Министерство науки и высшего образования Российской Федерации

Пензенский государственный университет

Кафедра «Вычислительная техника»

**ОТЧЕТ**

по лабораторной работе №4

по курсу «Логика и основы алгоритмизации в инженерных задачах»

на тему «Бинарное дерево поиска»

Выполнили:

студенты группы 22BВВ1

Митрошин Ю.Е.

Коннов А.Д.

Приняли:

Акифьев И.В.

Юрова О.В.

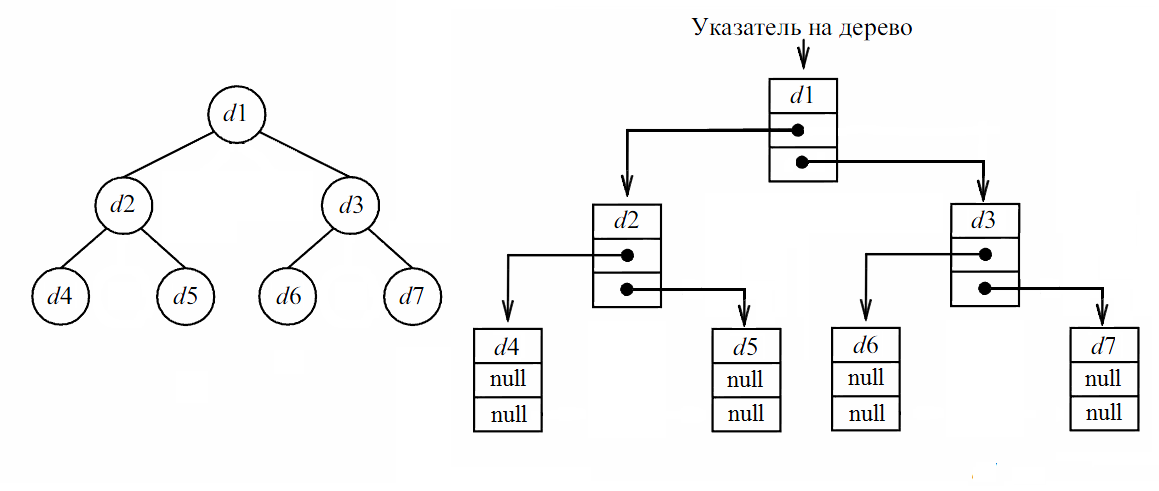
Пенза 2023

**Название**

Бинарное дерево поиска

**Общие сведения.**

Бинарные деревья – это деревья, у каждого узла которого возможно наличие только двух сыновей. Двоичные деревья являются упорядоченными.



Двоичное дерево можно представить в виде нелинейного связанного списка.

Бинарное дерево поиска — это бинарное дерево, обладающее дополнительными свойствами:

– значение левого потомка меньше значения родителя;

– значение правого потомка больше значения родителя.

Такие структуры используются для сохранения данных в отсортированном виде.

В простейшем случае для создания элемента списка используется структура, в которой объединяются полезная информация и ссылка на следующий элемент списка:

struct Node {

int data;

struct Node \*left;

structNode \*right;

};

В качестве информации в дереве хранятся целые числа.

Обращение к дереву и его элементам осуществляется посредством указателей:

structNode \*root;

Так как деревья по своей сути являются рекурсивными структурами данных, то и большинство функций, работающих с деревьями, рекурсивны.

Функция создания дерева выделяет память под каждый новый элемент и добавляет его в дерево:

struct Node \*CreateTree(struct Node \*root, struct Node \*r, int data)

{

if (r == NULL)

{

r = (struct Node \*)malloc(sizeof(struct Node));

if (r == NULL)

{

printf("Ошибкавыделенияпамяти");

exit(0);

}

r->left = NULL;

r->right = NULL;

r->data = data;

if (root == NULL) return r;

if (data > root->data) root->left = r;

else root->right = r;

return r;

}

if (data > r->data)

CreateTree(r, r->left, data);

else

CreateTree(r, r->right, data);

returnroot;

}

Функция вывода дерева на экран (дерево выводится повёрнутым на 90 градусов, корень находится слева):

void print\_tree(struct Node \*r, int l)

{

if (r == NULL)

{

return;

}

print\_tree(r->right, l + 1);

for(inti = 0; i< l; i++)

{

printf(" ");

}

printf("%d\n", r->data);

print\_tree(r->left,  l+1);

}

Программа, использующая приведенные функции:

intmain()

{

setlocale(LC\_ALL, "");

int D, start = 1;

root = NULL;

printf("-1 - окончание построения дерева\n");

while (start)

{

printf("Введитечисло: ");

scanf\_s("%d", &D);

if (D == -1)

{

printf("Построение дерева окончено\n\n");

start = 0;

}

else

root = CreateTree(root, root, D);

}

print\_tree(root,0);

scanf\_s("%d", &D);

return 0;

}

**Задание**

1. Реализовать алгоритм поиска вводимого с клавиатуры значения в уже созданном дереве.
2. Реализовать функцию подсчёта числа вхождений заданного элемента в дерево.
3. Оценили сложность процедуры поиска по значению в бинарном дереве.

**Cложность в бинарном дереве зависит от того, сбалансировано ли оно. Если дерево сбалансировано, то сложность поиска – O(log n). При поиске элемента искомое значение сравнивается с корнем. Если искомое больше корня, то поиск продолжается в правом потомке корня, если меньше, то в левом.**

**Листинг**

class TreeNode:

def \_\_init\_\_(self, key):

self.left = None

self.right = None

self.val = key

definsert(root, key):

if root is None:

return TreeNode(key)

else:

if root.val< key:

root.right = insert(root.right, key)

else:

root.left = insert(root.left, key)

return root

defsearch(root, key):

if root is None or root.val == key:

return root

if root.val< key:

return search(root.right, key)

return search(root.left, key)

defcount\_occurrences(root, key):

if root is None:

return 0

count = 0

if root.val == key:

count += 1

count += count\_occurrences(root.left, key)

count += count\_occurrences(root.right, key)

return count

defprint\_tree(root, level=0):

if root is not None:

print\_tree(root.right, level + 1)

print(" " \* (level \* 4) + str(root.val))

print\_tree(root.left, level + 1)

root = None

root = insert(root, 50)

while(True):

key = int(input("Введите значение элемента дерева: "))

if key == 0:

break

else:

insert(root, key)

search\_value = int(input("Введите значение для поиска: "))

result = search(root, search\_value)

occurrences = count\_occurrences(root, search\_value)

if result:

print(f"Значение {search\_value} найдено в дереве.")

else:

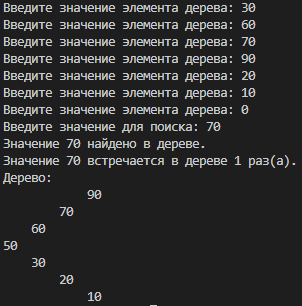
print(f"Значение {search\_value} не найдено в дереве.")

print(f"Значение {search\_value} встречается в дереве {occurrences} раз(а).")

print("Дерево:")

print\_tree(root)

**Результат работы программы:**



**Вывод:** в ходе лабораторной работы мы научились создавать бинарные деревья и работать с ними, оценивать сложность процедуры поиска по значению в бинарном дереве.