CONFIDENTIAL

C Programming Basic – week 7

Searching (part 2)

Lecturers :
Cao Tuan Dung
Le Duc Trung
Dept of Software Engineering
Hanoi University of Technology

Binary Search

1	3	5	6	10	11	14	25	26	40	41	78

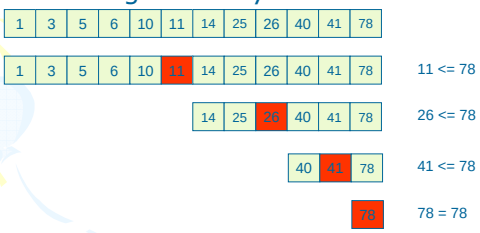
- The binary search algorithm uses a divide-and-conquer technique to search the list.
- First, the search item is compared with the middle element of the list.
- If the search item is less than the middle element of the list, restrict the search to the first half of the list.
- Otherwise, search the second half of the list.

Binary Search

- Binary Search is an incredibly powerful technique for searching an ordered list
- It is familiar to everyone who uses a telephone book!

Illustration

- Searching for a key=*78*



4 opérations necessary for finding out the good element.
How many operations in case of sequential search?

Example

- First, compare 75 with the middle element in this list, L[6] (which is 39).
- Because 75 > L[6] = 39, restrict the search to the list L[7 . . . 12], as shown in Figure.

Binary Search Code

```
int binSearch(int List[], int Target, int Size) {
    int   Mid,
          Lo = 0,
          Hi = Size – 1;
    while ( Lo <= Hi ) {
          Mid = (Lo + Hi) / 2;
          if ( List[Mid] == Target )
                return Mid;
          else if ( Target < List[Mid] )
                Hi = Mid – 1;
          else
                Lo = Mid + 1;
    }
    return -1;
}
```

Test Program

```
#include <stdio.h>
#define NotFound (-1)
typedef int ElementType;

int  BinarySearch(ElementType A[ ], ElementType X, int N )  {
     int Low, Mid, High;
     Low = 0; High = N - 1;
     while( Low <= High ) {
               Mid = ( Low + High ) / 2;
               if( A[ Mid ] < X )
                         Low = Mid + 1;
               elseif( A[ Mid ] > X )
                         High = Mid - 1;
               else
                         return Mid;  /* Found */
     }
     return NotFound;     /* NotFound is defined as -1 */
}
main( )
{
     static int A[ ] = { 1, 3, 5, 7, 9, 13, 15 };
     int SizeofA = sizeof( A ) / sizeof( A[ 0 ] );
     int i;
     for( i = 0; i < 20; i++ )
          printf( "BinarySearch of %d returns %d\n",
                    i, BinarySearch( A, i, SizeofA ) );
     return 0;
}
```

Exercise: Recursive Binary Search

- Implement a recursive version of a binary search function.

Big O Notation

- Definition: Suppose that f(n) and g(n) are nonnegative functions of n. Then we say that f(n) is O(g(n)) provided that there are constants C > 0 and N > 0 such that for all n > N, f(n) ≤ Cg(n).
- This says that function *f(n)* grows at a rate no faster than *g(n)*; thus *g(n)* is an upper bound on *f(n)*.
- Big-O expresses an upper bound on the growth rate of a function, for sufficiently large values of n.

Running time analysis in searching algorithms

- Mesure the number of comparison operations
- Compare results with the problem's size (size of input data)
- Sequential Search: O(n)
- Binary Search: $O(\log_2 n)$

Exercise

- Define an array of integers, load from 1 to 100 in order to the array.
- Read a number from the standard input, perform the binary search for an array. Output "Not Found" if the array does not have it.
- When you perform the binary search, output the array index compared to the standard output. Also, display the number of comparisons achieved until the target number is found.

Hint

- With each comparison:
  - increment a global variable counter

Execise

- Use recursive function for binary search operation
- Print out the number of function call of the Binary Search until the target number is found
- Compare it with the non recursive version.

Dictionary Order and Binary Search

- When you search for a string value, the comparison between two values is based on dictionnary order.
- We have:
  - 'a' < 'd', 'B' < 'M'
  - "acerbook" < "addition"
  - "Chu Trong Hien" > "Bui Minh Hai"
- Just use: strcmp function.

Exercise

- We assume that you make a mobile phone's address book.
- Declare the structure which can store at least "name", "telephone number", "e-mail address.". And declare an array of the structure that can handle about 100 address data.
- Read this array data of about 10 from an input file, and write a name which is equal to a specified name and whose array index is the smallest to an output file. Use the binary search for this exercise

Exercise

- Return to SortedList exercise in Week4 (student management) (Linked List) with structure of an element:

```
typedef struct Student_t {
   char   id[ID_LENGTH];
   char   name[NAME_LENGTH];
   int    grade;

   struct Student_t *next;
} Student;
```

implement the function BinarySearch for this list based on
- the name
- the grade

of students

List verification

- Compare lists to verify that they are identical or identify the discrepancies.
- example
  - international revenue service (e.g., employee vs. employer)
- complexities
  - random order: O(mn)
  - ordered list: O(tsort(n)+tsort(m)+m+n)

List verification

- Given two list whose elements are in the same type. Find
- (a) all records found in list1 but not in list2
- (b) all records found in list2 but not in list1
- (c) all records that are in list1 and list2 with the same key but have different values for different fields.