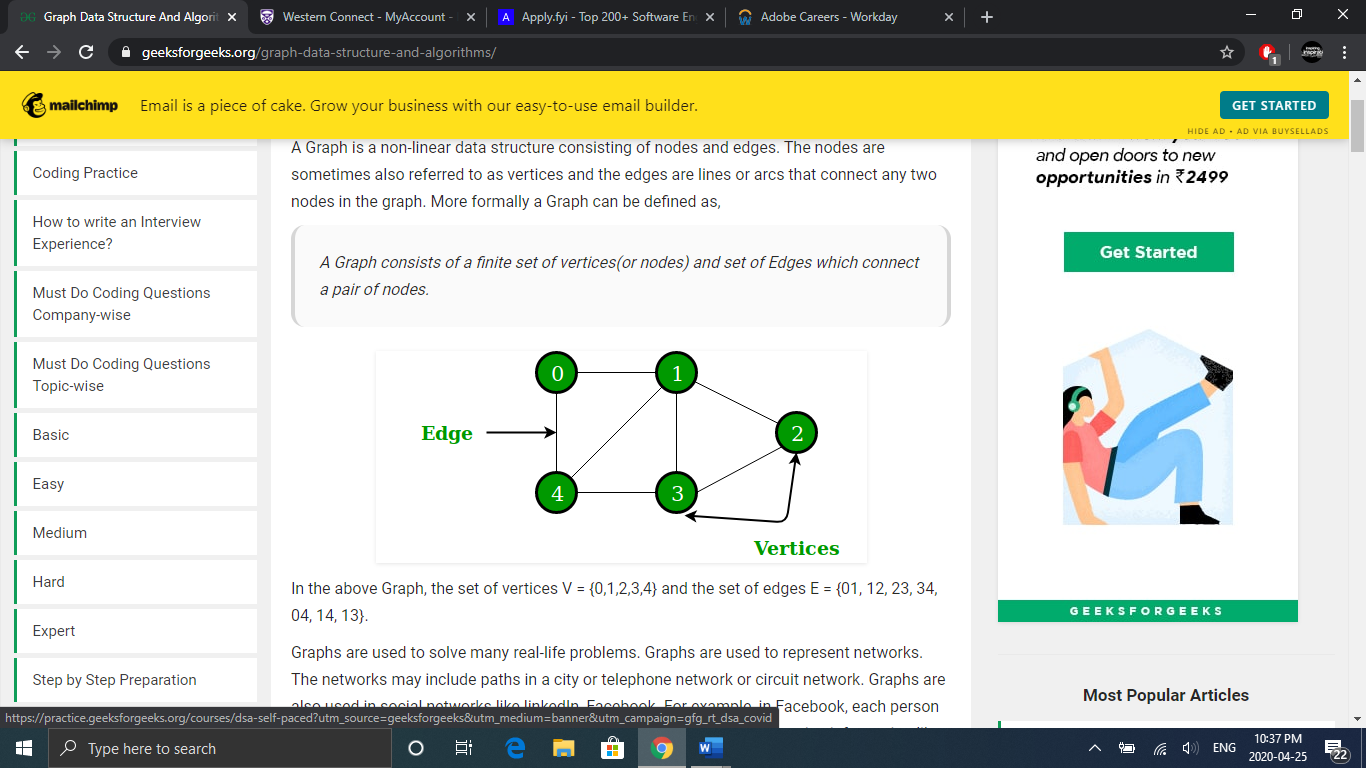
Graphs

* Non-linear data structures which consist of vertices and edges
* Graphs are typically used to represent networks
* Trees are a type of graph



# Breadth First Search

* Similar to BFS for trees except there is a possibility we may run into a cycle
* So when traversing through the graph, we use a Boolean variable to mark the nodes we have already visited
* This avoids us getting stuck in a cycle and our search never ending

 void BFS(int s)

    {

        // Mark all the vertices as not visited(By default set as false)

        boolean visited[] = new boolean[V];

        // Create a queue for BFS

        LinkedList<Integer> queue = new LinkedList<Integer>();

        // Mark the current node as visited and enqueue it

        visited[s]=true;

        queue.add(s);

        while (queue.size() != 0)

        {

            // Dequeue a vertex from queue and print it

            s = queue.poll();

            System.out.print(s+" ");

            // Get all adjacent vertices of the current vertex

            Iterator<Integer> i = adj[s].listIterator();

            while (i.hasNext())

            {

                int n = i.next();

                if (!visited[n])

                {

                    visited[n] = true;

                    queue.add(n);

                }

            }

        }

    }

# Applications

* Minimum Spanning Tree
* Social Networking websites
* GPS Navigation
* Path Finding

# Depth First Search

* Similar to DFS for trees except there is a possibility we may run into a cycle
* So when traversing through the graph, we use a Boolean variable to mark the nodes we have already visited
* This avoids us getting stuck in a cycle and our search never ending

void DFSUtil(int v,boolean visited[])

    {

        // Mark the current node as visited and print it

        visited[v] = true;

        System.out.print(v+" ");

        // Recursively call all the vertices adjacent to current vertex

        Iterator<Integer> i = adj[v].listIterator();

        while (i.hasNext())

        {

            int n = i.next();

            if (!visited[n])

                DFSUtil(n, visited);

        }

    }

    // The function to do DFS traversal. It uses recursive DFSUtil()

    void DFS(int v)

    {

        // Mark all the vertices as not visited(set as

        // false by default in java)

        boolean visited[] = new boolean[V];

        // Call the recursive helper function to print DFS traversal

        DFSUtil(v, visited);

    }

# Applications

* Detecting a Cycle
* Path Finding
* Topological sort
* Solving Maze problems