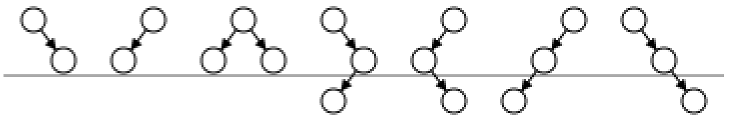
維基百科
自由的百科全書

AA树

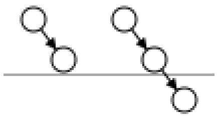
维基百科，自由的百科全书

AA樹在電腦科學一種形式的自平衡二元搜尋樹用於高效存儲和檢索序數據。AA樹的名稱是由它的發明者阿爾尼·安德森（Arne Andersson）而來。

AA樹是紅黑樹的一種變種，是安德森教授在1993年年在他的論文《Balanced search trees made simple》中介紹，設計的目的是減少紅黑樹考慮的不同情況，區別於紅黑樹的是，AA樹的紅節點只能作為右葉子，從而大大簡化了維護2-3樹的模擬。維護紅黑樹的平衡需要考慮7種不同的情況:



因為AA樹有嚴格的條件(紅節點只能為右節點)，故只需考慮2種情形:



旋轉平衡

平衡一顆紅黑樹需要記錄其顏色，而AA樹是在每個節點記錄其"level"這相當於紅黑樹節點的黑高度

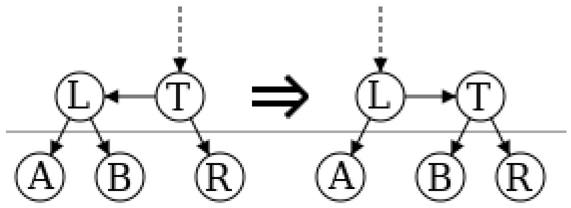
1. 所有葉節點的level都是1
2. 每個左孩子的level恰好為其父親的level減一
3. 每個右孩子的level等於其父親的level或為其父親的level減一
4. 每個右孫子的level嚴格小於其祖父節點的level
5. 每一個level大於1的節點有兩個子節點

兩個level相同的點之間的邊*水平邊*，也就是紅黑樹上的紅邊。往右的水平邊是允許的，但不可連續(紅黑樹性質)；不能有向左的水平邊(AA樹性質)。因為AA樹的條件比紅黑樹嚴格，所以重新平衡一顆AA樹會比重新平衡一顆紅黑樹容易。

插入和刪除會讓AA樹變的不平衡(即違反它的性質)。恢復平衡只需兩種操作:"skew"和"split". Skew是一個右旋轉使得子樹中向左的水平邊變成向右的水平邊；Split是一個左旋並增加子樹根節點的level(請看範例)使得連續向右的水平邊消失。平衡插入和刪除操作的實現是由skew及split決定是否旋轉，而不是在主程式中判斷。

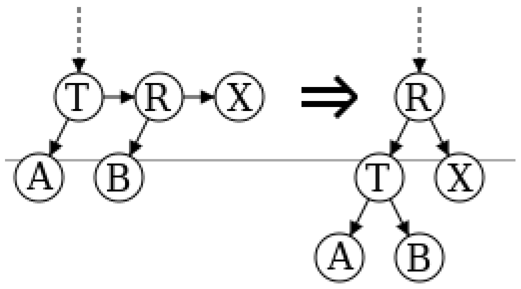
```
function skew is
    input: T, a node representing an AA tree that needs to be rebalanced.
    output: Another node representing the rebalanced AA tree.

    if nil(T) then
        return Nil
    else if nil(left(T)) then
        return T
    else if level(left(T)) == level(T) then
        Swap the pointers of horizontal left links.
        L = left(T)
        left(T) := right(L)
        right(L) := T
        return L
    else
        return T
    end if
end function
```

Skew:


```
function split is
    input: T, a node representing an AA tree that needs to be rebalanced.
    output: Another node representing the rebalanced AA tree.

    if nil(T) then
        return Nil
    else if nil(right(T)) or  nil(right(right(T))) then
        return T
    else if level(T) == level(right(right(T))) then
        We have two horizontal right links.  Take the middle node, elevate it, and return it.
        R = right(T)
        right(T) := left(R)
        left(R) := T
        level(R) := level(R) + 1
        return R
    else
        return T
    end if
end function
```

Split:


插入

在遞迴的實做中，除了葉節點之外，在每次的遞迴結束後呼叫skew和split即可

```
function insert is
    input: X, the value to be inserted, and T, the root of the tree to insert it into.
    output: A balanced version T including X.

    Do the normal binary tree insertion procedure. Set the result of the
    recursive call to the correct child in case a new node was created or the
    root of the subtree changes.
    if nil(T) then
        Create a new leaf node with X.
        return node(X, 1, Nil, Nil)
    else if X < value(T) then
        left(T) := insert(X, left(T))
    else if X > value(T) then
        right(T) := insert(X, right(T))
    end if
    Note that the case of X == value(T) is unspecified. As given, an insert
    will have no effect. The implementor may desire different behavior.

    Perform skew and then split. The conditionals that determine whether or
    not a rotation will occur or not are inside of the procedures, as given
    above.
    T := skew(T)
    T := split(T)

    return T
end function
```

刪除

在大部分的二元搜尋樹，刪除一個內部節點可以轉換成交換內部節點及其最接近的前驅或後繼節點，這取決於使用者。 為了平衡這顆樹，有幾中方法，Andersson教授描述的original paper (http://user.it.uu.se/~arnea/abs/simp.html)（页面存档备份 (https://web.archive.org/web/20141221213549/http://user.it.uu.se/~arnea/abs/simp.html)，存于互联网档案馆）是最基本的，儘管它還能再被優化。刪除後第一件事是降低其level(如果可以)，於是，整個level必須skew和split，這個方法最受到歡迎的，因為它的概念易懂，可以列舉成下列三個簡單步驟:

1. 如果可以的話，減少其level
2. Skew其level.
3. Split其level.

```
function delete is
    input: X, the value to delete, and T, the root of the tree from which it should be delete
d.
    output: T, balanced, without the value X.

    if nil(T) then
        return T
    else if X > value(T) then
        right(T) := delete(X, right(T))
    else if X < value(T) then
        left(T) := delete(X, left(T))
    else
        If we're a leaf, easy, otherwise reduce to leaf case.
        if leaf(T) then
            return Nil
        else if nil(left(T)) then
            L := successor(T)
            right(T) := delete(value(L), right(T))
            value(T) := value(L)
        else
            L := predecessor(T)
            left(T) := delete(value(L), left(T))
            value(T) := value(L)
        end if
    end if

    Rebalance the tree. Decrease the level of all nodes in this level if
    necessary, and then skew and split all nodes in the new level.
    T := decrease_level(T)
    T := skew(T)
    right(T) := skew(right(T))
    if not nil(right(T))
        right(right(T)) := skew(right(right(T)))
    end if
    T := split(T)
    right(T) := split(right(T))
    return T
end function
```

```
function decrease_level is
    input: T, a tree for which we want to remove links that skip levels.
    output: T with its level decreased.

    should_be = min(level(left(T)), level(right(T))) + 1
    if should_be < level(T) then
        level(T) := should_be
        if should_be < level(right(T)) then
            level(right(T)) := should_be
        end if
    end if
    return T
end function
```

這個網站展示了良好的刪除示範Andersson paper (http://user.it.uu.se/~arnea/abs/simp.html)（页面存档备份 (https://web.archive.org/web/20141221213549/http://user.it.uu.se/~arnea/abs/simp.html)，存于互联网档案馆）.

效能

AA樹的性能和紅黑樹是很類似的。儘管AA樹比紅黑樹做較多次旋轉，卻較容易實做，故二者效能相似。但是AA樹高度較淺，故查找時間較快[1]

參見

- 紅黑樹
- B樹
- AVL樹

- 左傾紅黑樹

引用

1. A Disquisition on The Performance Behavior of Binary Search Tree Data Structures (pages 67-75) (PDF). [2015-03-17]. （原始内容 (PDF)存档于2014-03-27）.

外部連結

- A. Andersson. Balanced search trees made simple (http://user.it.uu.se/~arnea/abs/simp.html)（页面存档备份 (https://web.archive.org/web/20141221213549/http://user.it.uu.se/~arnea/abs/simp.html)，存于互联网档案馆）
- A. Andersson. A note on searching in a binary search tree (http://user.it.uu.se/~arnea/abs/searchproc.html)（页面存档备份 (https://web.archive.org/web/20151219234930/http://user.it.uu.se/~arnea/abs/searchproc.html)，存于互联网档案馆）
- AA-Tree Applet (http://people.ksp.sk/~kuko/bak/index.html)（页面存档备份 (https://web.archive.org/web/20150304195520/http://people.ksp.sk/~kuko/bak/index.html)，存于互联网档案馆） by Kubo Kovac
- BSTlib (https://web.archive.org/web/20110807202006/https://bitbucket.org/trijezdci/bstlib/src/) - Open source AA tree library for C by trijezdci
- AA Visual 2007 1.5 - OpenSource Delphi program for educating AA tree structures (https://web.archive.org/web/20071022052324/http://www.softpedia.com/get/Others/Home-Education/AA-Visual-2007.shtml)
- Thorough tutorial (https://web.archive.org/web/20160303175254/http://www.eternallyconfuzzled.com/tuts/datastructures/jsw_tut_andersson.aspx) Julienne Walker with lots of code, including a practical implementation
- Object Oriented implementation with tests (http://www.cs.fiu.edu/~weiss/dsaa_c++3/code/)（页面存档备份 (https://web.archive.org/web/20080516092914/http://www.cs.fiu.edu/~weiss/dsaa_c++3/code/)，存于互联网档案馆）
- A Disquisition on The Performance Behavior of Binary Search Tree Data Structures (pages 67-75) (https://web.archive.org/web/20140327140251/http://www.cepis.org/upgrade/files/full-2004-V.pdf) - Comparison of AA trees, red-black trees, treaps, skip lists, and radix trees
- An example C implementation (https://web.archive.org/web/20110716085721/http://www.rational.co.za/aatree.c)
- An Objective-C implementation (https://code.google.com/p/objc-aatree)（页面存档备份 (https://web.archive.org/web/20111222220652/http://code.google.com/p/objc-aatree)，存于互联网档案馆）

-