```c
#include <stdlib.h>
#include <stdio.h>

#include "mapa.h"
#include "arvore.h"

/* IMPLEMENTAÃ‡ÃƒO DE MAPA */

struct smapa {
  int chave;
  int dados;
  Mapa* esq;
  Mapa* dir;
};

/* FunÃ§Ãµes auxiliares */

static Mapa *cria_no (int c, int d) {
  Mapa *nn = (Mapa *)malloc(sizeof(Mapa));
  if (nn!=NULL) {
    nn->esq = nn->dir = NULL;
    nn->chave =c;
    nn->dados = d;
  }
  return nn;
}

/* FunÃ§Ãµes exportadas */

Mapa* cria (void) {
  return NULL;
}
int busca (Mapa *m, int chave) {
  while (m!=NULL) {
    if (chave < m->chave)
      m = m->esq;
    else if (chave > m->chave)
      m = m->dir;
    else
      return m->dados; /* achou */
  }
  return INT_MIN;
}

void destroi (Mapa *m) {
  if (m==NULL) return;
  destroi (m->esq);
  destroi (m->dir);
  free(m);
}

Mapa *insere (Mapa *m, int chave, int d) {
  if (m==NULL)
    return cria_no(chave, d);
  if (chave < m->chave)
    m->esq = insere(m->esq, chave, d);
  else
    m->dir = insere(m->dir, chave, d);
  return m;
}

Mapa *retira (Mapa *m, int chave) {
  Mapa *sucessor, *t;
  if (m==NULL) return NULL;
  if (chave < m->chave) m->esq = retira(m->esq, chave);
  else if (chave > m->chave) m->dir = retira(m->dir, chave);
  else {
            if ((m->esq == NULL) && (m->dir == NULL)) {
      free(m); m = NULL;
    }
    else if (m->esq == NULL)  {
      t = m; m = m->dir; free(t);
    }
    else if (m->dir == NULL)  {
      t = m; m = m->esq; free(t);
    }
    else {
      sucessor = m->dir;
      while (sucessor->esq) sucessor = sucessor->esq;
```

```
      m->chave = sucessor->chave;
      m->dados = sucessor->dados;
      sucessor->chave = chave;
      m->dir = retira(m->dir, chave);
    }
  }
  return m;
}

/* IMPLEMENTAÃ‡ÃƒO DE ÃRVORE */

void mostra(Mapa* m) {
  printf("[");
  if (m != NULL) {
    printf("<%d %d> ", m->chave, m->dados);
    mostra(m->esq);
    mostra(m->dir);
  }
  printf("] ");
}

int num_nos (Mapa *m) {
  if (m == NULL) return 0;
  return num_nos(m->esq) + num_nos(m->dir) + 1;
}

int maior_chave (Mapa *m) {
  if (m == NULL) return INT_MIN;
  if (m->dir) return maior_chave(m->dir);
  return m->chave;
}
Mapa * menor_no (Mapa *m) {
  if (m == NULL) return m;
  if (m->esq) return menor_no(m->dir);
  return m;
}

int num_maiores_que (Mapa *m, int n) {
  int nos = 0;
  if (m == NULL) return 0;
  if (m->chave > n)
    nos += num_maiores_que(m->esq, n);
  nos += num_maiores_que(m->dir, n);
  if (m->chave > n)
    nos += 1;
  return nos;
}




/*int altura(Mapa * m)
{
  if(m==NULL)
    return -1;
  else if(altura(m->dir) > altura(m->esq))
    return altura(m->dir) +1;
  else
    return altura(m->esq) +1;


}*/


int altura(Mapa * m)
{

  if(m==NULL)
    return -1;
  else
    {
      int lAltura = altura(m->esq);
      int rAltura = altura(m->dir);
      if(lAltura > rAltura)
        return lAltura +1;
      else
        return rAltura +1;
    }

}
```

```
160
161
162
163
164 int e_balanceada (Mapa *m) {
165   if(m==NULL)
166     return 0;
167   else if( abs(altura( m->esq) -altura( m-> dir) ) > 1)
168     return 1;
169   else
170     return e_balanceada(m->esq) + e_balanceada(m->dir);
171 }
172
173
174
175
176 Mapa *balancear(Mapa *m) {
177   int temp;
178   if(m == NULL)
179     return NULL;
180   else if( abs(num_nos(m->dir) - num_nos(m->esq)) <= 1)
181   {
182     m->dir = balancear(m->dir);
183     m->esq = balancear(m->esq);
184   }
185   while(num_nos(m->dir) > (num_nos(m->esq) +1))
186     {
187       temp = m->chave;
188       m = retira(m,temp);
189       m = insere(m,temp,temp*2);
190       m->dir = balancear(m->dir);
191       m->esq = balancear(m->esq);
192     }
193   while(num_nos(m->esq) > (num_nos(m->dir) +1))
194     {
195       temp = m->chave;
196       m = retira(m,temp);
197       m = insere(m,temp,temp*2);
198       m->dir = balancear(m->dir);
199       m->esq = balancear(m->esq);
200     }
201   return m;
202 }
203
204 Mapa *balancear2(Mapa *m) {
205   int temp, val, nd, ne;
206   if(m == NULL)
207     return NULL;
208   nd = num_nos(m->dir);
209   ne = num_nos(m->esq);
210   if( abs(nd - ne) > 1)
211   {
212   while(nd > (ne+1))
213     {
214       temp = m->chave;
215       val = m->dados;
216       m = retira(m,temp);
217       m->esq = insere(m->esq,temp,val);
218       ne++; nd--;
219     }
220   while(ne > (nd +1))
221     {
222       temp = m->chave;
223       val = m->dados;
224       m = retira(m,temp);
225       m->dir = insere(m->dir,temp,val);
226
227       nd++; ne--;
228
229     }
230   }
231   m->dir = balancear(m->dir);
232   m->esq = balancear(m->esq);
233   return m;
234 }
235
236
237
238
```