# **Internship Document**

## **Day 1**

* Introduced ourselves to the mentor.
* The mentor provided an overview of the internship project.
* The project is a website focused on air quality information.
* Key features discussed:
  + Integration of charts for visualizing air quality data.
  + Development of a machine learning model related to air quality prediction or analysis.
  + A backend server should be there.
* Then for the next day the mentor asked us to decide which language we will be using for frontend and backend, the dataset we will be using and the database for the project.

## **Day 2**

* The mentor guided us to make key technology decisions for the project.
* We were asked about:
  + The programming language and framework for the **frontend**.
  + The language and framework for the **backend**.
  + The **dataset** we would be using for air quality analysis.
  + The **database** for storing project data.
* The mentor then divided us into groups.

## **Day 3**

* Mentor taught us about **APIs** and how backend frameworks handle requests.
* Learned the basics of **Flask** and **FastAPI**.
* Discussed about payload and its structure.
* Understood different HTTP request methods:
  + **GET** – Retrieve data from the server.
  + **POST** – Send data to the server.
  + **PUT** – Update existing data.
  + **DELETE** – Remove data from the server.
* Mentor introduced **Postman** and showed how to test API endpoints.

## **Day 4**

* Mentor taught us essential **Git commands** for version control.
* Commands covered and their functions:
  + **git add** – Stages changes so they can be committed.
  + **git commit** – Saves the staged changes to the local repository with a message.
  + **git push** – Uploads local commits to the remote repository (e.g., GitHub).
  + **git pull** – Fetches and merges the latest changes from the remote repository.
  + **git fetch** – Downloads the latest changes from the remote repository *without* merging them.
  + **git branch** – Shows available branches or is used to create a new branch.
  + **git checkout** – Switches between branches or restores files.
* Learned how Git helps in managing project changes and team collaboration.

## **Day 5**

* The mentor taught us the basics of **Database Management Systems (DBMS)**.
* Topics covered:
  + **DDL (Data Definition Language)** – Commands used to define or modify database structures (e.g., CREATE, ALTER, DROP).
  + **DML (Data Manipulation Language)** – Commands used to manipulate data (e.g., INSERT, UPDATE, DELETE).
  + **Relational Databases** – Databases that store data in structured tables (e.g., MySQL, PostgreSQL).
  + **Non‑Relational Databases** – Databases that use flexible formats like documents, key-value pairs, or graphs (e.g., MongoDB).
  + **1NF (First Normal Form)** – Ensures that each column contains atomic (indivisible) values and each record is unique.
  + **2NF (Second Normal Form)** – Ensures all non-key attributes are fully dependent on the primary key.
  + **3NF (Third Normal Form)** – Removes transitive dependencies (non-key attributes depending on other non-key attributes).
  + **BCNF (Boyce-Codd Normal Form)** – A stronger version of 3NF; every determinant must be a candidate key.

## **Day 6**

* The mentor taught us about different **LLM models** and their applications
* Examples of LLMs explained: GPT series,BERT,LLaMA.
* Introduced us to **NLTK (Natural Language Toolkit)** and explained how it is used for text preprocessing, tokenization, stemming, lemmatization, and other NLP tasks.
* Explained various types of **Machine Learning models** and how they are used in real‑world scenarios.
* The categories explained were:
  + **Supervised Learning** – The model learns from labeled data. Examples: classification and regression.
  + **Unsupervised Learning** – The model identifies patterns in unlabeled data. Examples: clustering and dimensionality reduction.
  + **Reinforcement Learning** – The model learns through rewards and penalties while interacting with an environment.
* Examples of ML models discussed:
  + **Linear Regression** – Predicting numerical values like house prices or pollution levels.
  + **Decision Trees** – Classification tasks such as identifying whether air quality is good or bad.
  + **K‑Means Clustering** – Grouping similar data points, such as clustering pollution patterns.

## **Day 7**

## Discussed about the upcoming **25% project milestone** with the mentor.

## The mentor clarified expectations and guided us on deliverables for the milestone.

* Our personal repository links were collected.

## Cleared doubts and questions raised by team members regarding workflow, tasks, and project progress.

* Learned how to install git on the computer and push the code we have done so far into the repository.

## 

## **Day 8**

* Discussed about what we have learned so far.
* Prepared document and ppt for the 25% milestone.

## **Day 9**

* We discussed **Artificial Intelligence (AI)** and its main subsets.
* Mentor explained how AI branches into various domains and where each component fits in the hierarchy.
* Subsets covered:
  + **Machine Learning (ML)** – Systems that learn from data.
  + **Deep Learning (DL)** – Neural‑network‑based approach for solving complex tasks.
  + **Natural Language Processing (NLP)** – AI focused on language understanding and generation.
  + **Computer Vision (CV)** – AI focused on image and video understanding.
* We also discussed the main **Deep Learning architectures**:
  + **ANN (Artificial Neural Network)** – Basic building block of deep learning models, made of interconnected neurons.
  + **CNN (Convolutional Neural Network)** – Used mainly for image‑based tasks such as classification and object detection.
  + **RNN (Recurrent Neural Network)** – Designed for sequential data like time‑series or text.
  + **LSTM (Long Short‑Term Memory)** – An advanced RNN architecture used for long‑sequence tasks, preventing vanishing gradients.

## **Day 10**

* Discussed **Natural Language Processing (NLP)** in deeper detail.
* Mentor explained how text data is processed and understood by machines.
* Topics covered:
  + **Tokenization** – Breaking text into words, sentences, or tokens.
  + **Stop Word Removal** – Removing commonly used words that do not add significant meaning (e.g., "is", "the").
  + **Stemming** – Reducing words to their root form (e.g., "running" → "run").
  + **Lemmatization** – Converting words to their base dictionary form using linguistic rules.
  + **Parts of Speech (POS) Tagging** – Identifying grammatical roles such as nouns, verbs, adjectives.
  + **Named Entity Recognition (NER)** – Detecting entities like names, locations, organizations.
  + **Term Frequency (TF)** – Measures how frequently a word appears in a document.
* Mentor also introduced **SVM (Support Vector Machine)**:
  + Explained how SVM is used for classification tasks.
  + Discussed how it finds the optimal hyperplane to separate different classes.
  + **Inverse Document Frequency (IDF)** – Measures how important a word is by reducing weight for commonly occurring words.
  + **TF-IDF** – A combined metric used to convert text into meaningful numerical features.

## **Day 11**

* Learned about **Support Vector Machine (SVM)** in more detail.
* Mentor explained how SVM uses **margins** to classify data points.
* Topics covered:
  + **Margin** – The distance between the separating hyperplane and the nearest data points from each class.
  + **Hard Margin SVM**:
    - Used when data is **perfectly separable**.
    - **No misclassification** is allowed.
    - Data is assumed to be **clean** and contains **no noise**.
    - Requires that all points be correctly classified.
  + **Soft Margin SVM**:
    - Used when data is **not perfectly separable**.
    - Allows some misclassification to achieve better generalization.
    - Introduced the **C parameter**, which controls the trade‑off between maximizing the margin and minimizing classification error.
      * Higher C → less tolerance for misclassification.
      * Lower C → more tolerance, larger margin.
* Mentor explained the various levels of autonomy in agents:
  + **Agents** – Systems that perceive their environment and take actions to achieve goals.
  + **Human‑in‑the‑loop Agents** – Require human interaction or feedback during decision‑making.
  + **Agents without Human Interruption** – Can operate independently once given instructions.
* Understood how agents are used in real‑world AI systems such as chatbots, robotics, automation tools, and recommendation engines.

## **Day 12**

* Attended a seminar conducted by Kapil on **CSS Flexbox**.
* Learned about the **overflow** property and its values:
  + **auto** – Shows scrollbars only when needed.
  + **visible** – Content overflows and remains visible outside the container.
  + **scroll** – Always shows scrollbars.
* Understood layout sizing concepts:
  + **min-height** and **max-height** for controlling element height limits.
* Learned **box sizing concepts** using:
  + **padding, margin, and border** to manage spacing and layout.
* Introduced to Flexbox core properties:
  + **display: flex** – Enables flex layout.
  + **flex-direction: row / column** – Controls the layout direction.
  + **justify-content: center** – Aligns items along the main axis.
  + **align-items: center** – Aligns items along the cross axis.

## **Day 13**

* Learned advanced Flexbox properties:
  + **align-self** – Individual item alignment: *flex-end, flex-start, stretch, center*.
  + Understood how **align-self follows the flex-direction** rule for alignment.
  + **order** property – Controls the visual order of flex items without changing HTML structure.
* Introduced to using **OpenAI API in Python**:
  + Installed the SDK using **pip install openai**.
  + Learned to import and configure the client:
    - from openai import OpenAI
    - client = OpenAI()
  + Understood the structure of a chat completion request including:
    - **model** – Specifies which LLM to use.
    - **messages** – Conversation input structured as roles:
      * **system** – Sets behavior of the assistant.
      * **user** – Provides user prompts.
    - **max\_tokens** – Limits the length of the generated response.
* Learned how to structure prompts and test basic LLM outputs such as poem generation.

## **Day 14**

* Learned about commonly used **Machine Learning models** for both regression and classification tasks.
* Topics covered:
  + **Random Forest** – An ensemble learning method that builds multiple decision trees and combines their outputs for more accurate and stable predictions.
  + **Linear Regression** – A regression model used to predict continuous numeric values based on linear relationships between variables.
  + **XGBoost (Extreme Gradient Boosting)** – A powerful boosting algorithm used for high‑performance classification and regression, known for efficiency and accuracy.
  + **KNN (K-Nearest Neighbors)** – A classification algorithm that predicts the class of a data point based on the majority class of its nearest neighbors.
  + **Decision Tree** – A model that splits data based on feature values to make predictions in a tree-like structure.
* Understood where each model is best applied and their advantages in solving real‑world prediction problems.

## **Day 15**

* Learned basic **SQL commands** used to interact with relational databases.
* Topics covered:
  + **SELECT** – Retrieve data from tables.
  + **INSERT** – Add new records into tables.
  + **UPDATE** – Modify existing records.
  + **DELETE** – Remove records from tables.
  + **WHERE** – Filter data based on conditions.
  + **LIKE** – Pattern matching for text filtering.
  + **BETWEEN** – Filter values within a specific range.
  + **UNION** – Combine results of multiple SELECT queries.
  + **ORDER BY** – Sort query results.
  + **GROUP BY** – Group records for aggregation in different ways.
  + **HAVING** – Apply conditions to grouped data.
  + **Aggregate functions** – **COUNT, SUM, AVG, MIN, MAX** used with GROUP BY.
  + **TRUNCATE** – Quickly removes all records from a table while keeping its structure.
  + **DROP** – Deletes a table or database permanently.
* Understood how SQL is used for data storage, retrieval, aggregation, and management in backend applications.

## **Day 16**

* Worked on the project for milestone 2.
* Prepared doc and ppt for milestone 2.

**Day17 -18**

Mileston2

## **Day 19**

* Learned about Python built‑in and string methods.
* Built‑in string case and split methods:
  + **lower()** – Converts string to lowercase.
  + **upper()** – Converts string to uppercase.
  + **split()** – Splits a string into a list based on a delimiter.
* String handling methods:
  + **strip()** – Removes leading and trailing whitespace.
  + **join()** – Joins list elements into a single string.
  + **replace()** – Replaces characters or substrings.
  + **startswith()** – Checks if a string starts with a specific value.
  + **endswith()** – Checks if a string ends with a specific value.
  + **find()** – Finds the position of a substring.
  + **isdigit()** – Checks if the string contains only digits.
  + **isalpha()** – Checks if the string contains only letters.

## **Day 20**

* Learned about Python data structure methods and general utilities.

### **List Methods**

* **append()**, **extend()**, **insert()**, **remove()**, **pop()**, **index()**, **count()**, **sort()**, **reverse()**

### **Dictionary Methods**

* **get()**, **keys()**, **values()**, **items()**, **update()**, **pop()**, **popitem()**, **clear()**

### **Set Methods**

* **add()**, **remove()**, **discard()**, **pop()**, **clear()**, **union()**, **intersection()**, **difference()**
* Method behavior summary:
  + **remove()** – Removes an element (raises error if not found).
  + **discard()** – Removes an element if present (no error if missing).
  + **pop()** – Removes and returns a random element.
  + **clear()** – Removes all elements from the set.

### **File I/O Methods**

* **open()**, **read()**, **readline()**, **readlines()**, **write()**, **writelines()**, **close()**

### **General‑Purpose Functions**

* **len()**, **range()**, **print()**, **type()**, **id()**, **sorted()**, **enumerate()**, **zip()**

## **Day 21**

* **Conversion Functions**:
  + int(): Converts value to integer.
  + float(): Converts value to floating-point number.
  + str(): Converts value to string.
  + list(): Converts value to list.
  + dict(): Converts value to dictionary.
  + set(): Converts value to set.
  + tuple(): Converts value to tuple.
* **Mathematical Functions**:
  + abs(): Absolute value.
  + sum(): Sum of elements.
  + min(): Minimum value.
  + max(): Maximum value.
  + pow(): Power of a number.
  + round(): Rounds a number.
* **Functional Programming Tools**:
  + filter(): Filters elements based on a function.
  + map(): Applies a function to all elements.
  + reduce(): Reduces elements to a single value.
  + lambda: Anonymous function. Example: a = lambda a,b: a\*b; print(a(3,4))
* **Input/Output**:
  + input(): Take user input.
  + format(): Format strings.
* **Class/Object Related**:
  + getattr(): Get attribute of an object.
  + setattr(): Set attribute of an object.
  + hasattr(): Check if object has attribute.
  + delattr(): Delete attribute.
  + isinstance(): Check object type.
  + issubclass(): Check class inheritance.
* **Miscellaneous**:
  + globals(), locals(): Access global/local variables.
  + callable(): Check if object is callable.
  + eval(): Evaluate a string as Python code.
  + exec(): Execute Python code.
* **Error Handling Functions**: try, except, finally blocks, raise exceptions.
* **Decorators and Metaprogramming**:
  + @staticmethod: Define static method.
  + @classmethod: Define class method.

## **Day 22**

* **Key Features of Python**
  + Interpreted, high-level, dynamically typed, supports multiple paradigms
  + **PEP 8** – Python style guide for readable and consistent code
  + Mutable vs Immutable data structures
* **Python Memory Management**
  + **Private Heap** – All Python objects reside here, managed by Python Memory Manager
  + **Automatic Allocation/Deallocation** – Memory allocated when objects created and freed when not needed
  + **Reference Counting** – Tracks how many references point to an object; zero references triggers deletion
  + **Garbage Collector (GC)** – Detects and cleans up cyclic references
  + **Memory Pools & Caching** – Reuse small objects to improve speed
* **Python Interpretation** – Python code is executed line by line by the interpreter
* **Namespaces** – Containers mapping names to objects, e.g., local, global, built-in
* **Dictionary** – Key-value storage; merge using dict1.update(dict2)
* Remove duplicates from a list: use set() or list comprehension
* **Flattened Nested List** – Convert nested lists into a single list
* **Shallow vs Deep Copy** – Shallow copies references, deep copies nested objects
* **Slicing** – Extract subsequences from lists, strings, etc. using [start:stop:step]
* **is vs ==** – is checks identity, == checks equality
* **Frozen Set** – Immutable set, cannot be modified after creation

## **Day 23**

* **Generator Function** – Uses yield to return values one at a time without storing the entire sequence in memory.
* **Static Method** – Defined with @staticmethod, does not take self or cls; behaves like a normal function inside a class.
* **Class Method** – Defined with @classmethod, takes cls and can modify class-level data.
* **Monkey Patching** – Dynamically modifying classes or modules at runtime.
* **Custom Exception** – User-defined exceptions created by inheriting from Exception.
* **Exception Chaining** – Using raise ... from ... to show the original exception cause.
* **Model vs Package** – Model refers to a class or data representation; package is a collection of modules.
* **Pivot in Pandas** – Reshape data using column values as new headers.
* **GroupBy in Pandas** – Group data based on column values and apply aggregation.
* **GIL (Global Interpreter Lock)** – Python mutex that allows only one thread to execute Python bytecode at a time.
* **Metaclass** – Class that defines the behavior of other classes.
* **Multithreading vs Multiprocessing** – Multithreading shares memory but limited by GIL; multiprocessing runs parallel processes using separate memory.
* **Train-Test Split** – Divide dataset using train\_test\_split() for model training and evaluation.
* **JDK** – Java Development Kit containing JRE + development tools.
* **JVM** – Java Virtual Machine that runs Java bytecode.
* **JRE** – Java Runtime Environment required to run Java applications.
* **== vs .equals()** – == checks reference equality; .equals() checks value equality.
* **Private Constructor** – Allowed; used in Singletons or factory patterns.
* **Method Overloading** – Same method name, different parameters.
* **Method Overriding** – Subclass redefines a superclass method.
* **Static vs Non-static Variables** – Static belongs to class; non-static belongs to objects.
* **final, finally, finalize** – final prevents modification, finally ensures execution, finalize() is a cleanup method.
* **Collection Framework** – Set of classes like List, Set, Map for data structures.
* **Array vs ArrayList** – Array is fixed size; ArrayList is dynamic.
* **ArrayList vs LinkedList** – ArrayList is faster for access; LinkedList is faster for insert/delete.
* **throw vs throws** – throw explicitly throws an exception; throws declares exceptions in method signature.
* **Serialization** – Converting object to byte stream.
* **Method Hiding** – Static methods in subclass hide superclass static methods

## **Day 24**

* Learned about different **roles and processes** involved in a software project lifecycle.

### **Functional Consultant / Business Analyst**

* Responsible for understanding business requirements.
* Prepares the **Functional Specification Document (FSD)** detailing what the system should do.

### **Developer / Programmer**

* Converts functional requirements into technical solutions.
* Prepares the **Technical Specification Document (TSD)** explaining how the system will be built.
* Develops the application based on specifications.
* Performs **Unit Testing** on individual components.

### **Prototype**

* A preliminary working model used to demonstrate features and gather feedback before full development.

### **Server Environments**

* **Development Server** – Used for coding and initial testing.
* **Quality Server** – Used by QA team for testing.
* **Production Server** – Live environment used by end users.

### **Testing / QA (Quality Assurance)**

* Ensures the application meets functional and quality standards.
* Prepares **Test Scripts** based on the Functional Specification Document.
* Executes test cases and records **Test Results**.

### **Administration**

* Manages overall IT infrastructure.
* Includes:
  + **Network Administration**
  + **Database Administration**
  + **Application Administration**

### **Security**

* Ensures data protection, access control, and system security.

### **Product / Project Management**

* Plans, monitors, and controls project progress, timelines, and deliverables.

### **Sales & Marketing**

* Responsible for product promotion, sales strategies, and customer outreach.

### **Human Resource (HR)**

* Manages recruitment, training, and employee-related activities.

### **Documentation Team**

* Prepares user manuals, technical documents, and project documentation.

### **Support Team**

* Provides post-deployment support and issue resolution.

### **Software Development Models**

* **Waterfall Model** – Sequential development process where each phase is completed before the next begins.
* **Agile Model** – Iterative development approach with continuous feedback and incremental delivery.

## **Day 25**

* Learned about **A\* (A-star) Algorithm**:
  + A graph traversal and pathfinding algorithm.
  + Uses cost function **f(n) = g(n) + h(n)**, where g(n) is the cost from start to current node and h(n) is the heuristic estimate to the goal.
  + Commonly used in navigation, maps, and AI pathfinding.
* Learned about **Sorting Algorithms** and their working principles:
  + **Bubble Sort** – Repeatedly swaps adjacent elements if they are in the wrong order.
  + **Selection Sort** – Selects the minimum element and places it at the correct position.
  + **Insertion Sort** – Builds the sorted list one element at a time by inserting elements in the correct position.
  + **Merge Sort** – Divide-and-conquer algorithm that splits the array and merges sorted halves.
  + **Quick Sort** – Uses a pivot element to partition the array and sort recursively.
  + **Heap Sort** – Uses a heap data structure to repeatedly extract the maximum or minimum element.
* Understood time complexity differences and use cases for each sorting technique.

**Day 26**

* We worked on the project for milestone 3.

**Day 27-28**

* Milestone 3

**Day 29**

* Held a group ppt presentation showing what each team has done so far.

## **Day 30**

* Learned the fundamentals of **Reinforcement Learning (RL)**.
* Key concepts discussed:
  + **Q-value (Q(s, a))** – Represents the expected future reward of taking action *a* in state *s*.
  + **Q-table** – A table that stores Q-values for all state–action pairs.
  + **Rewards** – Feedback signals received after taking an action.
  + **Episodes** – A complete sequence of states, actions, and rewards from start to terminal state.
  + **Greedy Policy** – Always chooses the action with the highest Q-value.
* **Temporal Difference (TD) Update Rule**:
  + Q(S, A) ← Q(S, A) + α (R + γ Q(S′, A′) − Q(S, A))
  + Where:
    - **α (alpha)** – Learning rate
    - **γ (gamma)** – Discount factor
    - **R** – Reward received
* **Q-Learning Algorithm**:
  + A model-free reinforcement learning algorithm.
  + Learns the optimal policy by updating Q-values based on experience.
* **Bellman Equation**:
  + Q(s, a) = R(s, a) + γ maxₐ Q(s′, a)
  + Describes the recursive relationship between current and future rewards.
* Understood how reinforcement learning agents learn optimal behavior through interaction with the environment.

**Day 31**

**What is DFS?**

**Depth First Search (DFS)** is a graph/tree traversal algorithm that explores **one path completely** before trying another path.

👉 It goes **deep first**, then comes back (backtracking).

**How DFS Works**

1. Start from a node.
2. Visit one neighbor.
3. Keep moving forward until no more neighbors exist.
4. Go back to the previous node.
5. Continue until all nodes are visited.

**Data Structure Used**

* **Stack** (or recursion)

**Example**

Graph:

A

/ \

B C

/

D

DFS traversal:

A → B → D → C

**Applications**

* Maze solving
* Cycle detection
* Topological sorting
* Connected components

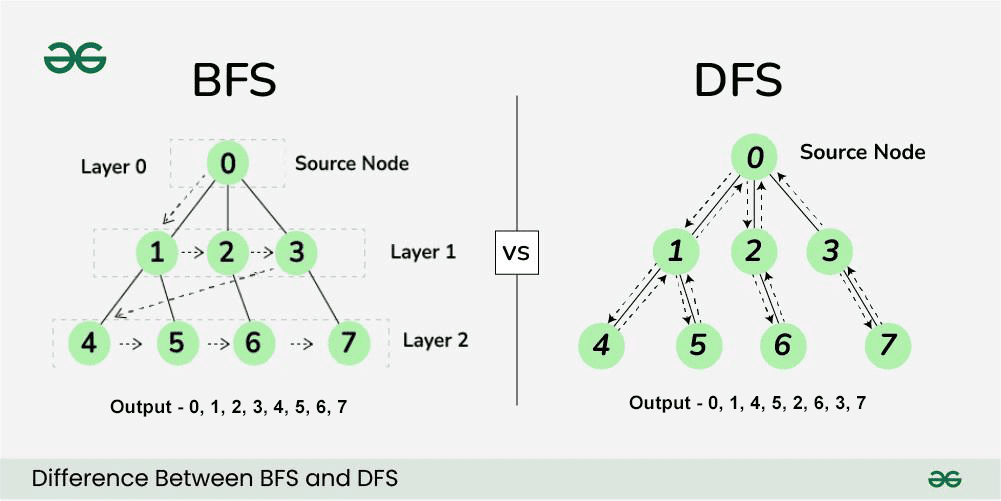
**Advantages**

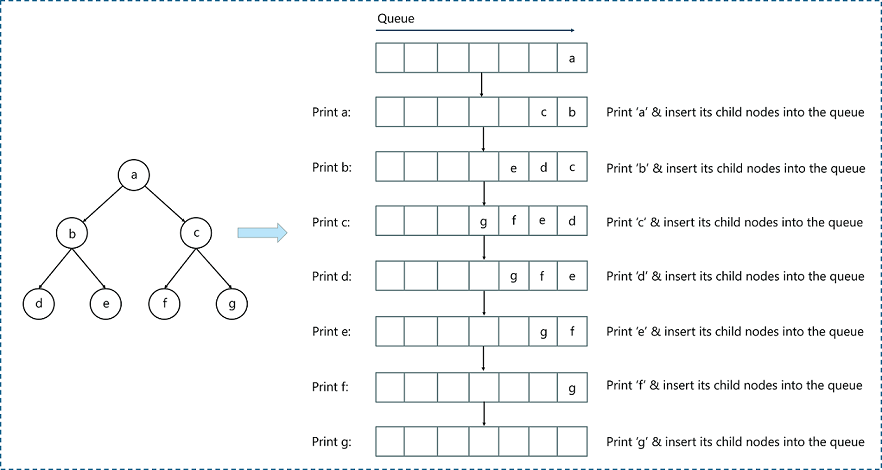
* Uses **less memory**
* Simple logic

**Disadvantages**

* **Does not guarantee shortest path**
* Can go very deep unnecessarily

**🔹 Breadth First Search (BFS)**





**What is BFS?**

**Breadth First Search (BFS)** explores all nodes **level by level**.

👉 It checks all nearby options before moving deeper.

**How BFS Works**

1. Start from the root node.
2. Visit all its neighbors.
3. Then visit neighbors of those neighbors.
4. Repeat until goal is found.

**Data Structure Used**

* **Queue**

**Example**

Same graph:

A

/ \

B C

/

D

BFS traversal:

A → B → C → D

**Applications**

* Shortest path in **unweighted graphs**
* Social networks
* Web crawling
* Level order traversal

**Advantages**

* **Finds shortest path**
* Complete (will find solution if exists)

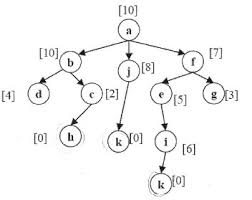
**Disadvantages**

* Uses **more memory**
* Slower for very large graphs

**🔹 DFS vs BFS (Easy Difference)**

| **Feature** | **DFS** | **BFS** |
| --- | --- | --- |
| Traversal | Depth wise | Level wise |
| Data Structure | Stack | Queue |
| Shortest Path | ❌ No | ✅ Yes |
| Memory | Low | High |

**🔹 Hill Climbing Algorithm**



**What is Hill Climbing?**

**Hill Climbing** is a **heuristic search algorithm** used in AI.

👉 It always moves to the **best immediate neighboring state**.

Think of climbing a hill:

* You only move **upward**
* You stop when no higher step exists

**How Hill Climbing Works**

1. Start with an initial solution.
2. Check neighboring solutions.
3. Move to the one with higher value.
4. Stop when no better neighbor exists.

**Problems in Hill Climbing**

* **Local Maximum** – stops before global best
* **Plateau** – flat area, no direction
* **Ridge** – zigzag movement

**Applications**

* Optimization problems
* Game AI
* Scheduling
* Machine learning tuning

**Advantages**

* Very **fast**
* Uses **very little memory**

**Disadvantages**

* Not guaranteed best solution
* No backtracking

**Day 32**

* Learned about **Graph Search Algorithms**:
  + **Depth Limited Search** – A variation of DFS with a predefined depth limit to avoid infinite traversal avoids infinite paths.
  + **Bidirectional Search** – Searches simultaneously from the start node and the goal node to reduce search space.
* Discussed **Travelling Salesman Problem (TSP)**:
  + An optimization problem where the goal is to find the shortest possible route that visits each city exactly once and returns to the origin city.
  + Commonly used in logistics, route planning, and optimization problems.
* Learned about **Principal Component Analysis (PCA)**:
  + A **dimensionality reduction technique** used to reduce the number of features while retaining most of the important information.
  + Works by transforming the original correlated features into a new set of **uncorrelated variables** called **principal components**.
  + **Principal Components**:
    - Ordered by importance.
    - The first principal component captures the **maximum variance** in the data.
    - Each subsequent component captures the remaining variance orthogonal to the previous ones.
  + PCA is based on **linear algebra concepts** such as covariance matrix, eigenvalues, and eigenvectors.
  + **Eigenvalues** represent the amount of variance captured by each principal component.
  + **Eigenvectors** determine the direction of the new feature axes.
  + Helps in:
    - Reducing computational complexity.
    - Removing noise and redundant features.
    - Visualizing high-dimensional data in 2D or 3D.
  + Commonly used as a **preprocessing step** before applying machine learning models.
  + Does not consider class labels (unsupervised technique).

**Day 33**

* Worked on the project

**Day 34**

* The mentor briefed us on the expectations for Milestone 4.
* Held a feedback session.
* We worked on the project.

**Days 35-37**

* We had milestone 4

**Day 38**

Learned the basics of **n8n**:

* Introduction to n8n as a **workflow automation tool**.
* Understood how nodes are connected to automate tasks.
* Learned how to run **Ollama models** locally and integrate them within n8n workflows.
* Explored use cases such as AI-powered automation and API orchestration.

Learned about **website development and hosting using modern AI tools**:

* **Lovable** – Used for rapid UI generation and frontend development with AI assistance.
* **Vercel** – Used for deploying and hosting frontend applications with continuous deployment.
* **Claude AI** – Used to assist in content generation, code suggestions, and rapid prototyping.

Understood the end-to-end flow of developing, deploying, and hosting a website using AI-assisted tools and cloud platforms.

**Day 39**

* We had a session with the higher ups and presented the ppt to them.
* Few suggestions were given after the presentation.

**Day 40**

* Learned about **Trees and Graphs** as fundamental data structures.

**Trees**

* A **tree** is a hierarchical data structure consisting of nodes connected by edges.
* Key concepts:
  + **Root** – Topmost node of the tree.
  + **Parent and Child** – Relationship between connected nodes.
  + **Leaf Nodes** – Nodes with no children.
  + **Height and Depth** – Measures of levels in a tree.

**Types of Trees**

* **Binary Tree** – Each node has at most two children.
* **Binary Search Tree (BST)** – Left subtree contains smaller values, right subtree contains larger values.
* **AVL Tree** – A self-balancing binary search tree where the height difference between left and right subtrees (balance factor) is at most **1**.
  + Ensures faster search, insertion, and deletion operations.
  + **AVL Tree Balancing**:
    - Uses **rotations** to maintain balance after insertions or deletions.
    - **Balance Factor** = height(left subtree) − height(right subtree).
    - If balance factor is outside −1, 0, 1, rotations are applied:
      * **LL Rotation (Left-Left)** – Right rotation when insertion happens in left subtree of left child.
      * **RR Rotation (Right-Right)** – Left rotation when insertion happens in right subtree of right child.
      * **LR Rotation (Left-Right)** – Left rotation followed by right rotation.
      * **RL Rotation (Right-Left)** – Right rotation followed by left rotation.

**Tree Traversals**

* **Inorder Traversal** – Left → Root → Right (used to get sorted order in BST).
* **Preorder Traversal** – Root → Left → Right.
* **Postorder Traversal** – Left → Right → Root.

**Expression Notations**

* **Infix Expression** – Operator placed between operands (e.g., A + B).
* **Postfix Expression** – Operator placed after operands (e.g., AB+).
* Trees are widely used in file systems, databases, compilers, and hierarchical data representation.

**Graphs**

* A **graph** is a collection of vertices (nodes) connected by edges.
* Can be **directed or undirected**, **weighted or unweighted**.
* Key concepts:
  + **Vertices and Edges** – Basic components of a graph.
  + **Degree** – Number of edges connected to a vertex.
  + **Path** – A sequence of vertices connected by edges.

**Graph Representations**

* **Adjacency Matrix**
* **Adjacency List**
* Graphs are used in social networks, maps, routing algorithms, and network analysis.

**Day 41**

* Made changes in the presentation as per suggestions given.
* Practiced presenting the project.