Pontificia Universidad Católica de Chile
Departamento de Ciencia de la Computación
IIC2133 - Estructuras de Datos y Algoritmos

Ayudantía 2

24 de abril de 2020

Árboles de Búsqueda Binarios (ABB)

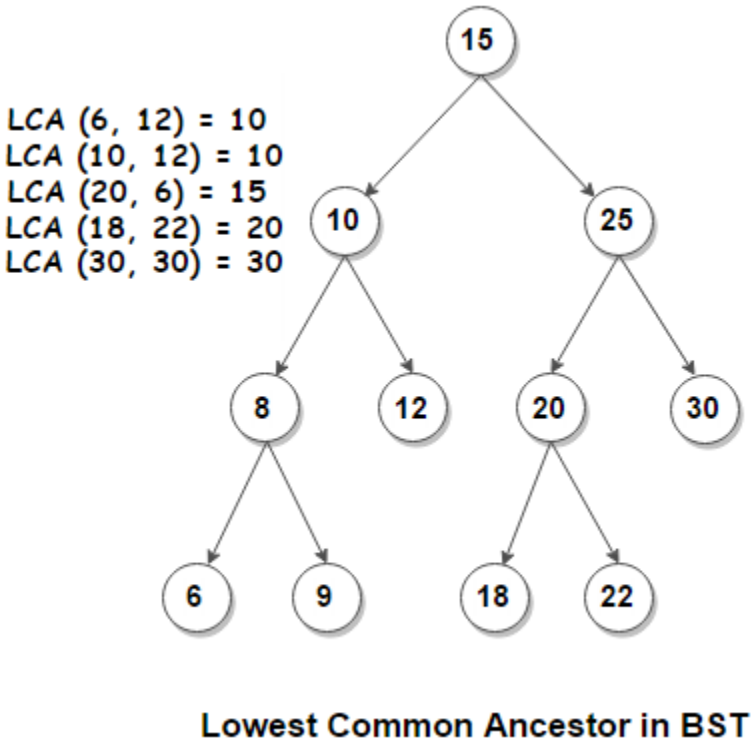
Pregunta 1 [I1 2014-1 P3]

a) Supongamos que tenemos una estimación por adelantado de cuán a menudo nuestro programa va a tener acceso a las claves de los datos (es decir, conocemos la frecuencia de acceso a las claves). Si queremos emplear un *ABB*, inicialmente vacío, para almacenar los datos (según sus claves), ¿en qué orden deberíamos insertar las claves en el árbol? Justifica.

b) Supongamos que cada vez que nuestro programa tiene acceso a una clave k, necesita saber cuántas claves en el árbol son menores que k. Esto se conoce como el rango de k. ¿Qué información adicional sería necesario almacenar en el árbol?¿Cómo se determinaría el rango de k y cuál sería la complejidad de esta operación?¿Cuánto costaría mantener actualizada la información adicional del árbol cuando se produce una inserción o una eliminación de una clave?

Pregunta 2 [Propuesta]

El *lowest common ancestor*(LCA) [más bajo común antecesor] entre dos nodos x e y en un ABB, es el nodo más bajo (profundo) que tiene tanto a x como a y como descendientes, donde cada nodo puede ser descendiente de sí mismo (si x es alcanzable desde y, entonces el LCA es y).
En otras palabras, el LCA entre x e y es el antecesor compartido (incluidos ambos como opciones) que está más lejos de la raíz.
Describa la lógica de cómo sería un algoritmo eficiente en términos de tiempo y de memoria, para encontrar el LCA entre dos nodos x e y.



Lowest Common Ancestor in BST

AVL

Pregunta 3

a) ¿ Es todo árbol AVL un árbol rojo-negro? Justifique o dé un contraejemplo. (I2 2015-2)

b) Describe un algoritmo de certificación para árboles AVL, suponiendo que sabemos que el árbol es un ABB. Es decir, tu algoritmo debe verificar que la propiedad de balance del árbol se cumple y que la información de balance mantenida en cada nodo esté correcta. Tu algoritmo debe correr en tiempo O(número de nodos del árbol) (I1 2014-1)

c) Propuesto: demuestra la correctitud y finitud del algoritmo anterior

SOLUCIÓN

Pregunta 1

a)

La primera que se ingrese será la raíz del árbol ABB. Para el resto, mientras antes se ingresen, más cerca quedaran de ella. Queremos que los que más consultemos queden lo más disponibles posbile, y por lo tanto lo más cerca de la raíz posible. Luego, para la mayoría de los casos será conveniente ingresar los datos en orden de mayor a menor frecuencia.

b)

Lógica

Algoritmo recursivo que dado un nodo sigue uno de tres casos al comparar su clave con la de la raíz:

- **Es raíz**: cantidad de nodos en subárbol izquierdo
- **Menor a raíz**: rango dentro de subárbol izquierdo (se descarta raíz y todo el subárbol derecho)
- **Mayor a raíz**: cantidad de nodos en subárbol izquierdo + 1 (raíz) + rango dentro de subárbol derecho

Implementación: información adicional

Para cada nodo t, un campo *size* con la cantidad de nodos en el subárbol con raíz t (cantidad de nodos del subárbol izquierdo de su nodo padre).

Implementación: mantención de información

Hay dos operaciones en las que se debe realizar mantención a la información:

1. **Inserción**: *n.size += 1* para cada nodo n en la ruta de inserción, *m.size = 1* para el nodo insertado m. (Se mantiene complejidad O(h) [= O(log(n)) u O(n)] para inserción).
2. **Eliminación**: *n.size -= 1* para cada nodo n en la ruta de búsqueda desde la raíz hasta el nodo padre del eliminado. (Se mantiene complejidad O(h) [= O(log(n)) u O(n)] para eliminación).

Complejidad

1. Consulta: O(h)
2. Mantención: O(h) para inserción y para eliminación. Conserva complejidad *big O* de estas operaciones.

— h = altura del árbol

Pregunta 2

Hay distintas formas de implementar una solución, con distintas complejidades.
h será la altura del árbol.

- **Naive [Ingenua]**:
  Dos arreglos que almacenen la ruta desde la raíz hasta la hoja consultada. Comparar hasta que sean distintos o uno termine, y el último comparado será el LCA.
  Complejidad de tiempo es 3*O(h) = O(h); de memoria auxiliar es 2*O(h) = O(h)
- Recursiva: Se llama a recursión que hace:
  - Caso base: Si el árbol es vacío retorna nulo.
  - Si x e y son ambos menores que n, llama a recursión con su hijo izquierdo
  - Si x e y son ambos mayores que n, llama a recursión con su hijo derecho
  - Si uno entre x e y es mayor (o igual) y el otro es menor (o igual) que n, retorna n como el LCA

  Complejidad de tiempo es O(h); requiere espacio O(h) para *stack call*
- Iterativa:
  - Si el árbol es vacío o x o y no están en él, retorna nulo.
  - Recorre en ciclo while desde n el nodo raíz realizando:
  - Si x e y son ambos menores que n, continúa reemplazando n con su hijo izquierdo
  - Si x e y son ambos mayores que n, continúa reemplazando n con su hijo derecho
  - Si uno entre x e y es mayor (o igual) y el otro es menor (o igual) que n, retorna n como el LCA

  Complejidad de tiempo es O(h); memoria auxiliar necesaria es O(1)

La implementación más eficiente al considerar tiempo y memoria es la iterativa.

Pregunta 3

a)

Sí, para esto analicemos las propiedades de los árboles:

Propiedades árbol AVL:

1. ABB

2. Las alturas de las ramas hijas difieren como máximo en 1
3. Todo nodo del árbol, es también la raiz de un árbol AVL

Propiedades árbol rojo-negro:

1. Todo nodo es rojo o negro
2. La raíz es negra
3. Todas las hojas son negras
4. Un nodo rojo, no puede tener hijos rojos
5. Todos los caminos desde un nodo a sus hojas descendientes tienen el mismo número de nodos negros

Utilizaremos el hecho de que la ruta (directa) más larga, no puede tener más del doble de nodos que la ruta más corta. De esta forma, sea d la diferencia entre el número de nodos de la ruta más corta con una ruta cualquiera. Hacemos que todos los nodos de la ruta más corta sean negros, luego debemos pintar d nodos rojos. Finalmente, partimos pintando la hoja roja y, de ahí hacia arriba nodo por medio, hasta completar d nodos pintados.

b)

El algoritmo asigna una "altura inverssa" a cada nodo (una hoja tiene altura 0) y la altura de un nodo que no es una hoja es uno más que la altura de su subárbol más alto.

El algoritmo comienza en la raíz del árbol. Se pregunta recursivamente por las alturas de sus hijos. Si el nodo es una hoja, entonces su balance debe ser 0. Si el balance no es 0, entonces el algoritmo responde "falso" y termina; de lo contrario, el algoritmo devuelve 0 (su altura).

Si estas ejecuciones recursivas vuelven con los valores h_{izq} y h_{izq}, entonces el algoritmo verifica la propiedad de árbol AVL (h_{izq} y h_{izq} no pueden diferir en más de uno y el balance del nodo debe ser 0, –1 o +1, dependiendo de la relación entre h_{izq} y h_{izq}). Si cualquiera de estas verificaciones no se cumple, entonces el algoritmo responde "falso" y termina; de lo contrario, el algoritmo devuelve la altura del nodo y, si el nodo es la raíz del árbol, el algoritmo responde "verdadero".