МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение

высшего образования

"МИРЭА - Российский технологический университет"

РТУ МИРЭА

Институт Информационных Технологий

Кафедра Вычислительной Техники

Отчет по выполнению практического задания №2

Тема. Нелинейные структуры данных. Бинарное дерево

Дисциплина Структуры и алгоритмы обработки данных Часть 2

Студент группы: ИКБО-04-20 Нгуен Ван Мань

(Фамилия студента)

Преподаватель Сорокин А.В.

(Фамилия преподавателя)

Москва 2021

СОДЕРЖАНИЕ

Тема: Нелинейные структуры данных. Бинарное дерево..

Цель: Получение умений и навыков разработки и реализаций операций над структурой данных бинарное дерево.

Задание

Для вариантов с 1 по 7

Разработать программу, которая создает идеально сбалансированное дерево из n узлов и выполняет операции.

1. Реализовать операции общие для всех вариантов

   1) Создать идеально сбалансированное бинарное дерево из n узлов. Структура узла дерева включает: информационная часть узла, указатель на левое и указатель на правое поддерево. Информационная часть узла определена вариантом.

   2) Отобразить дерево на экране, повернув его справа налево.

2. Реализовать операции варианта.

3. Разработать программу на основе меню, позволяющего проверить выполнение всех операций на ваших тестах и тестах преподавателя.

4. Оформить отчет.

   1) Для каждой представленной в программе функции предоставить отчет по ее разработке в соответствии с требованиями разработки программы (подпрограммы).

   2) Представить алгоритм основной программы и таблицу имен, используемых в алгоритме.

Вариант:

2	Целое число	Определить количество листьев с положительными значениями Определить, сколько узлов дерева содержат заданное число. Увеличить значения узлов вдвое, обходя дерево алгоритмом в ширину.

Разработка программы

```
#include<iostream>
#include<queue>
using namespace std;

struct Node {
	int data;
	Node* left;
	Node* right;
};

Node* newNode(int data) {
	struct Node* tmp = new Node();
	tmp->data = data;
	tmp->left = NULL;
	tmp->right = NULL;
	return tmp;
}


//insert 1 node into the tree
Node* insertNode(Node* root, int data) {
	if (root == NULL)
		return newNode(data);

	/* Otherwise, recur down the tree */
	if (data < root->data)
		root->left = insertNode(root->left, data);
	else
		root->right = insertNode(root->right, data);

	/* return the (unchanged) node pointer */
	return root;
}


//.determine the height of the tree
int height(Node* node) {
	if (node == NULL) {
		return 0;
	}
	else {
		int left;
		int right;
		left = height(node->left);
		right = height(node->right);
		if (left > right) {
			return left + 1;
		}
		else {
			return right + 1;
		}
	}
}


//determine the number of elements in the tree
int numberOfNode(Node* node) {
	int count = 1;
	if (node == NULL) {
		return 0;
	}
	else {
```

```
                count += numberOfNode(node->left);
                count += numberOfNode(node->right);
        }
        return count;
}



void printNode(Node* node) {
        if (node == NULL) {
                return;
        }
        else {
                if (height(node->left) == height(node->right) && numberOfNode(node->left) ==
numberOfNode(node->right)) {
                        cout << node->data;
                }
                else {
                        printNode(node->right);
                        printNode(node->left);
                }
        }
}


// print the number of leaves with a positive value
int numberOfPositiveLeaves(Node* node) {
        int count = 0;
        if (node == NULL) {
                return 0;
        }
        else {
                if (node->left != NULL || node->right != NULL) {
                        count += numberOfPositiveLeaves(node->left);
                        count += numberOfPositiveLeaves(node->right);
                }
                else if (node->left == NULL && node->right == NULL && node->data > 0) {
                        count++;
                }
        }
        return count;
}



//print the number of buttons with the same value
int notesOfTheSameValue(Node* node,int data) {
        int count = 0;
        if (node == NULL) {
                return 0;
        }
        else {
                if (node->data != data) {
                        count += notesOfTheSameValue(node->left,data);
                        count += notesOfTheSameValue(node->right,data);
                }
                else if (node->data == data) {
                        count += notesOfTheSameValue(node->left, data);
                        count += notesOfTheSameValue(node->right, data);
                        count++;
                }
        }
        return count;
```

```
}



//print values traverse from right to left
void printRightToLeft(Node* node) {
	if (node != NULL) {
		cout << node->data << " ";
		printRightToLeft(node->right);
		printRightToLeft(node->left);
	}
}



void printTree(Node* root, int level) {
	if (root) {
		printTree(root->right, level + 1);
		for (int i = 0; i < level; ++i)
			cout << "    ";
		cout << root->data << endl;
		printTree(root->left, level + 1);
	}
}


//double the value of nodes, traverse the tree by width
void printLevelOrder(Node* root) {
	if (root == NULL) return;
	queue<Node*> bf_queue;
	bf_queue.push(root);
	/* The loop ends when the queue is empty */
	while (!bf_queue.empty()) {
		Node* current = bf_queue.front();
		bf_queue.pop(); /* Get the first element out of the queue */
		current->data = (current->data) * 2;
		cout << current->data << ", ";
		/* Enqueue the left and right children of current into bf_queue.*/
		if (current->left != NULL) {
			bf_queue.push(current->left);
		}
		if (current->right != NULL) {
			bf_queue.push(current->right);
		}
	}
}

int main() {
	Node* root = NULL;
	cout << "--------------------Menu--------------------\n";
	cout << "[1] - Insert a node.\n";
	cout << "[2] - Print the number of leaves with a positive value.\n";
	cout << "[3] - Print a tree.\n";
	cout << "[4] - Calculate the number of nodes with the same value.\n";
	cout << "[5] - Print the tree after increasing the value of the traversal nodes by
width.\n";
	cout << "[0] - Exit program!\n";
	cout << "--------------------------------------------\n";

	int choice = 0;
	do
	{
		cout << "Input your choice: ";
		cin >> choice;
```

```
		cout << "\n--------------------------------------\n";
		switch (choice)
		{
		case 1:
			cout << "Enter size of tree: ";
			int n;
			cin >> n;
			cout << "Enter the data to insert into the tree: ";
			for (int i = 0; i < n; ++i) {
				int data;
				cin >> data;
				root = insertNode(root, data);
			}
			cout << "Done!";
			break;
		case 2:
			cout << "Number of leaves: " << numberOfPositiveLeaves(root);
			break;

		case 3:
			cout << "Tree: \n";
			printTree(root, 0);
			cout << "print Right to Left: ";
			printRightToLeft(root);
			break;
		case 4:
			cout << "Enter the value n: ";
			cin >> n ;
			cout << endl << "Number of nodes with the same value: "  <<
notesOfTheSameValue(root, n) <<endl;
			break;
		case 5:
			cout << "traverse the tree by width: ";
			printLevelOrder(root);
		case 0:
			cout << endl << "Thanks for using program!";
			break;
		default:
			break;
		}
		cout << "\n--------------------------------------\n";
	} while (choice != 0);

	system("pause");
	return 0;
                                        }
```

Тесты

Создать идеально сбалансированное бинарное дерево из n узлов.
Структура узла дерева включает: информационная часть узла, указатель на левое и указатель на правое поддерево. Информационная часть узла определена вариантом.

Отобразить дерево на экране, повернув его справа налево.

```
C:\Users\taoth\source\repos\cau_truc_du_lieu_va_giai_thuat_3\Debug\cau_truc_du_lieu_va_giai_thuat_3.exe
--------------------Menu--------------------
[1] - Insert a node.
[2] - Print the number of leaves with a positive value.
[3] - Print a tree.
[4] - Calculate the number of nodes with the same value.
[5] - Print the tree after increasing the value of the traversal nodes by width.
[0] - Exit program!
----------------------------------------
Input your choice: 1

----------------------------------------
Enter size of tree: 7
Enter the data to insert into the tree: 4
2
2
-1
6
5
7
Done!
----------------------------------------
Input your choice: 3

----------------------------------------
Tree:
        7
    6
        5
4
        2
    2
        -1
print Right to Left: 4 6 7 5 2 2 -1
```

Определить количество листьев с положительными значениями

```
Tree:
        7
    6
        5
4
        2
    2
        -1
print Right to Left: 4 6 7 5 2 2 -1
----------------------------------------
Input your choice: 2

----------------------------------------
Number of leaves: 3
```

Определить, сколько узлов дерева содержат заданное число.

```
Input your choice: 4

---------------------------------------
Enter the value n: 2

Number of nodes with the same value: 2
```

Увеличить значения узлов вдвое, обходя дерево алгоритмом в ширину.

```
Input your choice: 5

---------------------------------------
traverse the tree by width: 8, 4, 12, -2, 4, 10, 14,
---------------------------------------
Input your choice: 3

---------------------------------------
Tree:
        14
    12
        10
8
        4
    4
        -2
print Right to Left: 8 12 14 10 4 4 -2
---------------------------------------
Input your choice: 0

---------------------------------------

Thanks for using program!
---------------------------------------
Press any key to continue . . .
```

Выводы

Проведя работу по созданию, выводу и использованию идеально сбалансированного дерева были получены знания и умения по реализации данного вида бинарного дерева.

Ответы на вопросы.

1. Что определяет степень дерева?

   Степень дерева определяет максимальную степень его узлов.

2. Какова степень сильноветвящегося дерева?

   Степень сильноветвящегося дерева больше 2.

3. Что определяет путь в дереве?

   Путь в дереве определяет последовательность узлов от корня до нужного узла.

4. Как рассчитать длину пути в дереве?

   Чтобы рассчитать длину пути в дереве нужно посчитать сумму длин его ребер. (Длина пути дерева определяется как сумма длин путей ко всем его вершинам.)

5. Какова степень бинарного дерева?

   Степень бинарного дерева равна 2.

6. Может ли дерево быть пустым?

   Дерево называется пустым, если оно не содержит ни одной вершины.

7. Дайте определение бинарного дерева?

   Бинарное дерево - это дерево у каждого узла которого не более 2 потомков.

8. Дайте определение алгоритму обхода.

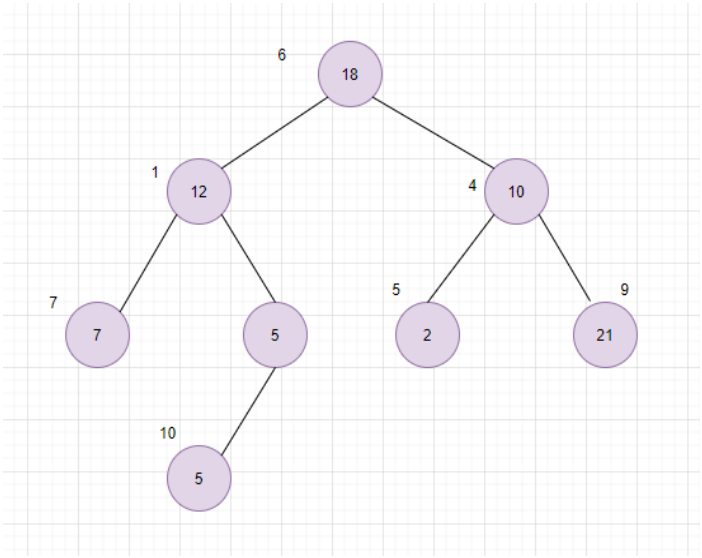
   Обход дерева – это упорядоченная последовательность вершин дерева, в которой каждая вершина встречается только один раз.

9. Приведите рекуррентную зависимость для вычисления высоты дерева.

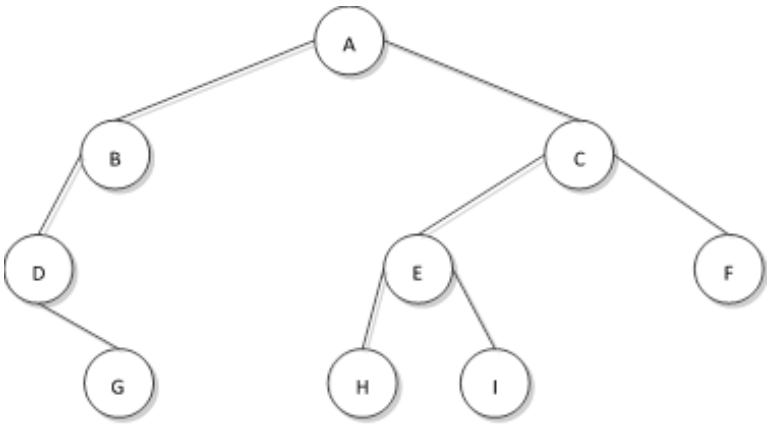
$$h(T) = \begin{cases} -1, & \text{если } T = NULL \\ 1 + max(h(T.left), h(T.right)), & \text{иначе} \end{cases}$$

10. Изобразите бинарное дерево, корень которого имеет индекс 6, и которое представлено в памяти таблицей вида:

Индекс	Key	left	right
1	12	7	3
2	15	8	NULL
3	4	10	NULL
4	10	5	9
5	2	NULL	NULL
6	18	1	4
7	7	NULL	NULL
8	14	6	2
9	21	NULL	NULL
10	5	NULL	NULL



11. Укажите путь обхода дерева по алгоритму: прямой; обратный; симметричный



Путь обхода дерева в прямом порядке: ABDGCEHIF.

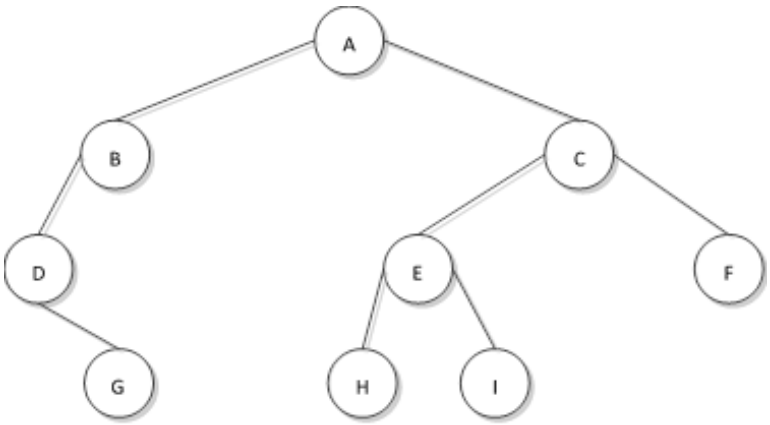
Путь обхода дерева в обратном порядке: GDBHIEFCA.

Путь обхода дерева в симметричном порядке: DGBAHEICF.

12. Какая структура используется в алгоритме обхода дерева методом в «ширину»?

В алгоритме обхода дерева методом в «ширину» используется очередь.

13. Выведите путь при обходе дерева в «ширину». Продемонстрируйте использование структуры при обходе дерева.



Путь обхода дерева в «ширину»: ABCDEFGHI.

Шаг 1. Очередь: A.

Шаг 2. Очередь: BC. Вывод: A.

Шаг 3. Очередь: CD. Вывод: B.

Шаг 4. Очередь: DEF. Вывод: C.

Шаг 5. Очередь: EFG. Вывод: D.

Шаг 6. Очередь: FGHI. Вывод: E.

Шаг 7. Очередь: GHI. Вывод: F.

Шаг 8. Очередь: HI. Вывод: G.

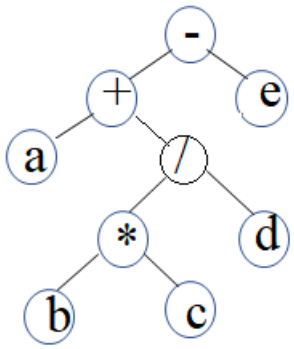
Шаг 9. Очередь: I. Вывод: H.

Шаг 10. Очередь: пусто. Вывод: I.

14. Какая структура используется в не рекурсивном обходе дерева методом в «глубину»?

    В не рекурсивном обходе дерева методом в «глубину» используется стек.

15. Выполните прямой, симметричный, обратный методы обхода дерева выражений.



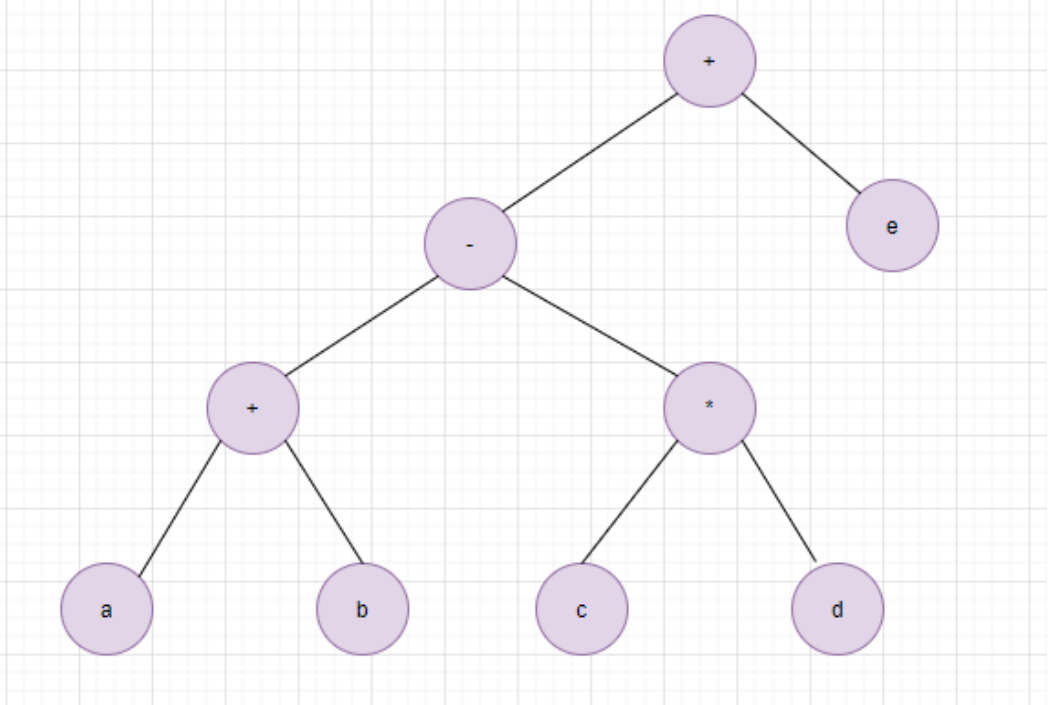
Путь обхода дерева в прямом порядке: -+a/*bcde.

Путь обхода дерева в симметричном порядке: a+b*c/d-e.

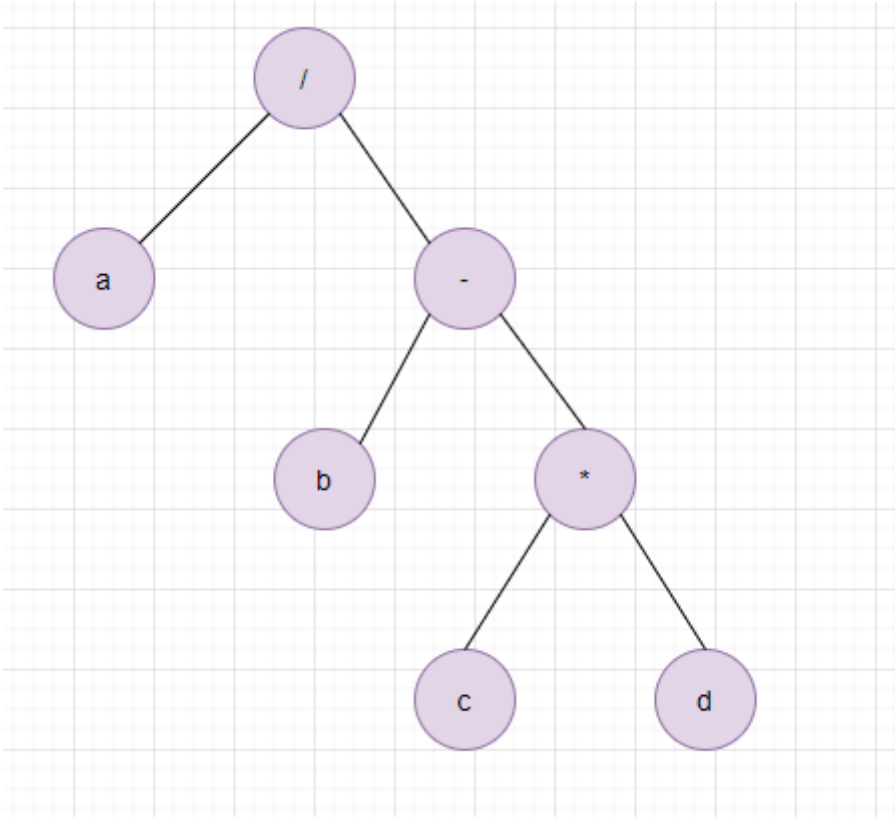
Путь обхода дерева в обратном порядке: abc*d/+e-.

16. Для каждого заданного арифметического выражения постройте бинарное дерево выражений:

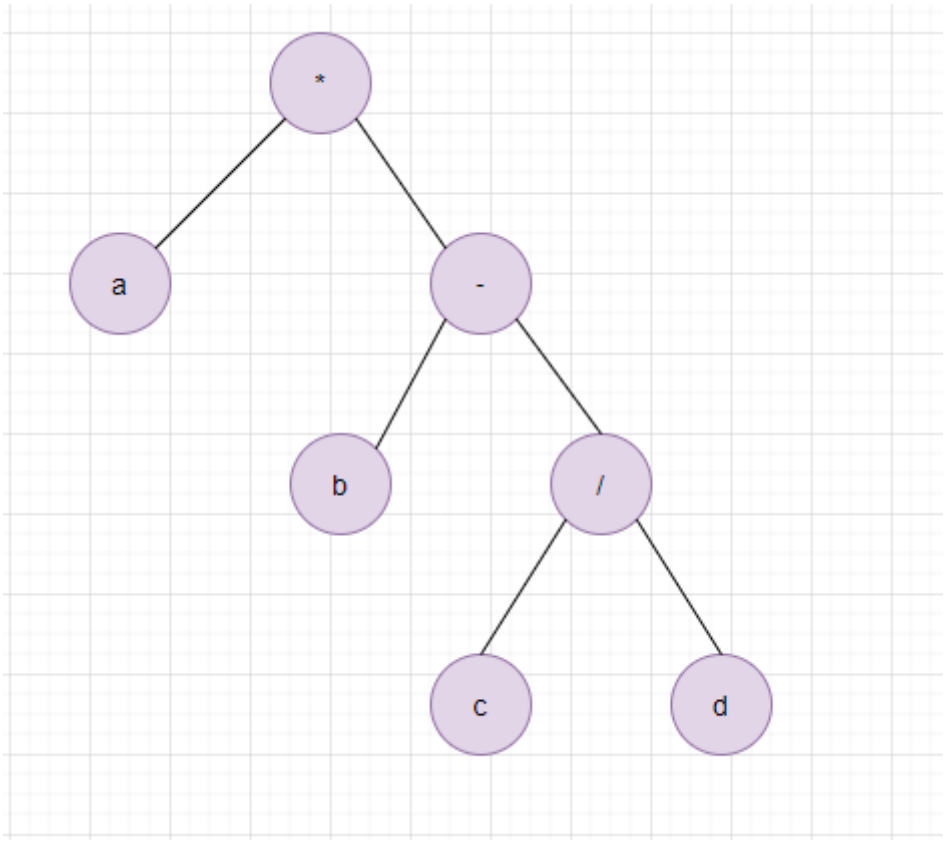
1. a+b-c*d+e



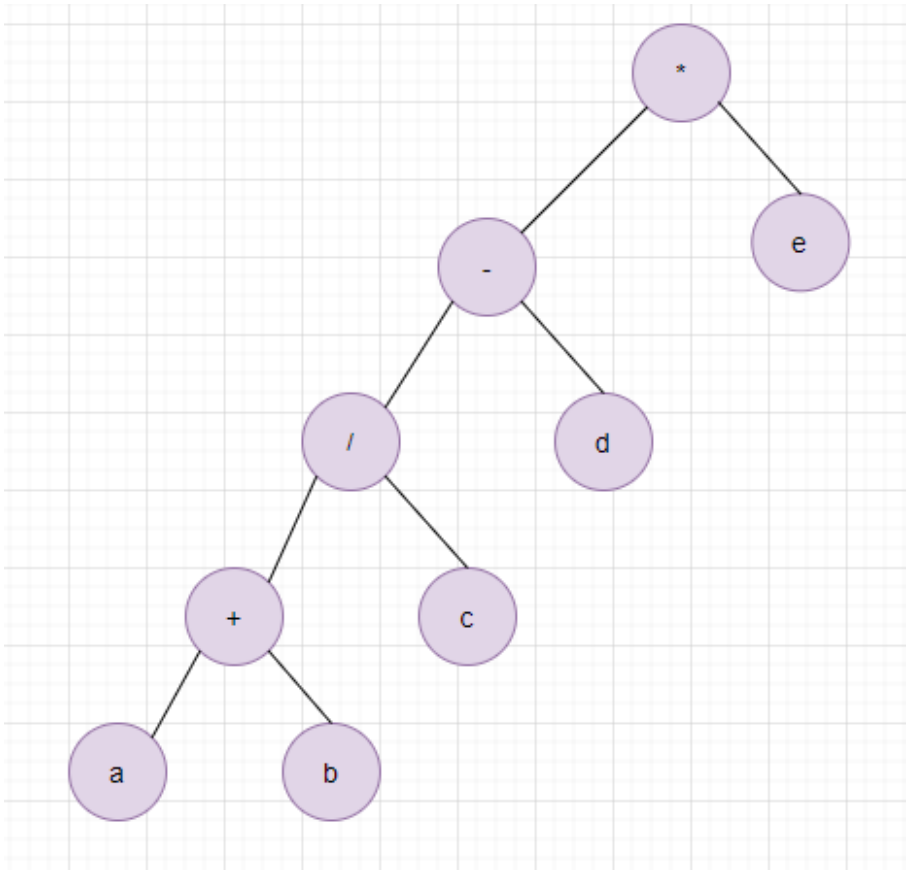
2. /a-b*c d



3. a b c d/ - *



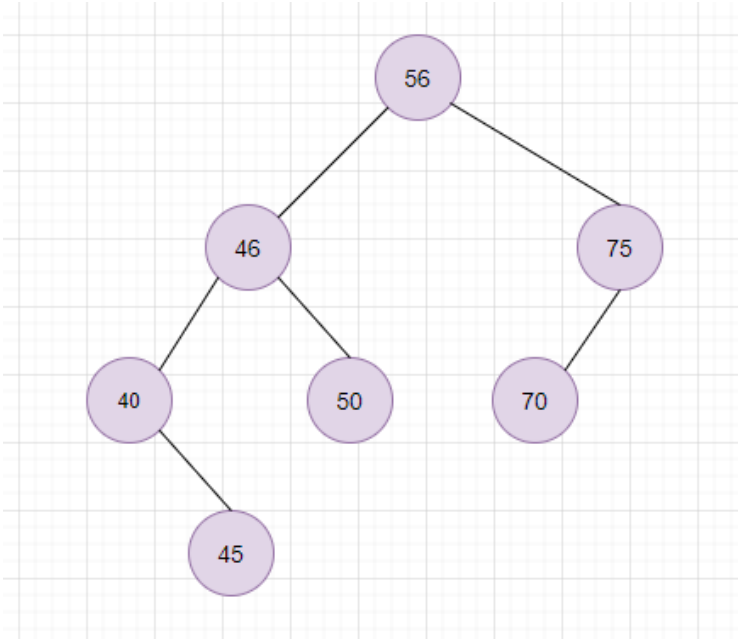
4. *-/+abcde



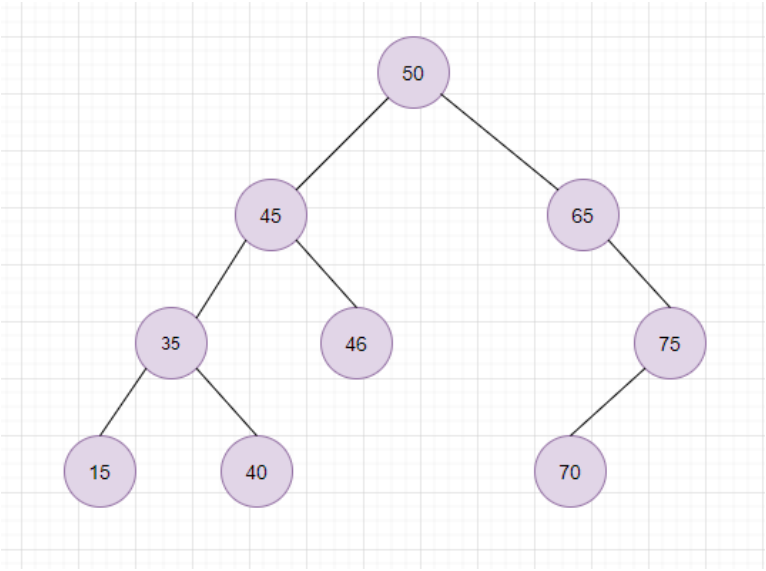
17. В каком порядке будет проходиться бинарное дерево, если алгоритм обхода в ширину будет запоминать узлы не в очереди, а в стеке?

    Если алгоритм обхода в ширину будет запоминать узлы в стеке, то бинарное дерево будет проходиться в прямом порядке.

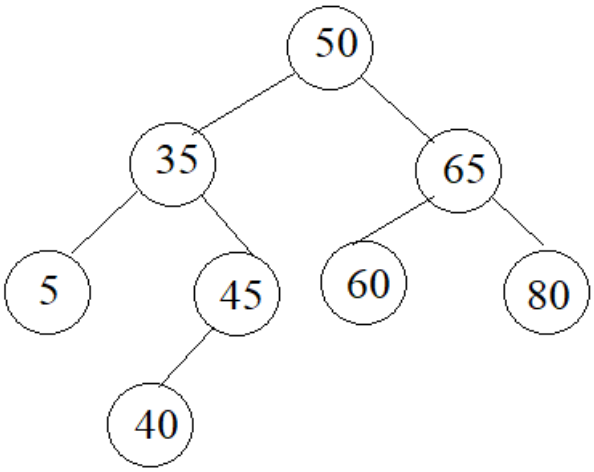
18. Постройте бинарное дерево поиска, которое в результате симметричного обхода дало бы следующую последовательность узлов: 40 45 46 50 65 70 75.



19. Последовательность {50 45 35 15 40 46 65 75 70} получена путем прямого обхода бинарного дерева поиска. Постройте это дерево.

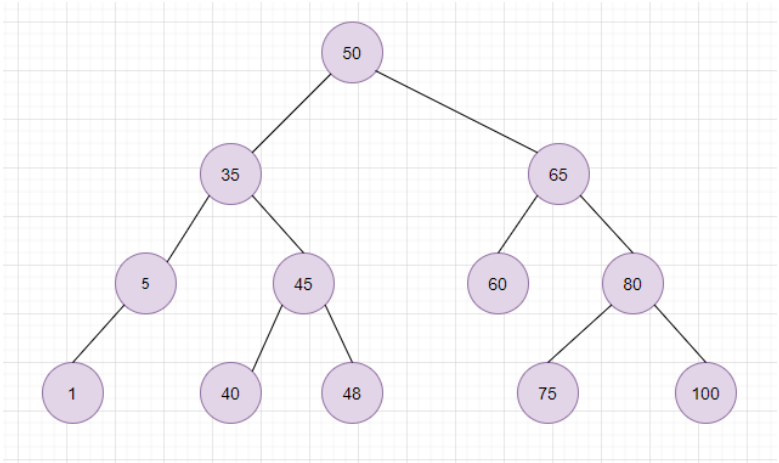


20.Дано бинарное дерево поиска.

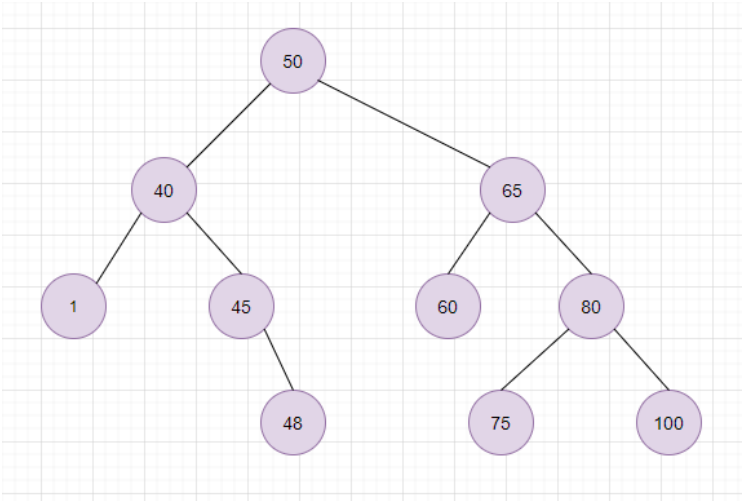


Выполните действия над исходным деревом и покажите дерево:

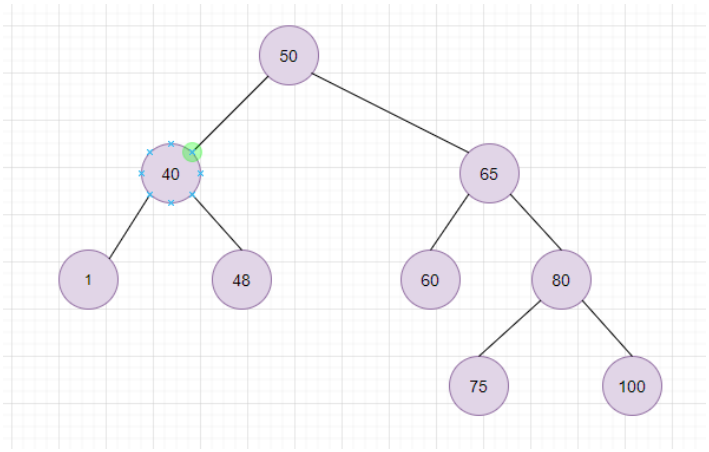
1) после включения узлов 1, 48, 75, 100



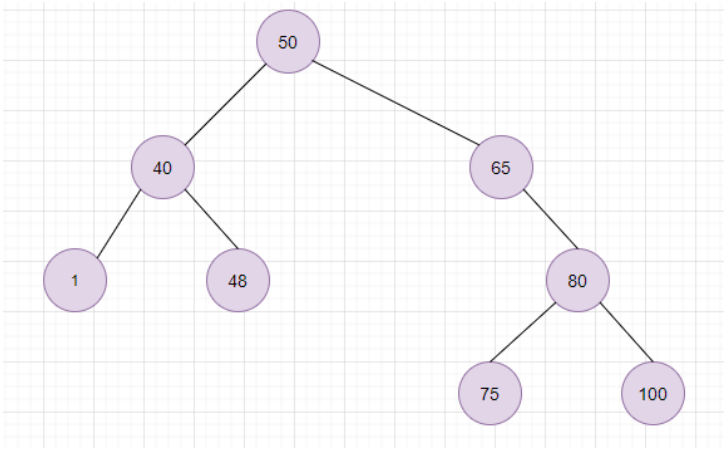
2) после удаления узлов 5, 35



3) после удаления узла 45



4) после удаления узла 50



5) после удаления узла 65 и вставки его снова