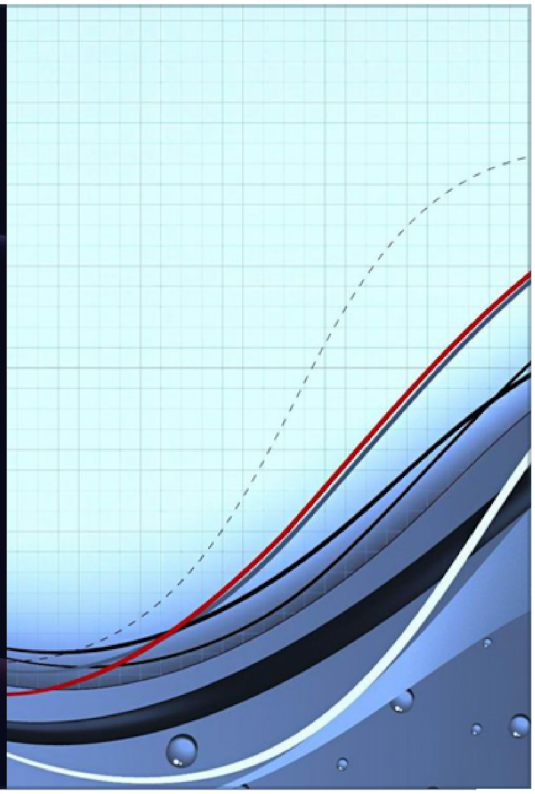
A.R GROUP
PRESENTATION

Presented by: Abdul Rehman

What is a Graph?

A graph is a data structure that represents connections between different objects or nodes. It consists of nodes and edges, where nodes represent the objects and edges represent the connections between them.



Components of a Graph

Nodes or Vertices

The fundamental units of a graph, representing the objects or entities.

Edges or Arcs

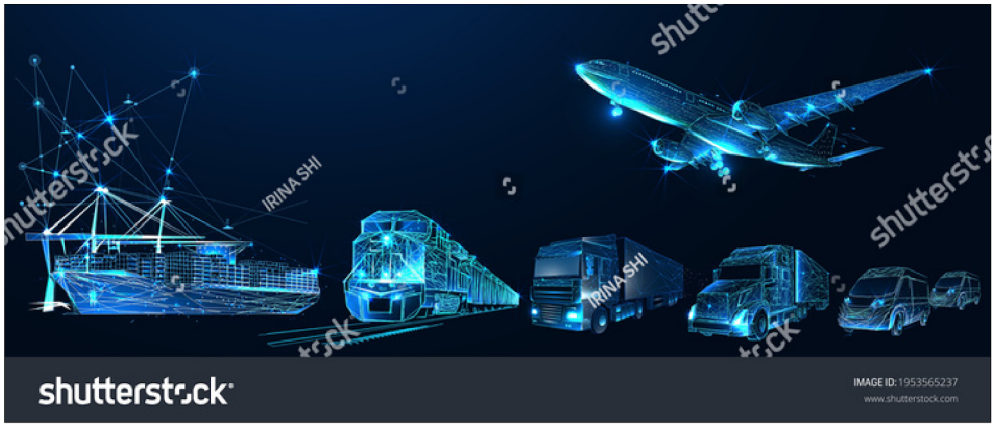
The connections or relationships between the nodes.

Weights

Optional values assigned to edges to indicate additional information, such as the distance or cost between nodes.

Real-World Explanation and Application

Graphs are a powerful tool used to model and solve various real-world problems. For example, they can represent social networks, transportation systems, computer networks, and more. By analyzing the connections between nodes, we can gain insights and make informed decisions.


shutterstock
IMAGE ID: 1953565237
www.shutterstock.com

Types of Graphs

1. **Undirected Graphs**
   Graphs where edges have no direction, and connections are bidirectional.

2. **Directed Graphs**
   Graphs where edges have a direction, indicating one-way connections.

3. **Weighted Graphs**
   Graphs with edges that have associated weights or values.

4. **Cyclic Graphs**
   Graphs that contain cycles, where it is possible to traverse from one node and return back to it.

Applications of Graphs

Social Networks

Graphs can represent connections between individuals on social media platforms.

Route Planning

Graphs are used to find the shortest or fastest routes between locations in maps and navigation systems.

Recommendation Systems

Graphs help in suggesting relevant products, movies, or recommendations based on user preferences and connections.

Graph Algorithms

1

Breadth-First Search (BFS)

Explores the graph by visiting neighbors of a node before moving to the next level.

2

Depth-First Search (DFS)

Explores the graph by recursively visiting the deepest node before backtracking.

3

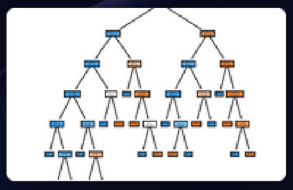
Dijkstra's Algorithm

Finds the shortest path from a source node to all other nodes in a graph with non-negative edge weights.

Graph Visualization

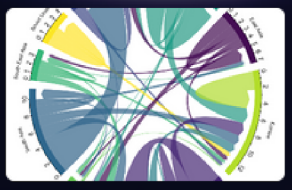
Network Visualization

Visualizes the connections and relationships between nodes using nodes and edges.



Tree Visualization

Represents a subset of graphs where each node has only one parent, forming a hierarchical structure.



Chord Diagrams

Displays relationships between data points using arcs connecting nodes or categories.

Conclusion

Graphs are widely used to model and solve complex problems in various domains. Understanding graph theory and its applications can provide valuable insights and help in making data-driven decisions.

What is Depth First Search?

1. **Graph traversal algorithm**

   Visits each vertex of a graph and explores as far as possible before backtracking.

2. **Depth-first nature**

   Focuses on exploring deeper branches or paths before returning to the previous levels.

3. **Stack-based approach**

   Uses a stack for implementation, allowing backtracking and exploration of unvisited vertices.

Why is Depth First Search Important?

Depth First Search is a fundamental graph algorithm due to its applicability in various problem-solving scenarios.

Graph traversal

DFS helps traverse and explore every vertex and edge of a graph, enabling analysis and pattern identification.

Maze solving

By exploring each path exhaustively, DFS can be used to solve mazes and find the shortest path.

Tree traversal

DFS plays a crucial role in traversing different types of trees, such as binary trees and expression trees.

Applications of Depth First Search

Network Exploration

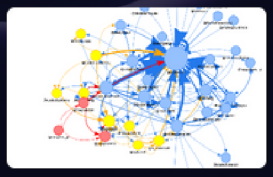
DFS can be used to explore computer networks, uncovering vulnerabilities, and analyzing connectivity.

Artificial Intelligence

DFS aids in pattern recognition and game tree searching, essential for AI algorithms and decision-making.

Software Engineering

DFS helps in dependency analysis, understanding relationships between modules and components.



Social Network Analysis

DFS assists in understanding social networks, identifying influential individuals and community structures.

Analysis of Depth First Search

Time Complexity

The time complexity of DFS is O(V + E), where V represents the number of vertices and E represents the number of edges in the graph.

Space Complexity

The space complexity of DFS depends on the implementation method, but it is generally O(V) for recursive DFS and O(V + E) for stack-based DFS.

Pros and Cons

Pros: Simplicity, easily implemented, detects cycles, suitable for analyzing graphs with deep branches.

Cons: May get stuck in infinite loops for cyclic graphs without proper termination conditions.

Conclusion and Summary of Key Points

1 **Depth First Search**

A powerful graph traversal algorithm that explores deeper branches first.

2 **Applications**

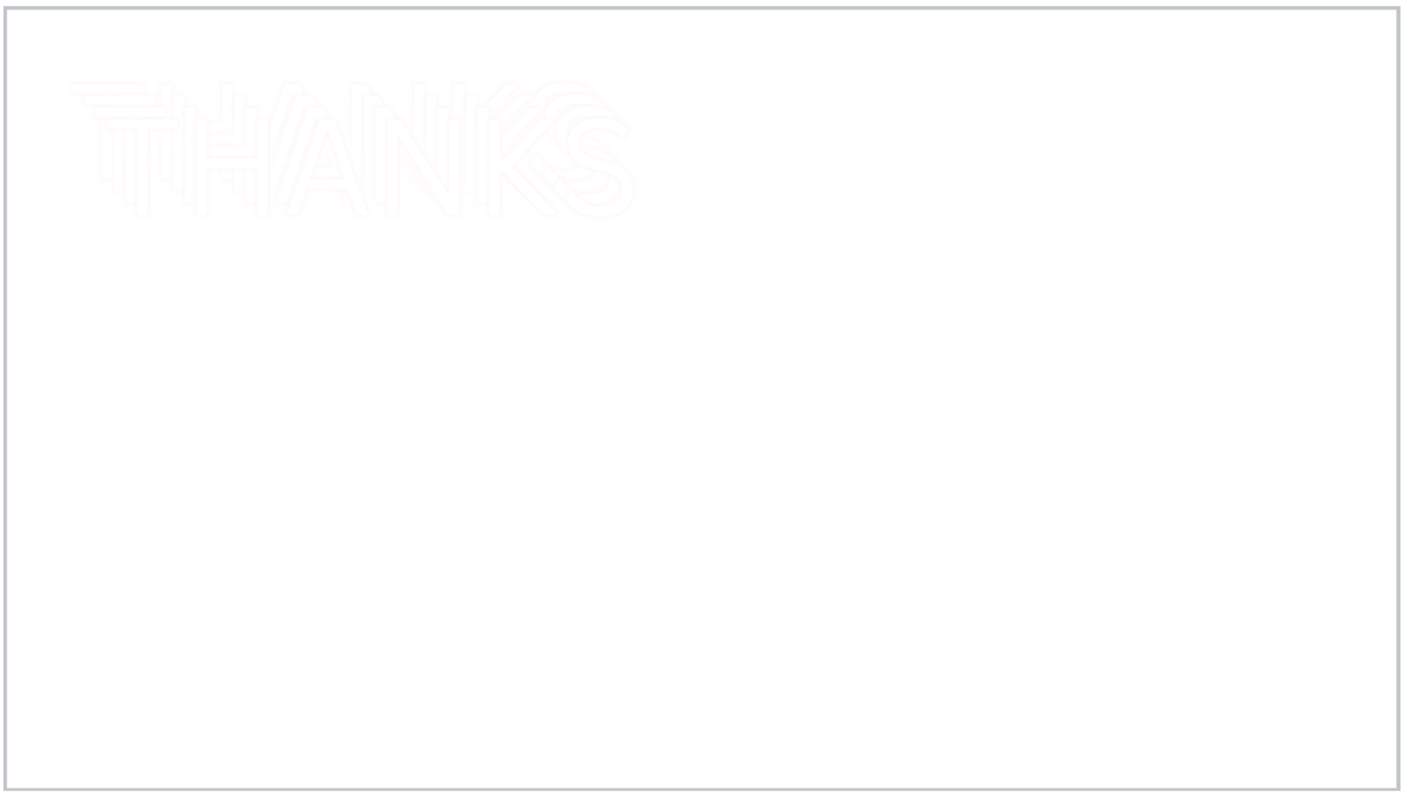
Used in graph traversal, maze solving, artificial intelligence, software engineering, and social network analysis.

3 **Implementation**

DFS can be implemented recursively or using a stack-based approach.

4 **Analysis**

Time complexity is O(V + E), and space complexity depends on the implementation method.


THANKS

Breadth First Searching in DSA

Breadth First Searching is a fundamental algorithm in Data Structures and Algorithms. Learn about its definition, implementation, and applications.

by Zubair Farooq

Definition and Basic Concepts

What is Breadth First Searching?

Breadth First Searching is a graph traversal algorithm that explores all vertices of a graph layer by layer.

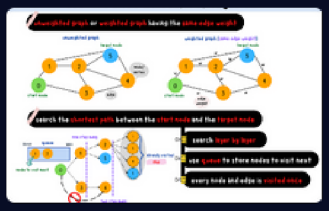
Graph Traversal

Traversal refers to visiting all the nodes/vertices of a graph in a specific order. Breadth First Searching ensures that nodes at the same depth are visited before going deeper in the graph.

Comparison to Depth First Searching

Breadth First Searching differs from Depth First Searching by exploring all vertices at the same level before backtracking. This leads to the discovery of the shortest path in an unweighted graph.

Example of Breadth First Searching Algorithm

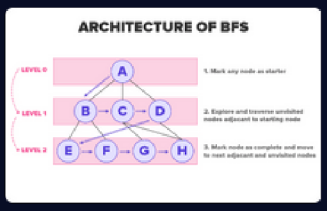


Step 1

Start the algorithm by visiting the initial vertex and enqueue it.

Step 2

Dequeue the visited vertex and process it. Enqueue all the unvisited neighbors of the vertex.



Step 3

Repeat Step 2 until all vertices in the graph have been visited.

Implementation of Breadth First Searching

Data Structures Used

- Queue

Pseudocode

```
function
BreadthFirstSearching(graph,
startVertex):
  create an empty queue
  enqueue(startVertex)
  mark startVertex as visited

  while queue is not empty:
    currentVertex = dequeue()
    process(currentVertex)

    for each neighbor of
currentVertex:
      if neighbor is not visited:
        enqueue(neighbor)
        mark neighbor as visited
```

Applications of Breadth First Searching

1

Shortest Path Finding

Breadth First Searching can be used to find the shortest path between two vertices in an unweighted graph.

2

Web Crawling and Search

Breadth First Searching is used by search engines to discover and index webpages in a systematic manner.

3

Social Network Analysis

Breadth First Searching helps in understanding the connectivity and influence among individuals in a social network.

Conclusion

1 Summary of Key Points

Breadth First Searching is a graph traversal algorithm that explores all vertices layer by layer. It is used to find the shortest path, crawl the web, and analyze social networks.

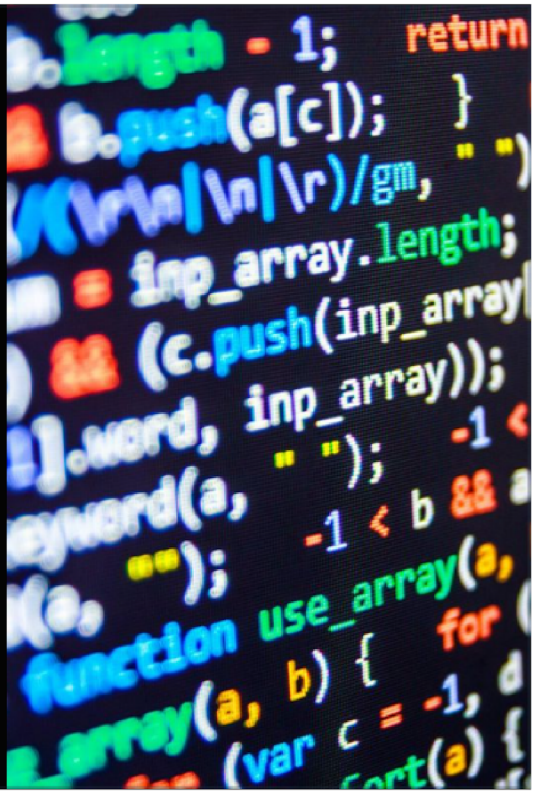
2 Importance in DSA

Breadth First Searching is a fundamental building block in many algorithms and forms the basis for more complex graph algorithms.

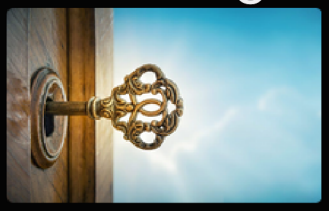
Hashing in Data Structures with Linear Probing

Hashing is a popular technique used in many data structures to achieve fast and efficient searching. In this presentation, we'll explore Linear Probing, one of the most common hashing techniques used today.

by mufeez khan



The Importance of Hashing



Secure Data Storage

To store and retrieve data efficiently, it's crucial to use a secure hashing algorithm to prevent hashing collisions and data theft.

Large Scale Data Processing

Hashing plays a significant role in big data applications, distributed systems, and search engines, enabling them to process large amounts of data quickly and efficiently.

Fast Search Results

Searching for an element in an array or a data structure can take time, but with the help of hashing, we can achieve search results in constant time, thus improving our application's performance.

Linear Probing Explained

Definition

Linear probing handles collisions by incrementing the index by one until a free slot is found.

Advantages

It's simple, fast, and is easy to implement. Linear Probing has fewer memory requirements than other collision resolution techniques.

Disadvantages

Linear probing clusters items, and it can cause a primary clustering issue, where the slots near the hash location are more likely to be occupied.

Implementing Linear Probing

1. **Insertion Process**

   When inserting an element, we first calculate its hash value, then check if the slot is free. If the slot is occupied, we move to the next slot and repeat the process until a free slot is found.

2. **Retrieval Process**

   During retrieval, we start from the hashed index and search for the item. If the element is not found, we move to the next slot linearly until we find the key, hit an empty slot, or backtrack to the start.

3. **Deletion Process**

   When deleting an item, we mark it with a flag (tombstone) instead of removing it, to avoid leaving gaps and breaking the linear search. During retrieval, we need to skip over tombstones and continue our search.

Implementing Linear Probing

1. **Insertion Process**

   When inserting an element, we first calculate its hash value, then check if the slot is free. If the slot is occupied, we move to the next slot and repeat the process until a free slot is found.

2. **Retrieval Process**

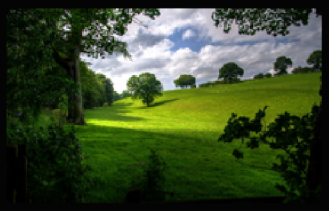
   During retrieval, we start from the hashed index and search for the item. If the element is not found, we move to the next slot linearly until we find the key, hit an empty slot, or backtrack to the start.

3. **Deletion Process**

   When deleting an item, we mark it with a flag (tombstone) instead of removing it, to avoid leaving gaps and breaking the linear search. During retrieval, we need to skip over tombstones and continue our search.

```
#include <iostream>
using namespace std;

const int TABLE_SIZE = 10;

void insertLinearProbe(int table[], int key) {
    int hash = key % TABLE_SIZE;

    while (table[hash] != -1) {
        hash = (hash + 1) % TABLE_SIZE;
    }

    table[hash] = key;
}

int main() {
    int hashTable[TABLE_SIZE];
    for (int i = 0; i < TABLE_SIZE; ++i) {
        hashTable[i] = -1; // Initialize hash table slots to -1 (indicating
empty)
    }

    int keys[] = { 5, 15, 25, 35, 45, 55, 65 };
    int numKeys = sizeof(keys) / sizeof(keys[0]);

    for (int i = 0; i < numKeys; ++i) {
        insertLinearProbe(hashTable, keys[i]);
    }

    // Display the hash table
    cout << "Hash Table:" << endl;
```

Examples of Linear Probing

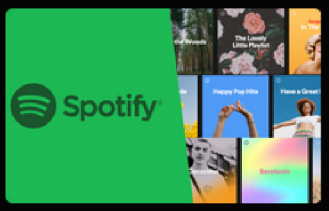


Geospatial Data

Linear Probing can be used to efficiently store and retrieve geospatial data, such as maps, positions, and locations.

Reservation System

Linear Probing can be used to manage restaurant reservations efficiently, ensuring that no tables are left empty or double-booked.



Playlist Management

Linear Probing can be used to efficiently store and manage music playlists, ensuring fast search and deletion times.

In Conclusion

1 **Recap**

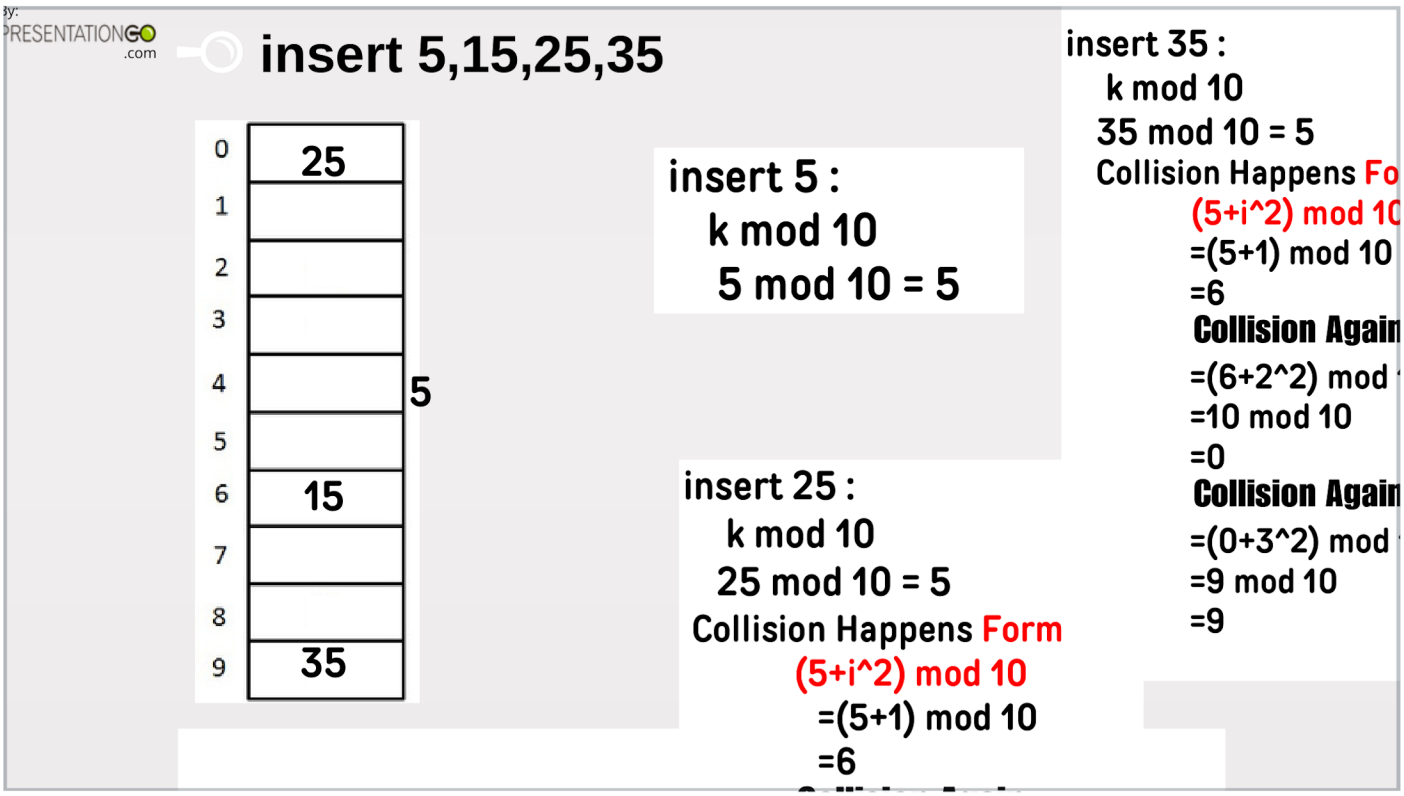
Hashing is a vital technique used in computer science, enabling efficient data storage, fast search results, and large scale data processing.

2 **Linear Probing**

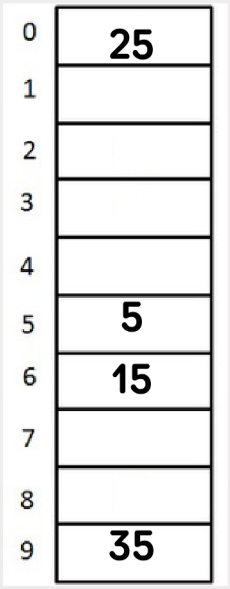
Linear Probing is an excellent collision resolution technique suited for many applications. Its advantages include simplicity, speed, and low memory requirements.

3 **Applications**

Linear Probing can be used for various tasks, such as geospatial data storage, reservation systems, and music playlist management.


insert 5,15,25,35
0 25
1
2
3
4
5
6 15
7
8
9 35
insert 5 :
k mod 10
5 mod 10 = 5
insert 25 :
k mod 10
25 mod 10 = 5
Collision Happens Form
(5+i^2) mod 10
=(5+1) mod 10
=6
insert 35 :
k mod 10
35 mod 10 = 5
Collision Happens Fo
(5+i^2) mod 10
=(5+1) mod 10
=6
Collision Again
=(6+2^2) mod
=10 mod 10
=0
Collision Again
=(0+3^2) mod
=9 mod 10
=9

insert 5,15,25,35



insert 35 :

k mod 10

35 mod 10 = 5

Collision Happens Formula=

(5+i^2) mod 10

=(5+1) mod 10

=6

Collision Again

=(6+2^2) mod 10

=10 mod 10

=0

Collision Again

=(0+3^2) mod 10

=9 mod 10

=9

insert 15 :

k mod 10

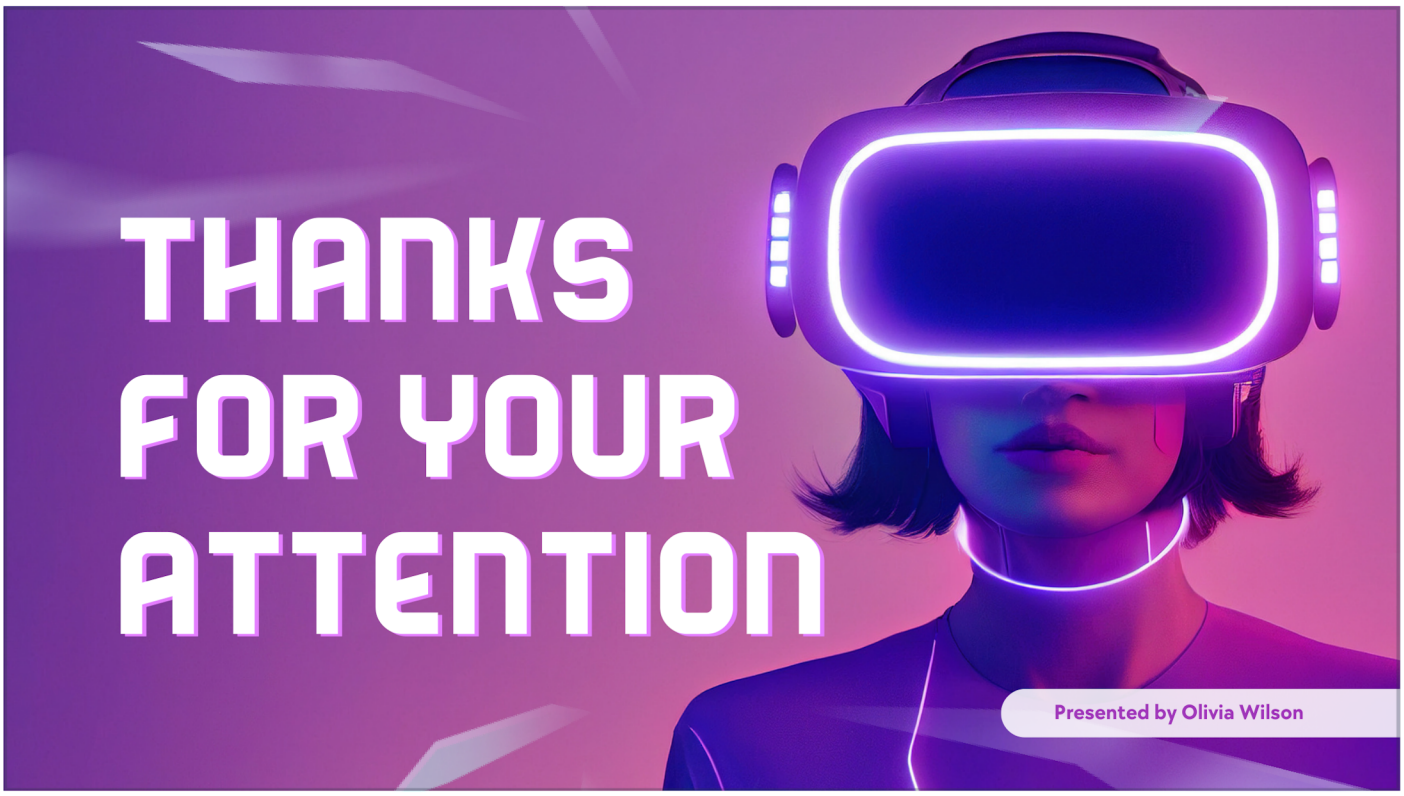
15 mod 10 = 5

Collision Happens

Formula=(5+i^2) mod 10

=(5+1) mod 10

=6


THANKS FOR YOUR ATTENTION
Presented by Olivia Wilson