IT2001
Data Structures
and
Algorithms

Sraban Kumar Mohanty

Slides and figures have been collected from various publicly available Internet sources for preparing the lecture slides of IT2001 course. I acknowledge and thank all the original authors for their contribution to prepare the content.

Introduction

- A famous quote: Program = Algorithm + Data Structure.
- All of you have programmed; thus, have already been exposed to algorithms and data structures.
  - Perhaps you have not seen them as separate entities.
  - Perhaps you have seen data structures as simple programming constructs (provided by libraries).
  - However, data structures are quite distinct from algorithms, and very important in their own right.

Data Structures and Algorithms

- Study of Data Structures & Algorithms: Fundamental to Computer Science
- Not only Computer Science and Engineering, but also other allied engineering disciplines such as
  - Computer Integrated Manufacturing,
  - Product Design,
  - Commerce, and
  - Communication Engineering, to list a few

Data Structures and Algorithms

- It is offered as a core or an elective course, enabling students to have the much-needed foundation for efficient programming, leading to better problem-solving skill.

What The Course Is About

- Foundations of Algorithm Analysis and Data Structures.
  - Algorithm: sequence of steps that results in the performance of a specific task
  - Data Structures: Organization of data needed to solve the problem
    - Data representation: int, float, char, double , string: usually an array of char
    - Program – an implementation of an algorithm in some programming language

What The Course Is About

- Analysis:
  - How to predict an algorithm's performance
  - How well an algorithm scales up
  - How to compare different algorithms for a problem
- Data Structures
  - How to efficiently store, access, manage data
  - Data structures effect algorithm's performance

Data Structures and Algorithms

- Efficient problem-solving using computers, irrespective of the discipline or application, calls for the design of efficient algorithms
- Inclusion of appropriate data structures is of critical importance to the design of efficient algorithms
- In other words, good algorithm design must go hand in hand with appropriate data structures for efficient program design to solve a problem

Example Algorithms

- Two algorithms for computing the Factorial
- Which one is better?

```
int factorial (int n) {
  if (n <= 1)    return 1;
  else   return n * factorial(n-1);
       }
```

```
int factorial (int n) {
     if (n<=1)    return 1;
         else {
             fact = 1;
             for (k=2; k<=n; k++)
                 fact *= k;
                 return fact;
              }
        }
```

Examples of famous algorithms

- Euclid algorithm
- Newton's root finding
- Fast Fourier Transform
- Compression (Huffman, Lempel-Ziv, GIF, MPEG)
- DES, RSA encryption
- Simplex algorithm for linear programming
- Shortest Path Algorithms (Dijkstra, Bellman-Ford)
- Error correcting codes (CDs, DVDs)
- TCP congestion control, IP routing
- Pattern matching (Genomics)
- Search Engines

Role of Algorithms in Modern World

- Enormous amount of data
  - E-commerce (Amazon, Ebay, Flipcart)
  - Network traffic (telecom billing, monitoring)
  - Search engines (Google)
  - Database transactions (Sales, inventory)
  - Scientific measurements (astrophysics, geology)
  - Sensor networks. RFID tags
  - Bioinformatics (genome, protein bank)

Data Structures

- How does Google find the documents matching your query so fast?
  - Uses sophisticated algorithms to create index structures, which are just data structures.
- With the amount of data created by the new technologies, the need to organize, search, and update MASSIVE amounts of information FAST is more severe than ever before.

Why study Data Structures (and algorithms)

- Using a computer?
  - Solve computational problems?
  - Want it to go faster?
  - Ability to process more data?
- Technology vs. Performance/cost factor
  - Technology can improve things by a constant factor
  - Good algorithm design can do much better and may be cheaper
  - Supercomputer cannot rescue a bad algorithm
- Data structures and algorithms as a field of study
  - Old enough to have basics known
  - New discoveries
  - Rapidly increasing application areas

Data Structures

- Data
  - Simply values or set of values
  - Group items: data items that are divided into sub-items, e.g. name
  - May be organized in many different ways
- Information: meaningful or processed data
- Data Structures
  - The logical or mathematical model of a particular organization of data
  - The choice of a particular data model depends on two considerations
    - It must be rich enough in structure to mirror the actual relationships of the data in real world
    - It should be simple enough that one can effectively process the data when necessary

What is Data Structure

- Data Structure can be defined as the group of data elements which provides an efficient way of storing and organizing data in the computer so that it can be used efficiently.
- Examples of Data Structures are:
  - Arrays, Linked List, Stack, Queue, Trees, Graphs etc.
- Data Structures are widely used in almost every aspect of Computer Science
  - E.g., Operating System, Compiler Design, Artificial intelligence, Graphics, Image processing and many more.

What is Data Structure

- Data Structures are the main part of many computer science algorithms as they enable the programmers to handle the data in an efficient way.
- It plays a vital role in enhancing the performance of a software or a program as the main function of the software is to store and retrieve the user's data as fast as possible

Data Structures

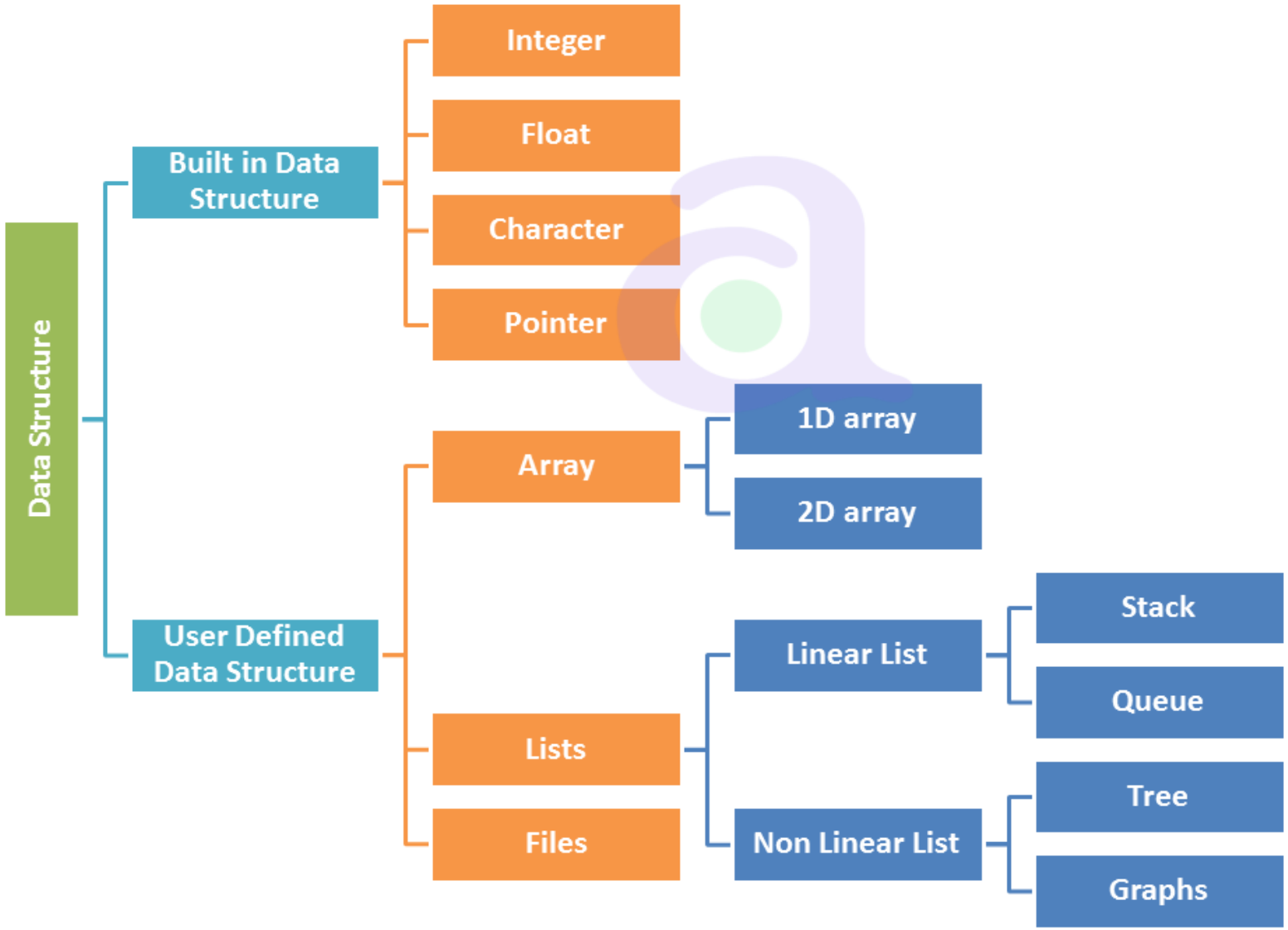
- The representation of a particular data structure in the main memory of a computer is called as storage structure.
- The storage structure representation in auxiliary memory is called as file structure.
- **Algorithm + Data Structure = Program**
- **Data Structure study covers the following points**
  1) Amount of memory require to store
  2) Amount of time require to process
  3) Representation of data in memory
  4) Operations performs on data

Operation on Data Structures

- Design of efficient data structure must take operations to be performed on the DS into account. The most commonly used operations on DS are broadly categorized into following types

1. **Create:** This operation results in reserving memory for program elements.
    1. Creation of DS may take place either during compile-time or run-time.
2. **Destroy:** This operation destroys memory space allocated for specified data structure .
3. **Selection:** This operation deals with accessing a particular data within a data structure.
4. **Updation:** It updates or modifies the data in the data structure.
5. **Searching:** It finds the presence of desired data item in the list of data items, it may also find locations of all elements that satisfy certain conditions.
6. **Sorting:** This is a process of arranging all data items in a DS in particular order, for example either ascending order or in descending order.
7. **Splitting:** It is a process of partitioning single list to multiple list.
8. **Merging:** It is a process of combining data items of two different sorted list into single sorted list.
9. **Traversing:** It is a process of visiting each and every node of a list in systematic manner.

Source: https://www.researchgate.net/publication/347303319_Data_Structure_Ppt



Types Of DS

The DS are divided into two types:

1) Primitive
2) Non primitive

Non primitive divided into two type

1) Linear DS
2) Non linear DS

DATA TYPES

- **Primitive Data Structure**

- Primitive Data Structure are basic structure and directly operated upon by machine instructions.
- Primitive data structures have different representations on different computers.
  - Integers, floats, character and pointers are example of primitive data structures.
- These data types are available in most programming languages as built-in type.
  - **Integer:** It is a data type which allows all values without fraction part.
  - **Float:** It is a data type which is use for storing fraction numbers.
  - **Character:** It is a data type which is used for character values.
  - **Pointer:** A variable that hold memory address of another variable are called pointer.

Non Primitive Data Type

- These are more sophisticated data structures.
- These are derived from primitive data structure.
- The non – primitive data structures emphasize structuring of a group of homogeneous or heterogeneous data items.
- Example of non – primitive data types are: Array, List, and File etc.
- A non – primitive data type is further divided into
  - Linear and non – Linear data structure.

Array: An array is a fixed size sequenced collection of elements of the same data type.

List: An ordered set containing variable number of elements is called as List.

File: A file is a collection of logically related information. It can be viewed as a large list of records consisting of various fields.

Course Structure

- Notion of Algorithm, Space and Time Complexity, Analyzing algorithms
- Static & Dynamic Memory Management: Arrays, Stacks, Queues, Linked Lists
- Trees, Binary Trees, Tree Traversals, Applications of Binary Tree
- Graphs and their representations, Graph Traversal Algorithms, Minimum Spanning Tree, Shortest Paths
- Searching Algorithms: Sequential Search, Binary Search
- Sorting Algorithms: Quick sort, Merge sort, Insertion sort, Selection sort, Heap & Heap sort
- Binary Search Tree, Balanced Tree, AVL Tree
- Files, Indexing: Hashing, Tree Indexing: B-tree
- Basic Algorithm Design Paradigms: Divide & Conquer, Greedy method, Dynamic Programming, Back tracking, Branch and Bound [Discussion with the help of some example which are already discussed].

Text Books

- Textbook
  - Introduction to Algorithms, Cormen, Leiserson, and Rivest, MIT Press/McGraw-Hill, Cambridge (Theory)
  - *Fundamentals of Data Structures* by Ellis Horowitz, Sartaj Sahni, Galgotia Booksource
- References
  - *Data Structures and Algorithm Analysis in C or C++ by Mark Allen Weiss*
  - *Data Structures* by Seymour Lipschutz, Schaum's Outlines, TMH
  - The C Programming language, Kernigham & Ritchie
  - Other material will be posted
  - Course home page web.iiitdmj.ac.in/~sraban

Grading Scheme

- Assignments: for your practice only
- Quiz1: 10%
- Quiz2:10%
- Mid Sem: 25%
- End Sem: 35%
- Lab work: 20%

Algorithm

- An algo is a sequence of computational steps that transform the input into output
  - The statement of the problem specifies in general terms the desired input/output relationship
  - The algo describes a specific computational procedure for achieving that input/output relationship
- An algorithm can be specified
  - in natural language like English,
  - as a computer program, or
  - as a hardware design
- The only requirement is that the specification must provide a precise description of the computational procedure to be followed

Algorithm

- Example: sorting
  - Fundamental operation in CS; a number of sorting algorithms are available
  - Which algorithm is best for a given application depends on a number of factors:
    - The number of items to be sorted
    - The extent to which the items are already sorted
    - Possible restrictions on the item values
    - The kind of storage device to be used
- Goal: To learn techniques of algorithm design and analysis so that you can
  - Develop algorithms on your own
  - Show that they give the correct answer, and
  - Understand their efficiency

Algorithm as a Technology

- Suppose computers are infinitely fast and computer memory is free
  would you have any reason to study algorithms?
- You would still like to demonstrate that your solution method terminates and does so with the correct answer
- Of course, computers may be fast, but not infinitely fast and memory may be cheap, but it is not free
- Computing time and memory space: bounded resources
- These resources should be used wisely, and algorithms that are efficient in terms of time/space will help you do so
- Algorithms devised to solve the same problem often differ dramatically in their efficiency
- These differences can be much more significant than differences due to h/w and s/w

Algorithm as a Technology: Efficiency

- Consider two sorting methods:
  - Insertion sort
    - Takes time roughly equal to c_1n^2 to sort n items
    - c_1 is a constant that does not depend on n
  - Merge sort
    - Takes time roughly equal to $c_2 n \log_2 n$ to sort n items
    - c_2 is another constant that also does not depend on n
  - usually $c_1 < c_2$

No matter how much smaller c_1 is than c_2, there will always be a crossover point beyond which merge sort is faster

- The constant factors are far less significant in the running time than the dependence on the input size n
- Merge sort: factor of $\log_2 n$; Insertion sort: factor of n in its running time, which is much larger
- Insertion sort is usually faster than merge sort for small input sizes, but once the input size n becomes large enough, merge sort's advantage of $\log_2 n$ Vs n will more than compensate for the difference in constant factors

Algorithm as a Technology: Efficiency

- Example in support of the concept:
  - Computer A: faster: running insertion sort
  - Computer B: slower: running merge sort
  - Input size: n= 1 million numbers
  - Computer A: speed: 1 billion instructions/sec (10^9 inst/sec)
  - Computer B: speed: 10 million instructions/sec (10^7 inst/sec)
  - To make the difference even more dramatic
    - Insertion sort: written by an efficient programmer in m/c language and resulting code requires $2n^2$ instructions to sort n numbers
    - Merge sort: written by an average programmer in a high level language with an efficient compiler, with the resulting code taking $50n\log_2 n$ instructions
  - To sort 1 million numbers:
    - Computer A takes: 2000 secs
    - Computer B takes: ~100 secs

    In general, as the problem size increases, so does the relative advantage of merge sort
  - By using an algorithm whose running time grows more slowly, computer B runs 20 times faster than computer A

Insertion sort takes approx 2.3 days

merge sort takes under 20 mins

- The advantage of merge sort is even more pronounced when we sort 10 million numbers

Algorithms and other Technologies

- So algorithms are also a technology like computer hardware
- Total system performance depends on choosing efficient algorithms as much as on choosing fast h/w

Algorithms and other Technologies

- Moreover, even an application that does not require algorithmic content at the application level relies heavily upon algorithms
- Does the application rely on fast h/w?

The h/w design used algorithm

- Does the application rely on GUIs?

The design of any GUI relies on algorithm

- Does the application rely on networking?

Routing in networks relies heavily on algorithm

- Does the application written in a language other than machine language?

Then it is processed by compiler, interpreter, or assembler, all of which make extensive use of algorithm

Algorithms and other Technologies

- Algorithms are at the core of most technologies used in contemporary computers
- Furthermore, with the ever-increasing capacities of computers, we can use them to solve larger problems than ever before
- At larger problem sizes, the differences in efficiencies between algorithms become particularly prominent

References:

- Slides and figures have been collected from various Internet sources for preparing the lecture slides of IT2001 course.
- I acknowledge and thank all the authors for the same.
- It is difficult to acknowledge all the sources though.
- https://www.researchgate.net/publication/347303319_Data_Structure_Ppt