Data Structures using Python

Shriram K. Vasudevan
Principal,
K. Ramakrishnan College of Technology,
Samayapuram, Trichy,
Tamil Nadu

Abhishek S. Nagarajan
Data Scientist, [24]7.ai Innovation Labs,
Bangalore

Karthick Nanmaran
Assistant Professor, Department of CSE,
SRM Institute of Science and Technology, Chennai

OXFORD
UNIVERSITY PRESS

OXFORD
UNIVERSITY PRESS

Oxford University Press is a department of the University of Oxford.
It furthers the University's objective of excellence in research, scholarship,
and education by publishing worldwide. Oxford is a registered trade mark of
Oxford University Press in the UK and in certain other countries.

Published in India by
Oxford University Press
22 Workspace, 2nd Floor, 1/22 Asaf Ali Road, New Delhi 110002

First Edition published in 2021

ISBN-13 (print edition): 978-0-19-012408-3
ISBN-10 (print edition): 0-19-012408-3

eISBN-13 (eBook): 978-0-19-099236-1
eISBN-10 (eBook): 0-19-099236-0

Typeset in Times New Roman and Helvetica LT Std
by Ideal Publishing Solutions, Delhi
Printed in India by

Cover image: © Alok Rawat

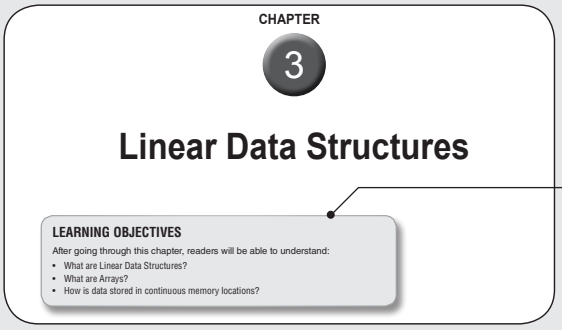
For product information and current price, please visit www.india.oup.com

Dedicated to

Baby Hasini, Sai Lakshmi and Master Saihari
Shriram K. Vasudevan

My parents, Usharani and Nagarajan
Abhishek S. Nagarajan

Features of


CHAPTER

3

Linear Data Structures

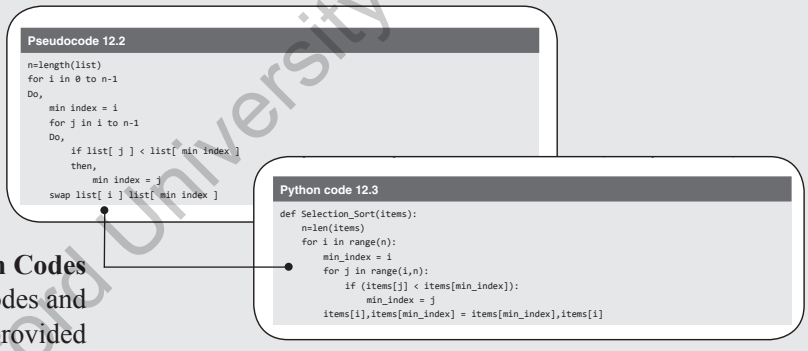
LEARNING OBJECTIVES

After going through this chapter, readers will be able to understand:

- What are Linear Data Structures?
- What are Arrays?
- How is data stored in continuous memory locations?

Learning Outcomes

Each chapter begins with learning outcomes listing the topics covered in detail.


Pseudocode 12.2

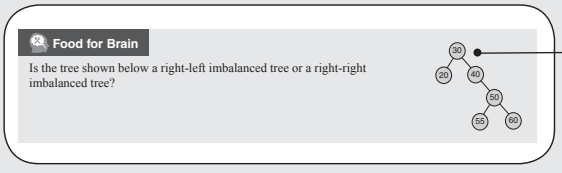
```
n=length(list)
for i in 0 to n-1
Do,
    min index = i
    for j in i to n-1
    Do,
        if list[ j ] < list[ min index ]
        then,
            min index = j
    swap list[ i ] list[ min index ]
```

Python code 12.3

```
def Selection_Sort(items):
    n=len(items)
    for i in range(n):
        min_index = i
        for j in range(i,n):
            if (items[j] < items[min_index]):
                min_index = j
        items[i],items[min_index] = items[min_index],items[i]
```

Pseudocodes and Python Codes

Numerous pseudocodes and Python codes have been provided to help readers improve their implementation skills.


Food for Brain

Is the tree shown below a right-left imbalanced tree or a right-right imbalanced tree?

Food for Brain

Mid-chapter 'Food for Brain' questions given in each chapter help readers think out of the box.

the Book

Question 8.2 Delete '50' from the Figure 8.5.

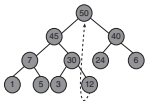


Figure 8.5 Initial max heap

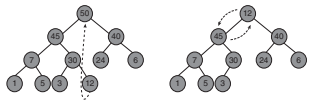


Figure 8.6 Initial max heap. Delete root '50' and replace with last leaf '12'. Property violation at the new root. Swap '12' and '45'. Swap '30' and '12' due to property violation Final max heap post-deletion

Mid-chapter Solved Questions with Pictorial Representation

Each chapter includes solved questions with pictorial representation of data structures' operation to ensure proper understanding of feature and visualization of transformation.

Chapter-end Exercises

The book comes with numerous objective questions with answers, theoretical review questions, exploratory application exercises, pictorial puzzles, and mini projects for self-check and practice.

Exercises

Multiple-choice Questions

1. An array is a collection of elements in _____ memory locations.
   (a) Distributed (b) Random
   (c) Composite (d) Continuous
2. A 'list' is a collection of elements of _____ type.
   (a) Same (b) Similar
   (c) Different (d) Custom/User Defined
3. Lists are _____ based.
   (a) Static allocation
   (b) Dynamic memory allocation
   (c) Fixed memory size
   (d) Not disclosed
4. A _____ is used to access an element in a list.
   (a) Subscript
   (b) Index
   (c) Relative position
   (d) All of the above
5. Reversal of a 'List' can be done using _____.
   (a) Additional List
   (b) Loops
   (c) Recursions
   (d) All of the above
6. _____ is the complexity of merging two lists of length 'm' and 'n'.
   (a) m (b) n
   (c) m+n (d) m*n
7. The last element from a Python list can be removed using the _____ function.
   (a) delete (b) remove
   (c) pop (d) del
8. All elements of lists are listed using _____ operator.
   (a) print (b) iterator
   (c) traversal (d) peek
9. A 'list' retains the order in which the elements are inserted, unless changed.
   (a) True (b) False
10. Python lists are implemented using OOP

Theoretical Review Questions

1. How are lists organized in memory?
2. How is memory allocated to lists when elements of different types are added dynamically?
3. Elaborate reversal of a list with illustration and example.
4. Explain the process of sorting elements in a list. How will elements of different type be sorted?
5. Explain some use cases of lists.

Exploratory Application Exercises

1. Write a code to merge two lists such that all odd indices contain elements from the first list and all the elements in even indices are from the second list.
2. Write a basic code to sort the given list in ascending order without using any built-in functions.
3. Develop a function to remove duplicates from a list.
4. Using two-dimensional arrays (nested lists) to implement an element counter. Given a list of elements, the final structure should have the distinct elements and the count of each element in a table format.
5. Implement 'find' and 'replace' as a function that finds a given element from a list and replaces all the instances with another element. In addition, try implementing 'find' and 'replace' with regular expressions for partial match and searches that are not case sensitive.

Picto Puzzles

1. 1 3 4 5 1 4 → 1 3 4 5
2. A1S9D3L → MODULE → ADLS139
3. 98413185 → 9 8 4 1 3 1 8 5

Mini Projects

1. Implement ADT for matrices along with arithmetic functions such as addition, subtraction

Key Points to Remember

- Linked lists are non–primitive data structures. It is a linear collection of elements or nodes not stored in continuous memory locations.
- A linked list has no particular order of motion. It has no constraint on data insertion and removal.
- The different types of linked lists are as follows:
  - Singly linked list
  - Doubly linked list
  - Circular linked list
  - Circular doubly linked list
- A singly linked list has only one pointer or link to the next node present in the list.
- In a doubly linked list, each node contains two pointers called 'NEXT' and 'PREV' pointing to the next and previous node, respectively.
- In circular linked lists, the last node is connected to the first node, that is, the last node does not contain 'Null' but contains a pointer to the first node of the list.
- In a circular linked list with one element, the node should point to itself.

Summary and Glossary

Quick recap of the concepts learnt and glossary of key terms are provided at the end of each chapter.

Key Terms

Linked lists A linear data structure designed to store elements in non-continuous memory locations using pointers from one memory location to the next location.
Singly linked lists/SLLs A linked list with pointer to traverse in forward direction. Every node will have a 'Next' pointer to move to the consecutive node, whereas there will not be any way to reach the previous node.
Doubly linked lists/DLL A linked list where every node has both 'Next' and 'Previous' pointers.
Circular linked lists A linked list where the last element points back to the first element to create a circular structure.
Head pointer The dedicated pointer that marks the first node/start point of the list.
Pointer-based linear data structures A data structure that uses pointers to associate a memory location to another memory location.

Preface

"Data is the new oil"

– Clive Humby

In this data-driven technical era, the amount of data that a company holds decides its value and impact on the society. However, handling the data has its own challenges. Data is absolutely useless if no meaningful information is extracted from it. This is where the concept of 'Data Structures' comes into play. Data structures are formats that aid in efficient access and modification of data. These were designed for efficient storing and processing volumes of data. The concept of data structures is completely inspired from real-world structures and solution formats. Data structures can be implemented in any language, but object oriented programming concepts add value to it.

Data structures play an important role in solving everyday problems. The choice of the apt data structure optimizes the solution of any problem to a great extent. With the power of data structures on its side, Python can claim to be the fastest growing, major programming language. Python is capable of handling data and along with data structures concepts it can perform miracles in the data world.

"Data Structures is a difficult subject" is a myth that needs to be disproven. It is a skill and a way of thinking that can be developed with practice. The book *Data Structures using Python* helps students in doing just that. It helps them to learn the concepts of various data structures, and enables them to start implementing their solutions to strengthen their algorithmic and implementation skills with the help of codes and exploratory questions given at the end of each chapter in this book.

About the book

Data Structures using Python is a textbook cautiously designed for undergraduate and post-engineering students of computer science, information technology, and allied disciplines. The core objective of this book is to introduce different types of data structures and make the readers strong in data structure application for solution implementation. It will also serve as a go-to reference book for professionals to understand important data structures widely used in the industry.

The book starts with highlighting the importance of data structures and slowly moves towards the idea of basic data structures. The evolution of data structures along with the motivation of each structure from real-life objects is analogically explained in the book to aid in faster and clearer understanding for the readers. Data structures are broadly classified as continuous memory-based, pointer-based, hierarchical, non-linear, and non-hierarchical. The chapters are also organized in the same evolutionary order to enhance understanding of concepts.

Each data structure is first explained, followed with a question and its solution. Then, the algorithm/pseudocode is shared and finally the Python implementable code is provided. When there is scope for Python's capabilities to optimize the algorithm further, a concise version of the code is added as well. This particular order of explanation/code is designed in such a way that the readers can clearly comprehend the concepts and implement the same, without which the learning curve of data structures stays incomplete. All these programs have already been implemented and tested using Python 3.6 in Anaconda and Python 3 compilers online. The book also has an Appendix on the useful Python functions and libraries. To further enhance the understanding of

the subject, application ability and analytical ability of the students, there are numerous objective, subjective, and programming exercises at the end of each chapter. Above these, each chapter has a set of innovative pictorial puzzles, where the logic behind the questions to be solved and then coded using data structures is given. Finally, each chapter has a set of mini projects, which include some real-world complex problems where data structures have come to the rescue.

Salient Features

The salient features of the book include:

- *Simple and lucid explanations* for complex Data Structures concepts using analogy of real-world objects/systems.
- *Pictorial representation and problem solving* of each data structures operation to ensure proper understanding of feature and visualization of transformation.
- *Base case analysis* with pictorial solving for important corner cases of primary algorithms.
- Plenty of implementable *programs* to help readers improve their implementation skills.
- Mid-chapter exploratory questions in the form of *food for brain* to help think out of the box and explore upcoming concepts on their own.
- Implementation promoting coding style where *pseudocode* is given before *Python codes* using which the readers can try out the code before referring to the given snippet.
- *Case-studies* to show the power of each data structure within each chapter.
- *Evolutionary style explanation* where every data structure talks about the problem in the previous structure and how it is overcome with the abilities of the next structure.
- Abundant and variety of chapter-end exercises including *objective questions* with *answers, theoretical review questions, exploratory application exercises, pictorial puzzles*, and *mini projects* for self-check and practice.
- *Glossary* of key terms and point-wise *summary* at the end of each chapter to help readers quickly revise important concepts learnt.

Organization of the book

The book is divided into 13 chapters and one appendix.

Chapter 1 provides an introduction to Data Structures, why the concept was introduced, and how to read Data Structures.

Chapter 2 discusses the concept of Abstract Data Type and how is it useful in implementing data structures. It introduces the Asymptotic Notations which is the way to measure the cost of an algorithm. The chapter also deals with recursive functions and methods to measure their performance.

Chapter 3 deals with the most primitive and simplest data structure – Array, and operations on arrays. It also describes various built-in functions for Python Lists.

Chapter 4 is dedicated to linear data structures which are traditionally continuous memory- based, and implemented based on Arrays/List – Stack & Queue. It deals with different types of Queue structures along with the behavioural functionalities of all the structures.

Chapter 5 takes up the first non-continuous structure. The different types of Linked Lists, their implementation, and working functions are discussed in this chapter. Since Python does not have the concept of pointers, this chapter shows how nested objects are used to implement non-contiguous structures.

Chapter 6 unleashes the concept of hierarchical data structures. The concept of associating the object to multiple objects makes non-linear data structures possible. 'Tree' is the basic non-linear, hierarchical data structure and this chapter introduces the basic tree concepts and behaviour.

Chapter 7 presents one of the most widely used hierarchical data structures. It deals with different types of Search trees, where there is a constraint with respect to organizing elements in a tree. For all the search trees, the advantages of every structure over the previous are clearly depicted along with the operations aiding the apt use of the structure in data intensive applications.

Chapter 8 shows how a non-linear tree structure can be used for a dynamic linear requirement. This chapter is mainly based on Heaps which a binary tree-like structure using which priority queues are implemented at optimal cost.

Chapter 9 covers other non-linear data structures such as Trie, Sets, Hash Tables, and Dictionaries. Trie is a complex form of tree and hash table is used to aid faster storing and searching of data. Other structures deal with association of various data points, without any particular associativity amongst themselves.

Chapter 10 is dedicated to the B+ Tree structure which is widely used in data storage. It shows various functionalities and explains why it is faster while handling data. Other than B+ Tree, it also throws light on different data structures used in various instances of operating systems.

Chapter 11 handles the non-linear, non-hierarchical data structure, Graphs. First, the various representations of graph along with their implementations are shared followed with various algorithms for problems. Algorithms like connectivity, topological sorting, minimum spanning tree, and shortest distance between any given pair are all explained to aid realistic problem solving.

Chapter 12 explains various sorting techniques. Sorting is the technique of arranging the elements of a list in a specific order. This chapter deals with a wide range of sorting algorithms which vary in performance. Sorting techniques such as bubble sort, selection sort, insertion sort, and distribution sort are discussed. Non-sorting problem solving with techniques like divide and conquer is also explained.

Chapter 13 is dedicated to efficient accessing techniques. Binary search which utilizes the power of a sorted data is shared in this chapter. This chapter also shows how structures like tree and hash table can be used to search data efficiently.

The **Appendix** at the end of the book discusses some of the important utility functions and libraries in Python. This can help in optimizing the implementation codes.

Acknowledgments

The writing of this textbook was a mammoth task for which a lot of help was required from many people. We take this moment to thank all our family and friends who supported us in completing this book.

We would like to sincerely thank and acknowledge [24]7.ai family for their support in writing the book and Abhilaash Nagarajan, Dinesh S, and Apurva Mandalika for extending their help towards organizing the content and testing programs.

Last but not the least, we would like to thank the editorial team at the Oxford University Press, India for their help and support.

Comments and suggestions for the improvement of the book are welcome. Please write to us at shriramkv@gmail.com.

Shriram K. Vasudevan
Abhishek S. Nagarajan
Karthick Nanmaran

Detailed Contents

CHAPTER

1

Data Structures—Introduction

LEARNING OBJECTIVES

After going through this chapter, readers will be able to understand:

- What is a data structure?
- Why are data structures needed?
- How to study and understand data structures?
- How are data structures implemented?
- Why do data structures appear difficult?
- Real-life scenarios for data structures
- DBMS vs Data Structures

1.1 INTRODUCTION

Before taking a deep dive into the programming and algorithmic aspects of data structures, it is inevitable for someone to understand certain fundamental things. What is a data structure? Why do we need a concept called data structure? How to prepare for this subject? Why does it appear to be one of the toughest courses of all times in computer science? How to select your appropriate data structure? All these points shall be clarified in this chapter. It shall serve as a platform for understanding the rest of the chapters.

1.2 WHAT IS A DATA STRUCTURE?

A data structure is a format for storing data in an organized manner. The readers could even be surprised to know that they have already been familiar with data structures. If terms such as array, record, or file are familiar to the reader, then data structure is also familiar.

The next question in the minds of the readers would be, why should the data be organized or why do we need data structure? Assume that you have a rack with a lot of books catering to different subjects such as Computer Science, Biology, and Physics. The books are not organized and randomly piled up. One can refer to Figure 1.1 to understand what is being portrayed.

It is very easy to interpret that picking out one book on a particular topic from the unorganized rack is a tough task and a nightmare if the table is huge with more number of books in place.

Figure 1.1 An unorganized rack of books

Coming to the next scenario, assume a rack which is neatly organized with books properly sectioned as shown in Figure 1.2. The question now is, how much difficult it would be to spot a book from this organized rack? It would not take much time and shall be very easy. This is the difference a reader should understand.

Figure 1.2 An organized rack

Coming back to computer science, data structure is all about organizing, managing, and storing data. This shall enable efficient access with increased ease of access. Let us get the understanding better with more discussion.

1.3 WHY DO WE NEED DATA STRUCTURES?

Most of the interviews, examinations, and discussions will certainly have this question. The understanding of the need for data structures shall inculcate the interest in the reader to learn the subject and concepts deeper. Data structure is a method/technique for storing and organizing the information in a computing

machine. Since it is all organized and well maintained, storage is properly taken care of and the data being searched for can be retrieved very easily and fast, hence effectively increasing the productivity. There is no single type of data structure that the reader is going to be introduced with. There are many types of classifications and each of these is unique. Data structures are generally classified as primitive and non-primitive and one can choose the appropriate option based on the requirements and suitability.

Experienced programmers shall agree to this point that the efficiency and speed at which a program runs depends on the choice of data structures and its implementation. Hence, data structures are important to organize data and to make sure the retrieval happens fast during data search.

1.4 HOW TO STUDY/PREPARE DATA STRUCTURES? WHY DOES IT APPEAR DIFFICULT?

First and foremost point to be understood is that 'data structure is not tough'. It is like any other subject which needs attention and a bit of patience to understand the flow. Every concept in a data structure has to be related to a real-time/real-life example. This approach would solve half the problem. Every data structure is inspired by some real-life scenario and this is well articulated in this book. For instance, if you take the data structure 'queue', it is inspired by the queue we stand in everyday for one purpose or other. In case of 'stack', bread slices can be cited as an example. Likewise, there are many real-life examples present in our day-to-day situations and all that is required is to correlate the examples to these data structures.

To make it simple, first identify the example related to a particular data structure, understand it and then navigate to technical learning. Realizing the data structure through implementation is very important. For implementation and to build the code, C, C++, or Python are preferred. Hence, the fundamental knowhow of any of these languages is a must. Here, in this book, we have used Python and it is one of the best options.

Data Structures = Concepts + Programming Skills

1.5 DIFFERENT TYPES OF DATA STRUCTURES

There are three major types of data structures:

1. Linear data structures
2. Non-linear, hierarchical data structures
3. Non-linear, non-hierarchical data structures

As the name suggests, in linear data structures, every data point can associate itself to a maximum of two data points only, one before and one after it. Here, the data points can be in continuous memory location. However, it is not mandatory to have the data points in continuous memory location. In non-linear data structures, the data points can never be stored in continuous memory locations as every data point can be associated with more than two data points. In hierarchical data structures, the data points maintain a hierarchical relationship among themselves. The last form of non-hierarchical is the most random form of data structures. Here, any data point can be associated with anything and the complete relationship is to be captured in a single structure.

1.6 HOW TO SELECT A DATA STRUCTURE?

Selection of a data structure for a scenario is completely based on the scenario and answering the following questions in order will help:

1. What is the associativity/relation among the data points?
2. Does the order of element have any significance in the solution?

3. Does the solution involve any order while processing, that is, can any element be modified directly or will there be any constraints on it?
4. What is the most frequently done operation in the solution?
5. Is there any specific behaviour expected from the structure in the solution?

Every time when a data structure is to be selected, do not feel that a particular feature of the data structure is an obstacle in the solution. Instead, pick the closest data structure and modify it as required. Always remember that a data structure is used to find a solution and the problem/solution will usually not be tailor-made for it. You will need to find a closest match.

1.7 HOW ARE DATA STRUCTURES IMPLEMENTED?

Data structures are implemented based on the object oriented programming (OOP) methodology. The features and associativity are defined in the form of classes and applied for the solution. It uses the OOP concept of abstraction and functional programming more. By defining the behaviour as functions inside classes, code reusability is achieved. For a user, it is sufficient if the behaviour of the data structure is known. It is not mandatory for the user to understand how the particular feature is defined. Thus, the functions inside the class abstract the unwanted information from the user. These are achieved by defining data structures as ADTs, called Abstract Data Types, explained in detail in Chapter 2.

1.8 REAL-LIFE SCENARIOS FOR DATA STRUCTURES

Figures 1.3-1.6 show the data structures that you see in the real-life elements every day. This is the approach used in the book throughout. Figure 1.3 is a train which is exactly the concept behind the linked lists. Figure 1.4 has a set of books piled, which is the idea behind the stack data structure. Needless to say, Figure 1.5 shows a tree which was behind the tree concepts in data structure. Followed by these is the Queue in data structure which is inspired through a real-life queue (Figure 1.6).

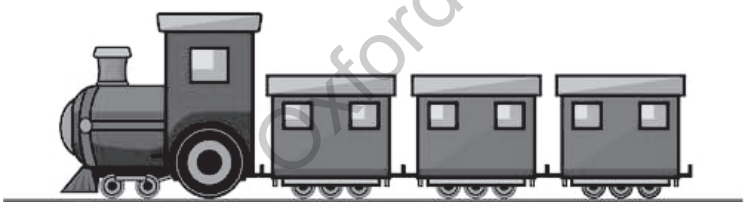


Figure 1.3 Train as 'Linked Lists'

Figure 1.4 Piled-up books as 'Stack'

Figure 1.5 Tree as 'Tree Data Structure'

Figure 1.6 Real-life queue—'Queue Data Structure'

Considering the above points, it can be understood that learning data structure with appropriate examples and concepts in place would be easier. The complete book is framed with a lot of examples and real-life references for detailed, fast learning. Examples of objects such as marbles, playing cards, buckets, etc. have been used in this book for easy understanding of concepts.

Food for Brain

Find some more examples of real-time data structures.

Question 1.1 How to represent the relationship of a grand-father to father to son while processing?

Solution: The relationship is shown in Figure 1.7.

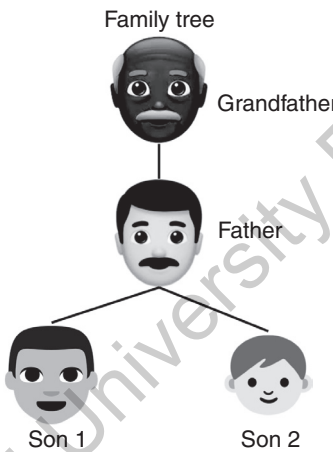


Figure 1.7 Relationship tree

Question 1.2 How to represent few cities and distance between them in order to plan a road trip easily using the information?

Solution: The information for the road trip is shown in Figure 1.8.

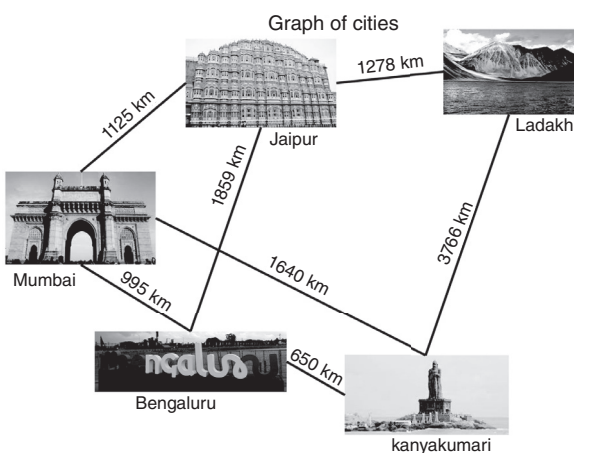


Figure 1.8 Road trip linked list charts

1.6 DIFFERENCE BETWEEN DATA STRUCTURES AND DATABASE MANAGEMENT SYSTEMS

All real-time data can be organized as objects, with each object having a lot of features. Multiple objects can have similar features if they ideally belong to the same class. At one time, two different objects may have a relationship. Database is a collection of similar objects. A table is a collection of similar objects, where each row is an object. The relationship among objects is represented as relation between tables. All the fields in the table are the features of that particular object. Database deals with permanent memory and stores data and relationship in real time, whereas data structure functions only at run time. Data Structures are more suitable for organizing data for efficient processing. In memory, all data is nothing but a collection of memory locations and values in them. In RAM, how these memory units are associated for efficient processing is all about data structures. They are not a permanent storage. In database, processing constraints cannot be applied as it only handles organizing and storing data by establishing relationships. All processing constraints are from data structures, with the key intent of improving performance of a solution.

Key Points to Remember

- A data structure is a format for storing the data in an organized manner.
- Data structure is all about learning how to store and retrieve data in an effective and efficient manner.
- The simplest data structure is an array.
- Data structures can be classified as primitive and non-primitive data structures.
- Only when the real-life examples are connected with learning, it becomes easier.
- Data structure is a combination of concepts and programming approach to implement the concepts.
- Things and components existing in real life can be related easily to data structures. Simple examples start with train, trees, books on the table, etc.

Exercises

Multiple-choice Questions

1. Data structures are broadly classified into _____categories.
   (a) 2 (b) 5
   (c) 3 (d) None of the above
2. What is the most important point while selecting the data structure for a problem?
   (a) Associativity among the data points
   (b) Most common operation optimally required in the solution
   (c) Any behaviour required while storing and retrieving the information
   (d) All of the above
3. Linear data structures can be significantly differentiated into how many types?
   (a) 2 (b) 3
   (c) 4 (d) 5
4. A data point can always be associated to a maximum of two data points only?
   (a) True (b) False
5. Data structures' behaviour is implemented in classes to achieve_____.
   (a) Abstraction
   (b) Functional programming
   (c) Polymorphism
   (d) Object oriented programming methodology

Theoretical Review Questions

1. Define data structures.
2. Explain why data has to be structured. What are the implications one would face in case the data is not structured.
3. Express your views about the connection between data structures and speedy execution.

4. Mention the types of data structures you are aware of.
5. Mention some of the real-life elements which can be connected to data structures.

Exploratory Application Exercises

1. Write a function to sort the given numbers.
2. From a file with one million numbers, find all the given numbers.
3. Develop a function for enquiry of subject's registration so that while enquiring for a process the prerequisites can be directly checked and registered if not done.
4. Write a program to check if a given string is a palindrome or not.
5. Represent places and distances in a structure and identify the shortest distance between any two places.

Picto Puzzles

Identify the logic for all the picto puzzles given below.

1. DATA STRUCTIRE ⟶ ERUTCURTS ATAD
2. '3 + (5 * 4)' ⟶ 23
3. '3x + 4y' + '4x – 7y' ⟶ 7x – 3y
4. 1 2 3 4 5 ⟶ 5 4 3 2 1
5. 7 2 9 4 3 ⟶ 2

Mini Projects

1. Develop a system to represent places and the distance among them. The system must also be able to identify if every place is connected to every other place either directly or indirectly.
2. Design a module to handle the tasks in an operating system. It should have the ability to take in new jobs as they are created along with a priority. Based on the priority, the next job is executed. Every job will have the total time for execution as well. When a job is waiting for more than a threshold, its priority will increase further. This module will act as a scheduler and return the order in which the jobs will be executed.
3. Design a structure to represent a family tree. The structure should maintain the hierarchy properly. It should also have a function to print the members in hierarchy.

Answers to Multiple-choice Questions

1. (c) 2. (d) 3. (a) 4. (b) 5. (d)