Foundations of Algorithm Design and Machine Learning - Graph Theory Basics

Prabuddha Durge — 24BM6JP17

27/01/2025

1 Introduction to Graph Theory

Graph theory is a fundamental area in computer science and mathematics, dealing with structures called graphs. A graph consists of:

- **Vertices (Nodes)**: Fundamental units that represent entities in a graph. Examples include:
  - Cities in a road network.
  - Users in a social network.
  - Computers in a network topology.
- **Edges**: Connections between vertices, defining relationships. Edges can be:
  - **Directed**: A one-way connection, such as a one-way street.
  - **Undirected**: A two-way connection, such as a friendship in a social network.

A graph is usually mathematically represented as $G = (V, E)$, where:

- V is the set of vertices (or nodes).
- E is the set of edges connecting pairs of vertices.

Graphs can be classified into different types based on properties such as being weighted, directed, cyclic, or connected.

2 Graph Representation

Graphs can be represented in multiple ways, depending on the application and efficiency considerations.

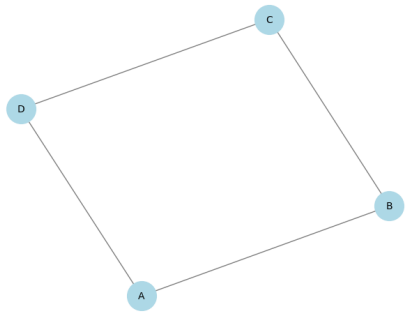

Figure 1: Adjacency Matrix and Adjacency list of this network could be found out in below

2.1 Adjacency Matrix

An adjacency matrix is a 2D array where each cell $matrix[i][j]$ stores a value that indicates whether there is an edge between the vertex i and the vertex j.

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

Pros:

- Quick edge look-ups in $O(1)$ time.
- Efficient for dense graphs where the number of edges E approaches V^2.

Cons:

- Uses $O(V^2)$ space even if the graph is sparse.
- Inefficient for graphs with very few edges.

Best Use Case: when the graph is dense or requires frequent edge lookups.

2.2 Adjacency List

An adjacency list is a collection of lists, one for each vertex, that contain its neighboring vertices.

Example for a graph with 4 nodes:

```
0 → [1, 3]
1 → [0, 2]
2 → [1, 3]
3 → [0, 2]
```

Pros:

- Space-efficient, using only $O(V + E)$.
- Faster for iterating over neighbors of a node.

Cons:

- Checking if an edge exists takes $O(V)$ time in the worst case.

Best Use Case: When the graph is sparse, saving memory compared to an adjacency matrix.

2.3 Time Complexity Analysis

Representation	Space Complexity	Edge Lookup	Iterating Over Edges
Adjacency Matrix	$O(V^2)$	$O(1)$	$O(V^2)$
Adjacency List	$O(V + E)$	$O(V)$	$O(V + E)$

3 Depth-First Search (DFS)

DFS explores each branch as much as possible before backtracking.

Algorithm 1 DFS Algorithm

```
Input: Graph G, starting node s
Output: Nodes visited in DFS order
procedure DFS(G, s)
    Mark s as visited
    for each neighbor v of s do
        if v is not visited then
            DFS(G, v)
        end if
    end for
end procedure
```

3.1 Handling Disjoint Graphs in DFS

If the graph contains multiple disconnected components, the DFS algorithm must be modified to ensure all nodes are visited.

4 Applications of DFS

4.1 Constraint Satisfaction Problems (CSPs)

DFS can help check the validity of paths under constraints. One such example is solving crossword puzzles.

Algorithm 2 Modified DFS for Disjoint Graphs

```
1: procedure DFS_Disjoint(G)
2:     Initialize visited set
3:     for each node v in G do
4:         if v is not visited then
5:             DFS(G, v)
6:         end if
7:     end for
8: end procedure
```

4.2 Crossword Puzzle as a CSP

A crossword puzzle can be modeled as a CSP:

- **Nodes** represent words to be filled.
- **Edges** represent intersections where words share common letters.
- **Constraints** define valid words fitting specific slots.

Solving the puzzle involves searching for word placements that satisfy the constraints.

4.3 Graph Cycle Detection

By tracking entry and exit numbers during DFS, we can detect cycles. This is useful in detecting infinite loops in scheduling problems.

4.4 Shortest Path in a Snake and Ladder Game

The game is modeled as a weighted graph where:

- Forward movement has weight 1.
- Backward movement (snake) has weight 0.

The shortest path to a target position can be computed using graph traversal algorithms like BFS or Dijkstra's algorithm.

5 Conclusion

This lecture covered:

- Basics of graph theory.
- Different graph representations and their complexities.
- DFS traversal and its applications.

- Handling disjoint graphs in DFS.
- Modeling Snake and Ladder as a graph problem.
- Detecting cycles using DFS with entry and exit numbering.
- Using DFS for CSPs like Sudoku and Crossword puzzles.

Understanding these concepts is crucial for solving real-world problems using graph-based algorithms.