

Introduction to Data Structures



Data structure – a way of organizing and storing data to enable efficient access and modification

Importance – Efficient data structures are crucial for optimizing the performance of algorithms and applications



Types of Data Structures

- linear data structures: Elements are arranged in a sequential manner (arrays, lists, stacks, queues)
- non-linear data structures: Elements are arranged in a hierarchical manner (trees, graphs)

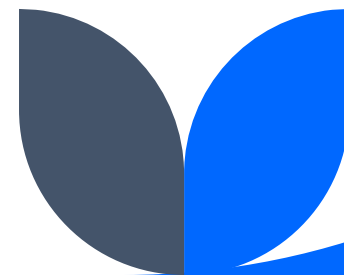
Benefits of Mastering Data Structures / Algorithms

- used an array (C++)(Java)
- dynamically sized data structures
- paramount to being a good programmer
- some data structures don't show up often – but ...
- DS / Alg are a toolkit inside your programming toolbox
- understanding their context – think about a problem from a different direction



“Learning DS / Algs is about becoming an ‘engineer’ instead of just being a programmer. It teaches you the ‘physics’ of programming and gives you technical mathematical concepts to ensure your solution is not only correct, but efficient and robust”

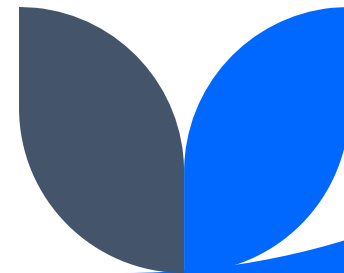
-- Elegant coding --





So ...

- Data structures make the code more efficient
- Data structures make the code easier to understand



List

- real-world example: simple to-do list application
- keep track of tasks you need to complete
- order they need to be completed in
- need to be able to add, remove and check off tasks



Python example:

```
todo_list = []
```

```
def add_task (task):
```

```
    todo_list.append(task)
```

```
    print("added ...")
```

```
def remove_task(task):
```

```
    try:
```

```
        removed_task=todo_list.pop(index)
```

```
        print("removed ...")
```

```
    except IndexError
```

```
        print("invalid ...")
```

```
...
```

completed_task, view_tasks, implementation of code

Linked List

- real-world example: Implement a basic contact list
- each contact has name, phone
- use linked list to store and manage contacts



Python example

class Node:

```
def __init__(self, name, phone):  
    self.name=name  
    self.phone=phone  
    self.next=None
```

class LinkedList:

constructor, add_contact, remove_contact, display_contacts

This class manages the linked list operations. LL's are linked by pointers to next node, have head node / tail node

Stacks

- real-world example: Web Browser History Manager
- uses a stack data structure to manage the forward and backward navigation history
- stack is perfect for this because it allows you to easily track the pages visited, and navigate back and forth between them using LIFO



Python example

- features: visit a new page, go back, go forward, display history
- components: stack for back and forward history – two stacks. Also – browser class – encapsulate the functionality of browser history manager
- more later

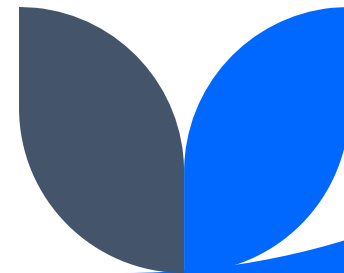
Queue

- real-world example: task management system
- tasks are processed in the order receive (FIFO)
- tasks: jobs in print queue, support tickets, scheduled tasks
- each task processed one at a time
- new tasks are added to the end of the queue
- system ensures tasks are handled in the order received



Python implementation:

- use deque from collections library
- class TaskQueue – add tasks, process tasks, complete tasks, view tasks



Trees

- real-world example: File system or Directory structure
- organized in a hierarchical structure
- makes it suitable for tree data structure – more later

Graphs

- real-world example: Representing and analyzing a social network
- find important individuals, detect communities, suggest new connections
- system will store and manipulate network data efficiently
- more later



Final note – using Generative AI's

