```
void countR(link h, int *cnt){
   if(h==NULL)
      return;

   if(h->l==NULL && h->r!=NULL ||
h->l!=NULL && h->r==NULL)
      (*cnt)++;
   countR(h->l,cnt);
   countR(h->r,cnt);
}
int oneChildCount(BST bst){
   int cnt=0;
   countR(bst->root,&cnt);
   return cnt;
}
```

```
void treeFree(link h){
   if(h==NULL)
      return;
   treeFree(h->l);
   treeFree(h->r);
   free(h);
}
void freeBST(BST bst){
   treeFree(bst->root);
   free(bst);
}
```

```
void findLeafR(link h, int *leaf, int
*min, int k){
   if(h==NULL)
      return;

   if(k<(*min)){
      if(h->l == NULL && h->r ==
NULL){
         *leaf=h->key;
         *min=k;
      }
   }
   findLeafR(h->l, leaf,min,k+1);
   findLeafR(h->r,leaf,min,k+1);
}
void findLeafMinHeight(BST bst){
   int leaf, min=INT_MAX;
   findLeafR(bst->root, &leaf, &min, 0);
}
```

```
void deleteR(link h, int k, int lvl){
   if(h==NULL)
      return;

   if(lvl==k){
      treeFree(h->l);
      h->l=NULL;
      treeFree(h->r);
      h->r=NULL;
   }
   deleteR(h->l,k,lvl+1);
   deleteR(h->r, k, lvl+1);
}
void deleteFromLevel(BST bst, int k){
   deleteR(bst->root, k,0);
}
```

```
void countLR_R(link h, int *l, int *r){
   if(h==NULL)
      return;
   if(h->l!=NULL)
      (*l)++;
   if(h->r!=NULL)
      (*r)++;
   countLR_R(h->l,l,r);
   countLR_R(h->r,l,r);
}
int countLR(BST bst){
   int l=0,r=0,tot;
   countLR_R(bst->root,&l,&r);
   tot=l+r+1;
   return tot;
}
```

```
void findMax_R(link h, int *max_key){
   if(h==NULL)
      return;
   if(h->key>(*max_key))
      (*max_key)=h->key;
   findMax_R(h->l,max_key);
   findMax_R(h->r, max_key);
}
int findMax(BST bst){
   int key=INT_MIN;
   findMax_R(bst->root, &key);
   return key;
}
```

```
int areEqual(link h1, link h2){
   if(h1==NULL && h2==NULL)
      return 1;

   if(h1==NULL || h2==NULL)
      return 0;

   return(h1->key==h2->key &&
areEqual(h1->l, h2->l) && areEqual(h1-
>r,h2->r));
}
int isSubTreeR(link h1, link h2){
   if(h2==NULL)
      return 1;
   if(h1==NULL)
      return 1;

   if(areEqual(h1,h2))
      return 1;

   return isSubTreeR(h1->l,h2) ||
isSubTreeR(h1->r,h2);
}
int isSubTree(BST b1, BST b2){
   return isSubTreeR(b1->root, b2-
>root);
}
```

```
link rotR(link h){
   link x=h->l;
   h->l=x->r;
   x->r=h;
   return x;
}
link rotL(link h){
   link x=h->r;
   h->r=x->l;
   x->l=h;
   return x;
}
link insertT(link h, int key){
   if(h==NULL)
      return NEW(key,NULL,NULL);
   if(key<h->key){
      h->l= insertT(h->l,key);
      h=rotR(h);
   }else{
      h->r= insertT(h->r,key);
      h=rotL(h);
   }
   return h;
}
void BSTinsert_root(BST bst, int key){
   bst->root= insertT(bst->root,key);
}
```

```
int sum(link h){
   if(h==NULL)
      return 0;
   return sum(h->l) + h->key + sum(h-
>r);
}
int isSumTreeR(link h){
   int ls,rs;
   if(h==NULL || h->l==NULL && h->r
== NULL)
      return 1;

   ls=sum(h->l);
   rs=sum(h->r);

   if((h->key == ls+rs) &&
isSumTreeR(h->l) && isSumTreeR(h-
>r))
      return 1;
   return 0;
}
int isSumTree(BST bst){
   return isSumTreeR(bst->root);
}
```

```
void countNR(link h, int *cnt){
   if(h==NULL)
      return;
   if(h->l != NULL && h->r !=NULL)
      (*cnt)++;
   countNR(h->l,cnt);
   countNR(h->r,cnt);
}
int countCompleteNodes(BST bst){
   int cnt=0;
   countNR(bst->root,&cnt);
   return cnt;
}
```

```
void greather_R(link h, int l1, int l2, int
c, int k, int *cnt){
   if(h==NULL)
      return;

   if((k>=l1 && k<=l2) && h->key>c)
      (*cnt)++;

   greather_R(h->l,l1,l2,c,k+1,cnt);
   greather_R(h->r,l1,l2,c,k+1,cnt);
}
int greatherNodes(BST bst, int l1, int l2,
int c){
   int cnt=0;
   greather_R(bst->root, l1,l2,c,0,&cnt);
   return cnt;
}
```

```
void nLeafR(link h, int lvl, int k, int
*cnt){
   if(h==NULL)
      return;

   if(k==lvl && h->l==NULL && h-
>r==NULL){
      (*cnt)++;
      return;
   }

   nLeafR(h->l, lvl, k+1, cnt);
   nLeafR(h->r, lvl, k+1, cnt);
}
int nLeaf(BST bst, int lvl){
   int cnt=0;
   nLeafR(bst->root, lvl, 1,&cnt);
   return cnt;
}
```

```
int f(link h1, link h2){
   if(h1 == NULL && h2 == NULL)
      return 0;

   if(h1 == NULL && h2 == NULL)
      return 0;

   if(h1->key != h2->key)
      return 0;

   return (f(h1->l,h2->l) && f(h1->r,h2-
>r) ||
         f(h1->l,h2->r) && f(h1->r, h2-
>l));
}
int isIsomorphic(BST bst, BST bst2){
   return f(bst->root, bst2->root);
```

```
void mirrorR(link h){
   if(h==NULL)
      return;

   link t;
   mirrorR(h->l);
   mirrorR(h->r);

   t=h->l;
   h->l=h->r;
   h->r=t;
}
void mirror(BST bst){
   mirrorR(bst->root);
}
```