Graphs

Vinay, Riddhi, Havish

February 05, 2024

1 Introduction

A graph is a data structure consisting of a finite set of vertices or nodes and a set of edges connecting these vertices. In this section, we will explore different graph representations along with their respective space and time complexities.

2 Graph Applications:

2.1 Map Coloring

In map coloring problems, the goal is to assign colors to regions on a map such that no two adjacent regions have the same color.

2.2 Route Planning

Graphs are extensively used in route planning algorithms.
For example, when seeking a flight with the shortest number of layovers, , we represent cities as vertices and the direct connections between them as edges. Moreover, if we want to the cheapest, fastest, or shortest path, we can achieve this by additionally assigning a weight to the corresponding edge.

2.3 Social Platform

In social media, individuals have diverse interests. Through the use of a graph data structure, we can identify groups of people who share similar interests.

2.4 Problem Solving

Various games and puzzles utilize graph data structures for solving, including chess, tic-tac-toe, and the well-known "Missionary Cannibal Problem". problems.

2.5 Graph Convolutional Network (GCN)

This is an advanced topic, we wont discuss this much here. This network uses matrix representation of the graph, which will be discussed in the later section.

In summary, a graph is a very important data structure extensively used in real-life scenarios.

3 Graph Representation

$G = (V, E)$ and $E \subseteq V \times V$
A graph G is defined by a set of vertices (V) and a set of edges (E) such that each edge is a subset of the Cartesian product of V with itself.

3.1 Undirected Graph

An undirected graph is a type of graph where the edges have no specified direction assigned to them. For any two vertices u and v, $(u, v) = (v, u)$.
The adjacency matrix A of an undirected graph represents the connections between vertices. Each entry a_{ij} is 1 if there is an edge between vertices i and j, and 0 otherwise.
The transpose of the adjacency matrix A is obtained by swapping its rows and columns.

$$A^T = \begin{bmatrix} 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \end{bmatrix}^T = \begin{bmatrix} 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \end{bmatrix}$$

The transpose of the adjacency matrix A for an undirected graph is equal to the original matrix A.

3.2 Directed Graph

A directed graph is a type of graph where the edges have a specified direction assigned to them. For any two vertices u and v, (u, v) may not be equal to (v, u).
where edge (u, v) exists does not mean (v, u) also exists
The adjacency matrix A of a directed graph represents the connections between vertices. Each entry a_{ij} is 1 if there is a directed edge from vertex i to vertex j, and 0 otherwise.

3.3 Weighted Graph

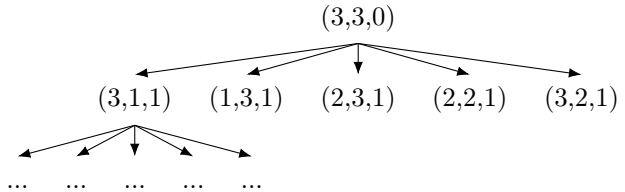
$G = (V, E, W)$
In a weighted graph, each edge($W : E \to \mathbb{R}$) or vertex($W : V \to \mathbb{R}$) has an associated weight or cost. This weight can represent distances, costs, or any other relevant metric.

3.4 Explicit Graph

An explicit graph is one where all vertices and edges are given.We have a finite set of vertices and edges.

3.5 Implicit Graph

In contrast to an explicit graph, in scenarios where we only know the start node, the subsequent nodes and edges are built gradually under a certain set of rules and conditions.
For example, in the Missionary Cannibal Problem, knowing the start and final states, we generate all possible paths and evaluate them against the problem's rules. The domain for people on the boat includes (2,0), (1,0), (0,1), (0,2), and (1,1), where (x,y) denotes the number of missionaries and cannibals in the boat. To further define the state, an additional factor indicates whether the boat is on the initial or destination side of the river.
We have the starting node as (3,3,0) and the destination node as (3,3,1).
We go on exploring the search space.



Theoretically, this graph can be infinite, depending on the domain of possible states.
We cannot represent the graph using a matrix because we don't know the set of vertices beforehand. Therefore, it is suggested to use an adjacency list.
Explanation: In the context of the Missionary Cannibal Problem, the tuple (x, y, z) represents the state of the system. Here, 'x' is the number of missionaries, 'y' is the number of cannibals, and 'z' is a

binary indicator (0 or 1) representing the boat's location (initial or destination side of the river). For instance, (3,3,0) denotes the starting state with 3 missionaries and 3 cannibals on the initial side of the river.

3.6 Representation with Adjacency Matrix or Adjacency List

3.6.1 Adjacency Matrix:

An NxN matrix where N is the number of vertices. Entry (i, j) is 1 if there is an edge between vertex i and vertex j, and 0 otherwise. For weighted graphs, the matrix holds the weights instead of 1s and 0s.
Example: Consider a simple undirected graph with 4 vertices and edges (1, 2), (2, 3), and (3, 4). The adjacency matrix for this graph is:

$$\begin{bmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

For a weighted graph, let's assign weights to the edges:

$$\begin{bmatrix} 0 & 2 & 0 & 0 \\ 2 & 0 & 3 & 0 \\ 0 & 3 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

3.6.2 Adjacency List:

A collection of lists where each list represents the neighbors of a vertex. For weighted graphs, each entry in the list contains the neighbor vertex and the corresponding weight.
Example: For the same graph as above, the adjacency list representation would be:

- Vertex 1: [2]
- Vertex 2: [1, 3]
- Vertex 3: [2, 4]
- Vertex 4: [3]

For the weighted graph:

- Vertex 1: [(2, 2)]
- Vertex 2: [(1, 2), (3, 3)]
- Vertex 3: [(2, 3), (4, 1)]
- Vertex 4: [(3, 1)]

Which is better form of representation?

The variable n represents the number of nodes or vertices in a graph. It is used to denote the size of the graph and is specifically referenced in the context of graph representation.

- For undirected graph maximum number of edges possible is $\binom{n}{2}$ and for directed graph it is n(n-1).
- In the case of an undirected graph, storing it in a matrix has the drawback of containing redundant data.
- In cases of storing sparse data, where the degree of nodes is significantly smaller than the number of nodes, it is desirable to use an adjacency list.

- For determining whether a vertex i is a neighbor of vertex j, the matrix representation can achieve it in O(1), whereas in the case of adjacency list representation, it takes O(branching factor).
- For determining all the neighbors of vertex i, the matrix representation can achieve it in O(n), whereas in the case of adjacency list representation, it takes O(branching factor).
- The vectorized operations that occur in our computer systems store data in matrices, enabling efficient matrix manipulations.

Therefore, there is a tradeoff between the matrix and adjacency list representations of a graph.

4 Graph Traversal

BFS (Breadth-First Search) Pseudo-code:

Two Data structures are used in this algorithm- Queue and array.

```
procedure BFS(graph, start_vertex):
    queue.enqueue(start_vertex)
    while queue is not empty:
        current_vertex = queue.dequeue()
        if current_vertex is not visited:
            visit(current_vertex)
            mark current_vertex as visited
            for each neighbor in graph.neighbors(current_vertex):
                queue.enqueue(neighbor)
```

Complexity for BFS:

Adjacency Matrix:

- **Space Complexity:** $O(|V|^2)$ for the matrix.
- **Time Complexity:** $O(|V|^2)$ due to the need to check all vertices.

Adjacency List:

- Space Complexity: $O(|V| + |E|)$ for the list.
- Time Complexity: $O(|V|+|E|)$, where V is the number of vertices and E is the number of edges. This complexity is approximately linear when the graph is sparse, offering significant efficiency improvements. However, in the case of a dense graph, where the number of edges approaches the maximum possible ($|V|^2$), the time complexity may exhibit characteristics closer to $O(|V|^2)$ due to the potential presence of a substantial number of edges.

Implicit Graph:

- Analogous to a binary tree, for an implicit graph, the time complexity of traversing the nodes through BFS is $O(b^d)$, where b is the branching factor and d is the depth of the graph.
  Explanation:
  In the context of an implicit graph, the time complexity for BFS is expressed as $O(b^d)$, drawing an analogy to a binary tree. This relationship arises from the fact that the traversal involves exploring all nodes up to a certain depth. The branching factor b represents the number of child nodes each node can have, and the depth d represents the maximum depth of the implicit graph. As BFS explores nodes level by level, the time complexity grows exponentially with the branching factor and the depth of the graph. Therefore, the notation $O(b^d)$ captures the computational cost associated with traversing the implicit graph through BFS.