Zabin Visram Room CS115

- Lectures :- Searching – Sequential, Binary, Hashing
  :- Sorting - selection sort, insertion sort, mergesort, quicksort, radix sort
- Books
- Java collections an introduction to abstract data types, data structures and algorithms David A. Watt and Deryck F. Brown
- Java structures Duane A. Bailey Chapter 5
- Data structures using Java D.S Malik, P.S Nair ISBN 0-619-15950-2 Chapters 8,9
- Data structures and abstractions with Java , Frank M, Carrano, Walter Savitch P356 - 361
- Web sites
- http://www.cosc.canterbury.ac.nz/people/mukundan/dsal/appldsal.html

Searching

Why Search ?

- Everyday life -We always Looking for something – builder yellow pages, universities, hairdressers
- Computers can search for us
- World wide web – different searching mechanisms, yahoo.com, ask.co.uk, google.com
- Spreadsheet – list of names – searching mechanism to find a name
- Databases – use to search for a record - select * from ..
- Large records – 1000s takes time - many comparison slow system – user wont wait long time

Search Algorithms

- Different types – Sequential Search, Binary Search
- Discuss the search algorithms analyze them
- Analysis of the algorithms enables programmers to decide which algorithm to use for a specific application

Some observations – Key

- Before describing the algorithm – clarify some terms
- Associated with each item in a data set is a special member that uniquely identifies the item in the data set
- For e.g in a data set containing student records – student ID uniquely identifies each student
- This unique member of the item is called Key of the item

Some observations - Key

- Keys of the item in a data set are used in operations such as searching, sorting, insertion, deletion
- When we search the data set for a particular item, we compare the key of the item for which we are searching with the keys of the items in the data set

Analysis of Algorithms

- In addition to describing the algorithms – analyse them
- In the analysis of algorithms – the key comparisons refer to comparing the key of the search item with the key of an item in the list
- Moreover – the number of key comparisons refers to the number of times the key of the item (in search & sort algorithms) is compared with the keys of the items in the list

Target

- When searching for an item e.g searching for a sweater in cupboard or a pen on you table , or a list of student names with exams results – search for your name on list
- Searching for a particular item – sometimes called the Target – among a collection of many items is a common task

Sequential search (linear search)

- Consider the problem of searching for a given element in a list.
- We may not know whether the target element occurs in the list in advance of the search.
- Sequential search works the same for both array-based and linked lists
- For simplicity here we shall assume an array of integers.

Sequential search (linear search)

- Typically we can have no guarantees about the order of elements in the list if (for example) insertions have been under a user's control.
- In such circumstances a sequential search from position 0 is as efficient as any method for finding a given target value.
- Thus search starts at the first element in the list and continues until either the item is found in the list or entire list is searched

A simple search method is as follows:

```
protected  int[] a;

 public  int linearSearch(int
target)  {

    for (int n = 0; n < a.length;
n++)

          if (a[n] == target)

          return  n;

    return  -1;

}
```

Task :- in pairs

- Write a method contains that returns the index of the first array element that equals anEntry. If the array does not contain such an element , return –1
- **Public boolean** contains(Object anEntry)

```
Public int contains (Object anEntry)
{
Boolean found = false;
Int result = -1;
For (int index = 0;   !found&& (index < length);
index++)
 {
  if (anEntry.equals(entry[index]))
    {
         found = true;
          result = index;
       }// end if
    } // end for
Return result;
```

Demonstration

- http://www.cosc.canterbury.ac.nz/people/mukundan/dsal/LSearch.html

Sequential search (linear search)

- If the search item is found, its index (that is its location in array) is returned. If search is unsuccessful, -1 is returned
- Note the sequential search does not require the list elements to be in any particular order

Sequential Search Analysis

- Statements before and after loop are executed once, - require very little computer time
- Statements in the for loop are repeated several times
- For each iteration in the loop, the search item is compared with an element in the list,and a few other statements are executed. Loop terminates when search item is found in list
- Therefore the execution of the other statement in loop is directly related to the outcome of the key comparisons.

Sequential Search Analysis

- When analysing a search algorithm, we count the number of key comparisons because this number gives us the most useful information, this criterion for counting the number of key comparisons can be applied equally well to other search algorithms

Sequential Search Analysis

- Suppose that the length of the list, say L, is n.
- We want to determine the number of key comparisons made by the sequential search, when the list L is searched for a given item
- If the item is not in the list then n comparisons are made (every item in list is compared) before return unsuccessful

Task :- in pairs – how many comparisons ?

An iterative sequential search of an array that {a) finds its target; (b) does not find its target

(a) A search for 8

Look at 9

9	5	8	4	7

8 <> 9 , so continue searching …

Look at 5

9	**5**	8	4	7

(b) A search for 6

Look at 9

9	5	8	4	7

6 <> 9 , so continue searching …

Look at 5

9	**5**	8	4	7

Sequential Search Analysis

- Suppose the search item is in the list
- Then number of key comparisons depends on where in the list the search item is located.
- If search item is first element of *L* – make one key comparison – best case
- Worst case search item is the last item – algorithm makes *n* comparisons

Sequential Search Analysis

- Best and worst cases are unlikely to occur
- More helpful – if we can determine the average behavior of the algorithm – I.e average number of key comparisons the sequential search makes in the successful case

Sequential Search Analysis

- If the search item ***Target*** , is the first element in list, one comparison is needed, if target is second, then two comparison are needed etc
- So if the target is the *kth* element in the list *k* comparisons are made
- Assuming the target can be any element in the list: that is, all elements are equally likely to be the target

Efficiency of a sequential Search of an Array

- In best case , you will locate the desired item first in the array
- You will have made only one comparison
- So search will be O(1)
- In worst case you will search the entire array
- Either desired item will be found at the end of array or not at all
- In either event you have made n comparisons for an array of n elements
- Sequential search in worst case is therefore O(*n*/2) which is just O(*n*)
- Typically you will look at about one-half of the elements in the array . Thus the average case is O(*n*/2), which is just O(*n*)

Sequential Search Analysis

- Suppose there are *n* elements in the list. The following expression, gives average number of comparisons
- 1+2+ …..+n / n
- 1+2+ …..+n = n(n+1)/2
- Thus the following expression gives the average number of comparisons made by sequential search in successful case :
- $\frac{1+2+\ldots\ldots+n}{n} = \frac{1}{n}\,\frac{n(n+1)}{2} = \frac{n+1}{2}$

Sequential Search Analysis

- so the algorithm is O(*n*)
- This expression shows that on average , the sequential search searches half the list.
- Thus sequential search is not efficient for large lists

Time Efficiency of a sequential search of an array

- Best case: O(1)
- Worst case : O(*n*)
- Average case O(*n*)

Sentinel

- Note that each iteration require two conditions to be checked and one statement to be executed.
- We can avoid checking for the end of the array on every iteration by inserting the target as an extra ‘sentinel’ element at the end of the array.
- We place it at position n and follow the algorithm:

Sentinel

- Search sequentially from position 0 until the target is found (it will definitely be found).
- If the target is found in position n then the sentinel has been found – search has ‘failed’,
- else search was successful, return first index where target was found.

Sentinel

- A method to implement this algorithm is as follows:

```
protected  int[] a;
public  int linearSentinel(int
   target)  {
   a[n] = target;
for (int j = 0; ;j++)
if (a[j] == target)
      return  j;
}
```

Sentinel

- Timing studies of lists in the range 100 – 1000 show a marked reduction in the average time to find a target value when using the sentinel method