```
#! /n/gnu/bin/gawk -f
# Last edited on 1999-09-21 16:47:52 by stolfi

BEGIN {
  NIL = -1;
  livre = 0;
  split("", esq);
  split("", dir);
  split("", ch);
  split("", alt);
  p = NIL;
  p = testa_insere(p);
  testa_procura(p);
  p = testa_remove(p);
  testa_procura(p);
}

# Aloca um novo nó com chave dada:

function novo(chave,  p)
{
  p = livre; livre++;
  esq[p] = NIL;
  dir[p] = NIL;
  alt[p] = 1;
  ch[p] = chave;
  return (p);
}

# A busca numa árvore balanceada não tem nada de mais:

function procura(p, chave)
{
  if (p == NIL)
    { return(NIL); }
  else if (chave < ch[p])
    { return(procura(esq[p], chave)); }
  else if (chave > ch[p])
    { return(procura(dir[p], chave)); }
  else
    { return(p); }
}

# Enumeração também é bico:

function lista(p)
{
  if (p == NIL)
    { return; }
  else
    { lista(esq[p]);
      printf "%2d = \"%s\"  (%d)\n", p, ch[p], alt[p] > "/dev/stderr";
      lista(dir[p]);
    }
}

# Vamos armazenar em cada nó p um campo alt[p] que dá a altura
# da sub-arvore com raiz p. (definida como o número de nós
# no caminho mais longo da raiz até alguma folha.

function altura(p)
```

```
{
  if (p == NIL) { return 0; } else { return alt[p]; }
}

function acerta_altura(p)
{
  alt[p] = max(altura(esq[p]), altura(dir[p])) + 1;
}

function max(x, y)
{
  if (x > y) { return(x); } else { return(y); }
}

# A inserção é quase igual à inserção numa árvore
# binária simples, exceto que precisamos manter os
# campos alt[p] atualizados, e rebalancear ocasionalmente
# algumas sub-árvores:

function insere(p, chave)
{
  # Insere "chave" na sub-arvore "p", devolve nova raiz.

  # printf "enter insert(%d, %s)\n", p, chave > "/dev/stderr";
  if (p == NIL)
    { return(novo(chave)); }
  else if (chave < ch[p])
    { esq[p] = insere(esq[p], chave);
      acerta_altura(p);
      if (altura(esq[p]) > altura(dir[p]) + 1) { p = bal_esq_maior(p); }
      return(p);
    }
  else if (chave > ch[p])
    { dir[p] = insere(dir[p], chave);
      acerta_altura(p);
      if (altura(dir[p]) > altura(esq[p]) + 1) { p = bal_dir_maior(p); }
      return(p);
    }
  else
    { return(p); }
}

# Remoção é mais complicada.
# O truque é dar um jeito de remover sempre um nó
# que tem esq[p] = NIL.

function remove(p, chave,  q,X,r)
{
  if (p == NIL)
    { return(p); }
  else if(chave < ch[p])
    { esq[p] = remove(esq[p], chave);
      acerta_altura(p);
      if (altura(esq[p]) + 1 < altura(dir[p])) { p = bal_dir_maior(p); }
      return(p);
    }
  else if (chave > ch[p])
    { dir[p] = remove(dir[p], chave);
      acerta_altura(p);
      if (altura(dir[p]) + 1 < altura(esq[p])) { p = bal_esq_maior(p); }
      return(p);
```

```
    }
  else
    { q = dir[p];
      X = esq[p];
      if (q == NIL)
        { return(X); }
      else
        { r = levanta(q);
          openbug();
          valida(dir[r], ch[p], "zzzzzzzzzzzzzzzzzzzzzzzz");
          closebug();
          esq[r] = X;
          acerta_altura(r);
          if (altura(dir[r]) + 1 < altura(esq[r])) { r = bal_esq_maior(r); }
          return(r);
        }
    }
}

function levanta(p,   r)
{
  # Encontra o primeiro nó da árvore "p" em pre-ordem
  # e translada-o para a raiz, balanceando o restante da árvore
  # e preservando a ordem dos nós.

  if (esq[p] == NIL)
    { return(p); }
  else
    { r = levanta(esq[p]);
      esq[p] = dir[r];
      acerta_altura(p);
      if (altura(esq[p]) + 1 < altura(dir[p])) { p = bal_dir_maior(p); }
      dir[r] = p;
      return(r);
    }
}


function bal_esq_maior(p,   q,r,X,Y,Y1,Y2)
{
  # Chamada quando a inserção de um nó na sub-arvore esq[p] ou deleção
  # de um nó na sub-árvore dir[p] fez com que altura(esq[p]) =
  # altura(dir[p]) + 2. Rearranja os nós para corrigir esse problema,
  # recalcula as alturas das sub-árvores, e devolve a nova raiz.

  # Suposição: a inserção/deleção numa árvore modifica a altura da mesma
  # em no máximo 1.

  q = esq[p];
  X = esq[esq[p]];
  Y = dir[esq[p]];
  Z = dir[p];

  if ((altura(Y) == altura(Z) + 1) && (altura(Y) == altura(X) + 1))
    { # "Dupla rotação" é necessária:
      r = Y; Y1 = esq[r]; Y2 = dir[r];
      esq[q] = X;
      dir[q] = Y1;
      acerta_altura(q);
      esq[p] = Y2;
      dir[p] = Z;
```

```
      acerta_altura(p);
      esq[r] = q;
      dir[r] = p;
      acerta_altura(r);
      openbug("esq_maior, dupla");
      valida(r, "", "zzzzzzzzzzzzzzzzzzzzzzzz");
      closebug();
      return(r);
    }
  else
    { # "Rotação simples" resolve:
      esq[p] = Y;
      dir[p] = Z;
      acerta_altura(p);
      esq[q] = X;
      dir[q] = p;
      acerta_altura(q);
      openbug("esq_maior, simples");
      valida(q, "", "zzzzzzzzzzzzzzzzzzzzzzzz");
      closebug();
      return(q);
    }
}

function bal_dir_maior(p,   q,r,X,Y,Y1,Y2)
{
  # Chamada quando a inserção de um nó na sub-árvore dir[p]
  # ou remoção de um nó na sub-árvore esq[p]
  # fez com que altura(dir[p]) = altura(esq[p]) + 2.
  # Rearranja os nós para corrigir esse problema,
  # recalcula as alturas das sub-árvores, e
  # devolve a nova raiz.

  # Suposição: a inserção ou deleção numa árvore modifica a
  # altura da mesma em no máximo 1.

  q = dir[p];
  X = dir[dir[p]];
  Y = esq[dir[p]];
  Z = esq[p];

  if ((altura(Y) == altura(Z) + 1) && (altura(Y) == altura(X) + 1))
    { # "Dupla rotação" é necessária:
      r = Y; Y1 = dir[r]; Y2 = esq[r];
      dir[q] = X;
      esq[q] = Y1;
      acerta_altura(q);
      dir[p] = Y2;
      esq[p] = Z;
      acerta_altura(p);
      dir[r] = q;
      esq[r] = p;
      acerta_altura(r);
      openbug("dir_maior, dupla");
      valida(r, "", "zzzzzzzzzzzzzzzzzzzzzzzz");
      closebug();
      return(r);
    }
  else
    { # "Rotação simples" resolve:
      dir[p] = Y;
```

```
      esq[p] = Z;
      acerta_altura(p);
      dir[q] = X;
      esq[q] = p;
      acerta_altura(q);
      openbug("dir_maior, simples");
      valida(q, "", "zzzzzzzzzzzzzzzzzzzzzzzz");
      closebug();
      return(q);
    }
}

# Validação da árvore:

function valida(p,chave_min,chave_max,   nbug,ae,ad)
{
  # Imprime bugs e retorna número de bugs
  nbug = 0;
  if (p == NIL)
    { return(nbug); }
  else
    { if ((ch[p] <= chave_min) || (ch[p] > chave_max))
        { printf "order bug: %d = %s : %s : %s\n", p, chave_min, ch[p], chave_max \
             > "/dev/stderr";
          nbug++;
        }

      ae = altura(esq[p]);
      ad = altura(dir[p]);
      if (alt[p] != max(ae,ad)+1)
        { printf "alt bug: %d = (%d, %d) -> %d\n", p, ae, ad, alt[p]  \
            > "/dev/stderr";
          nbug++;
        }
      if ((ae > ad + 1) || (ad > ae + 1))
        { printf "bal bug: %d = (%d, %d) -> %d\n", p, ae, ad, alt[p]  \
            > "/dev/stderr";
          nbug++;
        }
      return(nbug + valida(esq[p], chave_min, ch[p]) + valida(dir[p], ch[p], chave_max));
    }
}

# Testes:

function testa_insere(p)
{
  p = insere(p, "janeiro");
  p = insere(p, "fevereiro");
  p = insere(p, "marco");
  p = insere(p, "abril");
  p = insere(p, "maio");
  p = insere(p, "junho");
  p = insere(p, "julho");
  p = insere(p, "agosto");
  p = insere(p, "setembro");
  p = insere(p, "outubro");
  p = insere(p, "novembro");
  p = insere(p, "dezembro");
  p = insere(p, "thermidor");
  p = insere(p, "brumidor");
```

```
  p = insere(p, "messidor");
  p = insere(p, "pop");
  p = insere(p, "uo");
  p = insere(p, "zip");
  p = insere(p, "zotz");
  p = insere(p, "tzec");
  p = insere(p, "xul");
  p = insere(p, "yaxkin");
  p = insere(p, "mol");
  p = insere(p, "chen");
  p = insere(p, "yax");
  p = insere(p, "zac");
  p = insere(p, "ceh");
  p = insere(p, "mac");
  p = insere(p, "kankin");
  p = insere(p, "muan");
  p = insere(p, "pax");
  p = insere(p, "kayab");
  p = insere(p, "cumku");
  lista(p);
  printf "%d bugs\n", valida(p, "", "zzzzzzzzzzzzzzzzzzzzzzzzzzzz") > "/dev/stderr";
  return(p);
}

function testa_remove(p)
{
  p = remove(p, "janeiro");
  p = remove(p, "fevereiro");
  p = remove(p, "marco");
  p = remove(p, "abril");
  p = remove(p, "maio");
  p = remove(p, "junho");
  p = remove(p, "julho");
  p = remove(p, "agosto");
  p = remove(p, "setembro");
  p = remove(p, "outubro");
  p = remove(p, "novembro");
  p = remove(p, "dezembro");
  lista(p);
  printf "%d bugs\n", valida(p, "", "zzzzzzzzzzzzzzzzzzzzzzzzzzzz") > "/dev/stderr";
  return(p);
}

function testa_procura(p)
{
  testa_procura_um(p, "fevereiro");
  testa_procura_um(p, "dezembro");
  testa_procura_um(p, "brumidor");
}

function testa_procura_um(p, chave,  q)
{
  q = procura(p, chave);
  printf "procura(%d, \"%s\") = %d", \
    p, chave, q > "/dev/stderr";
  if (q != NIL)
    { printf "  (%s, %d)", ch[q], alt[q] > "/dev/stderr"; }
  printf "\n" > "/dev/stderr";
}
```

```
function openbug(title)
{
  printf "--- begin %s ---\n", title > "/dev/stderr";
}

function closebug()
{
  printf "--- end ---\n" > "/dev/stderr";
}
```