**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра МО ЭВМ**

отчет

**по лабораторной работе №3**

**по дисциплине «Алгоритмы и структуры данных»**

Тема: **Программирование алгоритмов с бинарными деревьями**

|  |  |  |
| --- | --- | --- |
| Студентка гр. 9381 |  | Москаленко Е.М. |
| Преподаватель |  | Фирсов М.А. |

Санкт-Петербург

2020

**Цель работы.**

Ознакомиться с такой динамической структурой данных, как бинарное дерево, и реализовать его и функции для работы с ним на языке программирования С++, используя объектно-ориентированное программирование.

**Задание.**

**Вариант 4д.** Для заданного бинарного дерева *b* типа *BT* с произвольным типом элементов определить, есть ли в дереве *b* хотя бы два одинаковых элемента.

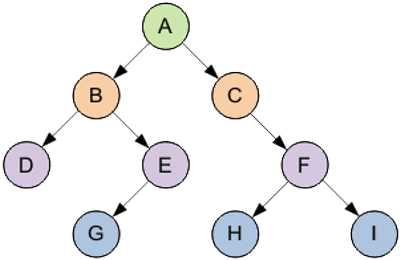
На базе указателей (динамическая связанная память).

**Основные теоретические положения.**

Дерево – структура данных, представляющая собой древовидную структуру в виде набора связанных узлов.

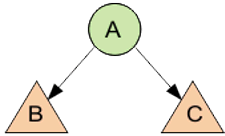
Бинарное дерево — это конечное множество элементов, которое либо пусто, либо содержит элемент (корень), связанный с двумя различными бинарными деревьями, называемыми левым и правым поддеревьями. Каждый элемент бинарного дерева называется узлом. Связи между узлами дерева называются его ветвями.

Способ представления бинарного дерева:



* A — корень дерева
* В — корень левого поддерева
* С — корень правого поддерева

Обход дерева осуществляется в порядке КЛП**.** Обход дерева сверху вниз (в прямом порядке): A, B, C — префиксная форма.



Пример дерева, обрабатываемого программой:

/ - пустой элемент (нет листьев)

* - abd//e//cf//g//

**Ход работы.**

Для создания бинарного дерева реализован класс его элемента **Elem.**

Он имеет приватные поля:

**tree left** – указатель на левое поддерево

**tree right** – указатель на правое поддерево

**T data** – данные (в программе используется char)

Указатель на объект Elem (Elem\*) будем называть tree.

И следующие приватные функции:

**Elem():left(nullptr), right(nullptr), data('\0') {}** – конструктор по умолчанию, правое и левое деревья, а так же значение обнулены.

**tree getLeft()** - возвращает указатель на левое поддерево типа tree.

**void setLeft(tree l)** – устанавливает в левое поддерево  **l** - указатель на переданный элемент типа **tree**

**void setRight(tree r)** – устанавливает в правое поддерево **r** - указатель на переданный элемент типа **tree**

**tree getRight()** – возвращает указатель на правое дерево (tree)

**T getData() const** – возвращает значение поля data типа Т

**void setData(T t)** – устанавливает в поле data переданное значение типа T

Так же реализован класс линейного списка, в котором хранятся все элементы дерева и их число повторений.

Он имеет поля:

**T data** - значение типа T (char в программе)

**SimpleList \*next** - ссылка на следующий элемент

**int countH** – количество повторений элемента в бинарном дереве

И функции:

**SimpleList() : data('\0'), next(nullptr), countH(1) {}; -** конструктор по умолчанию

**SimpleList(T sign, Simple elem = nullptr, int count = 1) : data(sign), next(elem), countH(count) {}; -** конструктор с присваиванием

**void push(T sign)** – добавление элемента список. Параметр - значение типа Т, которое присвоится data добавляемого элемента

**void listPrint()** – печать всего списка

**void initHead(T sign)** - инициализация головы линейного списка. Параметр – значение типа Т, которое присвоится data

**Simple checkSimple(T sign**) – функция проверки вхождения элемента дерева в линейный список. Передается значение типа T

Для рекурсивного вывода дерева реализована функция **void recTreePrint(tree node),** параметром которой является указатель на элемент дерева tree. Она выводит значени элемента, а затем рекурсивно вызывается для левого и правого поддеревьев.

Реализована рекурсивная функция **считывания строки д**ерева и его создания **tree readBT(string input)**, которая рекурсивно заполняет корень, левое и правое поддеревья.

**void treePrint(Simple head, tree tree)** – основная функция программы. Обходом КЛП создается линейный список и в него добавляются элементы. С помощью **checkSimple** ведется подсчет каждого элемента. **Simple head –** указатель на голову линейного списка, **tree tree –** указатель на элемент бинарного дерева.

**Описание алгоритма.**

Для начала программа должна считать данные и создать бинарное дерево. Дерево реализуется на базе указателей: в полях каждого узла должен храниться указатель на левый и правый элемент узла. Если узел в дереве пустой, то хранится указатель на nullptr.

Для перевода введенной строки в дерево поочередно обрабатываются символы строки. Сначала заполняются левые поддеревья узла (сначала левое поддерево корня, потом левое поддерево узла и т.д.), если встречается символ, означающий указатель на nullptr, - “/”, то начинается заполнение правых поддеревьев.

Если левые и правые поддеревья очередного узла заполнены, то происходит возврат на узел выше и рекурсивное заполнение оставшихся узлов дерева.

У пользователя есть выбор: ввод через консоль или через файл. Пустых элементов должно быть на 1 больше, иначе строка неверная.

Для подсчета количества повторений каждого элемента в дереве создается линейный список, элементы которого являются объектами класса SimpleList. Инициализируется указатель на объект класса SimpleList head – голову списка.

Затем вызывается рекурсивная функция treePrint(), которая проверяет элемент на пустоту и если он не пуст, вызывает метод checkSimple класса SimpleList, который проверяет наличие элемента в вспомогательном линейном списке. Если он есть, то поле countH увеличивается на 1, если нет, то элемент добавляется в список. Затем таким же образом проверяются левое и правое поддеревья.

После этого вызывается метод listPrint класса SimpleList, который проверяет значение поля countH каждого элемента линейного списка. Если хоть раз оно больше 1, то в дереве есть одинаковые элементы, о чем и выводится информация.

**Тестирование.**

|  |  |  |
| --- | --- | --- |
| **№** | **Входные данные** | **Вывод** |
| 1 | a/bc/// | Введенное дерево: a/bc///  Головой вспомогательного линейного списка будет a  Добавляем b в линейный список  Добавляем c в линейный список  В бинарном дереве элемент a повторяется 1 раз  В бинарном дереве элемент b повторяется 1 раз  В бинарном дереве элемент c повторяется 1 раз  В дереве нет одинаковых элементов |
| 2 | abc//d//bc//ef/// | Введенное дерево: abc//d//bc//ef///  Головой вспомогательного линейного списка будет a  Добавляем b в линейный список  Добавляем c в линейный список  Добавляем d в линейный список  Добавляем e в линейный список  Добавляем f в линейный список  Посчитаем количество повторений каждого элемента в дереве  В бинарном дереве элемент a повторяется 1 раз  В бинарном дереве элемент b повторяется 2 раз  Обнаружен повтор элемента b  В бинарном дереве элемент c повторяется 2 раз  Обнаружен повтор элемента c  В бинарном дереве элемент d повторяется 1 раз  В бинарном дереве элемент e повторяется 1 раз  В бинарном дереве элемент f повторяется 1 раз  В дереве 2 одинаковых элементов разных видов |
| 3 | abd/g///cd//gi//ba/ | Данные некорректны |
| 4 | ab//c/d// | В дереве нет одинаковых элементов |
| 5 | abee///e//kme//q//l// | В бинарном дереве элемент e повторяется 4 раз  Обнаружен повтор элемента e  В дереве 1 одинаковых элементов разных видов |
| 6 | /a//b// | Данные некорректны |
| 7 | 4/?\*//+// | В дереве нет одинаковых элементов |
| 8 | aeee///e//eee//e//e// | В бинарном дереве элемент e повторяется 9 раз  Обнаружен повтор элемента e  В дереве 1 одинаковых элементов разных видов |
| 9 | a/ | Данные некорректны |

**Выводы.**

Были освоены принципы работы с бинарным деревом, и реализована данная структура данных на языке программирования С++. Созданы два класса (элемент бинарного дерева и линейного списка) для решения задания варианта лабораторной работы и функции, позволяющие определить, есть ли в веденном бинарном дереве повторяющиеся элементы.

ИСХОДНЫЙ КОД

Файл main.cpp

|  |
| --- |
| #include <fstream>  #include "Elem.h"  #include "SimpleList.h"  static int readIndex; *//переменная, отвечающая за индекс элемента в строке*  tree readBT(string input){  T sign = input[readIndex];  readIndex++;  if (sign == '/'){ *//если элемент пустой*  return nullptr;  }  else{  tree buf = new Elem(); *//если нет, создаем листок*  buf->setData(sign);  buf->setLeft(readBT(input));  buf->setRight(readBT(input));  return buf;  }  }  int count(string str, char c){ *//подсчет конкретного символа в строке*  int count = 0;  for (int i = 0; i < int(str.length()); i++){  if (str[i] == c)  count++;  }  return count;  }  void treePrint(Simple head, tree tree) {  if (tree != nullptr) { *//Пока не встретится пустой узел*  cout << "Проверка\n";  Simple p = head->checkSimple(tree->getData());  if (!p) {  if (head->data == '\0') {  cout << "Головой вспомогательного линейного списка будет " << tree->getData() << "\n";  head->initHead(tree->getData());  }  else  head->push(tree->getData());  }  treePrint(head, tree->getLeft()); *//Рекурсивная функция для левого поддерева*  treePrint(head, tree->getRight()); *//Рекурсивная функция для правого поддерева*  }  }  int main() {  string input;  ifstream file;  string name;  cout << "Выберите:\n1. Ввод списка с консоли\n2. Ввод списка с файла\n";  int choice = 0;  cin >> choice;  switch(choice){  case 1:  cout << "Введите скобочную запись бинарного дерева: \n";  cin >> input;  break;  case 2:  cout << "Введите полный путь до файла: \n";  cin >> name;  file.open(name);  if (!file.is\_open()){  cout << "Файл не может быть открыт!\n";  exit(1);  }  getline(file, input); *//считывание из файла строки с данными*  file.close(); *//закрытие файла*  break;  default:  cout << "Вы должны ввести 1 или 2";  return 0;  }  if ((2\*count(input, '/') - input.length() != 1) || (input[0] == '/')){ *//проверка на то, что '/' на одну больше, чем остальных символов*  cout << "Данные некорректны";  return 0;  }  tree root = readBT(input);  cout << "Введенное дерево: ";  recTreePrint(root);  cout << "\n";  auto head = new SimpleList();  treePrint(head,root);  head->listPrint();  return 0;  } |

Файл Elem.h

|  |
| --- |
| #ifndef LAB3\_4\_ELEM\_H  #define LAB3\_4\_ELEM\_H  #include <iostream>  using namespace std;  class Elem;  typedef Elem\* tree;  typedef char T;  *//template <typename T>*  class Elem {  tree left;  tree right;  T data;  public:  Elem():left(nullptr), right(nullptr), data('\0') {};  tree getLeft() {  return left;  }  void setLeft(tree l) {  left = l;  }  void setRight(tree r) {  right = r;  }  tree getRight() {  return right;  }  T getData() const {  return data;  }  void setData(T t) {  data = t;  }  };  void recTreePrint(tree node) {  if (!node) {  cout << '/';  return;  }  cout << node->getData();  recTreePrint(node->getLeft()); *//печать левого*  recTreePrint(node->getRight()); *//печать правого*  }  #endif *//LAB3\_4\_ELEM\_H* |

Файл SimpleList.h

|  |
| --- |
| #ifndef LAB3\_4\_SIMPLELIST\_H  #define LAB3\_4\_SIMPLELIST\_H  #include <iostream>  using namespace std;  class SimpleList;  typedef SimpleList\* Simple;  typedef char T;  static int countSame = 0;  class SimpleList {  public:  T data; *//значение*  SimpleList \*next; *//ссылка на следующий элемент*  int countH;  SimpleList() : data('\0'), next(nullptr), countH(1) {};  SimpleList(T sign, Simple elem = nullptr, int count = 1) : data(sign), next(elem), countH(count) {};  void push(char sign);  void listPrint();  void initHead(char sign)  {  data = sign; *// инициализация головы линейного списка*  }  Simple checkSimple(T sign);  };  void SimpleList::push(T sign) {  cout << "Добавляем " << sign << " в линейный список" << "\n";  Simple current = this;  while (current->next != nullptr) *//пока не достигнем конца списка*  current = current->next;  auto node = new SimpleList(sign);  current->next = node;  }  void SimpleList::listPrint() {  Simple p = this;  do {  cout << "В бинарном дереве элемент " << p->data << " повторяется "; *// вывод значения элемента p*  cout << p->countH << " раз\n";  if (p->countH > 1) {  cout << "\033[31m Обнаружен повтор элемента " << p->data << "\033[0m \n";  countSame++;  }  p = p->next; *// переход к следующему узлу*  } while (p != nullptr);  if (countSame == 0)  cout << "\033[31m В дереве нет одинаковых элементов \033[0m" << "\n";  else  cout << "\033[31m В дереве " << countSame << " одинаковых элементов разных видов \033[0m" << "\n";  }  Simple SimpleList::checkSimple(T sign){  Simple p = this;  do {  if (p->data == sign) {  p->countH++; *//если в линейном списке уже есть такой элемент, то его количество увеличивается на 1*  return p;  }  p = p->next; *// переход к следующему узлу*  } while (p != nullptr);  return nullptr;  }  #endif *//LAB3\_4\_SIMPLELIST\_H* |
|  |