Data Structures - Arrays 01

Dr. TGI Fernando [1] [2]

[1] Email: tgi.fernando@gmail.com
[2] URL: http://tgifernando.wordpress.com/

Introduction - Arrays

- Fundamental data structure
- **HOMOGENEOUS** collection of values (all of the same type)
- Store values sequentially in memory
- Associate an **INDEX** with each value
- Use array name and the index to quickly access an element of an array
- Concise and efficient method for working with large collections of data values
- Limitation - need to know the size ahead of time
- Natural applications - vectors, matrices, string of characters, etc.
- Computer memory is a huge array.

Example of array use

Symbolic manipulation of polynomials

Representation of $x^9 + 3x^5 + 6$:

```
int a[10];
for (i=0; i<10; i++)
   a[i] = 0;
a[0] = 6; a[5] = 3; a[9] = 1;
```

Use exponents as array indices.
Store the coefficients in the array.
Advantages
- Can get each item quickly.
- Index carries implicit information, takes no space

Disadvantage
- Uses up space for for unused items.

Operations

Fundamental operations

- Insert
- Search
- Delete

E.g.

- Insert a student into the array data structure when the student arrives to the school.
- Check to see whether a particular student is present, by searching for the student number in the array data structure.
- Deletes a student from the data structure when that student leaves the school.

Insertion

- New item is always inserted to the first vacant cell in the array.
- Array data structure knows how many items are already in the array or next vacant cell in the array.
- In **NO-DUPLICATES** situation, the algorithm must ensure that not to insert an item with the same key as an existing item.

Searching

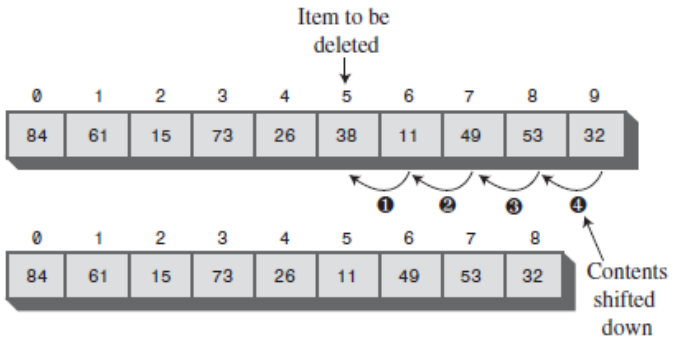
In **NO-DUPLICATES** mode

- The search will terminate as soon as an item with the specified key value is found.
- If the selected number is not in the array, the algorithm must check every element in the array before telling the item is not found in the array.
- If N is the number of items in the array,
  - the average number of steps needed to find an item is $N/2$.
  - in the **worst-case scenario**, the specified item is in the last occupied cell, and N steps will be required to find it.

Note: The time an algorithm takes to execute is proportional to the number of steps, so searching takes much longer on the average ($N/2$ steps) than insertion (one step).

Deletion

- To delete an item, the algorithm must first find it.
- **HOLES** are not allowed in the array.
- Therefore, after locating the item, the algorithm must shift the contents of each subsequent cell down one space to fill in the hole.



Deletion (Contd.)

A deletion requires (assuming no duplicates are allowed) searching through an average of elements and then moving the remaining elements (an average of $N/2$ moves) to fill up the resulting hole. This is N steps in all.

The Duplicates Issue

Insertion without duplication

- If the data structure does not allow duplicates, the algorithm must guard against human errors during an insertion.
- The algorithm must check every element of the array to ensure that none of them already has same key value as the item being inserted.
- This check is inefficient - increase the number of steps from one to N.

The Duplicates Issue (Contd.)

Searching with duplicates

- Algorithm must find all entries that match with the search key.
- Even if it finds a match, it must continue looking for additional matches until the last occupied cell.

Insertion with duplicates

- A single step inserts the new item.
- Do not need to check for item is already in the array.

The Duplicates Issue (Contd.)

Deletion with duplicates

- Delete every item with a specified key value.
- Same operation may require multiple deletions.
- Each time an item is deleted, subsequent items must be shifted further.
- Such operation requires checking N cells and moving more than $N/2$ cells.
- The average depends on how duplicates are distributed throughout the array.

Duplicates OK Vs. No Duplicates

	No Duplicates	**Duplicates OK**
Search	N/2 comparisons	N comparisons
Insertion	?	No comaparisons, one move
Deletion	N/2 comparisons, N/2 moves	N comparisons, more than N/2 moves

Array Class

Lab Work 02 - NoDupArray

Description: The class 'NoDupArray' given in 'NoDupArrayApp.txt' is written to demonstrate the array data structure without duplicates.
Tasks: Complete the coding of classes 'NoDupArray' and 'NoDupArrayApp' given in 'NoDupArrayApp.txt' text file.