
CHECKPOINT
Coding Club, IIT Guwahati
< WEEK 2 >

GRAPH THEORY

A Graph is a non-linear data structure consisting of vertices and edges. The vertices are also referred to as nodes and the edges are lines or arcs that connect any two nodes in the graph.

Components of a Graph -

1. **Vertices:** Fundamental units representing entities in a graph.
2. **Edges:** Connections between vertices indicating relationships.
3. **.Neighbors:** Nodes directly connected by edges.
4. **Degree**:Number of edges incident to a node, indicating its connectivity

Intro to Graphs and basic properties: **GFG | Youtube and GFG**

GRAPH REPRESENTATION

The following two are the commonly used representations of a graph.

1. **Adjacency Matrix** : Adjacency Matrix is a 2D array of size V x V where V is the number of vertices in a graph. Let the 2D array be adj[][], a slot adj[i][j] = 1 indicates that there is an edge from vertex i to vertex j. Adjacency matrix for undirected graph is always symmetric.
2. **Adjacency List** : An array of lists is used. The size of the array is equal to the number of vertices. Let the array be an array[]. For every index i, we store the no of vertices adjacent to vertex-i in array[i].

You can read more about this **here**.

GRAPH TRAVERSALS

The two most important graph algorithms are depth-first search and breadth-first search. These are graph traversal algorithms which begin at a starting node and visit all the nodes that can be reached from the starting node. The only difference between depth-first search and breadth-first search is in the order in which they visit the nodes.

Breadth First Search:

Breadth First Search (**BFS**) is a graph traversal algorithm that explores nodes level by level, starting from a given node. It calculates distances from the starting node to all other nodes. It's implemented by visiting nodes in increasing order of their distance from the starting node.

Time Complexity: O(n+m), where n is no. of nodes and m is no of edges.

Implementation: **CPH** Page 120

Practice Problems

- **Labyrinth**
- **Message Route**
- **Shortest Path(cf)**
- **Police Stations**

Multi-Source BFS:

BFS which has multiple starting points. It is convenient to think of it, both for implementation and understanding, is to imagine a “fake source”, which is connected to all of our actual sources, converting this into a normal BFS with an extra initial node.

Practice Problems

- **Nearest Opposite Parity**
- **Monsters**
- **Fair**

0-1 BFS:

0-1 BFS is a variation of BFS for graphs with edges of weight 0 or 1. It efficiently calculates distances from a start node to all nodes in such graphs, providing a faster alternative to Dijkstra's algorithm for certain constrained weight scenarios. **Implementation**

Practice Problems

- **Small Multiple**

Depth First Search

DFS is a graph traversal algorithm that explores as deeply as possible along each branch before moving to the next. It maintains a record of visited nodes to avoid revisiting them. In trees, it finds shortest paths (because there only exists one simple path), but not necessarily in general graphs.(Implementation: **CPH** Page 118)

Time complexity = O(m+n);

Practice Problems

- **Counting Rooms**
- **Leha and another game about graph**
- **Game Master**
- **Anton and Tree**
- **Snuke Maze**
- **Tree Queries**

Graph Edge Classification:

We can classify the edges using the entry and exit time of the end nodes u and v of the edges (u,v). These classifications are often used for problems like **finding bridges** and **finding articulation points.**

We perform a DFS and classify the encountered edges using the following rules:

If v is not visited:

- Tree edges: If v is visited after u then edge (u,v) is a tree edge. They are named thus as these edges are present in a DFS tree.

If v is visited:

- If v is an ancestor of u, (u,v) is a back edge.
- If v is a descendent of u (u,v) is a forward edge.
- If v is neither, (u,v) is a cross edge.

Here, forward and cross edges only exist in Directed Graphs.

DFS can check for bipartiteness by coloring nodes alternately during traversal. If any adjacent nodes end up with the same color, the graph is not bipartite; otherwise, it is.

SHORTEST PATH ALGORITHMS

DJIKSTRA'S ALGORITHM

Dijkstra's Algorithm is an algorithm that is used for finding the shortest distance, or path, from starting node to target node in a weighted graph.

Dijkstra's algorithm makes use of weights of the edges for finding the path that minimizes the total distance (weight) among the source node and all other nodes. This algorithm is also known as the single-source shortest path algorithm.

These resources will help you to get a better idea on this algorithm : -

- **CPH (pg 126-129)**
- Dijkstra's algorithm
- Dijkstra Algorithm - Single Source Shortest Path

Practice Problems

- **Moving Both Hands**
- **Shortest Routes I**
- **Flight Discount**
- **Jzzhu and Cities**

Bellman-Ford Algorithm

This is another algorithm for finding the shortest path between any pair of vertices. There is an upside and a downside associated with it. The advantage of using it is that it can detect negative cycles, which can't be done using Dijkstra (if you can recall). The disadvantage of using it is that it has a time complexity of O(VE), which is much slower than Dijkstra, which runs in O(Vlog(E)).
Resources: **CPH** (Page 123-125) | **G-41. Bellman Ford Algorithm**

So, how do you judge when to use Bellman-Ford?
The first hint lies within the time complexity constraints only, and then read the problem carefully to determine whether there can be negative weights or not. Generally, problems describe weights as distance; in that case, you can directly use Dijkstra.

Can you now answer why at most n-1 loops are necessary for its working?
The reason is that, in the worst case, in a single loop, at least one vertex apart from the starting vertex attains its minimum after relaxation. Then, there are n-1 vertices apart from the starting node left, so n-1 loops are required.
Now if the distance array changes after n-1 iterations then it is only possible if the graph contains a negative cycle.
Do try proving its correctness using induction or contradiction.

Practice Problems

- **All Pair Shortest Path**
- **Segments**

Further improvements (Additional Read)
The first modification is a very basic one: if the distance array remains the same in consecutive iterations, then terminate the loop.
Can you think of an algorithm in which this kind of technique was used?
The answer is very simple. Recall Bubble sort, where we terminated the loop immediately if no swaps were made in a pass.
Moreover there is a famous modification known as SPFA (Shortest Path Faster Algorithm). You can read about it in **CPH** (Page 126) or from **here.**

Floyd Warshall's Algorithm:

This algorithm is used to find the shortest paths between all pairs of nodes in a weighted graph having positive or negative weights in a single run. But it does not work for negative cycles (where the sum of edges in the cycle is negative).

This method uses dynamic programming. We maintain a 2D matrix where each entry represents the shortest path between vertices i and j. Initially, all entries are set to infinity. Then, the algorithm iterates through each edge, updating the shortest path weights for the start and end nodes if a shorter path is found. Once all edges are processed, the shortest paths between all pairs of nodes are determined.

This algorithm is mostly used when we are required to find the shortest distance between any two nodes. Note that the time complexity for this algorithm is O(n^3).

These resources will help you to get a better idea on this algorithm with example and implementation:

CPH (Page 129-131)
G-42. Floyd Warshall Algorithm

Practice Problem:

- **Shortest Routes II**
- **B. Greg and Graph**

TREE ALGORITHMS

A tree is a connected, acyclic graph that consists of n nodes and n-1 edges. Moreover, there is always a unique path between any two nodes of a tree.

Tree Terminology:

- A leaf of a tree is any node in the tree with degree 1
- A root of a tree is any node of the tree that is considered to be at the 'top'
- A parent of a node n is the first node along the path from n to the root
- The ancestors of a node are its parent and parent's ancestors
- The subtree of a node n are the set of nodes that have n as an ancestor
- The depth, or level, of a node is its distance from the root
- The diameter of a binary tree is the length of the longest path between any two nodes in a tree
- The centre is the middle vertex or middle two vertices in every longest path

Algorithms for Minimum Spanning Tree:

Prim's Algorithm:

Prim's algorithm is a method for finding a MST of a graph. The algorithm first adds an arbitrary node to the tree. After this, the algorithm always chooses a minimum-weight edge that adds a new node to the tree. When all the nodes are added to the tree the MST has been found.

These resources will help you to get a better idea on this algorithm:

CPH(pg 147 - 148)

- **Prim's algo implementation**

Kruskal's Algorithm:

The algorithm goes through the edges sorted by the edge weight, and always adds an edge to the tree if it does not create a cycle(this can be done using a data structure called DSU).

The algorithm maintains the components of the tree. Initially, each node of the graph belongs to a separate component. Always when an edge is added to the tree, two components are joined. Finally, all nodes belong to the same component, and a minimum spanning tree has been found.

These resources will help you to get a better idea on this algorithm:

CPH(pg 142 - 145)

- **Kruskal's algo implementation**

Union Find Data Structure(DSU):

This data structure is used in Kruskal's Algorithm in finding if an edge create a cycle or not.

A union-find structure maintains a collection of sets. The sets are disjoint, so no element belongs to more than one set. Two O(logn) time operations are supported: the unite operation joins two sets, and the find operation finds the representative of the set that contains a given element

These resources will help you to get a better idea on this data structure:

CPH(pg 145 - 147)

- **Implementation**

Practice Problems:

- **Design Tutorial: Inverse the Problem**
- **GCD and MST**
- **Make It Connected**
- **Choose Two and Eat One**
- **Microcycle**
- **Counting Graphs**

DP on Trees:-

Dynamic Programming (DP) on trees is a technique used to efficiently solve problems involving tree structures by breaking them down into smaller subproblems and storing intermediate results to avoid redundant calculations.
DP on trees does not have much theory part , you will get the intuition after doing problems .

Resource:-

- **DP ON TREES CF BLOG**

Before doing problems on dp on trees make sure you do these problems:-

- **Subordinates**
- **Tree Matching**
- **Tree Diameter**

Problems :-

- **Family and Insurance**
- **Anji's Binary Tree**
- **Copil Copac Draws Trees**
- **Tree Distances I**
- **Vlad and Trouble at MIT**
- **Fake Plastic Trees**
- **Tree Distances II**
- **Hanging Hearts**
- **Tree Tag**
- **Sasha and a Walk in the City**
- **Programming Competition**
- **Tree XOR (Rerooting on Trees)**
- **Company Queries I (Binary Lifting)**
-

Resource for binary lifting -

- **Binary Lifting Technique - GeeksforGeeks**