Mohamed Emam Sayed

Report: Hash Tables & Graphs

1. Introduction

This report provides an overview of Hash Tables and Graphs, focusing on their definitions, implementation outlines, and comparisons with other data structures, including linked lists, trees, stacks, and arrays. Visual representations are included for better understanding.

2. Hash Tables

2.1 What are Hash Tables?

A hash table is a data structure that maps keys to values using a hash function. It provides fast data access with an average time complexity of O(1) for search, insert, and delete operations.

2.2 Key Features

- Hash Function: Converts a key into an index.
- Buckets: Containers for values at specific indices.
- Collision Handling: Methods to resolve index conflicts (e.g., chaining, open addressing).

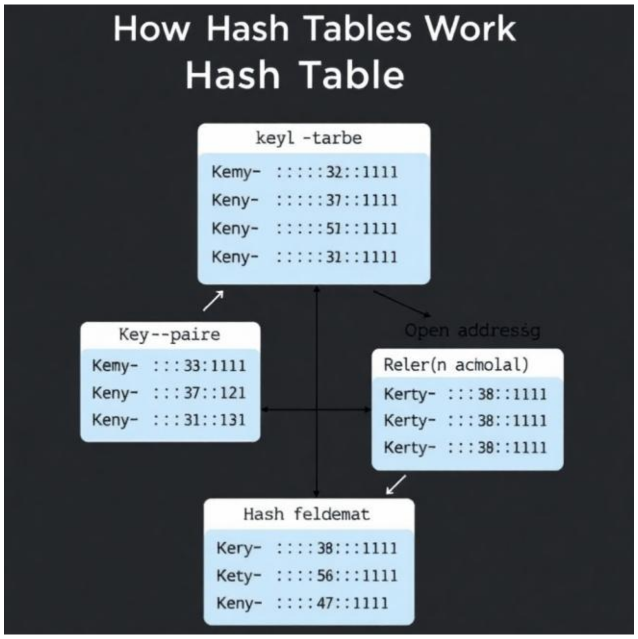
2.3 Implementation Outline

The implementation of a hash table includes designing a hash function to map keys to indices, handling collisions, and implementing basic operations like insertion, search, and deletion.

```cpp
class HashTable {
    vector<list<int>> table; // Using chaining
    int size;

    int hashFunction(int key) {
        return key % size;
    }

public:
    HashTable(int s) : size(s) { table.resize(size); }

    void insert(int key) {
        int index = hashFunction(key);
        table[index].push_back(key);
    }
    bool search(int key) {
        int index = hashFunction(key);
        for (int num : table[index])
            if (num == key) return true;
        return false;
    }

    void remove(int key) {
        int index = hashFunction(key);
        table[index].remove(key);
    }
}
```

2.4 Applications

- Caching
- Symbol tables in compilers
- Databases for quick lookups



3. Graphs

3.1 What are Graphs?

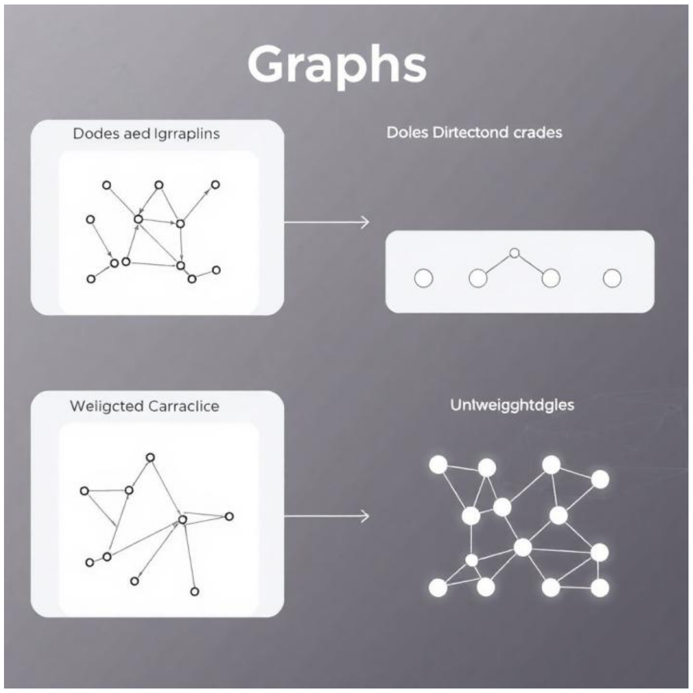
A graph is a collection of nodes (vertices) connected by edges. It can represent relationships between entities.

3.2 Types of Graphs
- Directed vs Undirected
- Weighted vs Unweighted
- Sparse vs Dense

3.3 Implementation Outline
Graphs can be represented using:
- Adjacency List: Efficient for sparse graphs.
- Adjacency Matrix: Efficient for dense graphs.

```cpp
class Graph {
    int vertices;
    vector<vector<int>> adjList;

public:
    Graph(int v) : vertices(v) { adjList.resize(v); }

    void addEdge(int u, int v) {
        adjList[u].push_back(v);
        adjList[v].push_back(u); // Remove for directed graphs
    }

    void display() {
        for (int i = 0; i < vertices; i++) {
            cout << i << " -> ";
            for (int neighbor : adjList[i])
                cout << neighbor << " ";
            cout << endl;
        }
    }
};
```

3.4 Applications
- Social networks
- Network routing
- Recommendation systems



4. Comparison with Other Data Structures

- Arrays: Hash tables offer faster lookups (O(1)), while arrays handle linear data efficiently.
- Linked Lists: Hash tables may use linked lists for collision handling. Adjacency lists in graphs use linked lists.
- Stacks: Depth-first search in graphs uses stacks, whereas hash tables store key-value pairs.
- Trees: Trees are hierarchical, while hash tables are unordered. Trees are a subset of graphs.

5. Strengths and Weaknesses

Hash Tables:
- Speed: Fast lookups (O(1))
- Structure: Key-value mapping
- Memory Use: Can waste space due to collisions
- Flexibility: Less flexible (fixed size)

Graphs:
- Speed: Traversals can be slower (O(V + E))
- Structure: Relationship-based
- Memory Use: Depends on representation
- Flexibility: Highly flexible for complex relationships

6. Conclusion

Both hash tables and graphs play pivotal roles in computer science:
- Hash tables excel in fast data lookups and caching.
- Graphs model complex relationships effectively.

Understanding their differences and use cases helps in designing efficient algorithms and systems.