**1. Деревья**

Основные понятия:

- Иерархия: Каждый узел имеет ровно одного родителя (кроме корневого узла), но может иметь много детей.

- Корень: Узел без родителей.

- Листья: Узлы без детей.

- Высота дерева: Количество уровней от корня до самого глубокого листа.

Типы деревьев:

- Двоичные деревья: Каждый узел имеет максимум два ребёнка.

- Бинарные деревья поиска (BST): Все элементы в левом поддереве меньше элемента узла, а в правом — больше.

- Полностью сбалансированное дерево: Высота дерева минимальна среди всех деревьев с таким количеством узлов.

Операции над деревьями:

- Обходы: Pre-order, In-order, Post-order.

- Вставка: Добавление новых узлов согласно правилам конкретного типа дерева.

- Удаление: Уничтожение узлов с учётом особенностей структуры дерева.

**2. Графы**

Основные понятия:

- Вершины: Точечные объекты.

- Ребра: Связи между вершинами.

- Степень вершины: Количество рёбер, инцидентных вершине.

Типы графов:

- Ориентированные: Рёбра имеют направление.

- Неориентированные: Нет направлений на рёбрах.

- Связные: Существует путь между любой парой вершин.

- Весовые: Каждая связь имеет числовой вес.

Способы представления:

- Матрица смежности: Таблица, показывающая наличие связей между парами вершин.

- Список смежности: Массивы/связные списки для хранения соседних вершин.

**Примеры:**

1. Python

#### Пример дерева:

class Node:

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

self.key = key

self.left = None

self.right = None

#### Пример графа:

graph = {

'A': {'B': 1, 'C': 3},

'B': {'A': 1, 'C': 2},

'C': {'A': 3, 'B': 2}

}

2. C++

#### Пример дерева:

struct TNode {

int Key;

TNode Left = nullptr;

TNode Right = nullptr;

};

#### Пример графа:

std::unordered\_map<char, std::unordered\_map<char, int>> graph = {

{'A', {{'B', 1}, {'C', 3}}},

{'B', {{'A', 1}, {'C', 2}}}

};

```

3. Java

#### Пример дерева:

class Node {

int key;

Node left;

Node right;

public Node(int key) {

this.key = key;

}

}

```

#### Пример графа:

```

Map<String, Map<String, Integer>> graph = new HashMap<>();

graph.put("A", Map.of("B", 1, "C", 3));

graph.put("B", Map.of("A", 1, "C", 2));

```

---

## Алгоритмы работы с деревьями и графами

### 1. Деревья

#### Рекурсивная вставка:

```python

def insert\_recursive(node, x):

if node is None:

return Node(x)

if x < node.key:

node.left = insert\_recursive(node.left, x)

elif x > node.key:

node.right = insert\_recursive(node.right, x)

return node

```

#### Итерация (нерекурсивная вставка):

```python

def insert\_iterative(x):

parent = None

node = root

while node is not None:

parent = node

if x < node.key:

node = node.left

else:

node = node.right

new\_node = Node(x)

if parent is None:

root = new\_node

elif x < parent.key:

parent.left = new\_node

else:

parent.right = new\_node

```

2. Алгоритм Дейкстры

Пример реализации на Python:

```python

from collections import defaultdict

import heapq

def dijkstra(graph, start):

distances = {vertex: float('inf') for vertex in graph}

distances[start] = 0

priority\_queue = [(0, start)]

while priority\_queue:

current\_dist, current\_vertex = heapq.heappop(priority\_queue)

if current\_dist > distances[current\_vertex]:

continue

for neighbor, weight in graph[current\_vertex].items():

dist = current\_dist + weight

if dist < distances[neighbor]:

distances[neighbor] = dist

heapq.heappush(priority\_queue, (dist, neighbor))

return distances

```

## Заключение

Использование деревьев и графов позволяет решать широкий спектр практических задач, включая обработку иерархических данных, управление сетями и анализ сложных взаимосвязей. Данные структуры необходимы при разработке программного обеспечения и анализе данных.