|  |  |  |
| --- | --- | --- |
|  | | |
| Федеральное государственное бюджетное  образовательное учреждение высшего образования «Новосибирский государственный технический университет» | | |
|  | | |
| Кафедра прикладной математики | | |
| Практическое задание № 3 | | |
| по дисциплине «Структуры данных и алгоритмы» | | |
| **Бинарные деревья** | | |
|  | | |
|  | Бригада 7 | Дроздов даниил |
| Группа ПМ-15 | иванова дарья |
| Вариант 3 | Чепурная Алина |
|  |  |
|  |  |
|  |  |
| Преподаватель | Тракимус Юрий Викторович |
|  |  |
| Новосибирск, 2022 | | |

**1 Задание**

В заданном двоичном дереве найти длину пути от корня до ближайшей вершины со значением равным заданному.

Реализовать с помощью прямого и рекурсивного обходов.

**2 Анализ задачи**

Входные данные: в первой строке искомый элемент, во второй строке двоичное дерево из файла.

Выходные данные: длину пути от корня до ближайшей вершины со значением равным заданному в файл.

При отсутствии заданного элемента в дереве вывести в файл соответствующее сообщение.

Для решения задачи определим структуру двоичного дерева, состоящую из элемента простого типа, хранящее значение, указатели на левое и правое поддерево. Также понадобятся структуры стека и списка.

В начале выполнения программы введем дерево из файла в функции input. Формат ввода следующий: <элемент вершины>(<левое поддерево>,<правое поддерево>).

Для решения задачи определим в структуре дерева функции:

1. pathlen – решение задачи прямым обходом.
2. pathlen\_r – решение задачи рекурсивным обходом.

Функции принимают char elem – искомый элемент, возвращают длину пути от корня до ближайшей вершины со значением равным заданному. Если искомый элемент отсутствует в дереве, возвращают специальное значение -1. В конце выполнения программы выведем результат в файл в функции output. Если результатом поиска является -1, то вместо значения вывести сообщение об отсутствии заданного элемента в дереве.

**3 Программа**

// tree.h

#ifndef TREE\_H

#define TREE\_H

#include <stdio.h>

struct tree

{

char elem;

tree \*left;

tree \*right;

tree(char \_elem = 0, tree \*\_left = NULL, tree \*\_right = NULL);

void input(FILE \*fp);

int pathlen(char elem);

int pathlen\_r(char elem);

};

#endif

// tree.cpp

#include <stdio.h>

#include "tree.h"

#include "stack.h"

tree::tree(char \_elem, tree \*\_left, tree \*\_right) : elem(\_elem), left(\_left), right(\_right) {}

void tree::input(FILE \*fp)

{

tree \*t = this;

stack \*s = new stack();

elem = fgetc(fp);

for (char c = fgetc(fp); c != EOF; c = fgetc(fp))

{

switch (c)

{

case '(':

s->push(t);

t = t->left = new tree(fgetc(fp));

break;

case ',':

t = s->top();

t = t->right = new tree(fgetc(fp));

break;

default:

s->pop(&t);

break;

}

}

};

int tree::pathlen(char tar)

{

stack \*s = new stack();

tree \*t = this;

int len = 0;

s->push(this);

for ( ; !s->empty() && t->elem != tar; )

{

if (t->left)

{

s->push(t);

t = t->left;

len++;

}

else

{

for ( ; !s->empty() && !(t == s->top()->left && s->top()->right); len--,

s->pop(&t));

if (!s->empty() && s->top()->right)

t = s->top()->right;

}

}

return t->elem == tar ? len : -1;

}

int tree::pathlen\_r(char tar)

{

if (elem == tar) return 0;

if (left)

{

int l = left->pathlen\_r(tar);

if (l != -1)

return l + 1;

}

if (right)

{

int r = right->pathlen\_r(tar);

if (r != -1)

return r + 1;

}

return -1;

}

// stack.h

#ifndef STACK\_H

#define STACK\_H

#include <stdio.h>

#include "list.h"

struct stack

{

list \*beg;

stack(list \*\_beg = NULL);

void push(tree \*elem);

bool pop(tree \*\*elem);

bool empty();

tree \*top();

};

#endif

// stack.cpp

#include <stdio.h>

#include "stack.h"

#include "tree.h"

stack::stack(list \*\_beg) : beg(\_beg) {};

void stack::push(tree \*elem)

{

beg = new list(elem, beg);

}

bool stack::empty()

{

return !beg;

}

bool stack::pop(tree \*\*elem)

{

bool is\_empty = empty();

if (!is\_empty)

{

\*elem = beg->elem;

list \*d = beg;

beg = beg->next;

delete d;

}

return is\_empty;

}

tree \*stack::top()

{

return beg->elem;

}

// list.h

#ifndef LIST\_H

#define LIST\_H

#include <stdio.h>

#include "tree.h"

struct list

{

tree \*elem;

list \*next;

list(tree \*\_elem = NULL, list \*\_next = NULL);

};

#endif

// list.cpp

#include <stdio.h>

#include "list.h"

list::list(tree \*\_elem, list \*\_next) :

elem(\_elem), next(\_next){}

// main.cpp

#include <stdio.h>

#include "tree.h"

bool input(tree \*t, char \*tar)

{

FILE \*fp;

fopen\_s(&fp, "input.txt", "r");

if (!fp) return false;

\*tar = fgetc(fp);

fgetc(fp);

t->input(fp);

return true;

}

bool output(int l)

{

FILE \*fp;

fopen\_s(&fp, "output.txt", "w");

if (!fp) return false;

fprintf\_s(fp, (l==-1) ? "Заданный элемент не найден." : "%d", l);

return true;

}

int main()

{

tree \*t = new tree();

char c;

if (!input(t, &c)) return 1;

output(t->pathlen\_r(c));

return 0;

}

**4 Тесты**

|  |  |  |
| --- | --- | --- |
| **№** | **Данные в файле input** | **Пояснение** |
| **1.1** | 1  1(2(4,5),3(6,7)) |  |
| **1.2** | 3  1(2(4,5),3(6,7)) |
| **1.3** | 5  1(2(4,5),3(6,7)) |
| **1.4** | A  1(2(4,5),3(6,7)) | Искомого элемента нет в дереве. |
| **2** | a  a | Дерево состоит из одной вершины, искомый элемент единственный. |
| **3.1** | a  a(b(d,e(g)),c(f(h))) |  |
| **3.2** | d  a(b(d,e(g)),c(f(h))) |
| **3.3** | g  a(b(d,e(g)),c(f(h))) |
| **3.4** | h  a(b(d,e(g)),c(f(h))) |
| **3.5** | c  a(b(d,e(g)),c(f(h))) |
| **3.6** | 0  a(b(d,e(g)),c(f(h))) | Искомого элемента нет в дереве. |

1. **Результаты работы программы**

|  |  |
| --- | --- |
|  | **Результаты тестов** |
| **1.1** | 0 |
| **1.2** | 1 |
| **1.3** | 2 |
| **1.4** | Заданный элемент не найден. |
| **2** | 0 |
| **3.1** | 0 |
| **3.2** | 2 |
| **3.3** | 3 |
| **3.4** | 3 |
| **3.5** | 1 |
| **3.6** | Заданный элемент не найден. |