Data Structure
&
Algorithm Analysis

Chandan Mazumdar

Professor

Department of Computer Science & Engineering

Jadavpur University

Calcutta 700 032

Computers are used to Solve Problems

3 Steps to solve a problem on a Computer :

- A **model** of the problem
- An **algorithm** within the framework of the model
- Computer representation of **data**

Knowledge of different **data models** and relevant **operations/algorithms** is essential for **objective** solution of problem.

Information & Data

- **Information**

  Some fact about the surrounding

  e.g.

  Height of Kamal is 6.5 ft.

  Rainfall today was 10 mm.

Information & Data …

- **Data**

  Abstraction of information in a Problem Solving System

e.g.

6.5

10

Data may be numeric or non-numeric.

Of course, at the machine level all kinds of data are represented as strings of bits : 1 and 0.

Data and Operations

- For representation of different forms of data, different data types are used.
- Each data type contains a set of allowable values and a set of allowable operations.
- Data values are interpreted according to their types.
  e.g. 123 – an integer
  "123" – a string of characters
- Operations also depend on the corresponding data types.
  e.g. 123 + 45 = 168 – integer addition

  "123"+"45" = "12345" – string concatenation

Program Structures

- Program = Algorithm + Data Structure

- Programming Languages provide facilities for algorithm representation and data representation.

- High Level Programming Languages like PASCAL and C facilitates structured and modular programming by providing algorithm structures.

- Algorithm structures are :
  1. Sequence
  2. Conditional
  3. Iteration
  4. Subprogram
- 5. Recursion
  6. Control transfer

Data Types

- **Scalar**

  Integer Real Character Boolean

  Pointer Subrange Enumerated

- **Data Aggregation Facilities**

  Arrays Records Sets

- Structured data types
- (1) **Components**
- (2) **Structure** defined by the set of rules that put the components together
- (3) **Set of operations**

ABSTRACT DATA TYPE (ADT)

- A **conceptual model** of information structure.
- An ADT **specifies** the **components**, their **structuring relationships** and a list of operations that are allowed to be performed.
- It is just a **specification**, no design or implementation info is included.
- The components themselves are other ADT's.

ADT …

- No assumption is made about the range of values of the components.
- Specification involves the **“what”**s of the operations, not the **“how”**s.
- ADT’s are generalizations of **primitive** data types.
- They **encapsulate** data values.

Data Structure

- A data structure is the **design representation** of an ADT.
- The same ADT may be represented by several data structures.
- Eg :

  Real nos : (1) <int> . <int>

  (2) (<int> , <int>)

Data Structure …

- There are many data structures corresponding to the ADT “set”.
- Operations on data structures are represented as algorithms.
- If the relations among components are not implicitly known,

  (1) they can be expressed separately (possibly through another data structure)

  (2) Components can be augmented to include additional fields that represent structural information. These components are termed as nodes.

Data Structure …

- A data structure is implemented using the facilities of a programming language.
- So what we use as a data structure is an ADT restricted at the levels of design and implementation.

Types of Data Structure

- Data Structures are of two generic types:

  Static and Dynamic

- The memory requirement of static data structures is fixed at the compile time.
- Dynamic data structures can shrink and grow at the run-time. Thus they require special memory allocation and deallocation functions from the programming system.
- **Algorithms and memory usage** of data structures should be as efficient as possible.

Array as a Data Structure

ADT array

- Objects Elements of the same type arranged in a sequence. An associated index has finite ordinal type. There is an one-to-one correspondence between the values of the index and the array elements.

- Operations
- (1) store_array (a,i,e) -- store e's value in the ith element of array a
- (2) retrieve_array (a,i) -> e -- return the value of the ith element of array a

Array as a Data Structure…

- **Design**

  The required no. of memory locations are statically allocated consecutively.

- **Implementation**

  Built into the language.

  What are the constraints ?

Higher Dimensional Arrays

- Two- and higher-dimensional arrays are extension of the same ADT.
- There are two design choices :

  1. **Row-major** – Elements are stored such that the last index increases most rapidly.
  2. **Column-major** – Elements are stored such that the first index increases most rapidly.

- Whenever there is a reference to an array element, the compiler generates codes for calculating the physical address of the element. This address is then used to access the element exactly like a simple variable.

Polynomials – Application of Array

- Operations
  - Is-zero – returns true if polynomial is zero.
  - Coef – returns the coeff. of a specified exponent.
  - add - add two polynomials
  - mult - multiply two polynomials
  - Cmult - multiply a polynomial by a const.
  - attach – attach a term to a polynomial
  - remove – remove a term from a polynomial
  - degree – returns the degree of the polynomial

- Representation decisions
- 1. Exponents should be unique and be arranged in decreasing order.
- 2. Storage alternatives ?

Sparse Matrix

It is natural to represent matrices as 2-d arrays. But for sparse matrices this involves wastage of a lot of memory space. The operations like transpose, add, multiply also takes a lot of time.

- An alternative approach :

  Store the nonzero elements of a sparse matrix in the form of 3-tuples (i, j, val) in an array.

  i = row-position

  j = column position

  val = value at position (i, j) [nonzero].

  [The first triple contains (m, n, t), for a m X n matrix having t non-zero values]

- This representation reduces the space requirement. Algorithms developed on this representation may be faster than the first approach.

Lists

The most obvious application of arrays is in representing lists of elements of the same type.

Some of the operations performed on lists :

- 1. find out the length of a list
- 2. read the list from either direction
- 3. retrieve the i^{th} element
- 4. store a new value into the i^{th} position
- 5. insert a new element at position i
- 6. delete the element at position i
- 7. search the list for a specified value
- 8. sort the list in some order on the value of the elements.