Array as a

Data Structures

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.
  - degree – returns the degree of the polynomial

- <u>Representation decisions</u>
- 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.

Summary

- Abstract Data Type for Array defined
- Design options of Array Data Structure and its Implementation discussed
- Application of array to implement other data structures discussed