Name: Youssef Ehab Nagy

Course: Data Structures & Algorithms

Hash table and Graphs

What is a Hashtable?

- **Definition**: A data structure that maps keys to values using a hash function.
- **Main Idea**:
    - Uses a hash function to calculate an index for a key in an underlying array.
    - Provides fast lookups, insertions, and deletions in O(1) average time.
- **Real-world analogy**: A dictionary where you look up the meaning of a word (value) using its spelling (key).

What is a Hash Function?

A **hash function** is a mathematical function that takes an input (key) and produces a fixed-size output, called a hash or hash code. This output is used as an index to locate the associated value in a hashtable.

Key Properties of a Hash Function

1. **Deterministic**: The same input always produces the same output.
2. **Fast Computation**: The function should be efficient to compute.
3. **Uniform Distribution**: Should spread keys uniformly across the hashtable to minimize collisions.
4. **Minimized Collisions**: Different keys should ideally produce different hash codes.

Example with Integer Keys

Suppose we have a hashtable with 10 slots (size = 10), and the keys are integers.

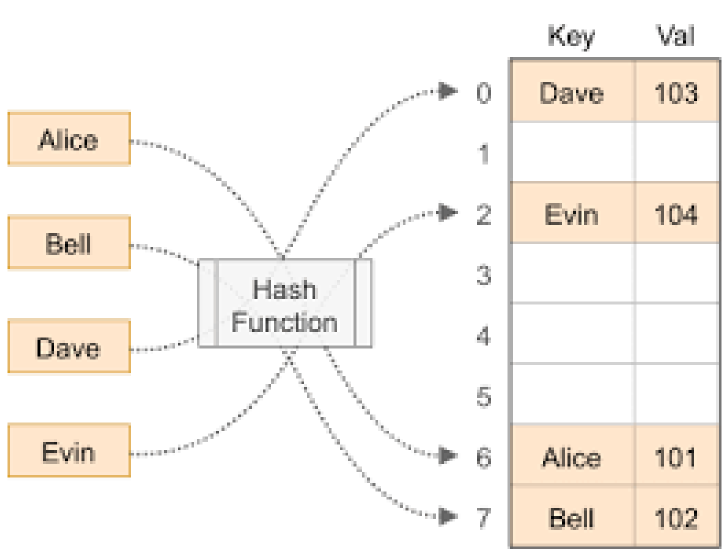
1. **Key**: 42
   Hash Code: 42%10=242 \% 10 = 242%10=2
   Index: 2
2. **Key**: 56
   Hash Code: 56%10=656 \% 10 = 656%10=6
   Index: 6
3. **Key**: 23
   Hash Code: 23%10=323 \% 10 = 323%10=3

Implementation Outline

1. **Core Components**:
   - **Array**: Stores key-value pairs.
   - **Hash Function**: Maps keys to specific indices.
   - **Collision Resolution**:
     - **Chaining**: Use linked lists to store multiple values at a single index.
2. **Steps**:
   - **Insertion**:
     - Compute the hash for a key using the hash function.
     - Place the key-value pair at the calculated index in the array.
     - Resolve collisions if necessary.
   - **Lookup**:
     - Compute the hash of the key.
     - Check the corresponding index for the key.
     - Follow the collision resolution strategy if needed.
   - **Deletion**:
     - Compute the hash and locate the key-value pair.
     - Remove the key-value pair and adjust for collisions.

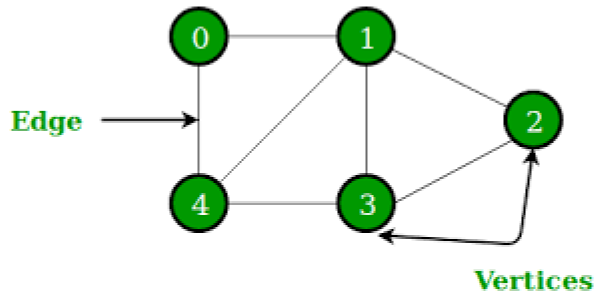
Practical Applications

- **Caching**: Storing frequently used data for fast retrieval.
- **Symbol Tables**: Used in compilers to manage variables and function names.
- **Sets and Dictionaries**: The backbone of these data structures in most programming languages.

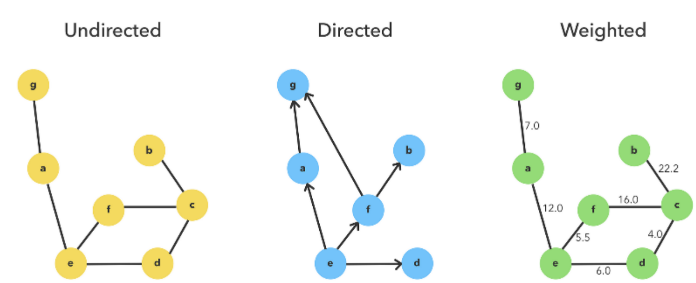


What is a Graph?

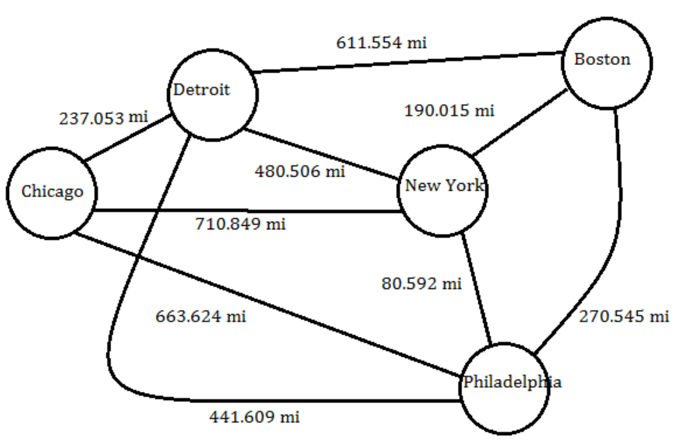
- **Definition**: A collection of nodes (vertices) connected by edges.



- **Main Idea**:
  - Represents relationships or connections between entities.
  - Can be directed (one-way edges) or undirected (two-way edges).
  - Edges can have weights to indicate cost or distance.



- **Real-world analogy**: A transportation map where cities are nodes and roads are edges.

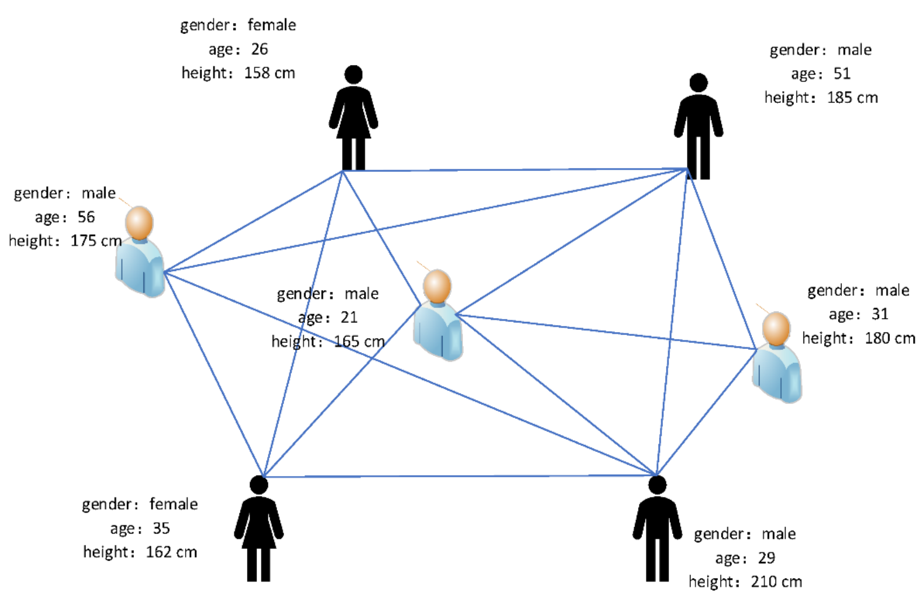


Implementation Outline

1. **Core Representations**:
   - **Adjacency Matrix**:
     - A 2D array where each cell indicates whether there is an edge between two vertices.
     - Space complexity: O(V^2).
   - **Adjacency List**:
     - An array of lists where each index corresponds to a vertex and its connected vertices.
     - More space-efficient for sparse graphs.
2. **Steps**:
   - **Add Vertex**: Extend the adjacency list or matrix.
   - **Add Edge**: Update the adjacency matrix or add to the adjacency list.
   - **Traverse**:
     - **Breadth-First Search (BFS)**: Explore neighbors level by level.
     - **Depth-First Search (DFS)**: Explore as far as possible along each branch before backtracking.

Practical Applications

- **Social Networks**: Representing friendships or connections.
- **Shortest Path Algorithms**: Used in GPS systems (e.g., Dijkstra, A*).



Feature	Hashtable	Graph
Primary Use Case	Key-value mapping	Relationship modeling (networks)
Access Time	O(1)* (average case)	O(V + E) for traversal
Insertion Time	O(1)* (average case)	O(1) for adjacency list
Deletion Time	O(1)* (average case)	O(1) for adjacency list
Order Maintenance	No	No
Search Time	O(1)* (average case)	Depends on traversal (O(V + E))
Space Complexity	O(n)	O(V + E)
Handling Duplicates	Supports duplicates in values	Supports duplicate edges/weights
Structure	Array-based with hash function	Nodes and edges (adjacency list/matrix)
Real-World Use Cases	Caching, dictionaries, symbol tables	Social networks, maps, web crawling

Stack	Queue
Last In, First Out (LIFO)	First In, First Out (FIFO)
O(n)	O(n)
O(1)	O(1)
O(1) (pop)	O(1) (dequeue)
Yes	Yes
O(n)	O(n)
O(n)	O(n)
Allows duplicates	Allows duplicates
Linear	Linear
Undo mechanisms, recursive calls	Task scheduling, BFS traversal

Linked List	Tree
Sequential access and insertion	Hierarchical data representation
O(n)	O(log n)** (balanced trees)
O(1)	O(log n)**
O(n)	O(log n)**
Yes	Yes
O(n)	O(log n)**
O(n)	O(n)
Allows duplicates	Configurable
Linear	Hierarchical
Dynamic memory allocation	Search engines, decision trees