Graphs – Graph Representation

1. Introduction

Graph Representation refers to the way a graph is stored in computer memory.

Since a graph consists of **vertices and edges**, we need an efficient method to represent their relationships.

Choosing the right representation is important because it affects:

- Memory usage
- Traversal speed
- Algorithm efficiency

---

2. Why Graph Representation is Needed

Graph representation is required to:

- Perform graph traversals (BFS, DFS)
- Apply shortest path algorithms
- Store large networks efficiently
- Process relationships between nodes

Different problems require different representations.

---

3. Types of Graph Representation

There are **two main methods** to represent graphs:

1. **Adjacency Matrix**
2. **Adjacency List**

---

4. Adjacency Matrix

What is an Adjacency Matrix?

An **Adjacency Matrix** is a **2D array** of size V × V, where:

- V is the number of vertices
- Each cell indicates whether an edge exists

Representation Rule

```
matrix[i][j] = 1 → edge exists
matrix[i][j] = 0 → no edge
```

Example

Graph:

```
A —— B
|
C
```

Adjacency Matrix:

```
   A B C
A [ 0 1 1 ]
B [ 1 0 0 ]
C [ 1 0 0 ]
```

Advantages of Adjacency Matrix

- Easy to implement
- Fast edge lookup (O(1))
- Suitable for dense graphs

Limitations of Adjacency Matrix

- Uses large memory
- Wastes space for sparse graphs

- Not efficient for large graphs

---

5. Adjacency List

What is an Adjacency List?

An **Adjacency List** stores:

- Each vertex
- A list of all vertices connected to it

It is usually implemented using **arrays or linked lists**.

Example

Graph:

```
A —— B
|
C
```

Adjacency List:

```
A → B → C
B → A
C → A
```

Advantages of Adjacency List

- Memory efficient
- Suitable for sparse graphs
- Easy to traverse neighbors
- Most commonly used

Limitations of Adjacency List

- Edge lookup is slower than matrix
- Slightly complex to implement
- Not ideal for very dense graphs

---

6. Adjacency Matrix vs Adjacency List

Feature	Adjacency Matrix	Adjacency List
Memory Usage	High	Low
Edge Lookup	O(1)	O(V)
Traversal	Slower	Faster
Best For	Dense Graphs	Sparse Graphs
Implementation	Simple	Moderate

---

7. Representation for Directed Graphs

Adjacency Matrix

```
matrix[i][j] = 1 → edge from i to j
```

Adjacency List

```
i → j
```

Direction is explicitly stored.

---

8. Representation for Weighted Graphs

Adjacency Matrix

- Store weight instead of 1

```
matrix[i][j] = weight
```

Adjacency List

- Store pairs (vertex, weight)

---

9. Which Representation to Use?

Scenario	Best Choice
Dense graph	Adjacency Matrix
Sparse graph	Adjacency List
Fast edge check	Adjacency Matrix
Fast traversal	Adjacency List

---

10. Applications of Graph Representation

- Social networks
- Road maps
- Network routing
- Dependency graphs
- Scheduling problems

---

11. Summary

- Graphs need memory representation
- Two main methods: matrix and list
- Matrix is simple but memory-heavy
- List is efficient and widely used
- Choice depends on problem requirements
