Introduction to Data Structures

Dr. Machbah Uddin
Associate Professor
Dept of Computer Science and Math.
machbah.csm@bau.edu.bd

Course Overview

- Week 1: Basic
- Week 2: Array, Binary Search
- Week 3: List, String
- Week 4: Stack
- Week 5: Queue
- Week 6: Hashing
- Week 7: Graph
- Week 8: Graph
- Week 9: Tree
- Week 10: Tree

Course Structure and Evaluation

Time	Work	
First day of week	• Theory Lecture • Problem discussion • Group Presentation	
2nd Day of week	• Problem-solving in PC based on theory • Evaluation • Demonstration of the given problem	

- We will provide all lecture slides, practice problems in google classroom
- CT on routine declared date
- A Project submission
- Detailed query or Feedback on Google Classroom

Dr. Machbah Uddin, Associate Professor, BAU, machbah.csm@bau.edu.bd

Evaluation Strategies

Theory Class

Attendance: 10%
Class Tests/Quizes: 10%
Assignment/Presentation: 10%
Final Exam: 70%

Lab Class

Lab Attendance: 10%
Continuous Performance Test: 20%
Problem solving: 40%
Viva: 5%
Project: 25%

Expected Outcome

- Develop skills to handle data in efficient ways for a target job.

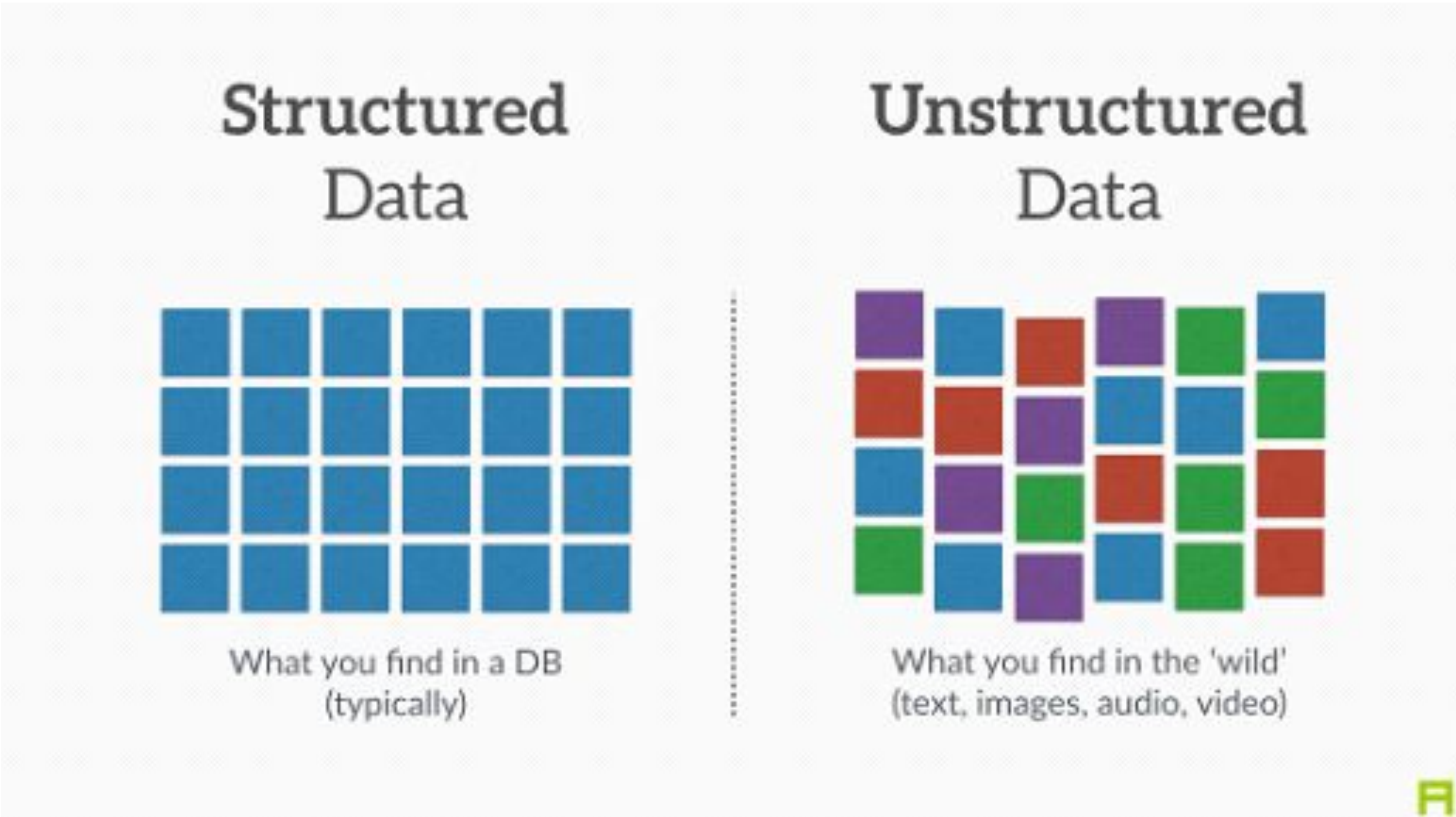
- Gain the ability to analyze and improve data retrieval and storing performance.

- Learn to apply data structure and algorithmic thinking in real-world scenarios

Language

- C/C++
- Java
- Python
- Any other

What is a Data Structure?



What is a Data Structure?

Dr. Machbah Uddin, Associate Professor, BAU, machbah.csm@bau.edu.bd

What is a Data Structure?

- A data structure is a way of organizing and storing data efficiently.
- Essential for efficient algorithm design and software development.

Operations in Data Structures

Dr. Machbah Uddin, Associate Professor, BAU, machbah.csm@bau.edu.bd

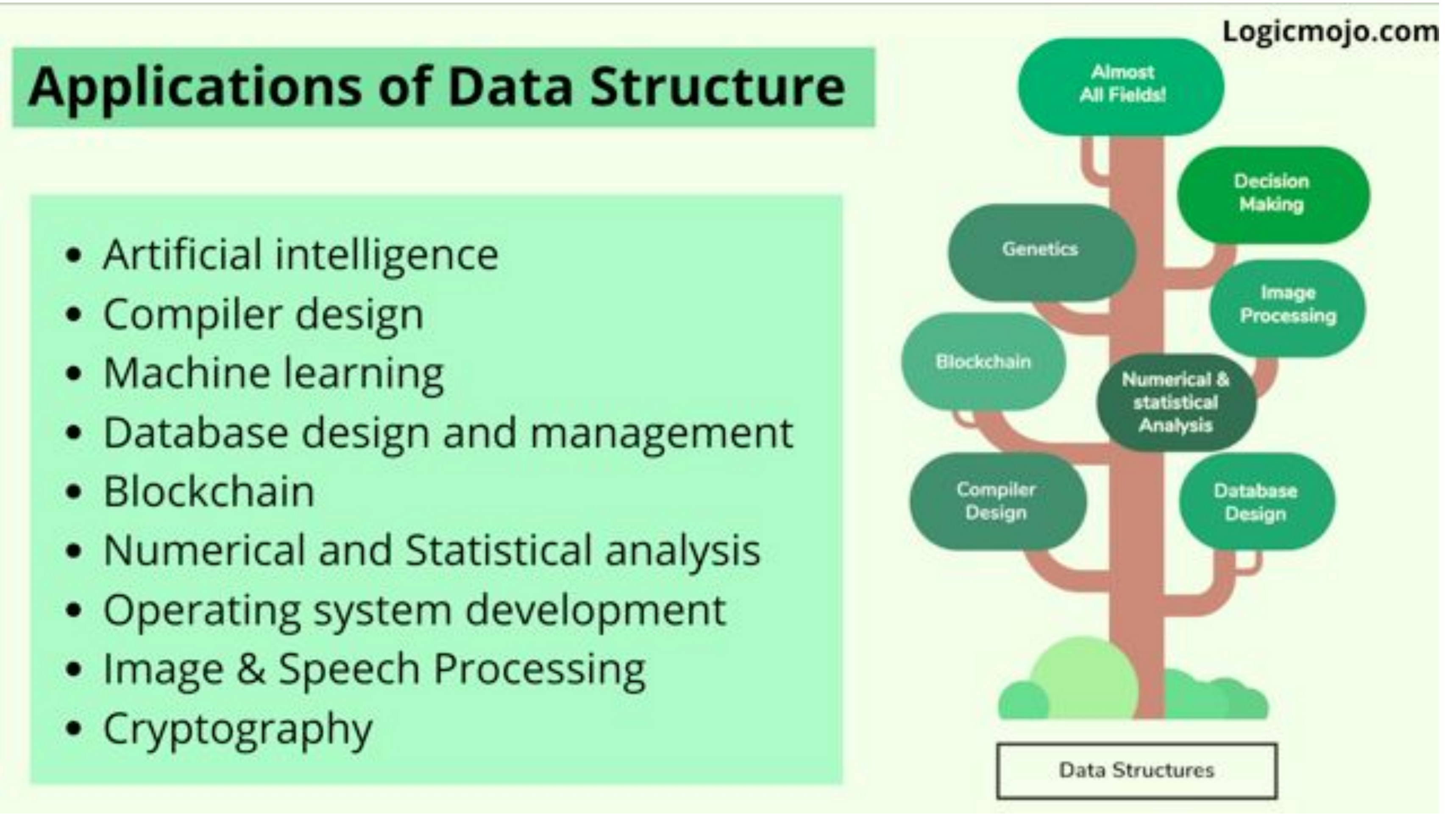
Importance of Data Structures

- Efficiency: Enables fast searching, sorting, and data manipulation.
- Memory Optimization: Reduces memory usage and improves performance.
- Foundation of Algorithms: Used in dynamic programming, graph algorithms, etc.
- Essential in Large-Scale Applications: Databases, operating systems, AI, etc.

Application Areas of Data Structures

- Software Development: Efficient data handling in applications.
- Databases & File Systems: Storing and retrieving large-scale data.
- Operating Systems: Memory management, process scheduling.
- AI & Machine Learning: Storing and accessing feature sets.
- Networking: Routing algorithms, data packet transmission.
- Bioinformatics: DNA sequence alignment, phylogenetic analysis.

Application Areas of Data Structures

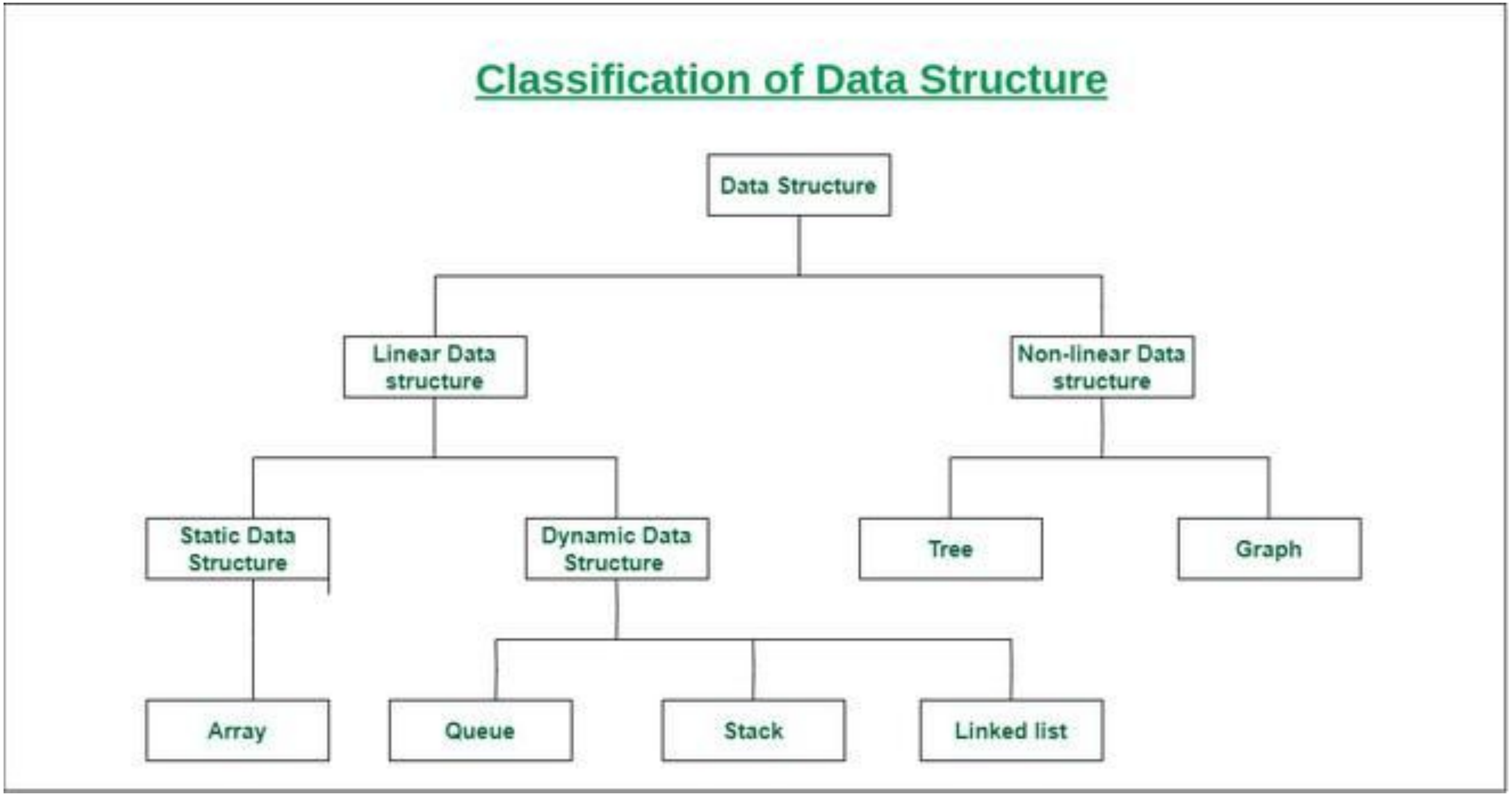


Dr. Machbah Uddin, Associate Professor, BAU, machbah.csm@bau.edu.bd

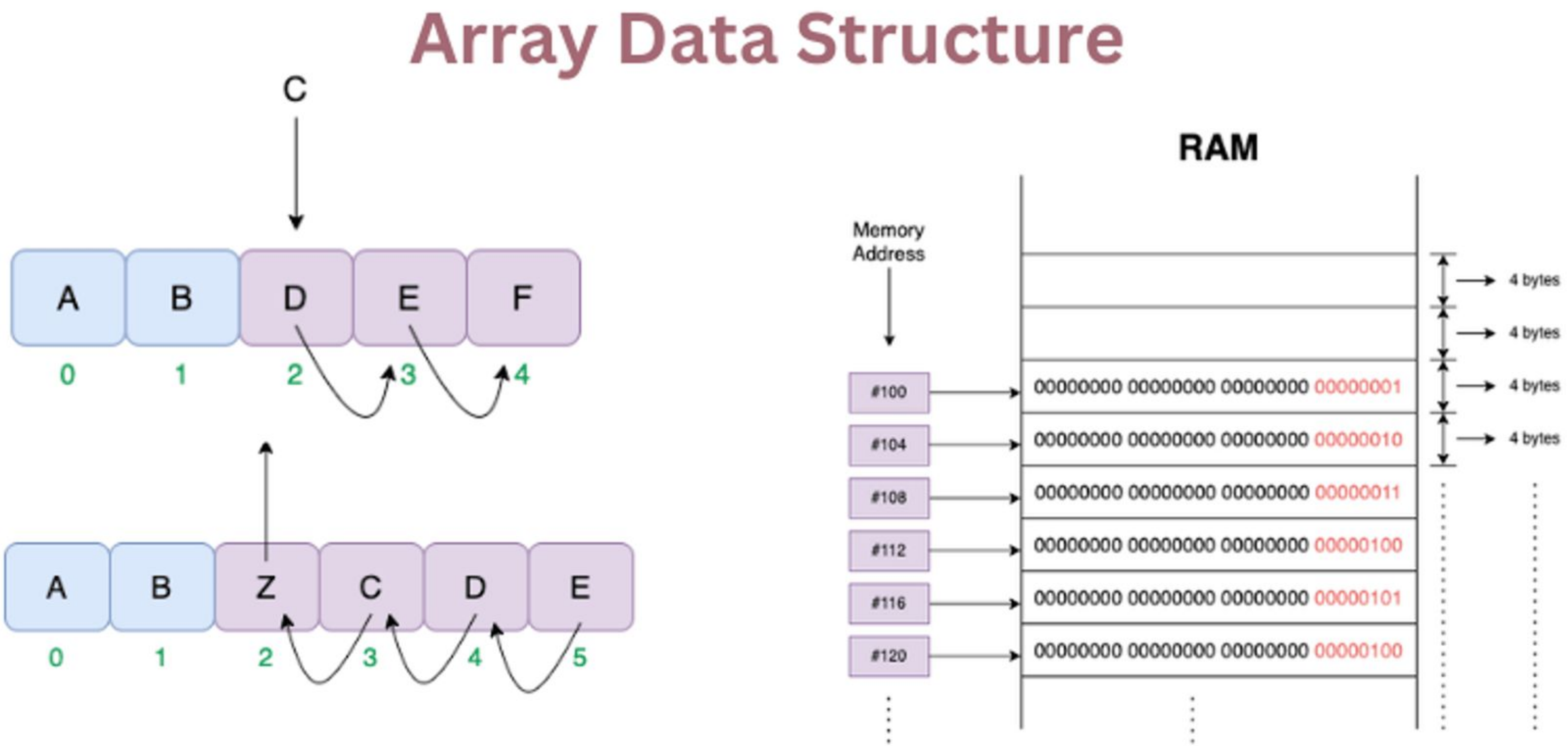
Classification of Data Structures

- Linear Data Structures: Arrays, Linked Lists, Stacks, Queues.
- Non-Linear Data Structures: Trees, Graphs.
- Hashing Techniques.

Classification of Data Structures

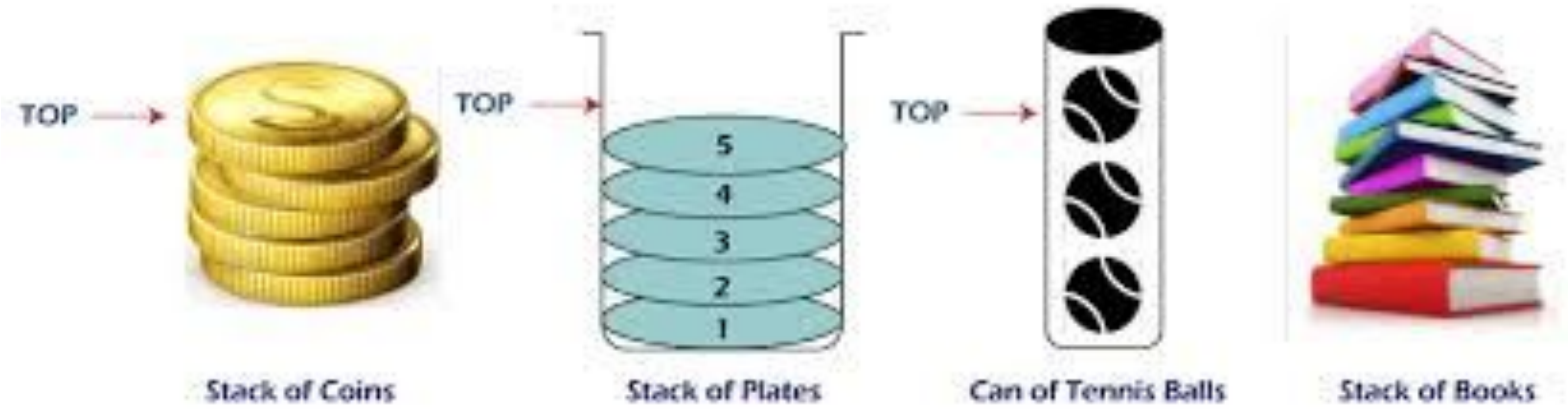


Array



Array Data Structure
C
A
B
D
E
F
0
1
2
3
4
A
B
Z
C
D
E
0
1
2
3
4
5
RAM
Memory Address
#100
#104
#108
#112
#116
#120
00000000 00000000 00000000 00000001
00000000 00000000 00000000 00000010
00000000 00000000 00000000 00000011
00000000 00000000 00000000 00000100
00000000 00000000 00000000 00000101
00000000 00000000 00000000 00000100
4 bytes
4 bytes
4 bytes
4 bytes

Stack

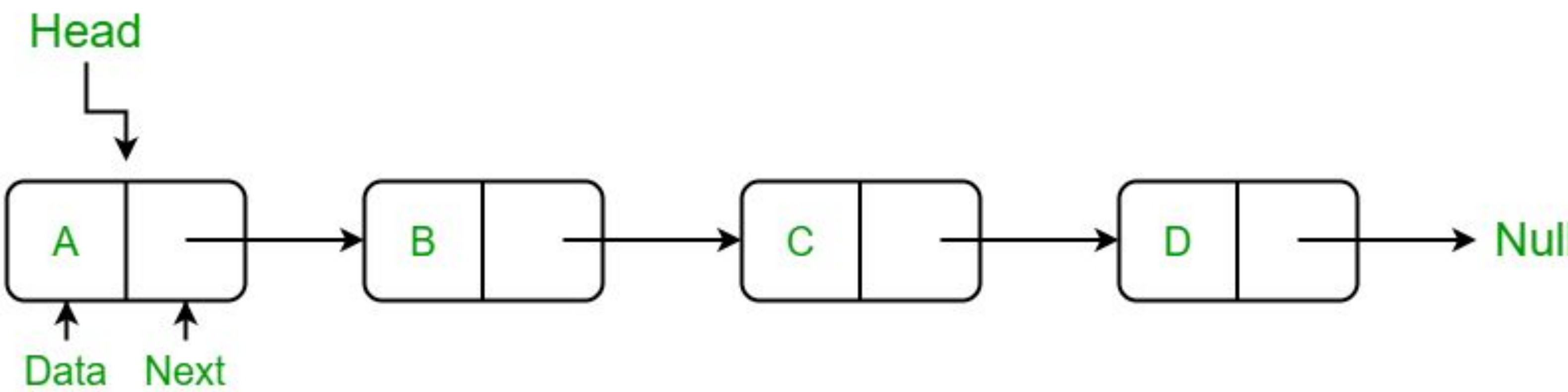


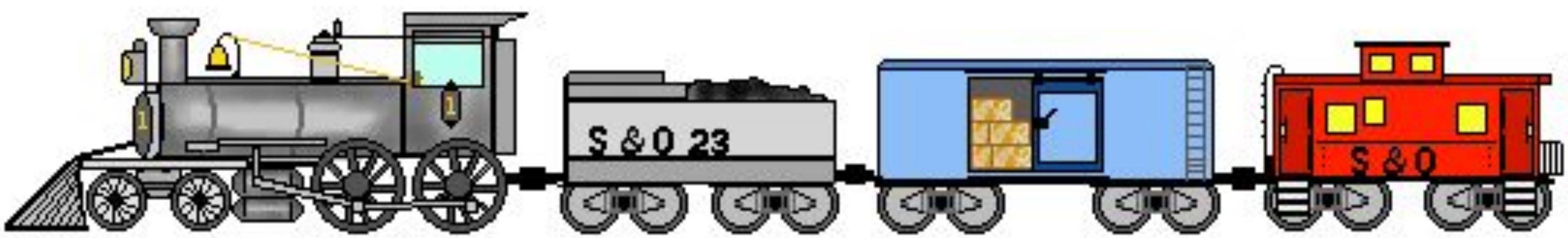
Dr. Machbah Uddin, Associate Professor, BAU, machbah.csm@bau.edu.bd

Queue

Dr. Machbah Uddin, Associate Professor, BAU, machbah.csm@bau.edu.bd

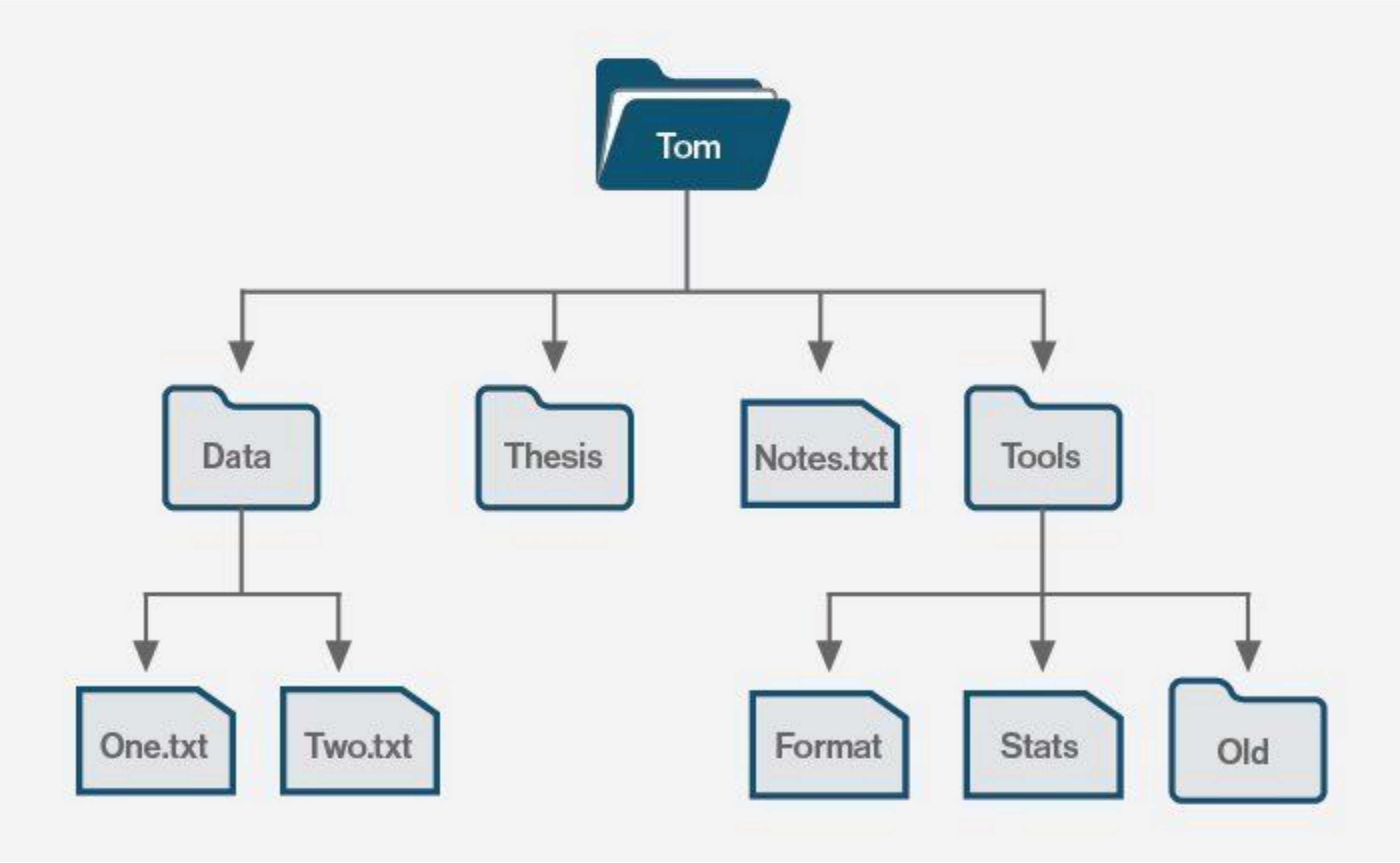
Linked List





Dr. Machbah Uddin, Associate Professor, BAU, machbah.csm@bau.edu.bd

Tree


Tom
Data
Thesis
Notes.txt
Tools
One.txt
Two.txt
Format
Stats
Old

Graph

Dr. Machbah Uddin, Associate Professor, BAU, machbah.csm@bau.edu.bd

Limitations of Data Structures

- Complexity: Some structures (e.g., graphs, trees) are difficult to implement.
- Memory Overhead: Linked structures require extra storage for pointers.
- Fixed Size (Arrays): Cannot dynamically grow or shrink.
- Time-Consuming Operations: Insertions and deletions may be expensive.

Challenges in Using Data Structures

- Choosing the Right Data Structure: Based on problem requirements.
- Scalability Issues: Some structures degrade in performance with large data.
- Optimization Trade-offs: Balancing speed vs. memory.
- Concurrency & Parallelism: Managing data in multi-threaded environments.

Summary

- Data structures improve computational efficiency.
- Linear vs Non-Linear structures.
- Applications in real-world scenarios.
- Challenges and limitations in practical use.

The Lost Artifact

- **Story:**
  A team of archaeologists has discovered an ancient underground maze where a legendary artifact is hidden. The maze consists of interconnected chambers, represented as a graph. Each chamber is a node, and the paths between them are edges. Some paths are one-way due to collapses. Your task is to help the archaeologists find the shortest path from the entrance to the chamber containing the artifact.
- **Input:**
  - A number of chambers (nodes) and paths (edges).
  - The start chamber (entrance) and the target chamber (artifact location).

The Lost Artifact

- **Solution Approach:**
- Use **Breadth-First Search (BFS)** to find the shortest path in an unweighted graph.
- If weighted paths are involved, use **Dijkstra's Algorithm**.

The Lost Artifact

```
from collections import deque

def find_shortest_path(n, edges, start,
target):
    graph = {i: [] for i in range(n)}
    for u, v in edges:
        graph[u].append(v)

    queue = deque([(start, [start])])
    visited = set()

    while queue:
        node, path = queue.popleft()
        if node == target:
            return path
        if node not in visited:
            visited.add(node)
            for neighbor in graph[node]:
queue.append((neighbor, path +
[neighbor]))

    return "No path found"

# Example usage:
n = 6
edges = [(0,1), (1,2), (2,3), (3,4), (4,5),
(1,3)]
start, target = 0, 5
print(find_shortest_path(n, edges, start,
target))
```

The Magic Spellbook

- **Story:**
  A young wizard has discovered an ancient spellbook, but the pages are cursed! The book must be read in reverse order to unlock its secrets. The wizard can only flip one page at a time. Given a sequence of pages read by the wizard, your task is to output the correct sequence in reverse order.
- **Input:**

A sequence of integers representing pages read in order.

The Magic Spellbook

- **Solution Approach:**
- Use a **Stack** (LIFO) to reverse the sequence.

```
def reverse_pages(pages):
    stack = []
    for page in pages:
        stack.append(page)

    reversed_order = []
    while stack:
        reversed_order.append(stack.pop())

    return reversed_order

# Example usage:
pages_read = [3, 5, 7, 9, 11]
print(reverse_pages(pages_read))  # Output:
[11, 9, 7, 5, 3]
```

The Treasure Hunt

- Story:A group of pirates is searching for hidden treasures scattered across an island. Each treasure has a value (gold coins) and a difficulty level (effort required to dig it up). The captain wants to prioritize treasures that give the most gold while requiring the least effort. Help the pirates decide which treasure to dig up first.
- Input:
  - A list of treasures, where each treasure is represented as (gold_coins, effort_required).

The Treasure Hunt

- **Solution Approach:**
- Use a **Priority Queue (Min-Heap or Max-Heap)** to prioritize treasures based on a gold-to-effort ratio.

```
import heapq

def prioritize_treasures(treasures):
    heap = []
    for gold, effort in treasures:
        heapq.heappush(heap, (-gold/effort, gold,
effort))  # Max heap based on gold/effort ratio

    best_treasures = []
    while heap:
        ratio, gold, effort = heapq.heappop(heap)
        best_treasures.append((gold, effort))

    return best_treasures

# Example usage:
treasures = [(100, 5), (200, 10), (50, 2), (300,
15)]
print(prioritize_treasures(treasures))
```

Questions & Discussion