Graph Representations: Adjacency Matrix vs. Adjacency List

Hussein ALarag

January 28, 2026

Adjacency Matrix vs. Adjacency List

1. Adjacency Matrix

An **Adjacency Matrix** is a 2D array (matrix) of size $V \times V$, where V is the number of vertices in a graph. Each cell (i, j) contains a binary value (1 or 0) or a weight, indicating whether an edge exists between vertex i and vertex j.

- **Pros:**
  - **Quick Edge Lookup:** Checking if an edge exists between two nodes takes $O(1)$ time.
  - **Simple Implementation:** It is easy to understand and code using a standard 2D array.
  - **Matrix Operations:** Allows for mathematical operations (like squaring the matrix) to find paths of specific lengths.
- **Cons:**
  - **Space Inefficient:** It always uses $O(V^2)$ space, regardless of the number of edges. This is wasteful for "sparse" graphs.
  - **Slow to Find Neighbors:** To find all nodes connected to vertex i, you must scan the entire row (V elements), taking $O(V)$ time.
- **Uses:**
  - Dense graphs where the number of edges is close to V^2.
  - Problems requiring frequent edge existence checks.

2. Adjacency List

An **Adjacency List** is an array of lists. The size of the array is equal to the number of vertices (V). Each index in the array represents a vertex and points to a linked list (or dynamic array) containing all the vertices it is connected to.

- **Pros:**
  - **Space Efficient:** It only stores existing edges, using $O(V + E)$ space. This is much better for sparse graphs.
  - **Fast Neighbor Traversal:** Finding all neighbors of a vertex is very fast, as you only iterate through its specific list.
- **Cons:**
  - **Slow Edge Lookup:** To check if an edge exists between vertex i and j, you have to search through the list of vertex i, which can take $O(V)$ in the worst case.

  - **Memory Overhead:** If implemented with pointers, there is a small overhead for storing pointer addresses.

- **Uses:**
  - Sparse graphs (e.g., social networks or maps).
  - Algorithms like BFS or DFS that require exploring neighbors.