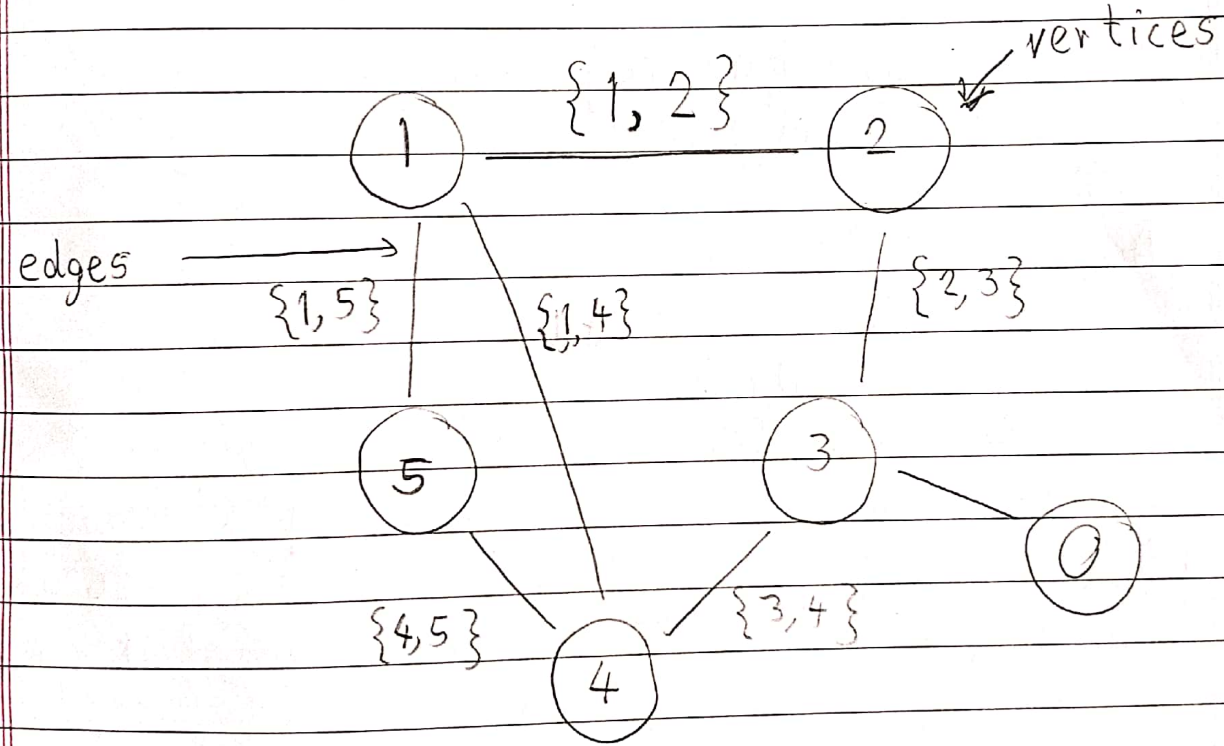
Graphs & Graph Algorithms

→ Why Graphs?

A. We can model various real-life problems and situations using graphs. - eg. rail networks
They ~~are~~ are extremely useful in simplifying stuff!

→ Example of a graph -



This is an example of an undirected graph.

It can be represented as follows -

$$G = (V, E)$$

where V, E denote the following sets -

$V = \{1, 2, 3, 4, 5, 0\}$

$E = \{(1,2), (2,3), (3,4), (4,5), (1,5), (1,4), (0,3)\}$

$|V|$: denotes the number of vertices.

$|E|$: denotes the number of edges.

Note: As the graph is undirected, (a, b) & (b, a) mean the same, ie they denote the same edge.

→ But this "mathematical representation" is hard for computers to process. Hence, we have come up with alternate representions -

① Adjacency Matrix
② Adjacency List

① Adjacency Matrix

DIY
(Do It Yourself)

Use geeks for geeks

PS: Let me know if you ever find a good application of this representation of graphs!

② Adjacency List

→ Defn: Degree of a vertex - V
Degree of a vertex V is the number of neighbors of V ≡ Number of vertices adjacent to V.

→ Question: Find the sum of degrees of all vertices of the example graph. Is it related to the number of edges?

Answer: For an undirected graph,
$\sum deg(V) = 2|E|$

→ Adjacency List for the Example Graph -
0 → (3)
1 → (2, 5, 4)
2 → (1, 3)
3 → (4, 2, 0)
4 → (1, 5)
5 → (4, 1)

→ Rule: If a is a neighour of b, then b is present in the adjacency list of a & a is present in the adjacency list of b.

→ Question: Find the error in the above example, if you haven't already!

→ An adjacency list can be represented as -
① a 2D Array or
② a Linked List

→ Now try the question "Representing Graphs" of the HackerRank Contest.
Pro Tip: Use the 2D Array representation, it is fairly easy to implement, as compared to the linked list representation.

Notes: → Revise your programming concepts if necessary.
→ Each row of the adjacency list matrix contains all neighbors of the vertex corresponding to the

row number.

- Each row is sorted in ascending order.
- The number of neighbors of each vertex is different, but is there an upper limit on this number?

→ Do not proceed without completing "Representing Graphs", with all test cases passed.

→ Next, try to solve the question "Modelling Graphs", wherein a scenario has to be modelled.

  - Notice the similarity between the 2 problems.
  - Notice the differences.
  - What is the maximum number of neighbors of any vertex?
  - Notice that each vertex has to be denoted by the x & y coordinates.

Hint: You can use the 2d → 1d mapping -

$$b = nx + y \quad \text{or}$$

$$x = b/n, \; y = b/n$$

to convert it to either previous problem (not necessary)

→ Do not proceed without completing "Modelling Graphs", with all test cases passed.

→ Now, take a look at the problem - "Maze Solving Robot"

→ This problem is an example of a reachability problem.

→ Reachability Problems-

1. Given a vertex $s \in V$, (which will be called the source vertex from now on), how do we find all the vertices that are reachable from s in one or more steps?
2. Given 2 vertices - s & t, is t reachable from s?

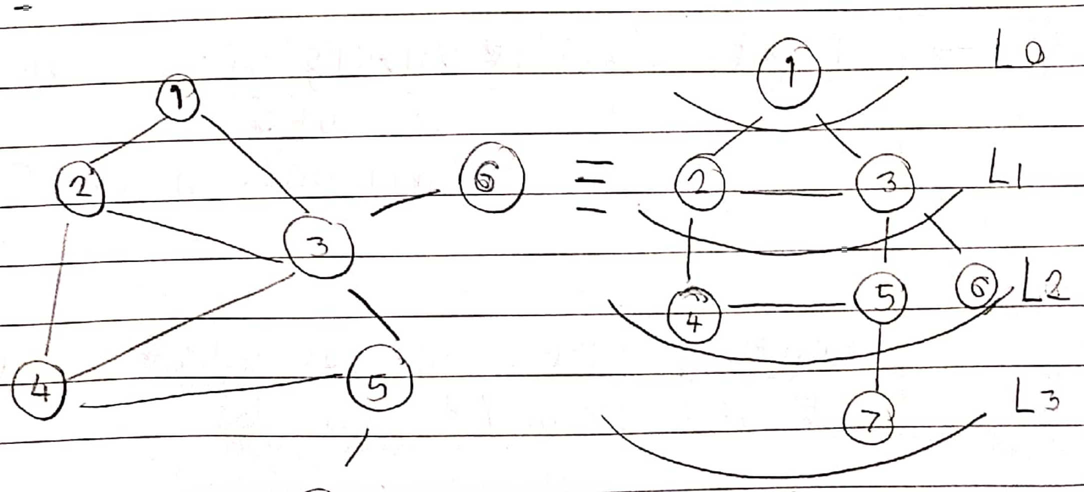
→ To solve problems of the above kind, there are 2 extremely useful and popular algorithms -
  1. Breadth First Search (BFS)
  2. Depth First search (DFS)

→ We will cover BFS here, as it has a wide array of applications. - including implementation of Maze Solving Algorithms for Robots!

→ Breadth First Search

Key idea - Exploring/Travelling the graph in layers.

Example -



BFS with source vertex 1.

Eg. All vertices reachable from vertex 1 in 2 steps are in L_2.

→ Hence, every vertex that can be reached from the source is assigned a layer number and the layer number is the length of the shortest path from the source vertex to the given vertex.

→ Can you connect this observation to the "Maze Solving Robot" Problem?

→ Assigning layer numbers to each vertex might appear to be an easy task, but it is not!

→ BFS is generally implemented using queues.

→ Now solve the question "Implementing Enqueue, Dequeue and Display functions"

→ If you are not able to reach this point, the task(s) become your homework!

→ Next Class :→ Understanding and Implementation of BFS.
→ Introduction to Flood fill Algorithm.

Contact Neil Shirude (9850892135) in case of any doubts/feedback