**Discrete Structures & Algorithms Practical Assignment 2**

Search Engine Optimisation Application

Text

Description automatically generated with medium confidence

**Student Name: Rachel Doogue**

**Student Number:C00237335**

**Lecturer: Áine Byrne**

**Due Date: 05/02/21**

Table of Contents

[Description of Problem 3](#_Toc64894992)

[Underlying Data Structures 4](#_Toc64894993)

[Pseudocode for Algorithms 5](#_Toc64894994)

[Copy of Code 7](#_Toc64894995)

[Description of Methods 10](#_Toc64894996)

[Test Data 11](#_Toc64894997)

# Description of Problem

This is a report on how to write a menu-driven application to implement a Search Engine Optimisation (SEO) application and its operations. The Search Engine Optimisation (SEO) application uses URL addresses and keywords to improve a site’s search visibility as the keywords will acts as an index to a URL. The application will store the URLs and up to 4 keywords per URL. The keywords are to be stored in a Binary Search Tree (BST) with links to the URLs which contained the keyword.

SEO needs to be able to perform the following operations implemented using a linked structure:

* **put(keyword, URL)**: Checks if the keyword already exists in the BST. If it isn’t, it will add a keyword into the BST and a link to the URL. If the keyword does exist, the URL is just added to the URL list for that keyword.
* **get(keyword)**: will return the list of URLs which uses the given keyword.
* **printOrder()**: will output the keywords list in alphabetical order.
* **deleteURL(keyword, URL)**: will delete the given URL from the keyword list. Note: the keyword is NOT deleted. Count(keyword): returns the number of URLs which reference a given keyword.
* **count(keyword)**: returns the number of URLs which reference a given keyword

Along with the listed operations above, 2 more operations which would be suitable and useful for SOE application must be included.

# Underlying Data Structures

To create a SEO, I used a Binary Search Tress (BST). A BST is a tree where each node has at most 2 children. These children can have their own subtrees, known as left subtree or right subtree. The BST starts at the root node, in this case 20. When values are placed in the BST it compares the value to a node. If it is a lower value, the value moves to the left. If it is a greater value, the value moves to the right. This comparison and shift will continue until the value finds a node with no leaf node in the position that it wishes to take.Diagram

Description automatically generated

# Pseudocode for Algorithms

* getNode()

{

TreeNode\* pointer;

pointer = (TreeNode\*)malloc(sizeof(TreeNode));

if (pointer == (TreeNode\*)NULL)

{

cout Failure

exit

}

return pointer

}

* insertTree(Treenode\*, Treenode\*)

{

if (!root)

{

root = newnode;

root->lchild = NULL;

root->rchild = NULL;

}

else if (newnode->data < root->data)

{

root->lchild = insertTree(root->lchild, newnode)

}

else

{

root->rchild = insertTree(root->rchild, newnode)

}

return root

* put(keyword, URL)

{

}

* get(keyword)

{

}

* printOrder()

{

}

* deleteURL()

{

}

* count(keyword)

{

}

# Copy of Code

//Student name: Rachel Doogue

//Student number:C00237335

//DS Assignment 2, create a Search Engine Optimisation application

#include <iostream>

#include <string>

#include <math.h>

#define MAX 80

typedef struct treenode

{

std::string data;

struct treenode\* lchild;

struct treenode\* rchild;

}TreeNode;

TreeNode\* insertTree(TreeNode\*, TreeNode\*);

TreeNode\* getNode(void);

int main()

{

TreeNode\* root = NULL;

TreeNode\* newnode;

int response = -1;

std::string input;

std::cin >> input;

newnode = getNode();

newnode->data = input;

root = insertTree(root, newnode);

while (response != 0)

{

std::cout << "\n";

std::cout << "Press 1 to check if keyword already exists\n";

std::cout << "Press 2 to return list of URLs which uses a given keyword\n";

std::cout << "Press 3 to output keywords in alphabetical order\n";

std::cout << "Press 4 to delete given URL\n";

std::cout << "Press 5 to return number of URLs that reference given keyword\n";

std::cout << "Press 6 to \n";

std::cout << "Press 7 to \n";

std::cout << "Press 0 to exit\n";

std::cin >> response;

switch (response)

{

case 1:

system("CLS");

//put();

std::cout << "\n Incomplete. \n ";

break;

case 2:

system("CLS");

//get();

std::cout << "\n Incomplete. \n ";

break;

case 3:

system("CLS");

//printOrder();

std::cout << "\n Incomplete. \n ";

break;

case 4:

system("CLS");

//deleteURL();

std::cout << "\n Incomplete. \n ";

break;

case 5:

system("CLS");

//count();

std::cout << "\n Incomplete. \n ";

break;

case 6:

system("CLS");

std::cout << "\n Incomplete. \n ";

break;

case 7:

system("CLS");

std::cout << "\n Incomplete. \n ";

break;

case 0:

system("CLS");

std::cout << "\n Thank You for Your Time. \n ";

break;

default:

system("CLS");

std::cout << "\n Invalid Input. Try Again. \n ";

break;

}

}

}

TreeNode\* getNode(void)

{

TreeNode\* pointer;

pointer = (TreeNode\*)malloc(sizeof(TreeNode));

if (pointer == (TreeNode\*)NULL)

{

std::cout << "Failed to allocate memory\n";

exit(1);

}

return pointer;

}

TreeNode\* insertTree(TreeNode\* root, TreeNode\* newnode)

{

if (!root) {

root = newnode;

root->lchild = NULL;

root->rchild = NULL;

}

else if (newnode->data < root->data)

{

root->lchild = insertTree(root->lchild, newnode);

}

else

{

root->rchild = insertTree(root->rchild, newnode);

}

return root;

}

void put()

{

return;

}

void get()

{

return;

}

void printOrder()

{

return;

}

void deleteURL()

{

return;

}

void count()

{

return;

}

# Description of Methods

* getNode() is used to allocate memory for a new node. Pointer is defined by allocation memory for the size of the tree node. If no memory has been allocated indicated by the pointer equalling NULL then a message will let the user know and then exit. Returning the pointer.
* insertTree() has three different functions. First it creates a root node if there is not one already present. Creating a new root node with an empty left and right child.

Second if there is a root node it checks to see if the new nodes data you want to enter has a value less than the root nodes. If so that new node will be inserted into the tree on the left-hand side.

Finally the only option left is that the new node has a value greater than the root node. This new node is inserted into the tree on the right-hand side.

* put() incomplete.
* get() incomplete.
* printOrder() incomplete.
* deleteURL() incomplete.
* count() incomplete.

# Test Data

I could not input any test data as I was unable to get the BST to properly initialise.

Text

Description automatically generated

When trying to initialise the BST it then exits the program, ignoring the menu display output and switch case.

Text

Description automatically generated