1. Introduction to System Design

What is LLD ?

DSA --> Has many algorithms --> Ex: Search an Element in an array
|
|---> If array sorted then BS
|
|---> If array unsorted then LS

So, DSA concepts are used to solve Isolated Problems.
|--> Ex: Searching an element in array (Problem)
| BS or LS (Solution)
|
|--> Sorting an Array (Problem).
| Quick Sort, Merge Sort, Insertion Sort, etc (Solution)

LLD involves building a complete application around one or multiple DSA concepts.

Let's understand this concept further by a story.

LLD vs DSA Story

There are two friends, Anurag & Maurya
gets placed in a |-->Knows DSA |--> Knows DSA
company called |-->Do not Know LLD |--> Knows LLD
Quick Ride
(Similar to Uber, Ola, etc)
(Ride Booking Platform)

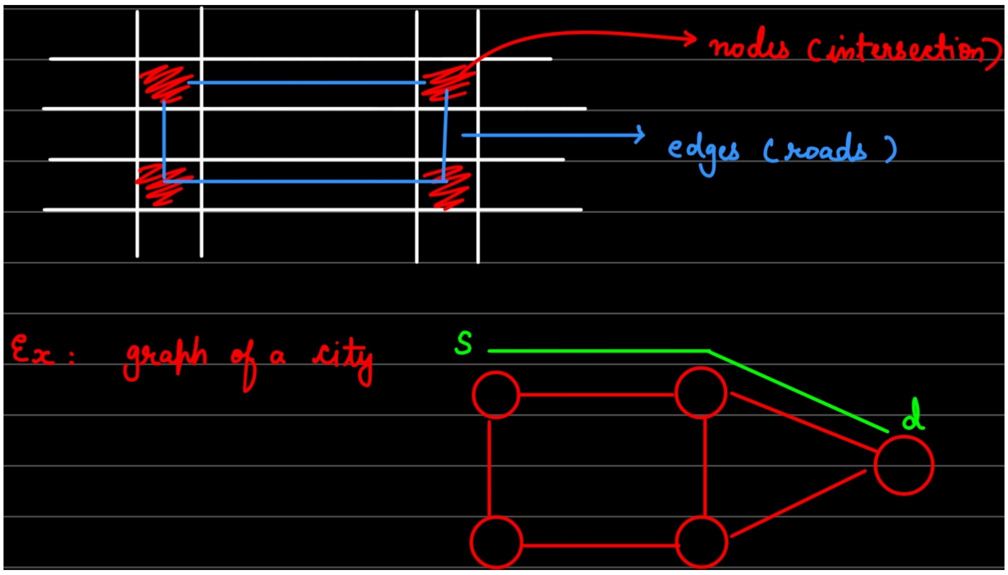
Let's look at perspective of both(Anurag & Maurya) solving the same problem.

Anurag's Approach --> Asked to figure out how to build Quick Ride Application. (By Manager)
| DSA ✓
| LLD ❌
| --> **Thinks about the problem from a DSA perspective focusing on writing Algorithms.**

Problem Identification

- **Problem 1:** Finding the path from source to destination.

- Anurag models the city as a graph, where intersection are nodes & roads are edges.

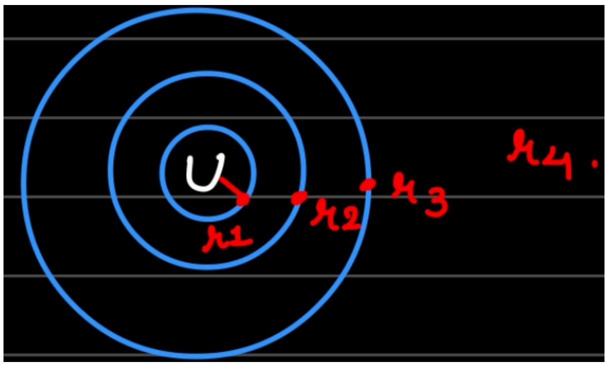


  - User --> Source to Destination --> Find shortest distance
  - To find shortest distance --> Anurag uses Dijkstra Algorithm

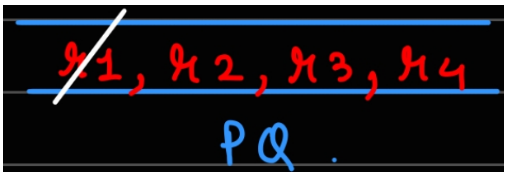
Problem 1 Solved.

  - **Problem 2:** Assigning a rider to user
  - U --> user's location

    Concentric circle --> Denotes nearby locations



    - Map a priority queue corresponding every user.



      - Put the nearby riders for that user in heap(PQ).
      - When we pop the heap, we will get the closest rider & we will assign it to user.

Problem 2 Solved.

Manager's Feedback

Anurag's manager tells him that what he solved were just algorithms, not an application. Anurag missed critical aspects of building a real-world application:

- Identifying the **objects or entities** in the application.
- Defining the **relationships** between these objects and how they interact.
- Considering **data security** (e.g., hiding user/rider phone numbers after a ride).

- Integrating features like **notifications.**
- Integrating services like **payment gateways.**
- Most importantly, how the application will **handle millions of users and scale.**

Anurag jumped directly into algorithms (DSA) without considering the overall application structure.

Maurya's Approach (DSA Expert, LLD Expert)

Maurya, who knows both DSA and LLD, approaches the problem differently. He believes building the application's structure (LLD) must come first, with DSA coming much later.

1. **Identify Objects/Entities:** Maurya starts by identifying the core objects or entities in the application.
    - User (the one booking the ride).
    - Rider (the one providing the ride).
    - Location.
    - Other entities like Notification and Payment gateway.
    2. **Define Relationships:** He then figures out how these objects are related and will interact (e.g., how a user interacts with a rider, how a rider interacts with a location).
    3. Consider Application-Level Concerns:
        - **Data Security:** How to handle sensitive data like phone numbers securely.
        - **Scalability:** How to build the application so it can sustain and handle millions of users without breaking.
        - Integrating external services like **notifications and payment gateways**.
    4. **Apply DSA (Later):** Only after designing the application's structure and considering these factors does Maurya think about the specific algorithms (like finding the shortest path or rider mapping) that Anurag focused on initially.

This comparison shows that LLD is the framework or structure upon which the application is built, and DSA is a tool used within that framework to solve specific technical problems.

Three Pillars of LLD

When building an application and creating its Low-Level Design, LLD primarily focuses on three key areas:

1. **Scalability**
    - Ensuring the application **can sustain and handle millions of users** as it grows.

- How easily **new features can be integrated** and the **application can scale**.

2. **Maintainability**
    - Writing code that is **easy to maintain**.
    - Integrating **new features should not break existing functionality**.
    - Requires **minimum effort to integrate new code**.
    - Code should be easily **debuggable** (easy to find and fix bugs). The goal is minimum bugs, acknowledging that 0% bugs is impossible.
3. **Reusability**
    - Writing code that is highly **reusable**.
    - Creating components or algorithms that are **application-independent**.
    - Follows a **"Plug and Play"** model where code can be easily moved to and work in different applications.
    - Code should not be **"tightly coupled"** or fixed only for one specific application.

Examples:

- **Notifications/Payment Gateways:** These are general functionalities that can be integrated into any application (Quick Ride, Zomato, Swiggy).
- **Rider Mapping Algorithm**: The core logic for assigning a delivery person (rider, delivery boy, delivery partner) to a user, restaurant, or order is a generic concept used in ride-hailing, food delivery (Zomato, Swiggy), e-commerce delivery (Amazon), and grocery delivery (Blinkit). A reusable algorithm for this mapping can be applied across different platforms with minimal changes.

What is NOT LLD: High-Level Design (HLD)

It's important not to confuse LLD with High-Level Design (HLD). While both are part of system design, they focus on different aspects.

HLD (High-Level Design) focuses on the System Architecture. It deals with broader, system-wide concerns that LLD typically does not.

HLD focuses on questions like:

- The application's **Tech Stack** (e.g., using Java Spring Boot).
- The choice of **Database** (SQL, NoSQL, or a hybrid model).
- How **Servers** will scale to handle bulk traffic by easily expanding and contracting.
- **Cost Optimization**, specifically minimizing the monetary cost of running servers (e.g., on AWS) by ensuring they scale efficiently

based on user load.

In an **HLD interview**, you typically focus on creating an **architectural design and system design diagrams, writing almost negligible code.**
This is different from LLD, where you think about the structure of the actual code and create diagrams like class diagrams.
Using the Quick Ride example, HLD questions would involve choosing the tech stack, deciding on the database, and planning server scaling and cost optimization.

The Relationship Between HLD, LLD, and DSA

•All **three (HLD, LLD, DSA) are used together to build a complete application or software**.
•**HLD** focuses on the **System Architecture**.
•**LLD** focuses on the **structure of the code**.
•**DSA** is a tool that **LLD uses to solve problems within the application's code structure.**

A key takeaway analogy is:"If DSA is the brain of an application, LLD is the skeleton."