Estructura de Datos Avanzadas

Prof.: Fernando Esponda

Tarea: Árboles B^+

Guillermo Arredondo Renero

C.U.: 000197256

Introducción.

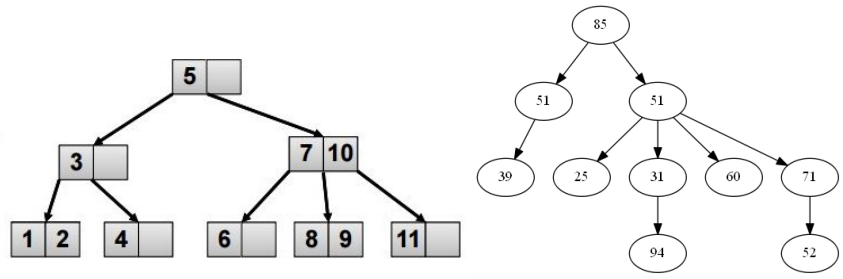
Los árboles B^+ son unas estructuras de datos que, al igual que los árboles generales, son no-lineales y dinámicas. Sin embargo, son una ampliación del volumen de datos que los árboles binarios pueden contener. Los árboles binarios son de los más utilizados en programación debido a su analogía con el lenguaje de máquina (binario), pero existen otro tipo de árboles comúnmente llamados *árboles multi-caminos.* Reciben este nombre puesto que utilizan tanto la memoria principal como la memoria secundaria de la computadora; desafortunadamente, realizar búsquedas en la memoria secundaria de la computadora involucra un mayor coste de tiempo y eficiencia, por ello es que se realizaron ciertas técnicas de grafos multi-caminos que permitieran la recuperación de datos en el menor tiempo posible a través de archivos con índices. Entre las múltiples estructuras de datos con índices, las más utilizadas son los árboles B^+; no obstante, los árboles B^+ son una variante de los árboles B, que explicaremos a continuación.

Árboles B

Los árboles B son árboles completos y compactos propuestos por Bayer y McCreighten en 1970 que generalizan el desarrollo de los árboles balanceados. En lugar de asegurar que cada nodo mantenga un factor de equilibrio entre los dos subárboles hijos guardando un solo dato por cada nodo, lo que hace un árbol B es asegurar que cada nodo contenga por lo menos la mitad de datos del orden (*d*) establecido para el árbol en general. A diferencia de los árboles normales, el orden no determina el número máximo de descendientes, sino que da la pauta para conocer las restricciones de número de datos y número de descendientes por nodo. Así, cada nodo puede contener entre *d* y *2d* datos y tener entre *d+1* y *2d+1* descendientes (exceptuando la raíz, que puede tener como mínimo un dato y dos descendientes).

Nótese que el orden de un árbol B es en realidad el número máximo de pares de claves de referencia que puede contener un nodo o página del árbol.

Lo más interesante de los árboles B es que se organizan de forma contraria a los árboles generales. En tanto que un árbol crece en forma descendiente añadiendo nuevos nodos enlazados a las hojas del árbol y partiendo de una raíz, los árboles B crecen de forma contraria, asegurando que todas las hojas se encuentran en el mismo nivel del árbol (el último), y al insertar debe agregarse en su respectivo lugar ordenado en este último nivel. En el caso en que el nodo en el que debe agregarse el dato se encuentre en su capacidad máxima es necesario recorrer uno de los datos (el de en medio) al nodo padre y así recursivamente hasta asegurar que se cumplan las condiciones establecidas anteriormente. De este modo, puede observarse que los árboles B en realidad crecen en dos sentidos, aumentando el número de datos de los nodos o hacia arriba aumentando niveles y creando nuevas raíces.



Imágenes Árbol B (izquierda) y Árbol General (derecha) obtenidas de Román, F. y Soto, F. (2016) *Árbol B.* URL: https://estructurasite.wordpress.com/ y Stackoverflow (2017) *Calcular altura [a]rbol orientado C++.* URL: https://es.stackoverflow.com/questions/108822/calcular-altura-arbol-orientado-c?rq=1, respectivamente.

Por último, es importante observar que los árboles B y, consecuentemente los árboles B^+, son estructuras de datos ordenadas, de forma que los elementos más chicos siempre se encuentran a la izquierda. A su vez, el árbol B cuenta con las operaciones básicas de las estructuras de datos: inserción, búsqueda y eliminación, aunque tanto la inserción como la eliminación pueden contar con casos especiales que impliquen la reorganización del árbol. En propia opinión, observo a los árboles B como una estructura de combinación entre árboles y arreglos (aunque imagino podrían ser listas enlazadas, cuyos nodos tengan dos enlaces).

Árboles B^+

Un árbol B^+ es un tipo de árbol B con la principal diferencia de la repetición de datos entre los nodos hijos y el padre. Los árboles B^+ son árboles que se caracterizan por mantener todos los datos en las hojas, mientras que los nodos "ancestros" sirven como índices de referencia para agilizar las operaciones en la estructura; sin embargo, es importante resaltar que la longitud del camino hasta el nodo que contiene un dato es siempre el mismo. Aunque los árboles B^+ ocupan un poco más de espacio de memoria que los árboles B lo compensan en eficiencia debido a su forma más compacta, eficiencia en búsqueda y que evita la utilización de reorganización constantemente, sobre todo después de operaciones de eliminación.

Inserción

La inserción en los árboles B^+, para empezar, debe realizarse a través de la búsqueda del lugar correspondiente en orden con respecto a las claves pasadas. El proceso es simple conforme menor sean los espacios llenos en el nodo hoja. Al igual que en su antecesor, los árboles B, la inserción presenta dificultades en los casos en que se intenta insertar un dato en un espacio que alcanza su máxima capacidad, pues en estos casos es necesario llevar a cabo el proceso de solución de desbordamiento. En el caso en que el nodo en el que se quiera insertar el dato se encuentre llena, es necesario dividir dicho nodo en dos, de tal forma que la primera y la segunda mitad (ya incluido el dato nuevo) pasen a ser hojas independientes. El caso del dato de en medio, debe considerarse a parte, puesto que es necesario generar una copia del mismo y añadirla al nodo padre (lo que genera la duplicidad de llaves, solo que la duplicada en realidad solo sirve de índice).

Asimismo, en el caso que la solución de desbordamiento genere un nuevo desbordamiento en el nodo padre, es necesario repetir el proceso, solo que esta vez no se duplicará la entrada del índice (este paso solo se presenta para los casos de desbordamientos en las hojas). Por último, en caso de generarse *h* desbordamientos, significa que se alcanzó a la raíz en la reestructuración, para lo cual será necesario crear una nueva raíz.

Búsqueda

La búsqueda en árboles B y B^+ es muy similar, con la diferencia de que en el segundo caso la búsqueda requiere alcanzar el nivel de las hojas aun cuando ya se haya encontrado el dato buscado. Dado a que los nodos o páginas de los árboles B y B^+ almacenan más de un dato, es necesario recorrer cada uno de ellos, para lo cual puede utilizarse los métodos de búsqueda secuencial o binaria, siendo la segunda más eficiente, sobre todo para grandes volúmenes de datos. La búsqueda debe realizar comparaciones con respecto a los datos almacenados, en caso de ser menor, se recorrerá una determinada rama izquierda, en caso de ser mayor, la del lado derecho, como puede verse en la imagen 2. Como se especificó anteriormente, en el caso de los árboles B^+, estos almacenan datos repetidos en distintas páginas a manera de índice, por lo cual, al igual que en el caso del s*kipList*, es necesario recorrer la

rama en la que se determinó se guardaran los datos iguales al índice (puede elegirse izquierda o derecha).

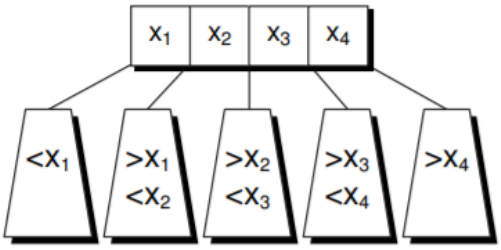


Imagen 2. Comparaciones de ramas en árboles de búsqueda n-arios. Recuperado de: http://dis.um.es/~ginesgm/files/doc/aed/sec4.4.pdf

Borrar

La eliminación de elementos en árboles B y B^+ involucra la necesidad de mantener las restricciones de estos árboles, es decir, el número de elementos mínimo y máximo que cada nodo debe contener, así como el número de descendientes. Esto resulta un tanto complicado en el caso de los árboles B; sin embargo, en el caso de los árboles B^+, la eliminación se facilita. Es muy importante notar que, a diferencia de las *skipList*, los árboles B^+ no consideran la necesidad de eliminar todos los datos que concuerden con el dato a eliminar, pues los datos repetidos en páginas que no sean hojas únicamente sirven a manera de índices; por lo que el único dato eliminado siempre se encuentra en el nivel de las hojas. No obstante, sí es importante asegurar que se mantenga el número deseado de descendientes y datos en la página, por lo que se distingue el caso de la redistribución (fusión de páginas o desplazamiento de elemento) de la siguiente forma:

- En caso de que, al eliminar una clave, la página o nodo contenga menos de *d* elementos, entonces debe tomarse una clave de la página inmediata a la izquierda, indicarla como índice y desplazarla a la página donde se eliminó el elemento. A su vez, si la página de la que se desea tomar este nuevo índice quedaría con menos elementos que *d*, entonces se lleva a cabo una fusión de ambas páginas.

Comparación con Árboles Balanceados (AVL)

Como se aclaró en la introducción, los árboles B son una generalización de los árboles balanceados diseñado para mayor número de datos. Tanto los árboles B como los árboles balanceado AVL se caracterizan por reducir el número de comparaciones durante las operaciones fundamentales por efecto de su reducción en la altura de sus componentes. De este modo, se permite acceder a los datos con un menor recorrido de nodos.

No obstante, los árboles B y árboles B^+ son árboles que buscan ser lo más compactos posibles ante un mayor número de datos. En primera instancia, los árboles AVL buscan mantener un equilibrio entre los nodos que guardan en cada rama, pero debido a que son binarios, son árboles que ante grandes volúmenes de datos sufren un alargamiento considerable; en cambio, los árboles B^+ mantienen el árbol compacto a través de un mayor número de descendientes, lo que permite un menor número de nodos que visitar al realizar las operaciones estructurales usuales. Es decir, existe un mayor número de nodos y, consecuentemente, mucho mayor número de datos almacenados a la misma altura con respecto a la raíz. En términos numéricos, los árboles B aseguran que cada nodo esté por lo menos lleno a la mitad de su capacidad antes de incrementar la altura, pero dado a que esta capacidad es

mayor a dos (cantidad de nodos que asegura un árbol AVL), asegura un mayor contenido a menor altura. Asimismo, los árboles B son aún más compactos, puesto que evitan la repetición de datos, mientras mantienen el mismo número de descendientes y datos almacenados por nodo.

Por último, una vez que los árboles B^+ mantienen a todos sus elementos en el nivel de hojas, se asegura un mismo recorrido hacia cada dato, aunque esto puede también jugar en su contra, sobre todo frente a un árbol AVL, el cual, en el mejor de los casos, puede hacer una sola comparación. Obsérvese que los árboles B y B^+ presentan similitudes en la organización de datos y, por ende, se asemejan en los procesos operacionales. De hecho, la complejidad de los árboles B y árboles B^+, también está representada por $O(\log_d(n))$, pero "[U]n árbol B tiene muchos hijos en un nodo y muchos punteros al nodo hijos. […] Esto permite que el tiempo de recuperación de datos se amortice al tiempo de registro (n), haciendo que el árbol B sea especialmente útil para implementaciones de recuperación de bases de datos y grandes conjuntos de datos." (Saharia, 2011).

En términos de eficiencia, un árbol B ejecuta un menor número de comparaciones que un árbol AVL, debido a la base logarítmica de su complejidad, la cual es mayor a dos. Sin embargo, este menor número de eficiencia es amortiguado debido a la necesidad de cambiar de locación de memoria en los árboles B. Pero, considerando que los árboles B se usan para volúmenes de datos mayores y en general el acceso de información en memoria secundaria es 1000 veces más lento, los árboles B son en extremo eficientes en su objetivo.

Similitud de Árboles B^+ y *SkipList*

Del mismo modo, excluyendo la característica probabilística de las *skipList*, su organización y visualización es muy similar a un árbol B^+, pues en ambos casos, los datos se repiten a manera de índices para facilitar las operaciones básicas de la estructura. Asimismo, se espera en el ideal que las listas auxiliares de la *skipList* se desenvuelvan de menor a mayor número en una forma proporcional, es decir, siguiendo una serie 1, 2, 4, …, n, aproximadamente. Además, en ambos casos es necesario alcanzar el último nivel, ya sea la lista enlazada o el nivel de las hojas, para poder realizar las operaciones indicadas de inserción, eliminación y búsqueda. No obstante, en la implementación de *skipList* visto en clase, la eliminación de datos incluye la eliminación de índices, mientras que en el árbol B esto no se efectúa. Por último, ambas estructuras requieren un método de reestructuración debido a la necesidad de mantener el estado ideal para la eficiencia de operaciones.

Aplicaciones y Usos

1. Los árboles B y B^+ son utilizados principalmente para el manejo eficiente de bases de datos a través de índices, enfocándose principalmente en la disminución del tiempo de entrada y salida de datos del disco duro o memoria secundaria. Esto funciona a partir del ajuste por parámetro de que cada uno de los nodos sea un bloque del disco duro. Dado a que el acceso en memoria secundaria es más tardado, se requieren de múltiples apuntadores que permitan un menor número de nodos que visitar compensado por el tiempo que tarda en entrar al bloque específico.
2. Los árboles B^+ tienen una variante conocida como de "prefijos simples", la cual permite dividir el contenido que se desea guardar a través de un prefijo general que abarque a varios datos específicos, y así categorizar bases de datos. En este sentido, son útiles para la definición e implementación de diccionarios, pues mientras guardan las claves (*keys*) en los índices superiores (los datos repetidos), los valores asociados a dichas claves pueden guardarse en los datos del nivel de hojas.

3. Los árboles B, debido a su eficiencia para el uso y manejo de amplios volúmenes de datos, son muy utilizados para la lectura y escritura de archivos. Debido a su facilidad para el acceso a bloques de memoria secundaria y la reestructuración de bloques mediante la fusión de páginas, pueden lograrse aplicaciones de escritura y unión de archivos. Asimismo, los árboles B son utilizados como sistemas de ficheros. Esto significó un gran progreso en la búsqueda de información, pues dejó de ser secuencial y pasó a ser "aleatoria", lo que permitió una conexión entre archivos, carpetas y subcarpetas más eficiente y versátil para el usuario.

Bibliografía:

- Guardati, S., & Cairó, O. (1993). *Estructuras de Datos* (Primera Edición). McGraw-Hill Education.
- Blancarte, O. (2020, 27 agosto). *Estructura de datos - Árboles*. Oscar Blancarte - Software Architecture. https://www.oscarblancarteblog.com/2014/08/22/estructura-de-datos-arboles/
- Román, F., & Soto, F. (2016, 7 junio). *Árbol // Árbol B // Árbol B+*. Estructura de Datos II. https://estructurasite.wordpress.com/arbol/
- Saharia, K. (2010, 29 abril). *data-structures — Árbol AVL vs. árbol B* [Foro]. it-swarm-es.com. https://www.it-swarm-es.com/es/data-structures/arbol-avl-vs.-arbol-b/970011400/
- *Sistemas de archivos: qué son y cuáles son los más importantes*. (2021, 21 octubre). IONOS Digitalguide. https://www.ionos.mx/digitalguide/servidores/know-how/sistemas-de-archivos/
- UAM. (2021). *Árboles B y B+* [Diapositivas]. PDF. https://academicos.azc.uam.mx/jfg/diapositivas/almacenamiento/Unidad_8.pdf
- Universidad de Murcia. (2003). Capítulo 4. Representación de conjuntos mediante árboles. En G. García, J. Cervantes, N. Marín, & D. Giménez (Eds.), *Algoritmos y Estructuras de Datos* (Estructuras de Datos ed., Vol. 1, pp. 168–177). Editorial Diego Marín. http://dis.um.es/~ginesgm/files/doc/aed/sec4.4.pdf