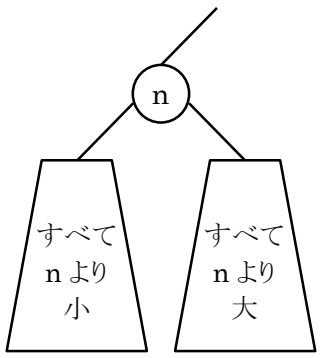
1. **2 分探索木**は，右図に示すように，その中のどのノード n をとっても，その左の部分木には n より小さい要素，右の部分木には n より大きい要素しかないような 2 分木である。このデータ構造は文字通り探索に適する。以下のプログラムは 2 分探索木を構築して，そこから指定のキーを持つノードを探索するものである。空欄を適切に埋めて実行し，探索の平均計算量を考察せよ。

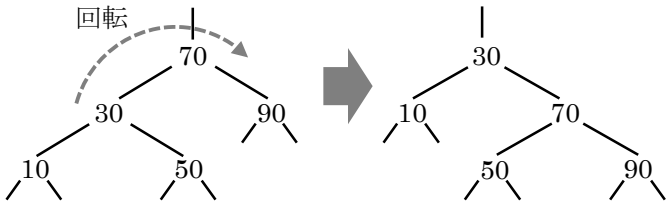



```
/* BSTree.java */
public class BSTree {

  private Node root = null;

  // クラス内での内部クラスの定義
  public class Node {
    private final String key;
    public int data;
    private Node left, right;

    private Node(String key) {
      this.key = key;
      this.data = 0;
      left = right = null;
    }
  }

  // キーを指定してノードを探索する
  // add が真なら，ない場合に新たに作って挿入
  public Node find(String key, boolean add) {

    // 木が空の場合は特別
    if (root == null) {
      if (add) root = new Node(key);
      return root;
    }

    // ループによるノードの探索
    Node n = root;
    while (true) {
      int cmp = key.compareTo(n.key);
      if (cmp < 0) {

        if (n.left ==         ) {

          if (add) n.left =
          return n.left;
        }
        // 木を降りていく
        n =

      } else if (cmp > 0) {

        if (n.right ==         ) {

          if (add) n.right =
          return n.right;
        }
        n =

      } else {  // cmp == 0 （発見）
        return n;
      }
    }
  }

  // 木のすべての要素を表示する
  public void traverse() {
    traverse(root, 0);
  }

  // 再帰で部分木の要素を昇順に表示する
  private void traverse(Node n, int level) {

    if (n == null) return;

    traverse(              , level + 1);

    for (int i = 0; i < level; i++)
      System.out.print("   ");
    System.out.printf("+ %s(%d)%n",
                n.key, n.data);

    traverse(              , level + 1);
  }
}
```

```
/* Program.java */
import java.util.Scanner;

public class Program {

  public static void main(String[] args) {

    BSTree tree =

    Scanner sc = new Scanner(System.in);

    // 木の構築（単語のカウント）
    while (true) {
      System.out.print("文字列？ ");
      String key = sc.next();

      // quit と入力したらプログラム終了
      if (key.equals("quit")) break;

      // 内部クラスの利用方法
      BSTree.Node n = tree.find(key, true);
      n.data = n.data + 1;

      tree.traverse();
      System.out.println();
    }
  }
}
```

2. 1.のクラスを，文字列の代わりに任意のクラス E を要素とするジェネリッククラス BSTree<E> となるように書き換え，適当なメインプログラムを添えてその動作を確認せよ。

3. 単純な 2 分探索木は, データの登録順によっては木の形が左右対称から大きく崩れ, 探索速度が低下する。**平衡木（AVL 木）**では，どのノードから見た左右の部分木の高さの差も 1 以内に収まるように，下図に示すような回転操作を利用してなるべく平衡を保つ。なお，矢印の方向を「右回転」と呼ぶことにする。



下記は，ノード n の左右の枝の先の長さ（n.left および n.right の部分木の高さ）が 2 以上離れたときに，それを 1 以内に収めるために，どのように回転操作を組み合わせればバランスを維持できるかを説明した疑似コードである。教科書および授業の説明を理解して適切に選択肢を選び，回転操作を図示せよ。

1: b = n.left の下の部分木の高さ（深さ）- n.right の下の部分木の高さ（深さ）

2: もし，b【 ≦　≧ 】2 ならば，n.left の下が深くなりすぎたので以下を実行

2.1: さらにもし，n.left.left の下の高さ【 ＜　＞ 】n.left.right の下の高さならば

2.1.1: n.left.right を持ち上げるために，n.left を軸に【 左　右 】回転する（図示せよ）

2.2: n.left を持ち上げるために，n を軸に【 左　右 】回転する（図示せよ）

3: もし，b【 ≦　≧ 】-2 ならば，n.right の下が深くなりすぎたので以下を実行

3.1: さらにもし，n.right.left の下の高さ【 ＜　＞ 】n.right.right の下の高さならば

3.1.1: n.right.left を持ち上げるために, n.right を軸に【 左　右 】回転する（図示せよ）

3.2: n.right を持ち上げるために，n を軸に【 左　右 】回転する（図示せよ）