

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

**ОТЧЕТ
По лабораторной работе №3
по дисциплине «Алгоритмы и структуры данных»
Тема: Программирование алгоритмов с бинарными деревьями**

Студент гр. 9381

Фоминенко А.Н.

Преподаватель

Фирсов М.А.

Санкт-Петербург
2020

1. Цель работы.

Ознакомиться с понятием деревьев в компьютерных науках, реализовать соответствующие алгоритмы и структуры данных.

2. Задание.

Вариант 3в.

3. Для заданного бинарного дерева b типа BT с произвольным типом элементов:

- напечатать элементы из всех листьев дерева b ;
- подсчитать число узлов на заданном уровне n дерева b (корень считать узлом 1-го уровня).

3. Основные теоретические положения.

Дерево – конечное множество T , состоящее из одного или более узлов, таких, что

- а) имеется один специально обозначенный узел, называемый *корнем* данного дерева;
- б) остальные узлы (исключая корень) содержатся в $m \geq 0$ попарно не пересекающихся множествах T_1, T_2, \dots, T_m , каждое из которых, в свою очередь, является деревом.

Деревья T_1, T_2, \dots, T_m называются *поддеревьями* данного дерева.

В данной работе дерево представляется на базе целочисленного массива. Таким образом, чтобы узнать индекс потомка элемента по индексу x , нужно взять элементы $tree[2*x]$ и $tree[2*x + 1]$ для левого и правого потомка соответственно.

Наиболее эффективным видом дерева для решения задачи данной работы есть

Бинарное дерево поиска.

Бинарное дерево поиска обладает следующим свойством: если x — узел бинарного дерева с ключом k , то все узлы в левом поддереве должны иметь ключи, меньшие k , а в правом поддереве большие k .

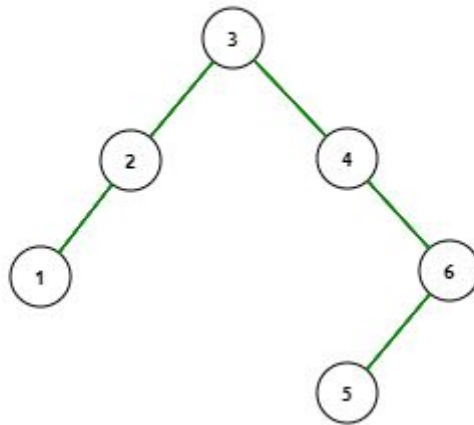


Рисунок 1.

4. Пример работы программы:

Основной тест №1:

Входные данные:

4 6 2 0 1

2

Выходные данные (с промежуточной информацией):

Depth : 4, nodes : 5

4

2

6

0

1

Printing leaves : 6 1 , number of leaves : 2

Printing nodes on depth 2 : 2 6 , number of nodes : 2. .

Дополнительное тестирование :

Номер теста	Входные данные	Результат
2	8 2 9 4 1 7 5 3	Printing leaves : 9 1 5 , number of leaves : 3 Printing nodes on depth 3 : 1 4 , number of nodes : 2
3	a 1	Printing leaves : a , number of leaves : 1 Printing nodes on depth 1 : a , number of nodes : 1
4	6 t 2 9 f e l 2	Printing leaves : 1 e , number of leaves : 2 Printing nodes on depth 2 : 2 t , number of nodes : 2
5	m a p k e q 4	Printing leaves : q e , number of leaves : 2 Printing nodes on depth 4 : e , number of nodes : 1
6	a qwe s 3	Error input, try again.

В последнем тесте были введены символы без пробела, поэтому некорректный ввод.

4. Выполнение программы:

1. Для ввода информации из файла или консоли необходимо ввести “1” или “0” на вопрос программы “0 - консоль, 1 - файл, 2 - запустить тесты”.
2. Для запуска заготовленных тестов нужно будет ввести “2” на тот же запрос.

Программе подается на вход строка для разбора.

И после подается глубина на которой надо посчитать узлы

5. Описание алгоритма:

На вход подается строка (символы разделенные пробелами).

Программа принимает эту строку и посимвольно добавляет к существующему дереву через функцию `insert(Node x)`, это происходит так:

- если значение элемента больше значения в узле \rightarrow идем вправо ($root = root * 2 + 1$), если вправо = пусто, то добавляем элемент направо от этого узла
- иначе идем влево ($root *= 2$), если влево = пусто, то добавляем элемент влево от этого узла
- а если по пути встречаем узел с таким же значением, то увеличиваем счетчик узла на 1 ($count++$).

Так происходит с каждым символом.

После построения дерева нам требуется найти все его листья.

Тут мы идем по дереву (итератором по массиву) и смотрим, если у узла нет правого и левого потомка \rightarrow это лист.

Также надо узнать количество (и сами) узлы на данной глубине.

Доходим в массиве до индекса 2^{dep} , где dep - глубина, и считаем кол-во узлов до индекса $2^{(\text{dep}+1)}$, это у будет количество узлов на глубине dep , так как изначально мы создавали массив так, что правый потомок = $root * 2 + 1$, левый = $root * 2$. значит расстояние до след. глубины будет $2^{(\text{dep}+1)} - 2^{\text{dep}}$. (там некоторые узлы могут быть пустыми, и, конечно, мы их не считаем и не выводим)

5. Описание функций:

```
/**
```

```
* функция добавления узла в дерево
```

```
* @param Node x
```

```
*/
```

```
void BT::insert(Node x)
```

```
-----  
/**
```

```
* функция находящая элемент в дереве и возвращающая индекс(если нет то -1)
```

```
* @param Node x
```

```

* @return index

*/

-----

/**

* функция создающая дерево из строки

* @param string s

*/

void BT::read_BT(string s)

-----

/**

* функция вывода дерева

*/

void BT::print()

-----

/**

* функция подсчета и вывода листьев дерева

*/

void BT::print_leaf()

-----

/**

* функция подсчета и вывода узлов на глубине dep

* @param int dep

*/

```

```
void BT::count_edges(int dep)
```

6. Описание структур данных

Класс элемента дерева.

```
class Node {  
  
public:  
  
    Data data = NULL;  
  
    int count = -1;  
  
};
```

Класс Бинарного дерева поиска в массиве.

Левый потомок элемента $a[x] := a[2*x]$

Правый потомок элемента $a[x] := a[2*x + 1]$

```
class BT {  
  
public:  
  
    int mx_dep = 0;  
  
    Node a[MaxNodes];  
  
    void insert(Node x) ;  
  
    int find(Node x) ;  
  
    void read_str(string &s);  
  
    void read_BT(string s) ;  
  
    void print();
```

```

    void print_leaf();

    void count_edges(int dep);

private:
    int count_nodes = 0;

};

```

8. Вывод:

В ходе выполнения лабораторной работы была создана программа, реализующее бинарное дерево поиска.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

Файл main.cpp:

```

#include "lab3.h"

#include "Test/Test.h"

/**
 * функция добавления узла в дерево
 * @param Node x
 */
void BT::insert(Node x) {
    int root = 1, dep = 1;

    if (count_nodes == 0) {
        count_nodes++;
        mx_dep = 1;
        a[1] = x;
        return;
    }
}

```



```

}

while (true) {

    int left = root * 2;

    int right = left + 1;

    if (a[root].data < x.data) {

        if (right < MaxNodes && a[right].count != -1) {

            root = right;

            dep++;

        } else {

            a[right] = x;

            count_nodes++;

            mx_dep = max(mx_dep, dep + 1);

            return;

        }

    } else if (a[root].data > x.data) {

        if (left < MaxNodes && a[left].count != -1) {

            root = left;

            dep++;

        } else {

            a[left] = x;

            count_nodes++;

            mx_dep = max(mx_dep, dep + 1);

            return;

        }

    } else if (a[root].data == x.data) {

        a[root].count++;

        return;

    }

}

}

/**

```

```

* функция находящая элемент в дереве

* @param Node x

* @return

*/

int BT::find(Node x) {

    int root = 0;

    if (count_nodes == 0) return -1;

    while (true) {

        if (a[root].data == x.data) return root;

        if (a[root].data > x.data) {

            if (a[root * 2].count == -1) return -1;

            root = root * 2;

        } else {

            if (a[root * 2 + 1].count == -1) return -1;

            root = root * 2 + 1;

        }

    }

}

/**

* функция создающая дерево из строки

* @param string s

*/

void BT::read_BT(string s) {

    int i = 0, count = 1;

    while (s[i] == ' ')i++;

    int k = i;

    for (; i < s.length(); i++) {

        if ((equal_Z2(i, k) && s[i] == ' ') || (!equal_Z2(i, k) && s[i] != ' ')) {

```

```

        cerr << "Error input, try again.\n";

        exit(0);
    }

    if (equal_Z2(i, k)) {

        Node x = {s[i], 1};

        insert(x);

        count++;

    }

}

count_nodes = count;

}

/**
 * функция вывода дерева
 */

void BT::print() {

    int dep = 0, k = 1, z = 1, p = 1, it = 1;

    while (dep <= mx_dep) {

        z *= 2;

        dep++;

    }

    cout << "Depth : " << mx_dep << ", nodes : " << count_nodes - 1 << '\n';

    for (int i = 1; i <= mx_dep; i++) {

        p = 1, k *= 2;

        if (i == 1) k = 1;

        for (int j = 0; j < z; j++) {

            if (p <= k && j == z / (k + 1) * p) {

                if (a[it].count != -1) {

                    cout << a[it].data;

                    it++, p++;

                } else {

                    cout << " ", it++, p++;

                }

            }

        }

    }

}

```

```

        }

    } else

        cout << " ";

    }

    cout << '\n';

}

cout << '\n';

}

/**
 * функция подсчета и вывода листов дерева
 */

void BT::print_leaf() {

    int count_leaf = 0;

    cout << "Printing leaves : ";

    for (int i = 1; i < MaxNodes; i++) {

        if (a[i].count >= 1 && (((i * 2 >= MaxNodes) || (a[2 * i].count == -1)) &&
                                ((i * 2 + 1 >= MaxNodes) || (a[2 * i + 1].count == -1))))

            cout << a[i].data << ' ', count_leaf++;

    }

    cout << ", " << "number of leaves : " << count_leaf << '\n';

}

/**
 * функция подсчета и вывода узлов на глубине dep
 * @param int dep
 */

void BT::count_edges(int dep) {

    int z = 1, dep2 = dep, k = 0;

    dep--;

    while (dep--)

        z *= 2;

    int next = 2 * z;

```

```

    cout << "Printing nodes on depth " << dep2 << " : ";

    for (; z < MaxNodes && z < next; z++)

        if (a[z].count != -1) cout << a[z].data << ' ', k++;

    cout << ", " << "number of nodes : " << k << '\n';

}

```

```

void BT::read_str(string &s) {

    char f;

    cin >> f;

    getline(cin, s);

    s = f + s;

}

```

```

int32_t main() {

    char f;

    cout << "0 - консоль, 1 - файл, 2 - запустить тесты\n";

    cin >> f;

    if(f == '2'){

        Test::runall();

        exit(0);

    }

    if (f == '1') {

        freopen("../Test/input.txt", "r", stdin);

        freopen("../Test/output.txt", "w", stdout);

    } else cout << "Введите строку :\n";

    BT bintree;

    string s;

    bintree.read_str(s);

    bintree.read_BT(s);

    bintree.print();

}

```

```
bintree.print_leaf();  
  
cout << "Введите глубину, на которой посчитать количество узлов :\\n";  
  
int k; cin >> k;  
  
bintree.count_edges(k);  
  
}
```