Contents

Programmer's HandBook

Akash Tesla

July 2025

Graph Theory

1 Types of Graphs

1.1 Undirected Graph

A Undirected graph is a graph in which edges have no orientation. the edge (u,v) is identical to (v,u)

1.2 Directed Graph(Digraph)

A directed graph is a graph in which edges have orientation. for example edge (u,v) is the edge from u to v

1.3 Weighted Graphs

Weighted graphs are graphs in which it's edges contains a certain value attributed to certain value such as cost, distance, quantity, etc...

1.4 Tree

A tree is a an Undirected graph with no cycles. Equivalently it's a connected graph with n nodes and n-1 edges

1.5 Rooted Trees

A rooted tre is a tree with designated roote node where every edge either ponts away from or towards the root node. When edges point away from the root node the graph is called arborescence or out tree and when the edges point towards the roote node the graph is called anti-arborescence.

1.6 Directed Acyclic Graphs(DAGs)

Directed Acyclic Graphs are Directed graphs with no cycles. These graphs are commonly used in representing structure with dependencies. Several efficient algorithms exist to operate on DAGS

1.7 Bipartite Graph

A Bipartite graph is one whose vertices can be split into two independend group U,V such that every edge connectes between U and V, and there is no odd cycles

1.8 Complete Graph

A complete graph is one where there is a unique edge between every pair of nodes. A complete graph with n vertices is denoted as K_n

2 Representing Graphs

2.1 Adjacency Matrix

A adjacency matrix m is a square matrix used to represent a graph. where each row and column corresponds to a vertex, populate the matrix with condition of m[from node id][to ndoe id] = weight of the edge 1 if it is unweighted
Example:

$$M = \begin{array}{c|ccc} & A & B & C \\ \hline A & 0 & 1 & 0 \\ B & 0 & 0 & 1 \\ C & 1 & 0 & 0 \end{array}$$

Pros

- Space efficient for dense graph
- Takes o(n) for look up

Cons

- Requires $O(v^2)$ space
- Requires $O(v^2)$ time to iterate over all edges
- where, v is the number of vertices in the graph

2.2 Adjacency List

A adjacency list represents graphs with a map with key as vertices that stores a list of neighbour nodes/ list of edges it connectes to with it's cost if it's weighted
Example: ”A”:[(C,2),(D,4)],”B”:[(A,4),(C,1),(D,7)]

Pros

- Space efficient for sparse graph
- Iterating over all edges is efficient
- Prefered for traversal

Cons

- Edge lookup is O(deg(v))
- Not ideal for representing denser graphs
- where, v is the number of vertices
- deg(v), is the degree of freedom of the vertices aka number of edges connected to it

2.3 Adjacency Map

A adjacency map represents graphs with a map with key as vertices that stores another map with key as edges and value as it's weight
Example: "A":"C":2,"D":6,B:"A":7,"C":4

Pros

- Edge weight lookup is O(1)
- Space efficient for sparse graph

Cons

- Space ineffient than adjacency list if you don't want o(1) weight lookup
- Not ideal for representing denser graphs

2.4 Edge List

Edge list represnts graph as list of unordered edges.
Example: [(C,A,4),(C,A,1),(B,C,6)]

Pros

- Space efficient for sparse graphs
- Iterating over all edges is efficient

3 Cons

- Less space efficient for denser graphs
- Edge weight lookup is O(E)
- where, E is number of edges in the graph

4 Graph problems and algorithms used

4.1 shortest path problem

given a weighted graph find the shortest path in the graph
Algorithms: BFS,Dijkstra's algorithm, A*, Bellaman-Ford, Floyd-Warshall...

4.2 Connectivity

Is there a path between Node A and Node B.
Algorithms: union find data structure, Any search algorithms(DFS, BFS)

4.3 Negative cyles

Does my weighted graph has a negative cycle if so where, Negative cycles is a path cycle in which the weights add up to a negative number
Algorithms: Bellaman-Ford and Floyd-Warshall

4.4 Strongly Connected Components

Strongly conected components are self contained cycles within a directed graph where every vertex can reach every other vetex in the same cycle
Algorithms:Tarjan's and Kosaraju's algorithm

4.5 Travelling Salesman Problem (TSP)

Given a list of cities and the distances between each pairs of cities, what is the shortest possible route that visites the city exactly once and returns to the origin city, this is an NP-Hard problem Algorithms: Held-Karp, branch and bound, Ant colony optimizationa and other approximation algorithms

4.5.1 Bridges

A bridge is an edge in a which graph in which it's removal would increase the number of connected components, They could hint at weak points, bottle neck or or vulnerabilities in a graph

4.6 Articulation Points

An ariticulation point is any node in the graph in which it's removal would increase the number of connected components, They could hint at weak points, bottle neck or or vulnerabilities in a graph

4.7 Minimum Spanning Tree(MST)

A Minimum spanning tree is a subset of a connecte, edge-weighted graph that connects all the vertices together, without any cycles and with minimum possible cost.MSTs are used in designing a least cost network, circuit design, trasportation networks and etc..
Algorithms: Krushkal's, Prim's, Boruvka's algorithm

4.8 Network flow: Max flow

With an infinite source how much flow can we push thorough the network consider edges as Pipes with water capacity or roards with car capacity.
Example: Number of cars that can sustain through the traffic.
Algorithms: Ford-Fulkerson, Edmonds-karp, Dinic's algorithm

5 Graph Algorithms

5.1 Dept First Search (DFS)

DFS works by exploring the deepst part of the network from a starting node, then back tracking once you hit the dead-end.

Time complexity : O(V+E)

5.1.1 Use cases:

- Compute a graph's minimum spanning tree
- detect and find cycles in an graph
- check if the graph is Bipartite
- Topologically sort the nodes of a graph

- Find the bridges and articulation points
- Generate mazes
- Find augumenting paths in a flow network

Pseudo code

Algorithm 1 Depth First Search (DFS)

```
function dfs(node)
if Node is visited then
    return
end if
Mark as visited
Problem specific logic goes here
for next in node.neighbours do
    dfs(next)
end for
```

Either implement logic in the given section or in a seperate function and just call the dfs function

5.2 Breath First Search (BFS)

A BFS traverses node layer by layer, it starts at some arbitrary node of a graph and explores the neighbours nodes first before moving on to the next level of neighbours, it uses queue to traverse the graph

Time complexity: O(V+E)

Algorithm 2 B

reath First Search (BFS)

```
  function bfs(node)
```