande(al menos (10^{40}) nodos).

6. Referencias

[1] Demaine, E. D.; Harmon, D.; Iacono, J.; Pătraşcu, M. (2007). "Dynamic Optimality—Almost". SIAM Journal on Computing.
[2] Sleator DD, Tarjan RE. Self-adjusting binary search trees. Journal of the ACM 1985.
[3] Prosenjit Bose, Karim Douïeb, Vida Dujmović, Rolf Fagerberg. An O(log log n)-Competitive Binary Search Tree with Optimal Worst-Case Access Times.
[4] Ron Wein. Efficient Implementation of Red-Black Trees with Split and Catenate Operations