МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра Вычислительной техники

ОТЧЕТ

по практической работе № 3
по дисциплине «Алгоритмы и Структуры данных»
Тема: «Деревья»

Студент гр. 1308, _____ Мельник Д. А.
Студент гр. 1308, _____ Лепов А. В.
Научный руководитель, _____ Манирагена В.

Санкт-Петербург
2022

СОДЕРЖАНИЕ

1. ЦЕЛЬ РАБОТЫ

Исследование алгоритмов для работы с троичным деревом.

2. ЗАДАНИЕ НА РАБОТУ С ДЕРЕВЬЯМИ

2.1. Общее задание

Написать и отладить программу для работы с деревьями по предложенному преподавателем варианту индивидуального задания (табл. П.2.2). Программа должна выводить на экран изображение дерева с разметкой его вершин, сделанной заданным способом, а под ним — последовательность меток вершин при обходе дерева и результат вычисления заданного параметра. Можно взять за основу учебный пример, убрав из него всё лишнее.

Сделать узел дерева и дерево в целом объектами соответствующих классов, а обходы дерева — функциями-членами для класса «дерево».

Объявить в классе «дерево» деструктор и все конструкторы, поддерживаемые по умолчанию. Сделать невозможным использование тех конструкторов, которые на самом деле не нужны. Сделать в тексте программы временные дополнения и убедиться, что это действительно так.

2.2. Вариант задания (вариант 4)

Таблица. № 1. Вариант задания

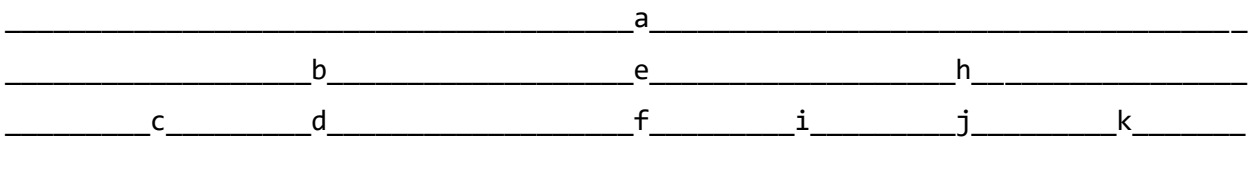
№ варианта	Вид дерева	Разметка	Способ обхода	Что надо вычислить
4	Троичное	Прямая	В ширину	Количество вершин, имеющих предков

3. ОБОСНОВАНИЕ ВЫБОРА СПОСОБА ПРЕДСТАВЛЕНИЯ ДЕРЕВЬЕВ В ПАМЯТИ ЭВМ.

По анализу предыдущих практических работ, в качестве структуры данных для троичного дерева был выбран список, так как он обладает, хоть и не самым быстрым способом обработки данных – как машинные слова, но для удобства использования и возможностью ссылаться от одного узла к другому.

4. ТЕКСТОВЫЙ ПРИМЕР

4.1. Изображение дерева



4.2. Порядок ввода дерева с клавиатуры

```
Node (a,0)1/0: 1
Node (b,1)1/0: 1
Node (c,2)1/0: 1
Node (d,3)1/0: 0
Node (d,3)1/0: 0
Node (d,3)1/0: 0
Node (d,2)1/0: 1
Node (e,3)1/0: 0
Node (e,3)1/0: 0
Node (e,3)1/0: 0
Node (e,2)1/0: 0
Node (e,1)1/0: 1
Node (f,2)1/0: 0
Node (f,2)1/0: 1
Node (g,3)1/0: 0
Node (g,3)1/0: 0
Node (g,3)1/0: 0
Node (g,2)1/0: 1
Node (h,3)1/0: 0
Node (h,3)1/0: 0
Node (h,3)1/0: 0
Node (h,1)1/0: 1
Node (i,2)1/0: 1
Node (j,3)1/0: 0
Node (j,3)1/0: 0
Node (j,3)1/0: 0
Node (j,2)1/0: 1
Node (k,3)1/0: 0
Node (k,3)1/0: 0
Node (k,3)1/0: 0
Node (k,2)1/0: 1
Node (l,3)1/0: 0
Node (l,3)1/0: 0
Node (l,3)1/0: 0
```

5. РЕЗУЛЬТАТЫ ПРОГОНА ПРОГРАММЫ

Ниже, на рис. 1-3 приведены примеры работы алгоритма программы.

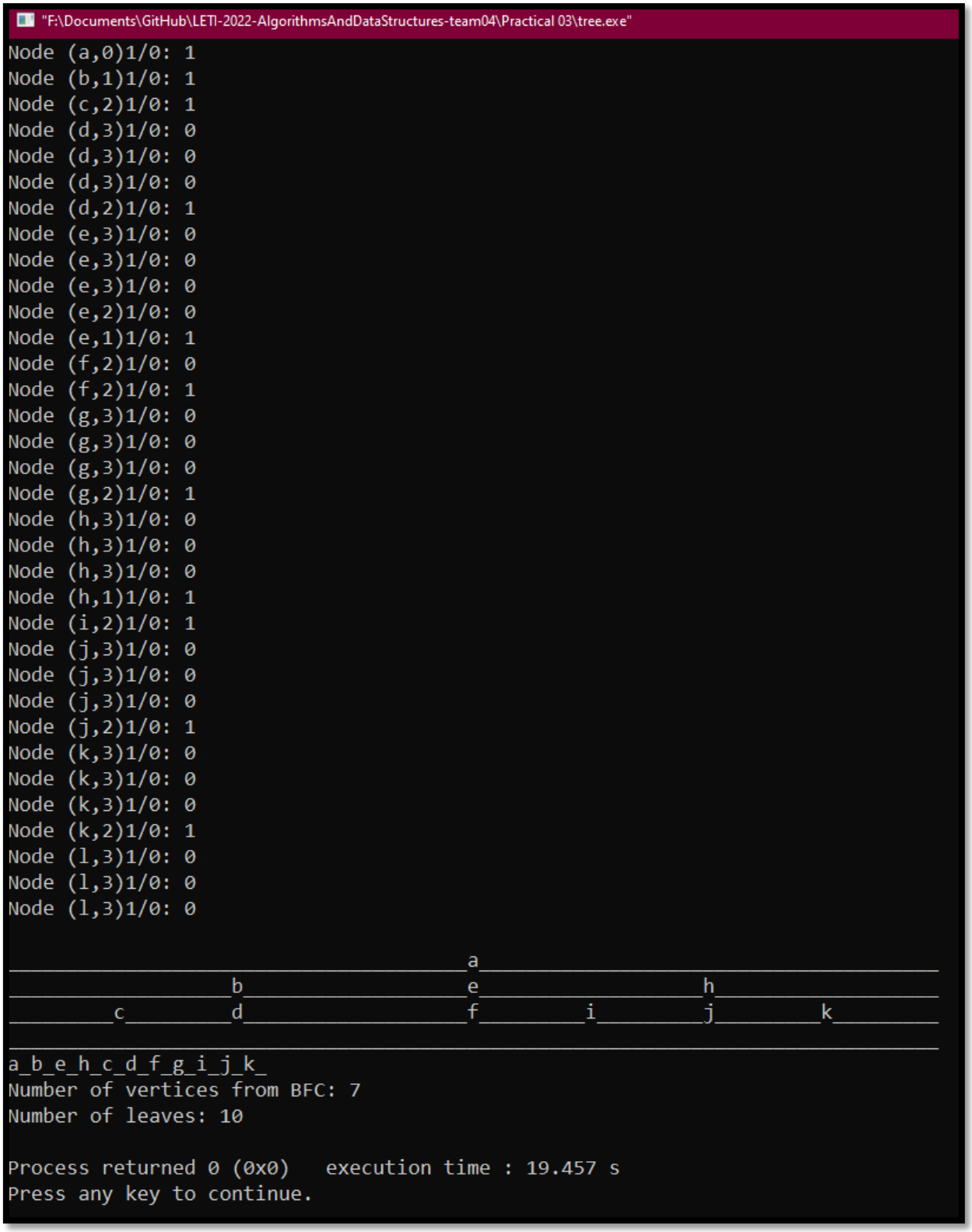


Рис. 1. Результат работы программы

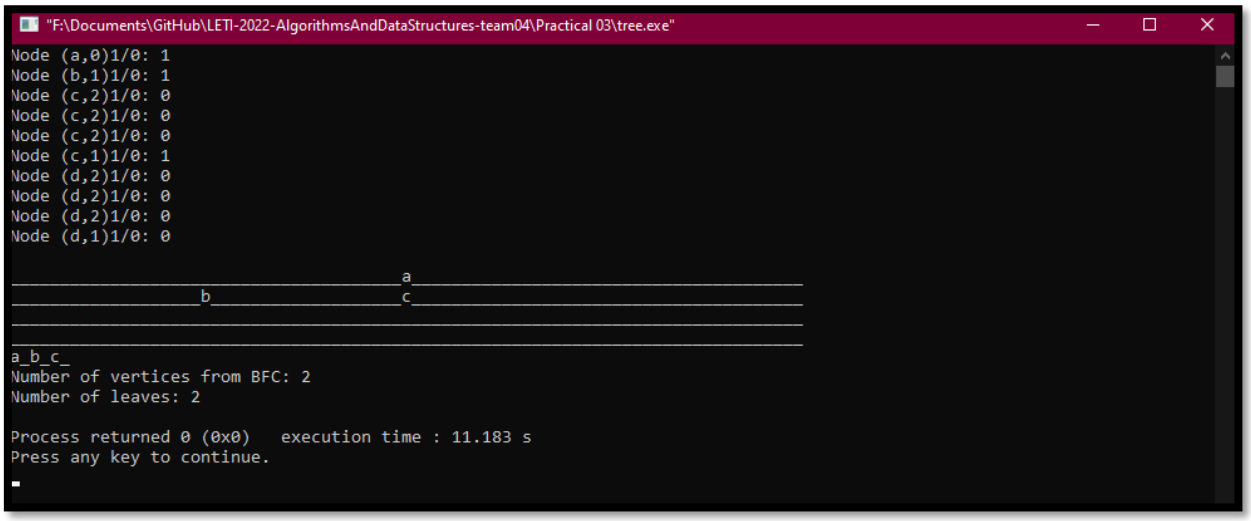


Рис. 2. Результат работы программы

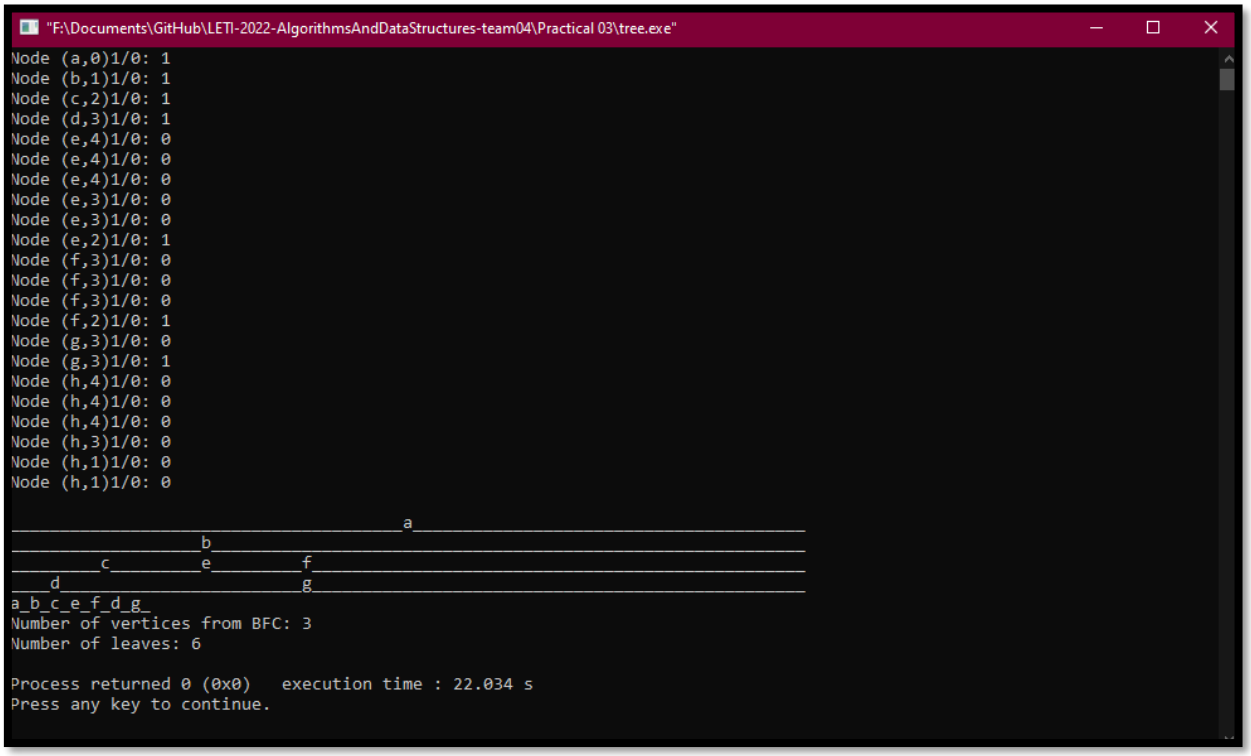


Рис. 3. Результат работы программы

6. ОЦЕНКА ВРЕМЕННОЙ СЛОЖНОСТИ ДЛЯ КАЖДОЙ ФУНКЦИИ ОБХОДА ДЕРЕВА, ИСПОЛЬЗОВАННОЙ В ПРОГРАММЕ

Таблица. № 2. Временные показатели

Создание дерева	Обработка	Вывод
2.89e-06 с.	4.25e-06 с.	0.00827543 с.

7. ВЫВОДЫ О РЕЗУЛЬТАТАХ ИСПЫТАНИЯ АЛГОРИТМОВ ОБХОДА ДЕРЕВЬЕВ

В результате выполнения практической работы и испытаний алгоритмов обхода троичного дерева студенты изучили необходимые для реализации данной программы теоретические основы структур данных.

8. СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ

1. П. Г. Колинько // «Методические указания по дисциплине «Алгоритмы и структуры данных, часть 1»: пользовательские структуры данных» // Издательство: СПб.: СПбГЭТУ «ЛЭТИ», 2022г. – [64 с.]
2. Dale, Nell B. «C++ plus data stractures» // Издательство: Prentice-Hall, Inc., 2003г. – [816 с.]
3. Robert L. Kruse, Alexander J. Ryba // «Data Structures and Program Design in C++» // Издательство: Prentice-Hall, Inc., 2000г. – [717 с.]

9. ПРИЛОЖЕНИЕ А. ЛИСТИНГ ПРОГРАММНОГО КОДА

```
1.  #include <iostream>
2.  #include <fstream>
3.  #include <stdlib.h>
4.  #include <queue>
5.  #include <cstring>
6.
7.  using namespace std;
8.
9.  class Node
10. {
11.     char d;
12.     Node *lft;
13.     Node *mdl;
14.     Node *rgt;
15.
16.     public:
17.     Node():lft(NULL), rgt(NULL), mdl(NULL){}
18.     ~Node(){if(lft) delete lft; if (rgt) delete rgt; if(mdl) delete mdl;}
19.     friend class Tree;
20. };
21.
22. class Tree
23. {
24.     Node * root;
25.     char num, maxnum;
26.     int maxrow, offset;
27.     int numel;
28.     char ** SCREEN;
29.     void clrscr( );
30.     Node* MakeNode(int depth);
31.     void OutNodes(Node * v, int r, int c);
32.     Tree (const Tree &);
33.     Tree (Tree &&);
34.     Tree operator = (const Tree &) const;
35.     Tree operator = (Tree &&) const;
36.
37.     public:
38.     Tree(char num, char maxnum, int maxrow);
39.     ~Tree();
40.     void MakeTree()
41.     {
42.         root = MakeNode(0);
43.     }
44.     bool exist( )
45.     {
46.         return root != nullptr;
47.     }
48.     int DFS( );
49.     int BFS( );
50.     void OutTree( );//*
51.     int show_numel()
52.     {
53.         return numel;
54.     }
55. };
56.
57. Tree ::   Tree(char nm, char mnm, int mxr): num(nm), maxnum(mnm), maxrow(mxr),
    offset(40), root(nullptr), SCREEN(new char * [maxrow]), numel(0)
58. {
```

```
59.      for(int i = 0; i < maxrow; ++i) SCREEN[ i ] = new char[80];
60.  }
61.
62.  Tree :: ~Tree( )
63.  {
64.      for(int i = 0; i < maxrow; ++i)
65.          delete [ ]SCREEN[i];
66.      delete [ ] SCREEN; delete root;
67.  }
68.
69.  Node * Tree :: MakeNode(int depth)
70.  {
71.      Node * v = nullptr;
72.      int Y;
73.      cout << "Node (" << num << ',' << depth << ")1/0: "; cin >> Y;
74.      if (Y)
75.      {
76.          v = new Node;
77.          v->d = num++;
78.          v->lft = MakeNode(depth+1);
79.          v->mdl = MakeNode(depth+1);
80.          v->rgt = MakeNode(depth+1);
81.          numel++;
82.      }
83.      return v;
84.  }
85.
86.  void Tree::OutTree()
87.  {
88.      clrscr();
89.      OutNodes(root,1,40);
90.      for (int i=0; i<maxrow; i++)
91.      {
92.          SCREEN[i][79]=0;
93.          cout << '\n' <<SCREEN[i];
94.      }
95.      cout << '\n';
96.  }
97.
98.  void Tree :: clrscr( )
99.  {
100.     for(int i = 0; i < maxrow; ++i)
101.         memset(SCREEN[ i ], '_', 80);
102. }
103.
104. void Tree::OutNodes(Node *v, int r, int c)
105. {
106.     int M[] = {40,20,10,5,0};
107.     if(r && c && (c<80)) SCREEN[r-1][c-1] = v->d;
108.     if (r < 4)
109.     {
110.         if(v->lft) OutNodes(v->lft, r+1, c - M[r]);
111.         if(v->mdl) OutNodes(v->mdl, r+1, c);
112.         if(v->rgt) OutNodes(v->rgt, r+1, c + M[r]);
113.     }
114. }
115.
116. int Tree :: BFS( )
117. {
118.     int countHasParant = 0; // количество узлов, имеющих предка
119.     int countLeaves = 0; // количество листьев
120.     queue < Node * > Q;     // создание очереди указателей на узлы
```

```
121.     Q.push(root);           // поместить в очередь корень дерева
122.     while (!Q.empty( ))     // пока очередь не пуста
123.     {
124.         Node * v = Q.front( ); Q.pop( );    // взять из очереди,
125.         cout << v->d << '_'; ++countHasParant;       // выдать тег, счёт узлов
126.         if (v->lft) Q.push(v->lft); // Q <- (левый сын)
127.         if (v->mdl) Q.push(v->mdl); // Q <- (средний сын)
128.         if (v->rgt) Q.push(v->rgt); // Q <- (правый сын)
129.         if ((v->lft)==NULL && (v->mdl)==NULL && (v->rgt)==NULL) countLeaves++;
130.     }
131.     return countLeaves;
132. }
133.
134. int main()
135. {
136.     int n=0;
137.     Tree Tr('a','z',4);
138.     Tr.MakeTree();
139.     if (Tr.exist())
140.     {
141.         Tr.OutTree();
142.         n=Tr.BFS();
143.         cout << '\n';
144.         cout << "Number of vertices from BFC: " << n << '\n';
145.         // cout << "Number of vertices that have ancestors: " <<
     Tr.show_numel()-1 << '\n'; // вывод количества узлов, имеющих предка
146.         cout << "Number of leaves: " << Tr.show_numel()-1 << '\n'; // вывод
     количества листьев
147.     }
148.     return 0;
149. }
```