```
// 左部分木の平衡化
// 入力引数: now 非平衡になったノード
//           mode 'a'(add) / 'd'(delete)
// 戻り値: now
NODE *balance_left(NODE *now, char mode) {
  NODE *p1,
       *p2;

  if (now->bal == 1) { // 元は右が高い場合
    now->bal = 0;      // 平衡になった
    (mode == 'a') ? (Ht = 0) : (Ht = 1);
  }
  else if (now->bal == 0) { // 元は平衡な場合
    now->bal = -1;          // 左が高くなった
    (mode == 'a') ? (Ht = 1) : (Ht = 0);
  }
  else if (now->bal == -1) { // 元は左が高い場合(再平衡化)
    p1 = now->left;
    if (p1->bal <= 0) { // 左の子の左部分木が高い; LL 一重回転
      cntLL++; // LL 回転数を加算
      now->left = p1->right;
      p1->right = now;
      if (mode == 'a')      // 追加時
        now->bal = 0;       // 平衡になった
      else {                // 削除時
        if (p1->bal == 0) {
          now->bal = -1;
          p1->bal = 1;
          Ht = 0;           // 低くなった
        }
        else {
          now->bal = 0; // 平衡になった
          p1->bal = 0;  // 平衡になった
          Ht = 1;
        }
      }
      now = p1; // 現在節が代わった
    }
    else { // 左の子の右部分木が高い; LR 二重回転
      cntLR++; // LR 回転数を加算
      p2 = p1->right;
      p1->right = p2->left;
      p2->left = p1;
      now->left = p2->right;
      p2->right = now;
      (p2->bal == -1) ? (now->bal = 1) : (now->bal = 0);
      (p2->bal == 1)  ? (p1->bal = -1) : (p1->bal = 0);
      p2->bal = 0; // 平衡になった
      now = p2;    // 根が代わった
    }
    height--; // 木の高さを更新
    if (mode == 'a') {
      now->bal = 0; // 平衡になった
      Ht = 0;       // 変更(0)
```

```
    }
  }

  return now;
}


// 右部分木の平衡化
// 入力引数: now 非平衡になったノード
//           mode 'a'(add) / 'd'(delete)
// 戻り値: now
NODE *balance_right(NODE *now, char mode) {
  NODE *p1, // now の子
       *p2;

  if (now->bal == -1) { // 元は左が高い場合
    now->bal = 0;       // 平衡になった
    (mode == 'a') ? (Ht = 0) : (Ht = 1);
  }
  else if (now->bal == 0) { // 元は平衡の場合
    now->bal = 1;           // 右が高くなった
    (mode == 'a') ? (Ht = 1) : (Ht = 0);
  }
  else if (now->bal == 1) {  // 元は右が高い場合; 再平衡化
    p1 = now->right;         // 非平衡ノードの右の子
    if (p1->bal >= 0) {      // 右の子の右部分木が高い; RR 一重回転
      cntRR++; // RR 回転数を加算
      now->right = p1->left; // 右の子を変更
      p1->left = now;        // 左の子を変更
      if (mode == 'a')       // 追加時
        now->bal = 0;        // 平衡になった
      else {                 // 削除時
        if (p1->bal == 0) {
          now->bal = 1;
          p1->bal = -1;
          Ht = 0;
        }
        else {
          now->bal = 0; // 平衡になった
          p1->bal = 0;
          Ht = 1;
        }
      }
      now = p1;
    }
    else { // 右の子の左部分木が高い; RL 二重回転
      cntRL++; // RL 回転数を加算
      p2 = p1->left;
      p1->left = p2->right;
      p2->right = p1;
      now->right = p2->left;
      p2->left = now;
      (p2->bal == 1) ? (now->bal = -1) : (now->bal = 0);
      (p2->bal == -1) ? (p1->bal = 1) : (p1->bal = 0);
```

```
      if (mode == 'd')
        p2->bal = 0; // 親が代わったため
      now = p2;
    }
    height--; // 木の高さを更新
    if (mode == 'a') {
      now->bal = 0; // 平衡になった
      Ht = 0;       // 全体の平衡度は 0
    }
  }
  return now;
}


// キーの追加
// 入力引数: now 非平衡になったノード
//           inv 入力値
// 戻り値: now
NODE *add(NODE *now, char *inv) {
  NODE *p1, // 非平衡ノードの子
       *p2, // 非平衡ノードの子の子
       *new;

  if (now == NULL) {
    new = set_key(inv); // ノードを生成し入力値を格納
    Ht = 1;             // 高くなった
    now = new;
  }
  else if (strcmp(inv, now->key) < 0) {         // 入力値 < 現在節のキー
    now->left = add(now->left, inv); // 左へ
    height++; // 木の高さを更新
    if (Ht != 0)
      now = balance_left(now, 'a'); // 再平衡化 @2009.02 追加
  }
  else if (strcmp(now->key, inv) < 0) {           // 入力値 > 現在節のキー
    now->right = add(now->right, inv); // 右へ
    height++; // 木の高さを更新
    if (Ht != 0)
      now = balance_right(now, 'a');   // 再平衡化 @2009.02 追加
  }
  else // 入力値 = 現在節のキー
    Ht = 0;
  return now;
}
```