**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 3](#_Toc118990295)

[2. ЗАДАНИЕ НА РАБОТУ С ДЕРЕВЬЯМИ 3](#_Toc118990296)

[2.1. Общее задание 3](#_Toc118990297)

[2.2. Вариант задания (вариант 4) 3](#_Toc118990298)

[3. ОБОСНОВАНИЕ ВЫБОРА СПОСОБА ПРЕДСТАВЛЕНИЯ ДЕРЕВЬЕВ В ПАМЯТИ ЭВМ. 3](#_Toc118990299)

[4. ТЕКСТОВЫЙ ПРИМЕР 4](#_Toc118990300)

[4.1. Изображение дерева 4](#_Toc118990301)

[4.2. Порядок ввода дерева с клавиатуры 4](#_Toc118990302)

[5. РЕЗУЛЬТАТЫ ПРОГОНА ПРОГРАММЫ 5](#_Toc118990303)

[6. ОЦЕНКА ВРЕМЕННОЙ СЛОЖНОСТИ ДЛЯ КАЖДОЙ ФУНКЦИИ ОБХОДА ДЕРЕВА, ИСПОЛЬЗОВАННОЙ В ПРОГРАММЕ 7](#_Toc118990304)

[7. ВЫВОДЫ О РЕЗУЛЬТАТАХ ИСПЫТАНИЯ АЛГОРИТМОВ ОБХОДА ДЕРЕВЬЕВ 7](#_Toc118990305)

[8. СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ 7](#_Toc118990306)

[9. ПРИЛОЖЕНИЕ A. ЛИСТИНГ ПРОГРАММНОГО КОДА 8](#_Toc118990307)

1. **ЦЕЛЬ РАБОТЫ**

Исследование алгоритмов для работы с троичным деревом.

1. **ЗАДАНИЕ НА РАБОТУ С ДЕРЕВЬЯМИ**
   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.

* 1. **Вариант задания (вариант 4)**

1. *Вариант задания*

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

1. **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.

1. **ТЕКСТОВЫЙ ПРИМЕР** 
   1. **Изображение дерева**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_a\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_b\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_e\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_h\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_c\_\_\_\_\_\_\_\_\_d\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_f\_\_\_\_\_\_\_\_\_i\_\_\_\_\_\_\_\_\_j\_\_\_\_\_\_\_\_\_k\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

* 1. **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

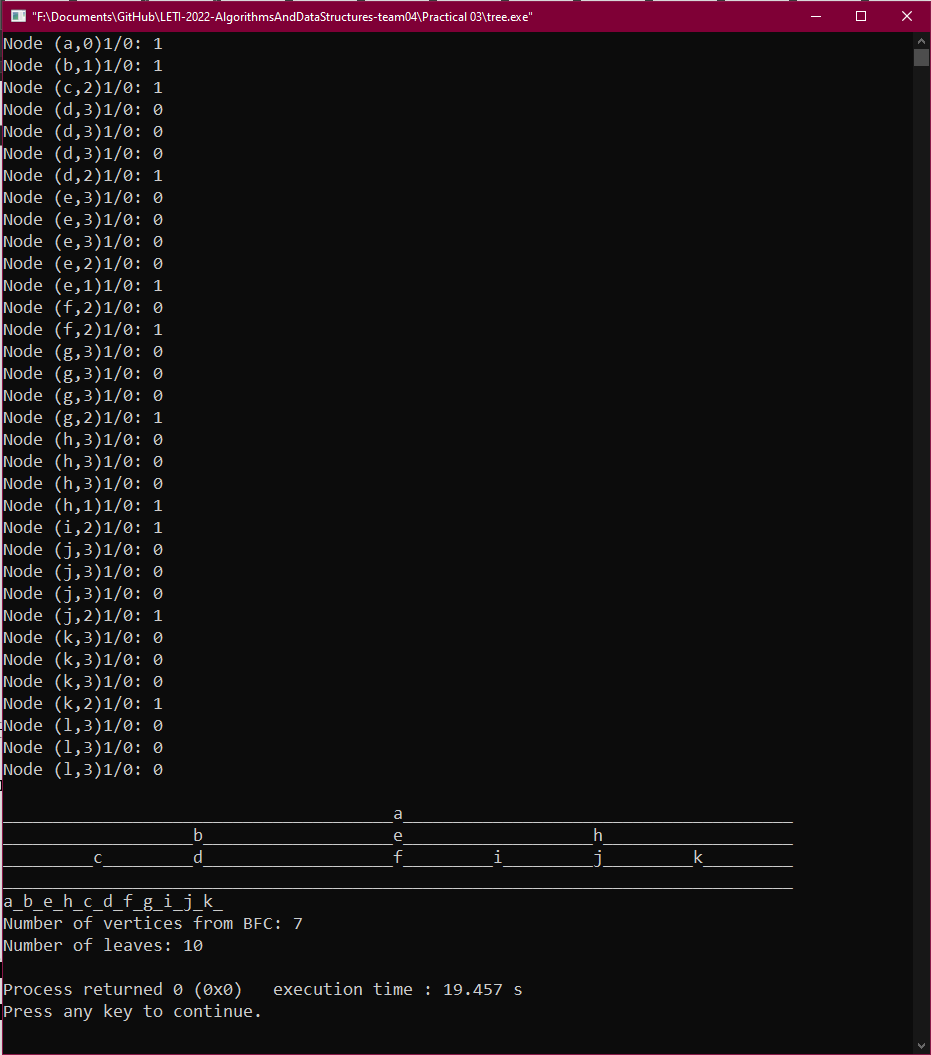
Node (l,3)1/0: 0

Node (l,3)1/0: 0

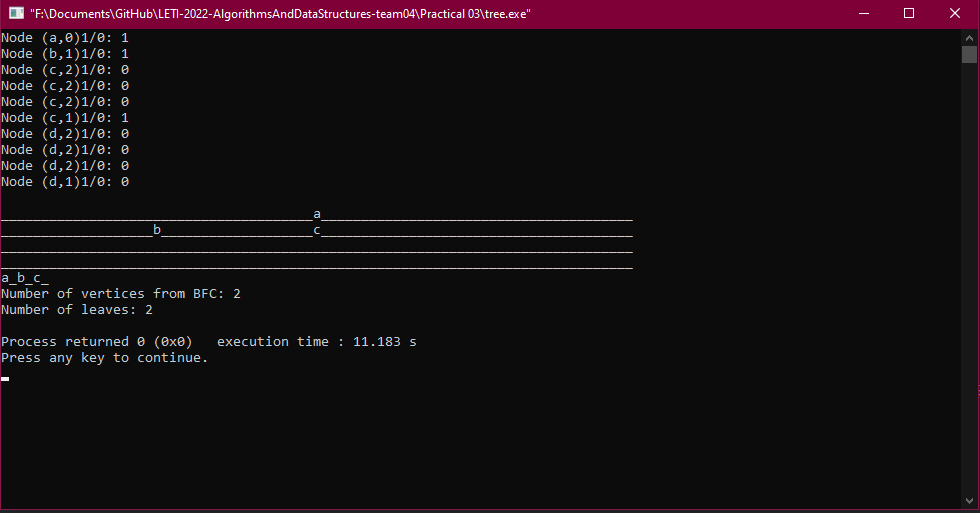
Node (l,3)1/0: 0

1. **РЕЗУЛЬТАТЫ ПРОГОНА ПРОГРАММЫ**

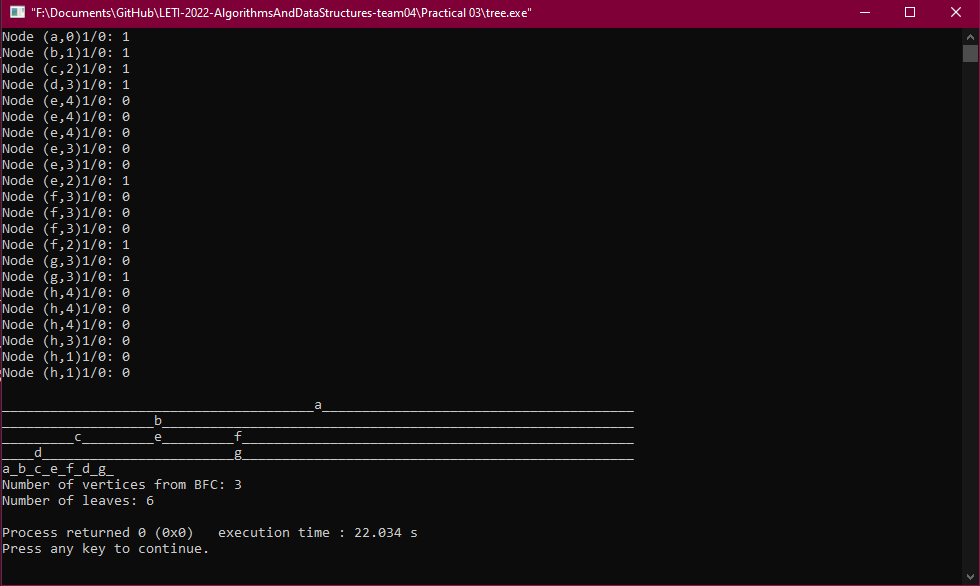
Ниже, на рис. 1-3 приведены примеры работы алгоритма программы.



1. *Результат работы программы*



1. *Результат работы программы*



1. *Результат работы программы*
2. **ОЦЕНКА ВРЕМЕННОЙ СЛОЖНОСТИ ДЛЯ КАЖДОЙ ФУНКЦИИ ОБХОДА ДЕРЕВА, ИСПОЛЬЗОВАННОЙ В ПРОГРАММЕ**
3. *Временные показатели*

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

1. **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.

1. **СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ**
2. П. Г. Колинько // «Методические указания по дисциплине «Алгоритмы и структуры данных, часть 1»: пользовательские структуры данных» // Издательство: СПб.: СПбГЭТУ «ЛЭТИ», 2022г. – [64 с.]
3. Dale, Nell B. «C++ plus data stractures» // Издательство: Prentice-Hall, Inc., 2003г. – [816 с.]
4. Robert L. Kruse, Alexander J. Ryba // «Data Structures and Program Design in C++» // Издательство: Prentice-Hall, Inc., 2000г. – [717 с.]
5. **ПРИЛОЖЕНИЕ A. ЛИСТИНГ ПРОГРАММНОГО КОДА**
6. #include <iostream>
7. #include <fstream>
8. #include <stdlib.h>
9. #include <queue>
10. #include <cstring>
11. using namespace std;
12. class Node
13. {
14. char d;
15. Node \*lft;
16. Node \*mdl;
17. Node \*rgt;
18. public:
19. Node():lft(NULL), rgt(NULL), mdl(NULL){}
20. ~Node(){if(lft) delete lft; if (rgt) delete rgt; if(mdl) delete mdl;}
21. friend class Tree;
22. };
23. class Tree
24. {
25. Node \* root;
26. char num, maxnum;
27. int maxrow, offset;
28. int numel;
29. char \*\* SCREEN;
30. void clrscr( );
31. Node\* MakeNode(int depth);
32. void OutNodes(Node \* v, int r, int c);
33. Tree (const Tree &);
34. Tree (Tree &&);
35. Tree operator = (const Tree &) const;
36. Tree operator = (Tree &&) const;
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. Tree ::   Tree(char nm, char mnm, int mxr): num(nm), maxnum(mnm), maxrow(mxr), offset(40), root(nullptr), SCREEN(new char \* [maxrow]), numel(0)
57. {
58. for(int i = 0; i < maxrow; ++i) SCREEN[ i ] = new char[80];
59. }
60. Tree :: ~Tree( )
61. {
62. for(int i = 0; i < maxrow; ++i)
63. delete [ ]SCREEN[i];
64. delete [ ] SCREEN; delete root;
65. }
66. Node \* Tree :: MakeNode(int depth)
67. {
68. Node \* v = nullptr;
69. int Y;
70. cout << "Node (" << num << ',' << depth << ")1/0: "; cin >> Y;
71. if (Y)
72. {
73. v = new Node;
74. v->d = num++;
75. v->lft = MakeNode(depth+1);
76. v->mdl = MakeNode(depth+1);
77. v->rgt = MakeNode(depth+1);
78. numel++;
79. }
80. return v;
81. }
82. void Tree::OutTree()
83. {
84. clrscr();
85. OutNodes(root,1,40);
86. for (int i=0; i<maxrow; i++)
87. {
88. SCREEN[i][79]=0;
89. cout << '\n' <<SCREEN[i];
90. }
91. cout << '\n';
92. }
93. void Tree :: clrscr( )
94. {
95. for(int i = 0; i < maxrow; ++i)
96. memset(SCREEN[ i ], '\_', 80);
97. }
98. void Tree::OutNodes(Node \*v, int r, int c)
99. {
100. int M[] = {40,20,10,5,0};
101. if(r && c && (c<80)) SCREEN[r-1][c-1] = v->d;
102. if (r < 4)
103. {
104. if(v->lft) OutNodes(v->lft, r+1, c - M[r]);
105. if(v->mdl) OutNodes(v->mdl, r+1, c);
106. if(v->rgt) OutNodes(v->rgt, r+1, c + M[r]);
107. }
108. }
109. int Tree :: BFS( )
110. {
111. int countHasParant = 0; // количество узлов, имеющих предка
112. int countLeaves = 0; // количество листьев
113. queue < Node \* > Q;     // создание очереди указателей на узлы
114. Q.push(root);           // поместить в очередь корень дерева
115. while (!Q.empty( ))     // пока очередь не пуста
116. {
117. Node \* v = Q.front( ); Q.pop( );    // взять из очереди,
118. cout << v->d << '\_'; ++countHasParant;       // выдать тег, счёт узлов
119. if (v->lft) Q.push(v->lft); // Q <- (левый сын)
120. if (v->mdl) Q.push(v->mdl); // Q <- (средний сын)
121. if (v->rgt) Q.push(v->rgt); // Q <- (правый сын)
122. if ((v->lft)==NULL && (v->mdl)==NULL && (v->rgt)==NULL) countLeaves++;
123. }
124. return countLeaves;
125. }
126. int main()
127. {
128. int n=0;
129. Tree Tr('a','z',4);
130. Tr.MakeTree();
131. if (Tr.exist())
132. {
133. Tr.OutTree();
134. n=Tr.BFS();
135. cout << '\n';
136. cout << "Number of vertices from BFC: " << n << '\n';
137. // cout << "Number of vertices that have ancestors: " << Tr.show\_numel()-1 << '\n'; // вывод количества узлов, имеющих предка
138. cout << "Number of leaves: " << Tr.show\_numel()-1 << '\n'; // вывод количества листьев
139. }
140. return 0;
141. }