Graph, Tables, and Sets – Cheat Sheet (Theory Only)

---

1. Graph Basics

Graph (G) = (V, E), where:

- **V (Vertices)** → Set of nodes.
- **E (Edges)** → Set of connections between nodes.

Types of Graphs:

- **Directed Graph (Digraph)** → Edges have a direction.
- **Undirected Graph** → Edges have no direction.
- **Weighted Graph** → Edges have weights (costs).
- **Connected Graph** → Every vertex is reachable.
- **Disconnected Graph** → Some vertices are unreachable.
- **Complete Graph** → Every node is connected to every other node.
- **Sparse Graph** → Few edges relative to vertices.
- **Dense Graph** → Many edges relative to vertices.

---

2. Graph Representation

(A) Adjacency Matrix:

- Uses a **2D array** where arr[i][j] = 1 if an edge exists, else 0.
- **Space Complexity:** $O(V^2)$.
- **Best for dense graphs.**

Example:

```
  0  1  2
0 [0, 1, 1]
1 [1, 0, 0]
2 [1, 0, 0]
```

(B) Adjacency List:

- Uses a **list of lists** where each node points to its neighbors.
- **Space Complexity:** $O(V + E)$.
- **Best for sparse graphs.**

Example in Python:

```
graph = {
    0: [1, 2],
    1: [0],
    2: [0]
}
```

---

3. Graph Traversal Algorithms

(A) Depth-First Search (DFS) [Recursive]

Uses **Stack (or Recursion)** → Explores as far as possible before backtracking.

Time Complexity: $O(V + E)$.

```
def dfs(graph, node, visited=set()):
    if node not in visited:
        print(node, end=" ")
        visited.add(node)
        for neighbor in graph[node]:
            dfs(graph, neighbor, visited)

graph = {0: [1, 2], 1: [0, 3], 2: [0, 3], 3: [1, 2]}
```

```python
dfs(graph, 0)  # Output: 0 1 3 2
```

(B) Breadth-First Search (BFS)
Uses **Queue** → Explores level by level.
Time Complexity: O(V + E).

```python
from collections import deque

def bfs(graph, start):
    queue = deque([start])
    visited = set([start])

    while queue:
        node = queue.popleft()
        print(node, end=" ")
        for neighbor in graph[node]:
            if neighbor not in visited:
                visited.add(neighbor)
                queue.append(neighbor)

bfs(graph, 0)  # Output: 0 1 2 3
```

---

4. Tables

A table is a structured format to store and manage data efficiently.
Types of Tables:

- **Relational Tables** → Used in databases (SQL).
- **Hash Tables** → Used for fast lookups and key-value storage.

---

5. Hash Table & Its Operations

A hash table stores key-value pairs using a hash function.
Common operations:

- **Insertion:** Store a key-value pair.
- **Deletion:** Remove a key.
- **Search:** Retrieve value using a key.

Example Implementation in Python:

```python
hash_table = {}

# Insert
hash_table["Alice"] = 25
hash_table["Bob"] = 30

# Search
print(hash_table["Alice"])  # Output: 25

# Delete
del hash_table["Bob"]
print(hash_table)  # Output: {'Alice': 25}
```

Applications of Hash Tables:

- **Fast lookups** (O(1) on average).
- **Used in databases, caching, symbol tables, and load balancing.**
- **Cryptography (hash functions in security algorithms).**

---

6. Sets & Their Operations

A set is an unordered collection of unique elements.

Common Set Operations:

- **Union (A | B)** → Combines all elements from both sets.
- **Intersection (A & B)** → Elements common to both sets.
- **Difference (A - B)** → Elements in A but not in B.
- **Symmetric Difference (A ^ B)** → Elements in A or B, but not both.

Example in Python:

```
A = {1, 2, 3, 4}
B = {3, 4, 5, 6}

print(A | B)  # Union: {1, 2, 3, 4, 5, 6}
print(A & B)  # Intersection: {3, 4}
print(A - B)  # Difference: {1, 2}
print(A ^ B)  # Symmetric Difference: {1, 2, 5, 6}
```

Applications of Sets:

- **Duplicate removal from lists.**
- **Mathematical computations in probability and statistics.**
- **Data science (e.g., finding common users between two datasets).**

Comparison of Graph, Tables, and Sets

Feature	**Graph**	**Hash Table**	**Set**
Usage	Network modeling, pathfinding	Fast lookup, caching	Unique item storage, operations
Time Complexity	O(V + E) for traversal	O(1) avg for search	O(1) avg for operations
Structure	Nodes & edges	Key-value pairs	Unordered collection
Example	Social networks, maps	Database indexing, caching	Unique words in a document

---

Key Takeaways

Graphs: Used in networks, pathfinding algorithms (DFS, BFS).
Hash Tables: Efficient key-value storage (O(1) lookups).
Sets: Unique element storage, mathematical operations.

This **Graph, Tables, & Sets Cheat Sheet** covers **terminologies, graph traversal algorithms, hash tables, and set operations**. Let me know if you need further explanations!