МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКОГО ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра АПУ

ОТЧЕТ

по лабораторной работе № 3

по дисциплине «Алгоритмы и структуры данных»

Тема: Рекурсивные алгоритмы

Студентка гр. 1361	_____	Горбунова Д. А.
Студентка гр. 1361	_____	Токарева У. В.
Преподаватель	_____	Беляев А. В.

Санкт-Петербург
2022

Цель работы: ознакомление с алгоритмами поиска в линейных структурах и оценкой эффективности данных алгоритмов.

Теоретическая часть

Рекурсивные алгоритмы отличаются вызовом самих себя один или более раз для реализации принципа итеративной декомпозиции задачи (выделение в процессе ее решения более простых задач, аналогичных по методу решения). При этом на определенном этапе снижения сложности задача должна обязательно решаться без использования рекурсивного вызова – наступает так называемое ограничение глубины рекурсии (при отсутствии него алгоритм будет зацикливаться).

Классическим примером рекурсивного алгоритма является вычисление факториала:

- для любого N, большего 1, факториал равен N*факториал(N-1)
- для 1 факториал равен 1 (ограничение глубины рекурсии).

Рекурсивный алгоритм, который в процессе исчисления вызывает себя строго один раз, может быть преобразован (ограничение глубины рекурсии при этом фактически превращается в условие выхода из цикла). Основное применение получили рекурсивные алгоритмы, которые осуществляют собственный вызов более 1 раза. Ранее был рассмотрен алгоритм сортировки QuickSort, который вызывает себя рекурсивно дважды:

- от левой части частично отсортированного массива
- от правой части частично отсортированного массива

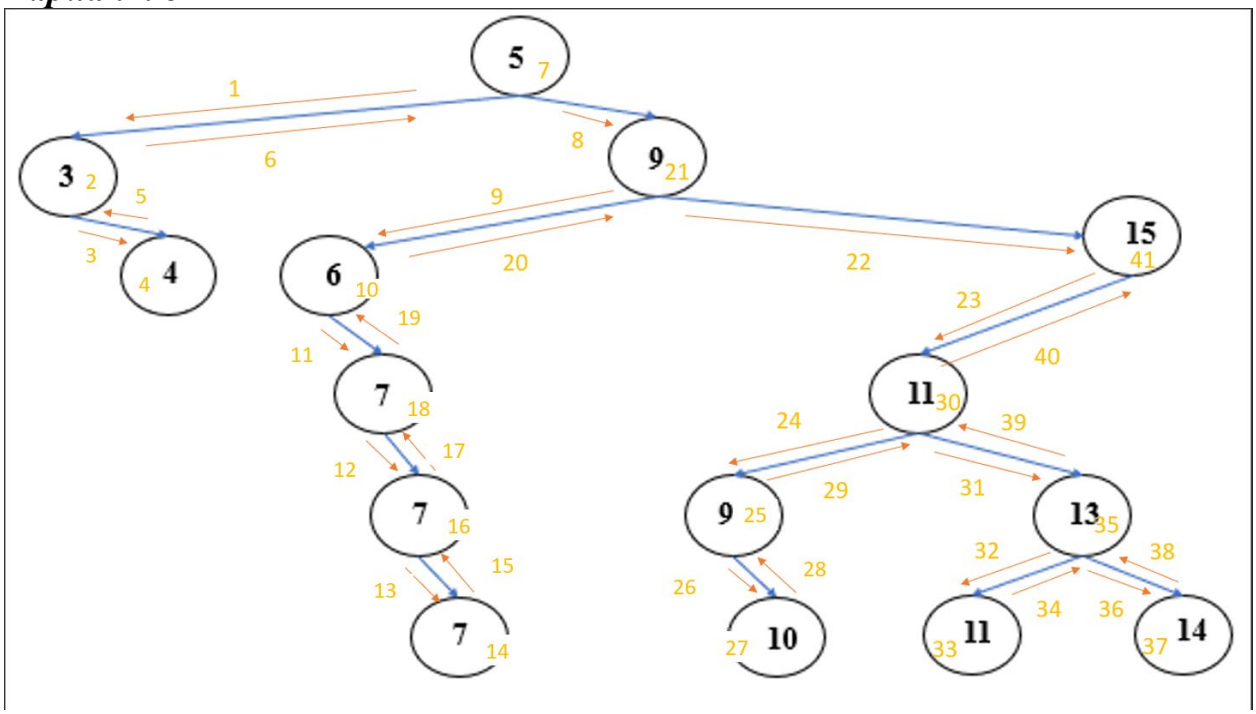
Классическим примером рекурсивных алгоритмов являются алгоритмы обхода деревьев: алгоритм запускается от корня дерева, и затем внутри тела процедуры вызывает ее саму некоторое количество раз в зависимости от количества дочерних ветвей. Частным случаем такого алгоритма является алгоритм обхода двоичного дерева поиска:

- алгоритм вызывает себя рекурсивно от левой дочерней ветви
- алгоритм обрабатывает (например, выводит на экран) значение, сохраненное непосредственно в текущем узле
- алгоритм вызывает себя рекурсивно от правой дочерней ветви

За счет описанной выше последовательности, все записи, размещенные в дереве поиска, будут обработаны в порядке их сортировки (например, по возрастанию числового значения).

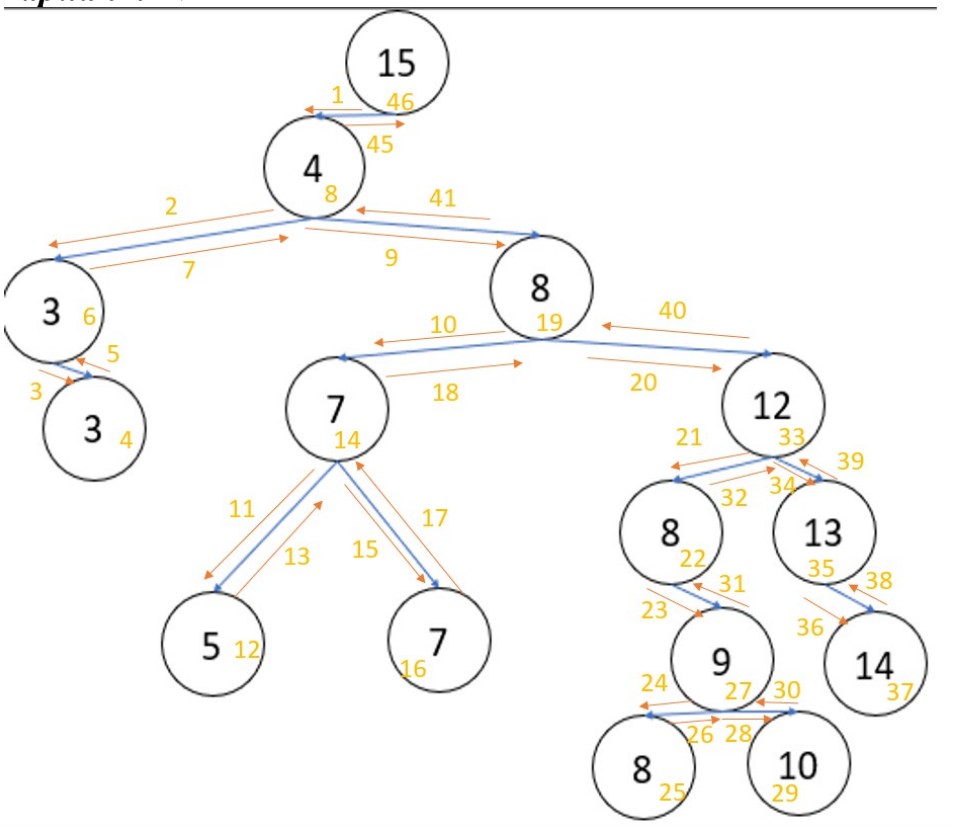
ИДЗ

Горбунова Дарья
Вариант 5



Отсортированный массив: 3 4 5 6 7 7 7 9 9 10 11 11 13 14 15

Токарева Ульяна
Вариант 17



Отсортированный массив: 3 3 4 5 7 7 8 8 8 9 10 12 13 14 15

Исходный код программы BinTree. Листинг дополненного алгоритма

```
#include <cstdio>
#include <iostream>
#include <list>
using namespace std;
int Index;
struct tree{
    float price;
    int article, fabricator;
    tree *left;
    tree *right;
    int articles [1000] = {0};
    int fabricators [1000] = {0};
    size_t way;
    size_t index_node=0;};
tree *create (float price, int article, int fabricator){
    tree *new_node = new tree;
    new_node -> price = price;
    new_node -> article = article;
    new_node -> fabricator = fabricator;
    new_node -> left = nullptr;
    new_node -> right = nullptr;
```

```
    new_node -> way = 0;
    return new_node;}
void add (tree * containingNode, tree * myNode){
    int i=0;
    if (containingNode->price == myNode -> price ){
        while (containingNode->articles [i] !=0)
            i++;
        containingNode -> articles [i] = myNode ->article;
        containingNode -> fabricators[i] = myNode -> fabricator;}
    if (myNode-> price > containingNode->price){
        if(containingNode->right!= nullptr){
            add(containingNode->right, myNode);
        }else{
            containingNode->right = myNode;
            containingNode -> right -> fabricators[i]= myNode -> fabricator;
            containingNode->right->articles[i] = myNode ->article;}
    }else if (containingNode -> price > myNode -> price){
        if(containingNode->left != nullptr){
            add(containingNode->left, myNode);}
        else{
            containingNode->left = myNode;
            containingNode -> left -> fabricators[i]= myNode -> fabricator;
```

```
            containingNode->left->articles[i] = myNode ->article;}}}
void *search (tree * myNode, float price){
    if ((myNode-> price == price))
    {   printf ("Article: %d     Fabricator: %d    Price: %f\n", (myNode->article),  (myNode->fabricator), (myNode-
>price));
        for (int i=1; myNode->articles [i] !=0 ; i++)
            printf ("Article: %d     Fabricator: %d    \n", (myNode->articles[i]), (myNode->fabricator)); }
    if (myNode-> left != nullptr)
    {return search (myNode -> left, price); }
    if (myNode -> right != nullptr)
    {return search (myNode-> right, price);}}
int preOrderTravers(tree * root, int * statistic) {
    if (root) {
        statistic[root->way]+=1;
        preOrderTravers(root->left, statistic);
        preOrderTravers(root->right,statistic); }}
int Way (tree * root){
    int level_tree = 0;
    if (root){
        int Way_left = Way(root->left);
        int Way_right = Way(root -> right);
        level_tree = max(Way_left+1,Way_right+1);}
```

```
    return level_tree;}
int count_way (tree * root, int Way){
    if (root == nullptr) return 0;
    else if (Way == 0) return 1;
    else return (count_way(root->left, Way-1) + count_way(root->right,Way-1));}
int Travel_Tree (tree * root){
    int i;
    if (root -> left !=0)
        Travel_Tree(root->left);
    root -> index_node = Index;
    Index++;
    if ((root->index_node >=0 && root-> index_node<=9) ||
            (root->index_node>=50000 && root->index_node<=50009)){
        cout << "\t\tPrice: "<< root->price<<endl;
        while (root->articles[i]!=0){
            cout << "Article: "<< root->articles[i] <<"\t Fabricator: "<< root->fabricators[i]<<endl;
            i++;}
        cout << endl;}
    if (root->right != 0){
        Travel_Tree(root->right);}
    return 0;}
int main() {
```

```
FILE *file;
int scan_error,i=0, count;
tree * BinTree;
tree * myNode;
float price, user_price;
int article,fabricator;
int* statistic;
file = fopen ("D:\\Users\\Leera\\CLionProjects\\LR2_AiCD\\ads_lab2.txt", "r");
if ( file == nullptr){
    printf ("Error fopen!");
    return 1;}
if (fscanf(file,"%d%f%d",&article, &price, &fabricator)==EOF){
    printf("error!");
    return 4;}
BinTree = create(price,article,fabricator);
myNode=BinTree;
do{
    scan_error=fscanf(file, "%d%f%d",&article, &price, &fabricator);
    myNode = create(price, article, fabricator);
    add (BinTree, myNode);
    ++i;
}while (scan_error!=EOF);
```

```
Travel_Tree(BinTree);
for (int i=0; i < Way (BinTree); ++i ) {
    cout << "Statistic: " << i << "\tWay: " << count_way(BinTree, i) << "\t\t\t";
    cout << "Statistic: " << ++i << "\tWay: " << count_way(BinTree, i) << "\n";}
printf("\n\nPlease, input price:");
scanf("%f", &user_price);
search (BinTree, user_price);
cout << endl;
if (fclose(file)!=0){
    printf("Error fclose!");
    return 2;}
delete (myNode);
delete (statistic);
delete (BinTree);
return 0;}
```

Результат работы программы.

```
Price: 100.1
        Article: 7542765        Fabricator: 8560
        Article: 7111724        Fabricator: 8394
Price: 100.11
        Article: 4585189        Fabricator: 2724
Price: 100.12
        Article: 6810187        Fabricator: 4814
        Article: 6094152        Fabricator: 3576
Price: 100.13
        Article: 4271227        Fabricator: 1609
Price: 100.15
        Article: 4750122        Fabricator: 9086
Price: 100.16
        Article: 4586751        Fabricator: 2176
        Article: 7537085        Fabricator: 9069
Price: 100.17
Price: 100.2
        Article: 1177260        Fabricator: 7238
Price: 100.21
        Article: 1784452        Fabricator: 4937
Price: 100.23
        Article: 2814662        Fabricator: 2791
Price: 883.42
Price: 883.43
        Article: 6401795        Fabricator: 6945
Price: 883.49
        Article: 4999545        Fabricator: 5949
        Article: 1841962        Fabricator: 5900
        Article: 2825738        Fabricator: 3037
```

Рисунок 1 – Вывод первых 13 значений

```
Price: 883.52
        Article: 2400164        Fabricator: 7414
Price: 883.54
        Article: 6227916        Fabricator: 2136
        Article: 2548909        Fabricator: 8909
Price: 883.55
        Article: 4890702        Fabricator: 7564
        Article: 5784308        Fabricator: 4621
        Article: 6128587        Fabricator: 3617
Price: 883.56
        Article: 4860101        Fabricator: 3836
        Article: 2620051        Fabricator: 8501
Price: 883.57
        Article: 8038700        Fabricator: 9980
        Article: 7298454        Fabricator: 4787
        Article: 7447483        Fabricator: 8142
        Article: 6257097        Fabricator: 1616
        Article: 5104696        Fabricator: 6568
        Article: 5461045        Fabricator: 4717
Price: 883.58
        Article: 8395759        Fabricator: 2178
Price: 883.59
        Article: 7712597        Fabricator: 2522
        Article: 1443226        Fabricator: 1610
```

Рисунок 2 – Вывод остающихся значений.

ВЫВОД

В ходе выполнения данной лабораторной работы был изучен рекурсивный алгоритм обхода двоичного дерева поиска. Программа, выполняющая построение двоичного дерева поиска, была дополнена алгоритмом обхода дерева.