AVL 木

AVL 木は通常の２分探索木を拡張して、最悪時の計算量を O(log n) に抑えたものである。基本的なデータ構造と操作は２分探索木と同じものを利用する。変更点として、挿入と削除の後で、各ノードでのバランス（左右の木の高さ（末端までのノード数）の差）をチェックして、バランスが崩れていたらバランスを回復する操作を行う。ここでいうバランスとは、左の木の高さから右の木の高さを引いたものであり、これが常に-1 から１に収まるようにすることで、全体の木の高さを log n の定数倍以内に抑えることができる。図に AVL 木の例を示す。

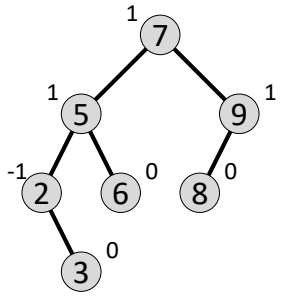

図　AVL 木の例。添え字はバランスを示す。

バランスをチェックして必要に応じて回復する操作は、末端で挿入や削除を行った後、根に向かってさかのぼりながら適用していく。たとえば、挿入においては、途中ノードで右の木や左の木の高さが増えた場合には、まずそのノードでバランスをチェック・回復操作を行う。その結果、そのノード自身の高さが増えかどうかをチェックし、その内容を親ノードに伝える。ノードの高さが変化している間は根に向かって処理が伝搬し、変化しなかったところで終了となる。

各ノードでのバランスのチェック・回復の詳細は以下のようになる。簡単のため、ここでは左の木に対して挿入が起きて左の木の高さが１増えた場合を考える（右の木への挿入は、左右を入れ替えただけで同じ操作。削除の場合はほぼ同じだが、場合分けの詳細が若干異なる。）。元のバランスが-１だった（右の木の方が高かった）場合(図１左)には、左の木の高さが増えてバランスが取れたことになるので、バランスを回復する必要はなく、また自身の高さも変わらないので親に対して false を返し処理が終了する。元のバランスが０だった場合(図１中)には、バランスが１になるが、これは許容範囲内なので、特にバランスを回復する操作はしない。しかし、自身の高さは１増えているので、親に対して true を返し、処理を根に向かって伝搬させる。元のバランスが１だった場合(図１右)には、そのままだとバランスが２になってしまうので、バランスを回復する操作が必要である。このバランスの回復操作は、左の子のバランスの状況に応じてさらに２つの場合に分けられる(図２)。

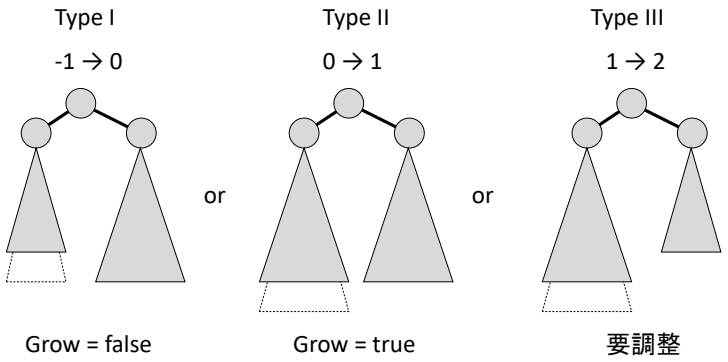

図１

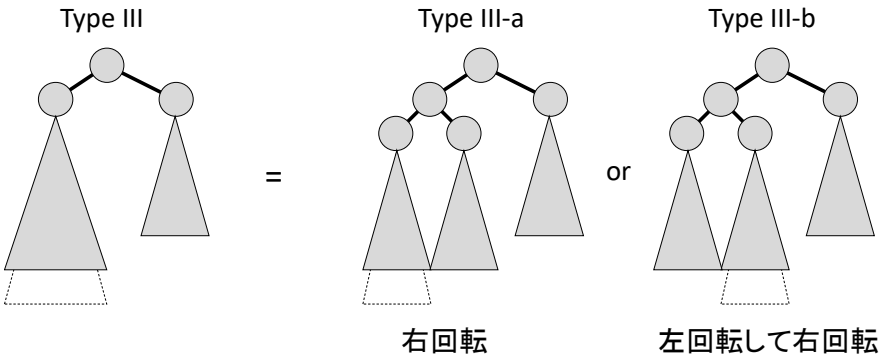

図２

まず、左の子の左の子の高さが１増えた場合（図２中）には、図３に示したような右回転と呼ばれる操作を行う。この操作を行うことで、左の木の高さが１減り、右の木の高さが１増えるので、バランスが回復される。しかし、左の子の右の子の高さが１増えた場合（図２右）には、そのまま右回転を行ってしまうとバランスが-2 になってしまうので、図４に示すように、まず左の子に対して左回転を行った後で自身に対して右回転を行うことでバランスを回復する。いずれの場合にも、現在ノードの高さは挿入前と変化しないので、親に対しては、false を返す。

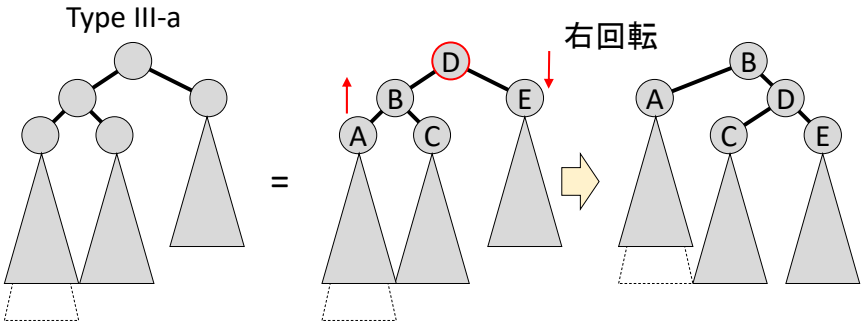

図３

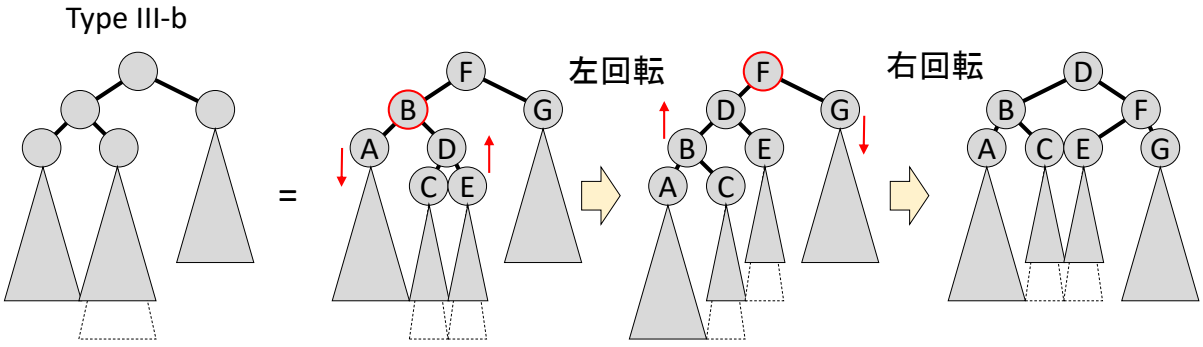

図４

疑似コード

```
insert(node, object)
    if (node == null)
        return {new Node(object), true};
    if (object < node.object)
        {node.left, grow} = insert(node.left, object);
        if (grow) {node, grow} = balance_left(node);
    else
        {node.right, grow} = insert(node.right, object);
        If (grow) {node, grow} = balance_right(node);
    return {node, grow};

balance_left(node)
    if (node.balance = -1)              // もともと左＜右
        node.balance = 0;
        return {node, false};
    else if (node.balance = 0)          // もともと左＝右
        node.balance = 1;
        return {node, true};
     else // node.balance = 1           もともと左＞右
     if (node.left.balance >=0)         // 左左＞＝左右
        node = rotateR(node);
        node.balance = 0;
        return {node, false};
     else // node.left.balance = -1   左左<左右
        node.left = rotateL(node.left);
        node = rotateR(node);
        node.balance = 0;
        return {node, false};
rotateR(node)
      left = node.left
      node.left = left.right;
      left.right = node;
      [ここでバランスの値の更新を行う]
      return left;
rotateL(node)
      right = node.right;
      node.right = right.left;
      right.left= node;
      [ここでバランスの値の更新を行う]
      return right;
```