```c
/*
 * João Paulo Batista Ferreira
 * 2009113274
 * Algoritmos e Estruturas de Dados - TP3 exF
 */

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>

typedef struct item {
    char word[256];
    int counter;
}item;

typedef struct node {
    item info;
    int weight;
    struct node *left;
    struct node *right;
}node;

typedef struct node* nodePtr;

void visita (nodePtr leaf)
{
    printf("%s %d\n", leaf->info.word, leaf->info.counter);
}

void emOrdem (nodePtr leaf)
{
    if (leaf != NULL)
    {
        emOrdem(leaf->left);
        visita (leaf);
        emOrdem(leaf->right);
    }
}

void eraseTree(nodePtr leaf)
{
    if (leaf == NULL)
        return;
    eraseTree(leaf->left);
    eraseTree(leaf->right);
    free(leaf);
}

nodePtr fixN(nodePtr leaf)
{
    if(leaf->left == NULL && leaf->right == NULL )
        leaf->weight = 1;

    else if(leaf->left == NULL)
        leaf->weight = leaf->right->weight + 1;
    else if(leaf->right == NULL)
        leaf->weight = leaf->left->weight + 1;

    else
        leaf->weight = leaf->left->weight + leaf->right->weight + 1;

    return leaf;
}

nodePtr rightRotation(nodePtr t)
{
    nodePtr t2 = t->left;
    t->left = t2->right;
    t2->right = t;
    t2 = fixN(t2);
    return t2;
}

nodePtr leftRotation(nodePtr t)
{
    nodePtr t2 = t->right;
    t->right = t2->left;
    t2->left = t;
    t2 = fixN(t2);
    return t2;
}

nodePtr searchN(char* word, nodePtr leaf)
{
    if (leaf == NULL)
        return NULL;

    if (strcmp(word, leaf->info.word)==0)
        return leaf;

    else if (strcmp(word,leaf->info.word) > 0)
        return searchN(word, leaf->right);

    else
        return searchN(word, leaf->left);
}

nodePtr createNode(char* word)
{
    nodePtr aux;
    aux = (nodePtr) malloc(sizeof(node));
    strcpy(aux->info.word, word);
    aux->left = NULL;
    aux->right = NULL;
    aux->weight = 1;
    aux->info.counter = 1;
    return aux;
}

nodePtr rootInsertion(char* word, nodePtr leaf)
{
    if(leaf == NULL)
```

```
        return createNode(word);

    if(strcmp(word,leaf->info.word ) < 0)
    {
        leaf->left = rootInsertion(word, leaf->left);
        leaf = rightRotation(leaf);
    }

    else if((strcmp(word,leaf->info.word) > 0))
    {
        leaf->right = rootInsertion(word,leaf->right);
        leaf = leftRotation(leaf);
    }

    return leaf;
}

nodePtr addNode(char* word,nodePtr leaf)
{
    if(leaf == NULL)
        return createNode(word);

    if( rand() < RAND_MAX/(leaf->weight+1) )
        return rootInsertion(word, leaf);

    if(strcmp(word,leaf->info.word) < 0)
        leaf->left = addNode(word, leaf->left);

    else if(strcmp(word,leaf->info.word) > 0)
        leaf->right = addNode(word, leaf->right);

    leaf->weight++;

    return leaf;
}

nodePtr worker(char* word, nodePtr tree)
{
    nodePtr aux;
    int i, size = strlen(word);

    for (i = 0; i < size; i++)
        word[i] = tolower(word[i]);

    aux = searchN(word, tree);

    if (aux)
        aux->info.counter++;
    else
        tree = addNode(word, tree);

    return tree;
}

int main()
{
    nodePtr tree=NULL;
    char word[256];

    while( (scanf("%s", word)) != EOF )
        tree = worker(word, tree);

    emOrdem(tree);

    eraseTree(tree);

    return 0;
}
```