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

ОТЧЕТ

по лабораторной работе №2

по дисциплине «Алгоритмы и структуры данных»

Тема: Алгоритмы поиска в линейных структурах данных

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

Двоичный (бинарный) поиск

Двоичный поиск применяется, если данные в анализируемом списке упорядочены по неубыванию или невозрастанию. В этом случае возможен эффективный поиск с оценкой сложности O(log 2 n) следующим способом.

Не теряя общности предположим, что массив упорядочен по неубыванию. Проверим, что ключ поиска не меньше крайнего левого элемента массива и не больше крайнего правого (если это не так, завершим алгоритм с сообщением о том, что искомый элемент отсутствует в массиве). Также проверим, не являются ли крайние значения искомым.

Введем два индекса:

- L (от англ. left), первоначально указывающий на самый левый элемент массива (в дальнейшем будет постепенно смещаться вправо);
- R (от англ. right), первоначально указывающий на самый правый элемент массива (в дальнейшем будет постепенно смещаться влево).

В цикле вычисляем новый индекс M (от англ. middle), равный среднему между L и R с округлением при необходимости (в любую сторону). Если элемент массива с индексом M равен ключу, прекращаем поиск.

Если элемент массива с индексом M меньше ключа, то искомый элемент, если и присутствует в массиве, то только в интервале индексов (M; R), т.к. массив упорядочен по неубыванию. Следовательно, необходимо заменить значение в индексе L значением из M. Аналогично, если элемент массива с индексом M больше ключа, то искомый элемент.

Цикл прекращается по условию L+1=R сообщением об отсутствии ключа в массиве.

Если обнаружено хотя бы одно вхождение ключа в массиве, а по условию поиска требуется вывести все записи из массива с данным значением, то необходимы дополнительные циклы влево и вправо от найденного элемента.

Построение двоичного дерева поиска

Двоичное дерево поиска представляет собой вспомогательную структуру в виде дерева, каждый узел которого содержит значение узлового элемента и три ссылки:

- на «левое» поддерево, в котором находятся все элементы, строго меньшие текущего
- на «правое» поддерево, в котором находятся все элементы, строго большие текущего
- на цепочку ссылок (например, индексов записей в массиве), равных текущей

Построение хеш-таблицы

Хеш-таблица представляет собой вспомогательную структуру в виде массива, каждая запись которого представляет собой указатель на цепочку записей, содержащих:

- значение элемента
- ссылку (например, индекс записи в массиве)
- ссылку на следующую запись цепочки

ИДЗ

Горбунова Дарья

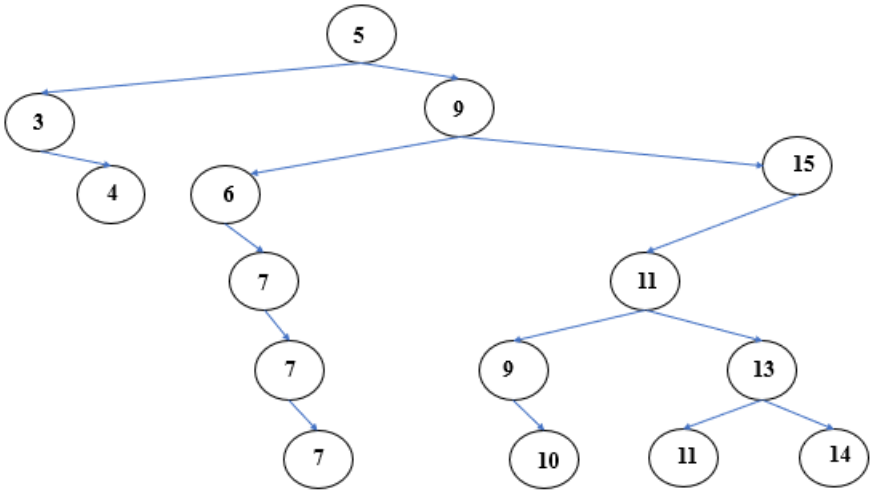
Вариант 5

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

Метод двоичного поиска:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
1	3	4	5	6	7	7	7	9	9	10	11	11	13	14	15
2	L														R
3								M							
4	L							R							
5				M											
6				L				R							
7						M									
8						L		R							
9							M								

Двоичное дерево для массива: 5 9 15 6 11 7 13 14 9 3 11 7 7 4 10



Найдем число 11:

11 > 5, тогда спускаемся в право;

11 > 9, тогда спускаемся ниже в право;

11 < 15, тогда спускаемся ниже влево;

11 = 11, число найдено.

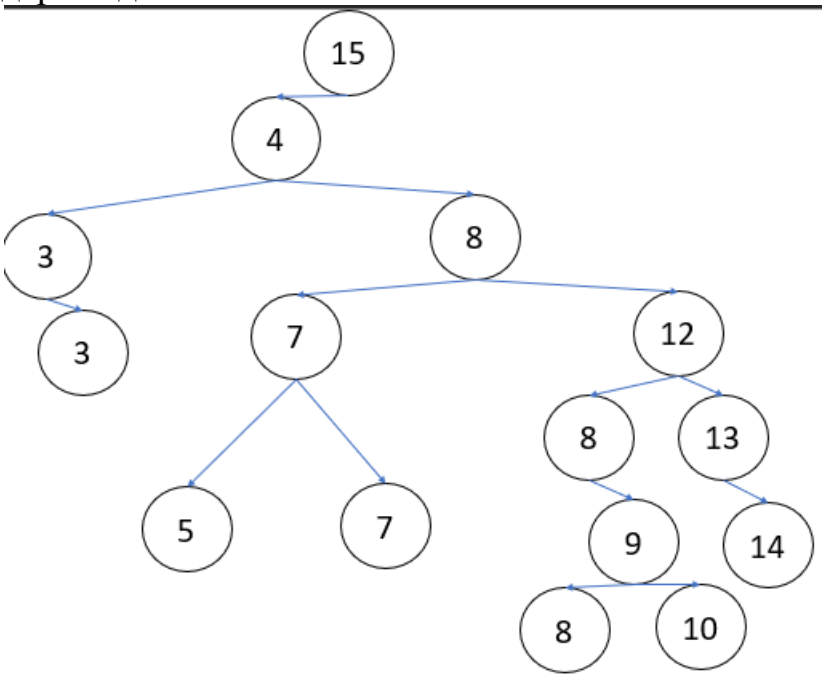
Токарева Ульяна
Вариант 17

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

Метод двоичного поиска:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
1	3	3	4	5	7	7	8	8	8	9	10	12	13	14	15
2	L														R
3								M							
4	L							R							
5				M											
6				L				R							
7						M									
8						L		R							
9							M								

Двоичное дерево для массива: 15 4 8 12 3 7 8 3 9 13 10 7 5 14 8



Найдем число 10:

10 < 15, тогда спускаемся влево;
10 > 4, тогда спускаемся ниже вправо;
10 > 8, тогда спускаемся ниже вправо;
10 < 12, тогда спускаемся влево;
10 > 8, тогда спускаемся ниже вправо;
10 > 9, тогда спускаемся ниже вправо;
10 = 10, число найдено.

Листинг дополненного алгоритма лабораторной работы №1

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/time.h>
#include <time.h>

int binarysearch(int a, long long int mass [], int n)
{int low, high, middle;
    low = 0;
    high = n - 1;
    while (low <= high)
    {middle = (low + high) / 2;
        if (a < mass[middle])
            high = middle - 1;
        else if (a > mass[middle])
            low = middle + 1;
        else
            return middle;
    } return -1;}
int main () {
    long long int* mass;
    long long int n,l,i,j,min,number,k,b;
    size_t t;
```

```
struct timeval t1,t2;
n = 10000;
l = 10000;
mass = (long long int*)malloc(l * sizeof(mass));
if (mass == NULL)
{printf("Error!\n");
    return 1;}
t = time(NULL);
if (t == -1)
{printf("Error!\n");
    return 1;}
srand(t);
for (i = 0; i < l; i ++) mass[i] = rand()%n;
printf("Unsorted array\n");
for (i = 0; i < 10; i ++) printf("%d\n", mass[i]);
printf("...");
for (i = l - 10; i < l; i ++) printf("%d\n", mass[i]);
printf("\n");
if (gettimeofday(&t1,NULL)==-1) {
    printf("Error t1");
    return 1;}
for (i = 0; i < l - 1; i ++)
{min = mass[i];
```

```
        for (j = i + 1; j < l; j ++)
        {if (mass[j] <= min)
            {min = mass[j];
                number = j;}}
        mass[number] = mass[i];
        mass[i] = min;}
    if (gettimeofday(&t2,NULL)==-1)
    {printf("Error t2");
        return 1;}
    printf("Sorted array\n");
    for (i = 0; i < 10; i ++) printf("%d\n", mass[i]);
    printf("...");
    for (i = l - 10; i < l; i ++) printf("%d\n", mass[i]);
    printf("\n");
    printf("\nmilli second algorithm: %lld \n",((t2.tv_sec *1000000 + t2.tv_usec) - (t1.tv_sec * 1000000 +
t1.tv_usec))/1000);
    printf("What number should I find: ");
    scanf("%d", &k);
    b=binarysearch( k, mass, l);
    if (b==-1) printf("NO RESULT");
    else printf("\nPosition number: %d",b);
    return 0;}
```

Исходный код BinTree

```
#include <cstdio>
#include <iostream>
#include <list>
using namespace std;
struct tree{
    float price;
    int article, fabricator;
    tree *left;
    tree *right;
    int articles [1000] = {0};
    size_t way;};
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; }
    if (myNode-> price > containingNode->price){
        if(containingNode->right!= nullptr){
            add(containingNode->right, myNode);
        }else{
            containingNode->right = myNode;
            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->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 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);
    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;
}}
```

Результаты работы программы. Статистика бинарного дерева

```
Statistic: 0    Way: 1                  Statistic: 1    Way: 2
Statistic: 2    Way: 4                  Statistic: 3    Way: 8
Statistic: 4    Way: 15                 Statistic: 5    Way: 30
Statistic: 6    Way: 57                 Statistic: 7    Way: 103
Statistic: 8    Way: 187                        Statistic: 9    Way: 315
Statistic: 10   Way: 505                        Statistic: 11   Way: 826
Statistic: 12   Way: 1253                       Statistic: 13   Way: 1839
Statistic: 14   Way: 2546                       Statistic: 15   Way: 3393
Statistic: 16   Way: 4145                       Statistic: 17   Way: 4832
Statistic: 18   Way: 5304                       Statistic: 19   Way: 5403
Statistic: 20   Way: 5202                       Statistic: 21   Way: 4771
Statistic: 22   Way: 4140                       Statistic: 23   Way: 3439
Statistic: 24   Way: 2738                       Statistic: 25   Way: 2011
Statistic: 26   Way: 1516                       Statistic: 27   Way: 1061
Statistic: 28   Way: 694                        Statistic: 29   Way: 449
Statistic: 30   Way: 286                        Statistic: 31   Way: 172
Statistic: 32   Way: 103                        Statistic: 33   Way: 55
Statistic: 34   Way: 29                 Statistic: 35   Way: 13
Statistic: 36   Way: 9                  Statistic: 37   Way: 3
```

Рисунок 3 – Табличная статистика бинарного дерева

Рисунок 4 – Статистика частот длин путей до узлов двоичного дерева

Исходный код Hash-Table

```
#include <iostream>
#include <fstream>
using namespace std;
#define Key 35129
#define SIZE  100000
struct HT_Param { ;
    int article;
    int fabricator;
    float price;
};
struct HT {
    int volume;
    HT_Param *parameters;
};
int Flag (int x){return x % Key;}
void nul_HT (HT * hash_table){
    for (int i=0; i< Key; i++){
        hash_table[i].volume=0;
        hash_table[i].parameters = nullptr;
    }
}
void create_HT_Param (HT_Param * curr_table, int curr_article, int curr_fabricator, float curr_price ){
```

```
    curr_table -> article = curr_article;
    curr_table -> price = curr_price;
    curr_table -> fabricator = curr_fabricator;
}
void add_HT (HT * new_table, HT_Param new_param){
    size_t flag = Flag (new_param.article);
    if (new_table[flag].parameters == nullptr){
        new_table[flag].parameters = (HT_Param *) malloc(sizeof (HT_Param));
        new_table[flag]. parameters[0] = new_param;
        new_table[flag].volume = 1; }
    else {
        new_table[flag].volume +=1;
        new_table[flag].parameters = (HT_Param *) realloc (new_table[flag].parameters, sizeof (HT_Param) *
(new_table[flag].volume));
        new_table[flag].parameters[new_table[flag].volume-1]=new_param;}}
void search(HT * ht, size_t user_article){
    int flag_user_article = Flag(user_article);
    if (ht[flag_user_article].parameters == nullptr){
        cout << "No elements";
        return;}
    else {
        for(int i=0;i < ht[flag_user_article].volume; i++){
            if (ht[flag_user_article].parameters[i].article == user_article){
```

```
                cout << "Price: " << ht[flag_user_article].parameters[i].price << endl;
                cout << "Fabricator: "<< ht[flag_user_article].parameters[i].fabricator << endl;
                return;}}}
    cout << "No element";}
void row_len (HT * ht, int * arr){
    int border = (SIZE / Key) + 5;
    for (int i=0; i < border; i++)
        arr[i] = 0;
    for (int i=0; i < Key; i++){
        if (ht[i].parameters != nullptr)
            arr[ht[i].volume]+=1;
        else arr[0]+=1;}}
int main() {
    HT * Hash_Table;
    ifstream file;
    file.open("ads_lab2.txt");
    int user_article, article, fabricator;
    int array[SIZE];
    float price;
    Hash_Table = (HT *) malloc(sizeof (HT)*Key);
    nul_HT(Hash_Table);
    for (int i=0; i< SIZE; i++){
        file >> article;
```

```
    file >> price;
    file >> fabricator;
    HT_Param new_param;
    create_HT_Param(&new_param, article, fabricator, price);
    add_HT(Hash_Table, new_param);}
cin >> user_article;
search (Hash_Table, user_article);
row_len(Hash_Table,array);
cout << endl;
for (int i=0; i<10; i++)
    if (array[i]!=0)
        cout << i<<" "<< array [i] << endl;
delete(Hash_Table);
return 0;}
```

Результаты работы программы. Статистика хэш-таблицы.

Таблица 1. Табличная статистика частот длин цепочек за ячейками хэш-таблицы

Длина цепочек	Количество
0	28
1	719
2	6987
3	24273
4	3122

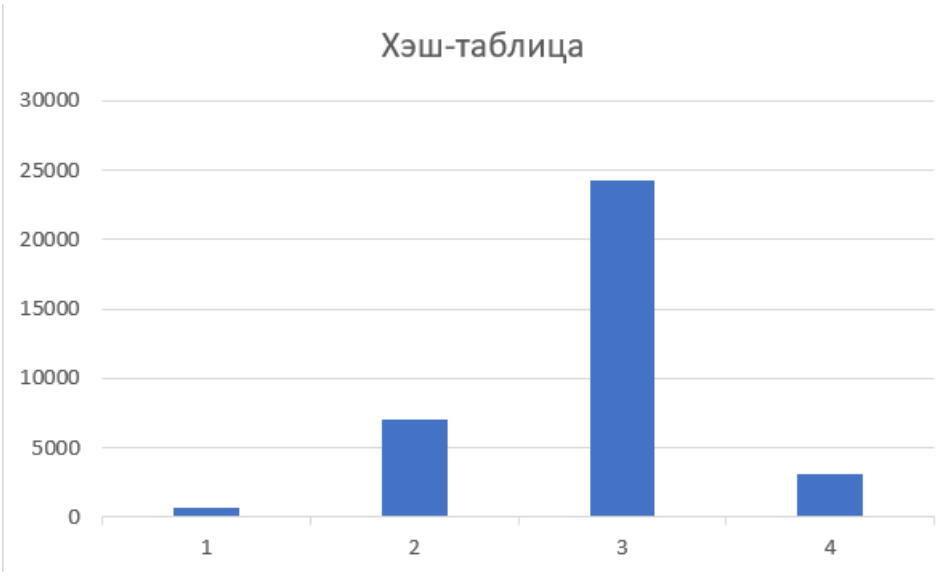


Рисунок 5 – Статистика частот длин цепочек за ячейками хэш-таблицы