Tema: Árboles de búsqueda autoajustables (*Splay Trees*)

Asignatura: Métodos Algorítmicos de Resolución de Problemas. **Autor:** Ricardo Peña

Curso 2015/16

1. Introducción

⋆ Los *árboles binarios de búsqueda autoajustables* (ABBA), llamados *splay trees* en los libros de texto, fueron inventados por D. Sleator y R. Tarjan en 1985, en un artículo de título *Self-adjusting binary search trees*[1]. Se trata de una más de las estructuras de datos que estos autores idearon en esos años para ilustrar la utilidad del análisis amortizado del coste, análisis del que también fueron autores.

⋆ Se trata de árboles binarios de búsqueda con las operaciones habituales de *buscar*, *insertar* y *borrar* una clave. A diferencia de los AVL, aquí no se hace ningún esfuerzo por mantener un invariante de equilibrio en altura, con lo que el caso peor de dichas operaciones puede llegar a tener un coste en tiempo en $O(n)$, siendo n el cardinal del árbol.

⋆ En su lugar, cada una de las operaciones mencionadas desencadena una transformación del árbol que afecta a todo el camino seguido para buscar, insertar, o borrar, la clave en cuestión, y que altera el coste de las mismas tan solo en una constante multiplicativa.

⋆ Esa transformación, llamada un "*splay*", que aquí traduciremos por una **flotación**, modifica la función de potencial de la estructura, de forma que es posible atribuir un coste amortizado en $O(\log n)$ a cada una de las operaciones, en cualquier secuencia de inserciones, búsquedas y borrados.

⋆ Una flotación es una **secuencia de rotaciones** muy similares a las que se han visto para los AVL. Cada rotación tiene también, como en los AVl, un coste constante,

⋆ Una posible ventaja de estos árboles con respecto a los AVL es que los nodos no necesitan llevar ninguna información adicional a la clave. En los AVL era necesario llevar en cada nodo, o bien la altura del subárbol que tiene por raíz dicho nodo, o bien un pequeño entero que informa del estado de equilibrio del subárbol.

2. Algoritmos

⋆ Los algoritmos para buscar, insertar y borrar una clave en un ABBA son **exactamente** los mismos que los de un árbol binario de búsqueda convencional, salvo que incorporan como última acción una flotación.

⋆ La flotación se inicia en un nodo, que llamaremos *origen* de la flotación, que es:

- En *buscar*, el nodo buscado, o el último nodo visitado, si el nodo buscado no se encuentra en el árbol.
- En *insertar*, el nodo insertado, o el nodo que ya contenía dicha clave, si no se ha realizado la inserción.
- En *borrar*, el padre del nodo borrado, o el último nodo visitado, si el nodo a borrar no se encuentra en el árbol.

[1] Journal of the ACM 32:3, pp. 652–686, 1985.

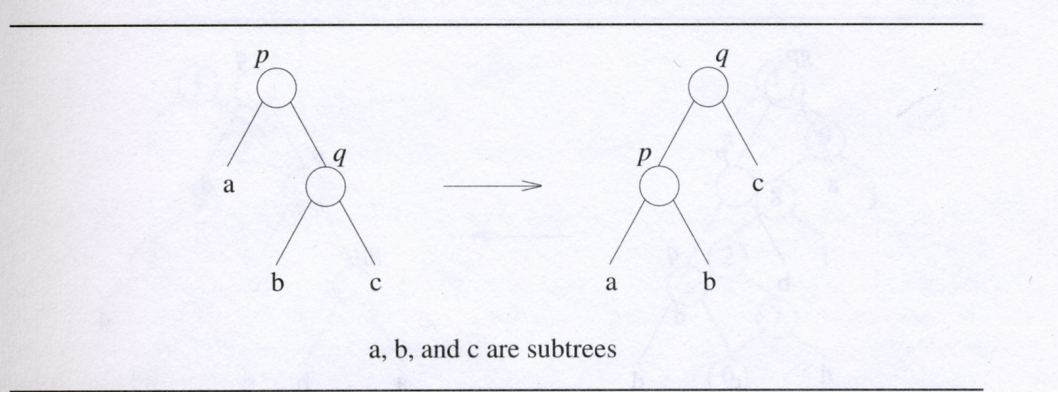

Figura 1: Rotación RR cuando q es hijo directo de la raíz

⋆ El objetivo de la flotación es **flotar el nodo origen** hasta convertirlo en la raíz del árbol. Para ello, se realiza una cadena de *rotaciones* en sentido ascendente, cada una de las cuales se asemeja a alguna de las rotaciones vistas en los AVL, aunque no son exactamente las mismas.

⋆ Sea q el nodo que desencadena la rotación. Inicialmente, q es el origen de la flotación. Casos:

1. Si q es el nodo raíz, la flotación termina.
2. Si q es hijo directo de la raíz, se desencadena una rotación RR o LL idéntica a la de los AVL, que hace que q pase a ser la raíz. En la Fig. 1 (tomada de [1]) se muestra el caso RR. La flotación termina.
3. Si q tiene padre p y abuelo gp, se desencadena una rotación LL, LR, RL o RR, según que q sea respectivamente el hijo izquierdo del hijo izquierdo de gp, el hijo derecho del hijo izquierdo de gp, el hijo izquierdo del hijo derecho de gp, o el hijo derecho del hijo derecho de gp. Las rotaciones LR y RL son exactamente las mismas que las de los AVL, mientras que las LL y RR son ligeramente distintas. En la Fig. 2 (tomada de [1]) se muestran las rotaciones RR y RL. La flotación continua en la nueva posición de q.

⋆ En la Fig. 3 (tomada de [1]) se muestra la secuencia completa de rotaciones a la que da lugar la flotación de la clave 5 (en gris).

3. Estudio del coste

⋆ Como las flotaciones tienen el mismo coste que la operación de inserción, búsqueda, o borrado que la desencadena, basta con estudiar el coste amortizado de una flotación.

⋆ Utilizaremos el **método del potencial**. Si q identifica un nodo del árbol, denotaremos por $n(q)$ al **cardinal** del subárbol que tiene por raíz dicho nodo.

⋆ Dado un nodo q del árbol, definimos el **rango** de q, denotado $r(q)$, como $r(q) = \lfloor \log n(q) \rfloor$. El logaritmo se entenderá en base 2.

⋆ El **potencial** de un árbol t se define como $\Phi(t) = \sum_{q \in t} r(q)$. Notese que $\Phi(avacio) = 0$ y que para todo árbol t, $\Phi(t) \geq 0$. En estas condiciones, sabemos que la suma de costes amortizados de una secuencia de operaciones es una cota superior de la suma de sus costes reales.

⋆ Consideraremos que el coste real de cualquier rotación es **una unidad**. Si q es el nodo del árbol que desencadena la rotación, denotaremos por $r(q)$ y $r'(q)$ a los rangos de q **antes** y **después** de la misma, y por t y t' a los árboles totales correspondientes. Si $\hat{c}$ designa el coste amortizado de una sola rotación, tenemos entonces:

$$\hat{c} = 1 + \Phi(t') - \Phi(t) = 1 + \Delta\Phi$$

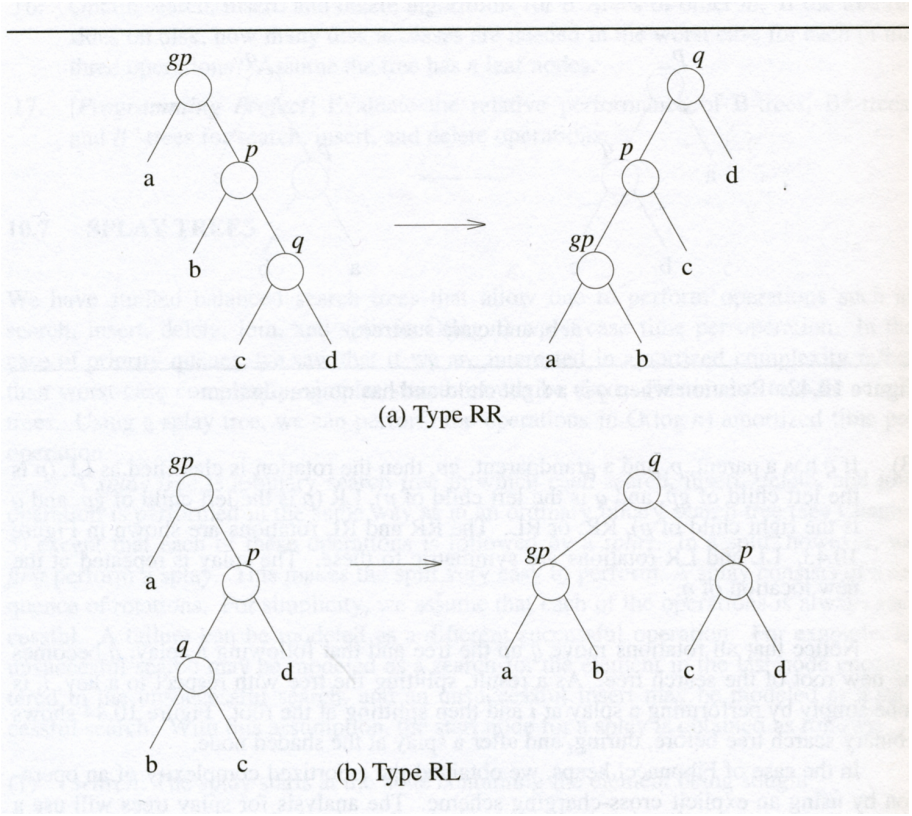

Figura 2: Rotación RR general y rotación RL

⋆ Estudiamos el incremento/decremento de potencial según el tipo de rotación:

1. Si q es hijo directo de la raíz y la rotación es RR (el caso LL es idéntico), según la Fig. 1 tenemos:
$$\Delta\Phi = r'(p) + r'(q) - r(p) - r(q)$$
ya que solo los nodos p y q cambian su rango. Al ser $r'(p) \leq r(p)$, tenemos $\Delta\Phi \leq r'(q) - r(q)$ y $r'(q) - r(q) \geq 0$. Por tanto, en este caso $\hat{c} \leq r'(q) - r(q) + 1 \leq 3(r'(q) - r(q)) + 1$. Obsérvese que este coste sólo se ha de cargar una vez en cada flotación.

2. Si q tiene padre p y abuelo gp, en cualquiera de las rotaciones q pasa a ser la raíz de todo el subárbol. Por tanto, tenemos $r'(q) = r(gp)$ y en consecuencia:
$$\Delta\Phi = r'(gp) + r'(p) + r'(q) - r(gp) - r(p) - r(q) = r'(gp) + r'(p) - r(p) - r(q) \qquad (1)$$
Consideremos las rotaciones RR y RL de la Fig. 2 (el razonamiento es el mismo para los casos LR y LL). Observando las posiciones inicial y final de q, p y gp, se deduce inmediatamente que $r'(p) \leq r'(q)$, $r'(gp) \leq r'(q)$ y $r(q) \leq r(p)$. Por tanto, tenemos $\Delta\Phi \leq 2(r'(q) - r(q))$ y $\hat{c} \leq 2(r'(q) - r(q)) + 1$, con $r'(q) - r(q) \geq 0$.
Si sumásemos los costes amortizados de todas las rotaciones a lo largo de la flotación, el sumando 1 haría que el coste amortizado de la flotación fuera $O(n)$, ya que el camino más largo a una hoja está en $O(n)$ en el caso peor. Lo que sigue tiene por objetivo eliminar dicho sumando, aun a costa de subir la constante multiplicativa 2. Distinguimos dos casos:

   $r'(q) > r(q)$ Como al menos hay una unidad de diferencia, deducimos $\Delta\Phi \leq 3(r'(q) - r(q)) - 1$ y $\hat{c} \leq 3(r'(q) - r(q))$.

   $r'(q) = r(q)$ Entonces tenemos también $r'(q) = r(p) = r(gp)$, ya que $r(q) \leq r(p) \leq r(gp)$. Por otro lado, en el caso RR, $n'(q) > n(q) + n'(gp)$ y $r'(gp) \leq r'(q)$ (ver figura). Veamos que, además, $r'(gp) < r'(q)$. En efecto, si fuera $r'(gp) = r'(q)$ y sabiendo que, por

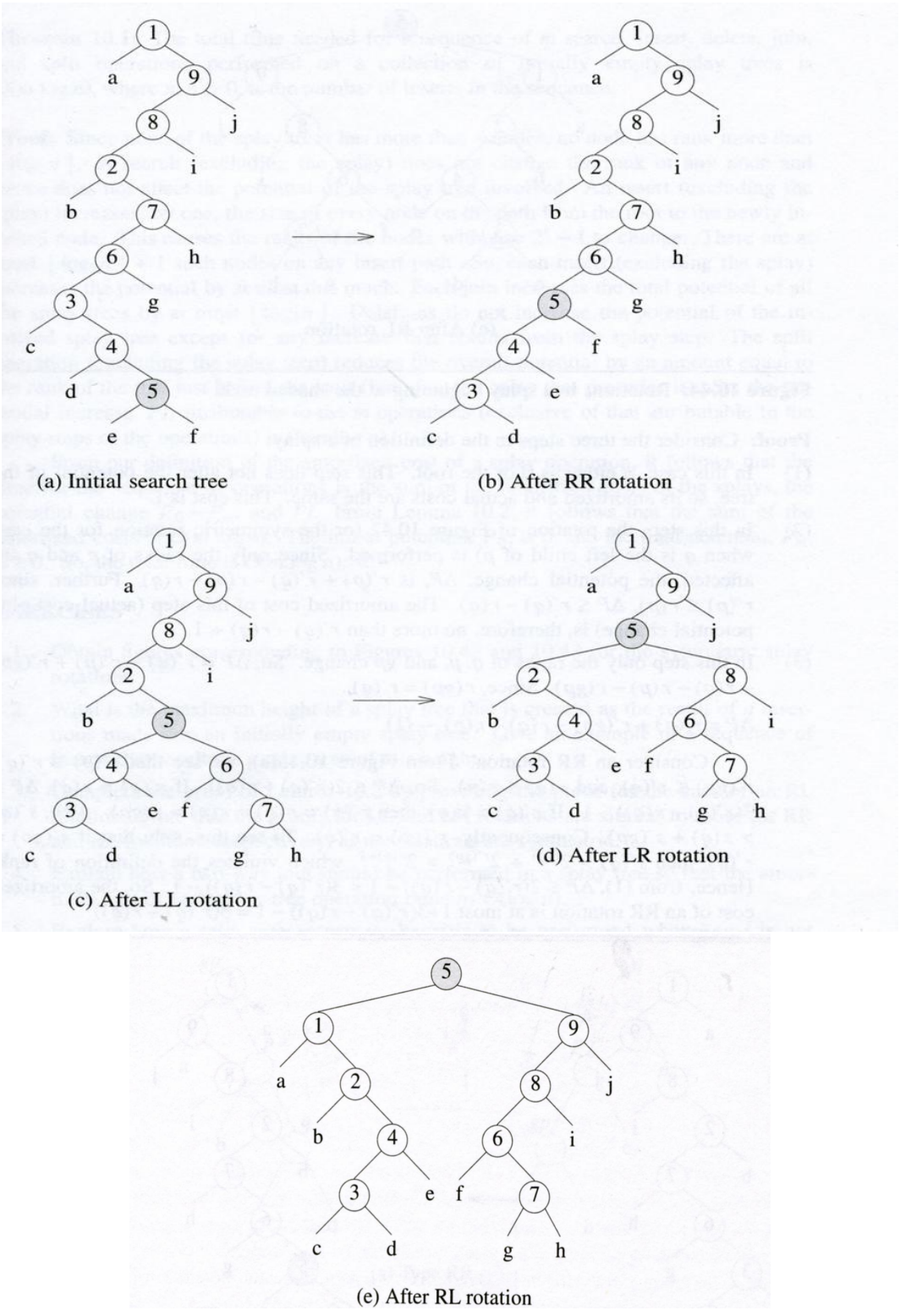

Figura 3: Secuencia de rotaciones en una flotación

definición de rango, $n(s) \geq 2^{r(s)}$ para cualquier nodo s, deduciríamos:

$$\begin{array}{ll} & n'(q) > n(q) + n'(gp) \\ \Rightarrow & n'(q) > 2^{r(q)} + 2^{r'(gp)} \\ \Rightarrow & n'(q) > 2^{r'(q)} + 2^{r'(q)} = 2^{r'(q)+1} \\ \Rightarrow & \log n'(q) > \lfloor \log n'(q) \rfloor + 1 \end{array}$$

que es un absurdo. A partir de (1) tenemos entonces:

$$\begin{array}{lll} & \Delta\Phi = r'(gp) + r'(p) - r(p) - r(q) & \\ \leq & r'(q) - 1 + r'(q) - r(p) - r(q) & \text{-- por ser } r'(gp) < r'(q) \text{ y } r'(p) \leq r'(q) \\ \leq & 2(r'(q) - r(q)) - 1 & \text{-- por ser } r(p) \geq r(q) \\ = & 3(r'(q) - r(q)) - 1 & \text{-- por ser } r'(q) = r(q) \end{array}$$

El caso RL es similar. Aquí tenemos $n'(q) > n'(p) + n'(gp)$, $r'(gp) \leq r'(q)$ y $r'(p) \leq r'(q)$ (ver figura). Veamos que no pueden darse a la vez $r'(gp) = r'(q)$ y $r'(p) = r'(q)$. Si así fuera, deduciríamos $n'(q) > 2^{r'(p)} + 2^{r'(gp)} = 2^{r'(q)+1}$ que, al igual que antes, sería absurdo. Por tanto, en uno de los dos casos al menos, la desigualdad es estricta. Si fuera $r'(gp) < r'(q)$, el resto del razonamiento es el mismo que el del caso RR. Si fuera $r'(p) < r'(q)$, el razonamiento es similar.
Concluimos que, si $r'(q) = r(q)$, también $\hat{c} \leq 3(r'(q) - r(q))$.

⋆ Sumando ahora los costes amortizados de todas las rotaciones i a lo largo de una flotación, y observando que $r'(q_i) = r(q_{i+1})$, donde q_i denota el nodo q durante la rotación i-ésima, obtenemos:

$$\begin{array}{ll} & \sum_i \hat{c}_i \\ \leq & \sum_i 3(r'(q_i) - r(q_i)) + 1 \\ = & 3(r(q_{t'}) - r(q_0)) + 1 \\ = & 3(\lfloor \log n \rfloor - r(q_0)) + 1 \\ \in & O(\log n) \end{array}$$

donde q_0 es el nodo origen de la flotación y $q_{t'}$ es la raíz del árbol final resultante.

⋆ Concluimos que el coste amortizado de cada flotación, y por tanto de la correspondiente operación de inserción, búsqueda, o borrado que la desencadena, es **logarítmico** en el cardinal del árbol.

⋆ La intuición que se esconde tras la definición de potencial dada por Sleator y Tarjan que ha conducido a este resultado, es que se asigna un potencial más alto a los árboles más desequilibrados, y más bajo los más equilibrados. Las operaciones de alto coste real que se producen en árboles del primer tipo se pagan en parte con el potencial del árbol, con lo cual el potencial disminuye y el árbol resultante resulta más equilibrado. Ello reduce el coste real de las subsiguientes operaciones. El nombre de *splay* dado a la acción de flotación quedaría entonces justificado ya que produce un ensanchamiento, a la vez que un aplastamiento, del árbol.

Lecturas complementarias

Estas notas están basadas en la Sección 10.7 de [1], donde hay más ilustraciones y explicaciones con las que el lector podrá completar lo resumido aquí.

Referencias

[1] E. Horowitz, S. Sahni, and D. Mehta. *Fundamentals of Data Structures in C++*. Computer Science Press, 4th edition, 1997.