**Деревья на Python**

class TreeNode:  
 def \_\_init\_\_(self, value):  
 self.value = value  
 self.left = None  
 self.right = None  
  
def inorder\_traversal(node):  
 *"""Центрированный обход (левое поддерево, корень, правое поддерево)"""* if node:  
 inorder\_traversal(node.left)  
 print(node.value, end=' ')  
 inorder\_traversal(node.right)  
  
# Создание дерева  
root = TreeNode(1)  
root.left = TreeNode(2)  
root.right = TreeNode(3)  
root.left.left = TreeNode(4)  
root.left.right = TreeNode(5)  
  
print("Центрированный обход:")  
inorder\_traversal(root) # Вывод: 4 2 5 1 3

**Деревья на Java**

class TreeNode {

int value;

TreeNode left;

TreeNode right;

TreeNode(int value) {

this.value = value;

this.left = null;

this.right = null;

}

}

public class BinaryTree {

public static void inorderTraversal(TreeNode node) {

if (node != null) {

inorderTraversal(node.left);

System.out.print(node.value + " ");

inorderTraversal(node.right);

}

}

public static void main(String[] args) {

TreeNode root = new TreeNode(1);

root.left = new TreeNode(2);

root.right = new TreeNode(3);

root.left.left = new TreeNode(4);

root.left.right = new TreeNode(5);

System.out.println("Центрированный обход:");

inorderTraversal(root); // Вывод: 4 2 5 1 3

}

}

**Граф на Python**

from collections import deque  
  
class Graph:  
 def \_\_init\_\_(self):  
 self.adj\_list = {}  
  
 def add\_vertex(self, vertex):  
 if vertex not in self.adj\_list:  
 self.adj\_list[vertex] = []  
  
 def add\_edge(self, v1, v2):  
 # Для неориентированного графа добавляем связь в оба направления  
 self.adj\_list[v1].append(v2)  
 self.adj\_list[v2].append(v1)  
  
 def bfs(self, start\_vertex):  
 *"""Обход в ширину"""* visited = set()  
 queue = deque([start\_vertex])  
 visited.add(start\_vertex)  
  
 while queue:  
 current\_vertex = queue.popleft()  
 print(current\_vertex, end=' ')  
  
 for neighbor in self.adj\_list[current\_vertex]:  
 if neighbor not in visited:  
 visited.add(neighbor)  
 queue.append(neighbor)  
  
# Создание графа  
graph = Graph()  
for i in range(5):  
 graph.add\_vertex(i)  
  
edges = [(0, 1), (0, 2), (1, 3), (2, 4)]  
for edge in edges:  
 graph.add\_edge(edge[0], edge[1])  
  
print("BFS обход, начиная с вершины 0:")  
graph.bfs(0) # Вывод: 0 1 2 3 4

**Графы на Java**

import java.util.\*;

class Graph {

private int numVertices;

private ArrayList<ArrayList<Integer>> adjList;

public Graph(int vertices) {

this.numVertices = vertices;

adjList = new ArrayList<>(vertices);

for (int i = 0; i < vertices; i++) {

adjList.add(new ArrayList<>());

}

}

public void addEdge(int v1, int v2) {

adjList.get(v1).add(v2);

adjList.get(v2).add(v1);

}

public void bfs(int startVertex) {

boolean[] visited = new boolean[numVertices];

Queue<Integer> queue = new LinkedList<>();

visited[startVertex] = true;

queue.offer(startVertex);

while (!queue.isEmpty()) {

int currentVertex = queue.poll();

System.out.print(currentVertex + " ");

for (int neighbor : adjList.get(currentVertex)) {

if (!visited[neighbor]) {

visited[neighbor] = true;

queue.offer(neighbor);

}

}

}

}

public static void main(String[] args) {

Graph graph = new Graph(5);

graph.addEdge(0, 1);

graph.addEdge(0, 2);

graph.addEdge(1, 3);

graph.addEdge(2, 4);

System.out.println("BFS обход, начиная с вершины 0:");

graph.bfs(0); // Вывод: 0 1 2 3 4

}

}

1. **Установите необходимые расширения:**
   * **Python:** Отличная поддержка "из коробки". Расширение **Pylance** для подсветки синтаксиса и автодополнения.
   * **C++:** Установите расширение **C/C++** от Microsoft. Для компиляции и отладки настройте компилятор (например, **MinGW-w64** для Windows или **GCC** для Linux/Mac).
   * **Java:** Установите **Extension Pack for Java**, который включает в себя всё необходимое. Убедитесь, что у вас установлен **JDK**.
2. **Поэкспериментируйте с кодом:**
   * **Деревья:** Реализуйте разные виды обхода (прямой, обратный). Попробуйте написать алгоритм поиска элемента в бинарном дереве поиска (BST).
   * **Графы:** Реализуйте обход в глубину (DFS). Попробуйте добавить веса ребрам и реализовать простейший алгоритм Дейкстры для поиска кратчайшего пути.
3. **Используйте отладчик:** VS Code имеет мощный отладчик. Установите точки останова и пошагово пройдитесь по алгоритмам обхода, чтобы увидеть, как программа "путешествует" по дереву или графу. Это лучший способ понять логику работы.