Министерство образования и науки Российской Федерации

Севастопольский государственный университет

Кафедра ИС

Отчет

по лабораторной работе № 11

«Программирование нелинейных

структур данных на языке С/С++»

Выполнил

ст. гр. И12д

Серегин А.В.

Проверил:

асс. Забаштанский А.К.

Севастополь

2015

1. Цель работы

Изучений нелинейных структур данных и приобретение навыков разработки и отладки программ, использующих древовидные структуры. Исследование особенностей работы с поисковыми бинарными деревьями на языке C/C++.

2. Варианты заданий

Представить приведенную в предыдущей работе таблицу в виде бинарного дерева. Написать функции создания и обхода дерева, а также одну из функций, приведенных ниже. Значения полей и количество записей в таблице студент выбирает сам. Программа должна сохранять дерево в файле и создавать его заново при её повторном запуске.

Таблица 1 - Расписание

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| № Поезда | Станция отправки | Станция назначения | Время отправки | Время прибытия | Стоимость билета |

Вариант 20: Функцию, которая подсчитывает число вершин на n-ом уровне непустого дерева T.

3. Структурная схема



а) б)

Рисунок 1 – Функции ввода-вывода:

a) ввод данных;

б) вывод данных.



Рисунок 2 – Функция DeleteTree.



Рисунок 3 – Функция PrintTree.



Рисунок 4 – Функция PrintTreeStruct.



Рисунок 5 – Функция LoadFile.



Рисунок 6 – Функция AddNode.



Рисунок 7 – Функция main.



Рисунок 8 – Функция SaveFile.

4. Код программы

#include <iostream>

#include <iomanip>

#include <fstream>

#include <conio.h>

using namespace std;

struct Time

{

int hour;

int minute;

};

//запись о поезде

struct Train

{

int number;

char StPoint[8];

char EndPoint[8];

Time StTime;

Time EndTime;

int Price;

};

//структура дерева

struct TrainTree

{

Train key;

TrainTree\* left;

TrainTree\* right;

};

//считывание данных о поезде

Train ScanTrain()

{

Train Train;

cout << "Введите номер поезда:" << endl;

cin >> Train.number;

cout << "Введите место отправки:" << endl;

cin >> Train.StPoint;

cout << "Введите место прибытия:" << endl;

cin >> Train.EndPoint;

cout << "Введите время отправки:" << endl;

cin >> Train.StTime.hour >> Train.StTime.minute;

cout << "Введите время прибытия:" << endl;

cin >> Train.EndTime.hour >> Train.EndTime.minute;

cout << "Введите стоимость билета:" << endl;

cin >> Train.Price;

return Train;

}

//вывод данных о поезде

void PrintTrain(const Train Train)

{

cout.setf(ios::left);

cout << setw(3) << Train.number;

cout << setw(9) << Train.StPoint;

cout << setw(9) << Train.EndPoint;

cout << setw(2) << Train.StTime.hour << ':' << setw(4) << Train.StTime.minute;

cout << setw(2) << Train.EndTime.hour << ':' << setw(5) << Train.EndTime.minute;

cout << Train.Price << endl;

}

//функция очистки памяти

void DeleteTree(TrainTree\* node)

{

if (!node) return;

DeleteTree(node->left);

DeleteTree(node->right);

delete node;

}

//добавление нового узла в дерево

TrainTree\* AddNode(TrainTree\* node, const Train NewTrain)

{

if (!node)

{

node = new TrainTree;

node->key = NewTrain;

node->left = NULL;

node->right = NULL;

return node;

}

if (node->key.number > NewTrain.number)

node->left = AddNode(node->left, NewTrain);

else

node->right = AddNode(node->right, NewTrain);

return node;

}

//вывод дерева

void PrintTree(TrainTree\* node)

{

if (!node) return;

PrintTree(node->left);

PrintTrain(node->key);

PrintTree(node->right);

}

//отображение структуры дерева

void PrintTreeStruct(TrainTree\* node, int level)

{

if (!node) return;

PrintTreeStruct(node->right, level + 1);

for (int i(0); i < level; i++)

cout << " ";

cout << node->key.number << endl;

PrintTreeStruct(node->left, level + 1);

}

//количество узлов n уровня

int NodeCount(TrainTree\* node, int level)

{

if (!node || level < 1) return 0;

if (level == 1) return 1;

if (level > 1)

return NodeCount(node->left, level - 1) +

NodeCount(node->right, level - 1);

}

//загрузка файла

void LoadFile(fstream& f, TrainTree\*\* root)

{

f.open("train.dat", ios::in | ios::binary);

if (!f) return;

Train temp;

while (!f.eof())

{

if (f.read(reinterpret\_cast<char\*>(&temp), sizeof(Train)))

\*root = AddNode(\*root, temp);

}

f.close();

}

//запись в файл

void PrintToFile(fstream& f, TrainTree\* node)

{

if (!node) return;

f.write(reinterpret\_cast<char\*>(&node->key), sizeof(Train));

PrintToFile(f, node->left);

PrintToFile(f, node->right);

}

//сохранение файла

void SaveFile(fstream& f, TrainTree\* root)

{

f.open("train.dat", ios::out | ios::binary);

PrintToFile(f, root);

f.close();

}

int main()

{

setlocale(LC\_ALL, "Russian");

fstream f;

TrainTree\* root = NULL;

LoadFile(f, &root);

char key;

int n=0;

do

{

system("cls");

cout << "1.Add 2.PrintTree 3.PrintTreeStruct 4.NodeCount Esc.Exit" << endl;

key = \_getch();

system("cls");

switch (key)

{

case '1':

root = AddNode(root, ScanTrain());

break;

case '2':

cout << "№ StPoint EndPoint StTime EndTime Price" << endl;

PrintTree(root);

break;

case '3':

PrintTreeStruct(root, 0);

break;

case '4':

cout << "Введите № уровня:" << endl;

cin >> n;

cout << NodeCount(root, n) << endl;

break;

system("pause");

} while (key != 27);

SaveFile(f, root);

DeleteTree(root);

return 0;

}

5. Тестовые примеры

На рисунках ниже представлены тестовые примеры работы программы.

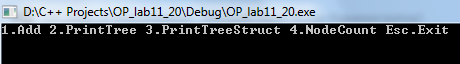


Рисунок 9 – Главное меню.

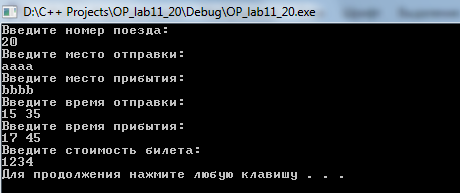


Рисунок 10 – Добавление узла.

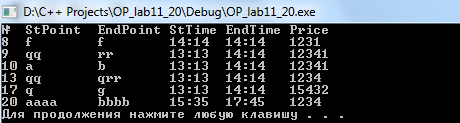


Рисунок 11 – Вывод всего списка.

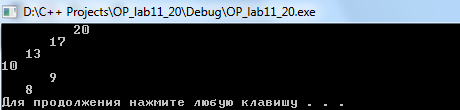


Рисунок 12 – Вывод структуры дерева.

Выводы

С помощью нелинейных структур данных можно управлять различными данными. Сложностью работы с деревьями являются рекурсивные алгоритмы. Одним из минусов работы с рекурсией является быстрое переполнение стека, а также рекурсивный алгоритм работает значительно медленнее чем итерационный. Плюсом древовидных структур является быстрота нахождения нужного элемента в бинарном дереве поиска.