Министерство науки и образования РФ
Федеральное государственное автономное образовательное
учреждение высшего профессионального образования
«Санкт-Петербургский государственный электротехнический
университет «ЛЭТИ» им. В. И. Ульянова (Ленина)»
(СПбГЭТУ «ЛЭТИ»)
Факультет компьютерных технологий и информатики

Кафедра ВТ

Отчёт
по лабораторной работе № 3
на тему: “Деревья”
по дисциплине “Алгоритмы и структуры данных”
Вариант 25

Выполнили: Студенты группы 6306
Гордиенко М. Е.
Пустовойтова А. А.
Принял: Колинько П. Г.

Санкт-Петербург
2017

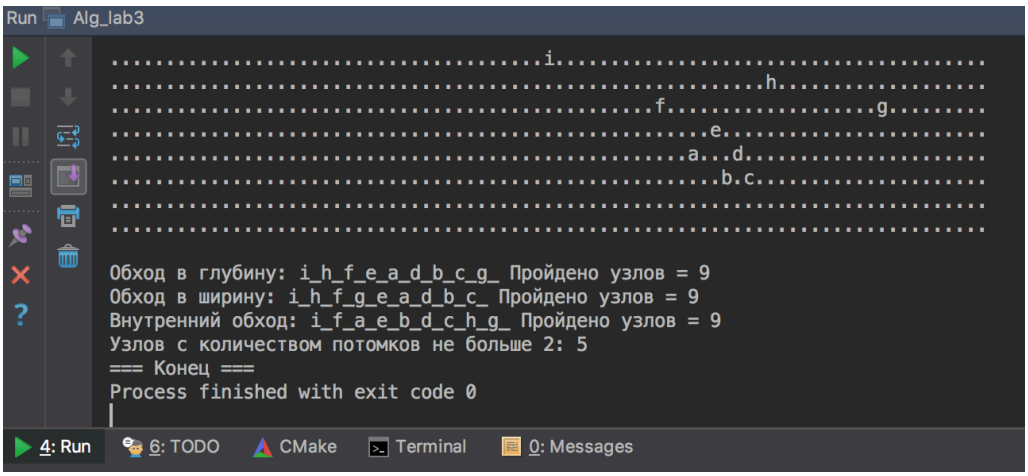
Задание

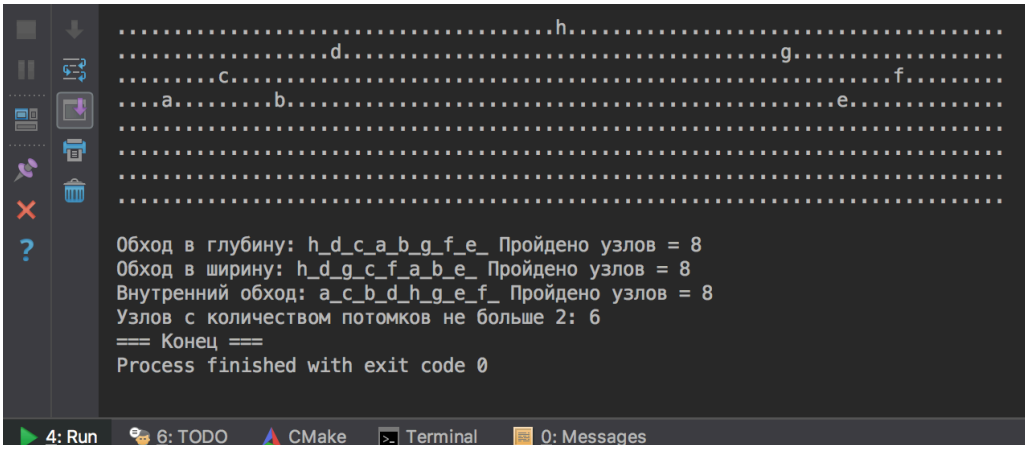
Вид дерева	Разметка	Способ обхода	Что надо вычислить
Двоичное	Обратная	Внутренний	Количество вершин, имеющих не более двух потомков

Способ представления списка в памяти ЭВМ

Список с двумя указателями (левый узел, правый узел).

Контрольные примеры





Временная сложность

Временная сложность представлена в таблице 2.

Таблица. 2. Временная сложность

Функция	Ожидаемая
Создание дерева	O(n)
Обход	O(n)
Вывод	O(n)

Вывод

При выполнении лабораторной работы были получены практические навыки работы с деревьями на языке программирования «C/C++».

Список используемых источников

- Алгоритмы и структуры данных: методические указания к лабораторным работам, практическим занятиям и курсовому проектированию. Федеральный образовательный стандарт / сост.: П.Г. Колинько. - СПб.: Изд-во СПБГЭТУ "ЛЭТИ", 2017. - 64 с.

main.cpp

```
#include <iostream>
#include "Tree.h"

int main() {
    int n = 0;
    Tree Tr('a', 'z', 8);
    srand(time(nullptr));
    setlocale(LC_ALL, "Russian");
    Tr.MakeTree();
    if (Tr.exist()) {
        Tr.OutTree();
        std::cout << '\n' << "Обход в глубину: ";
        n = Tr.DFS();
        std::cout << " Пройдено узлов = " << n;
        std::cout << '\n' << "Обход в ширину: ";
        n = Tr.BFS();
        std::cout << " Пройдено узлов = " << n;
        std::cout << '\n' << "Внутренний обход: ";
        n = Tr.IFS();
        std::cout << " Пройдено узлов = " << n;
        n = Tr.task();
        std::cout << "\nУзлов с количеством потомков не больше 2: " << n;
    }
    else std::cout << "Дерево пусто!";
    std::cout << "\n=== Конец ===";
    return 0;
}
```

node.cpp

```
#include <cstdlib>
#include "Node.h"
#include "Tree.h"

Node * Tree::MakeNode(int depth) {
    Node* v = nullptr;
    int Y = (depth < rand() % 6 + 1) && (num <= 'z');
    if (Y) {
        v = new Node;
        v->lft = MakeNode(depth + 1);
        v->rgt = MakeNode(depth + 1);
        v->d = num++;
    }
    return v;
}
```

node.h

```
#ifndef ALG_LAB3_NODE_H
#define ALG_LAB3_NODE_H


class Node {
   char d;
   Node* lft;
   Node* rgt;
public:
   Node(): lft(nullptr), rgt(nullptr) {}
   ~Node() {
      if(lft) delete lft;
      if(rgt) delete rgt;
   }
   friend class Tree;
};


#endif //ALG_LAB3_NODE_H
```

tree.cpp

```
#include <iostream>
#include <cstring>
#include "Tree.h"
#include "STACK.h"
#include "QUEUE.h"

Tree::Tree(char nm, char mnm, int mxr):
   num(nm), maxnum(mnm), maxrow(mxr), offset(40), root(nullptr),
   SCREEN(new char* [maxrow])
{
   for (int i = 0; i < maxrow; ++i) SCREEN[i] = new char[80];
}

Tree::~Tree() {
   for (int i = 0; i < maxrow; ++i) delete []SCREEN[i];
   delete []SCREEN;
   delete root;
}

void Tree::OutTree() {
   clrscr();
   OutNodes(root, 1, offset);
   for (int i = 0; i < maxrow; ++i) {
      SCREEN[i][79] = 0;
      std::cout << '\n' << SCREEN[i];
   }
   std::cout << '\n';
}
```

```
void Tree::clrscr() {
   for (int i = 0; i < maxrow; ++i) {
      memset(SCREEN[i], '.', 80);
   }
}

void Tree::OutNodes(Node *v, int r, int c) {
   if (r && c && (c < 80)) SCREEN[r - 1][c - 1] = v->d;
   if (r < maxrow) {
      if (v->lft) OutNodes(v->lft, r + 1, c - (offset >> r));
      if (v->rgt) OutNodes(v->rgt, r + 1, c + (offset >> r));
   }
}

int Tree::DFS() {
   const int MaxS = 20;
   int count = 0;
   STACK <Node*> S(MaxS);
   S.push(root);
   while (!S.empty()) {
      Node* v = S.pop();
      std::cout << v->d << '_';
      ++count;
      if (v->rgt) S.push(v->rgt);
      if (v->lft) S.push(v->lft);
   }
   return count;
}

int Tree::BFS() {
   const int MaxQ = 20;
   int count = 0;
   QUEUE <Node*> Q(MaxQ);
   Q.put(root);
   while (!Q.empty()) {
      Node* v = Q.get();
      std::cout << v->d << '_';
      ++count;
      if (v->lft) Q.put(v->lft);
      if (v->rgt) Q.put(v->rgt);
   }
   return count;
}

int Tree::OutNode(Node* root) {
   int count = 0;
   if (root->lft) count += OutNode(root->lft);
   std::cout << root->d << '_';
   if (root->rgt) count += OutNode(root->rgt);
   return count + 1;
```

```
}

int Tree::IFS() {
    return OutNode(root);
}

int Tree::CountOfDefs(Node* root) {
    int count = 0;
    if(root->lft) count += CountOfDefs(root->lft) + 1;
    if(root->rgt) count += CountOfDefs(root->rgt) + 1;
    return count;
}

int Tree::task (Node* root) {
    int count = 0;
    if (root->lft) count += task(root->lft);
    if (root->rgt) count += task(root->rgt);
    if (CountOfDefs(root) <= 2) count++;
    return count;
}

int Tree::task() {
    return task(root);
}
```

---

tree.h

```
#ifndef ALG_LAB3_TREE_H
#define ALG_LAB3_TREE_H


#include "Node.h"

class Tree {
    Node* root;
    char num, maxnum;
    int maxrow, offset, taskCount;
    char** SCREEN;
    void clrscr();
    Node* MakeNode(int depth);
    void OutNodes(Node* v, int r, int c);
    Tree(const Tree&);
    Tree(Tree&&){};
    Tree operator = (const Tree &) const;
    Tree operator = (Tree&&) const{};
    int OutNode(Node*);
    int task (Node*);
    int CountOfDefs(Node*);
public:
    Tree(char num, char maxnum, int maxrow);
    ~Tree();
```

```
    void MakeTree() {
        root = MakeNode(0);
    }
    bool exist() {
          return (root != nullptr);
    }
    int DFS();
    int BFS();
    int IFS();
    void OutTree();
    int task();
};


#endif //ALG_LAB3_TREE_H
```

STACK.cpp

```
#include “STACK.h"
```

STACK.h

```
#ifndef ALG_LAB3_STACK_H
#define ALG_LAB3_STACK_H


template <class Item> class STACK {
    Item* S;
    int t;
public:
    STACK(int maxt) : S(new Item[maxt]), t(0) {}
    int empty() const {
        return (t == 0);
    }
    void push (Item item) {
        S[t++] = item;
    }
    Item pop() {
        return (t ? S[--t] : 0);
    }
};


#endif //ALG_LAB3_STACK_H
```

QUEUE.cpp

```
#include “QUEUE.h"
```

QUEUE.h

```
#ifndef ALG_LAB3_QUEUE_H
```

```
#define ALG_LAB3_QUEUE_H


template <class Item> class QUEUE {
   Item* Q;
   int h, t, N;
public:
   QUEUE (int maxQ) : h(0), t(0), N(maxQ), Q (new Item[maxQ + 1]) {}
   int empty() const {
      return (h % N) == t;
   }
   void put (Item item) {
      Q[t++] = item;
      t %= N;
   }
   Item get() {
      h %= N;
      return Q[h++];
   }
};


#endif //ALG_LAB3_QUEUE_H
```