C Programming Basic – week 7

Searching (part 2)

Lecturers :

Tran Hong Viet

Dept of Software Engineering

Hanoi University of Technology

Binary Search

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

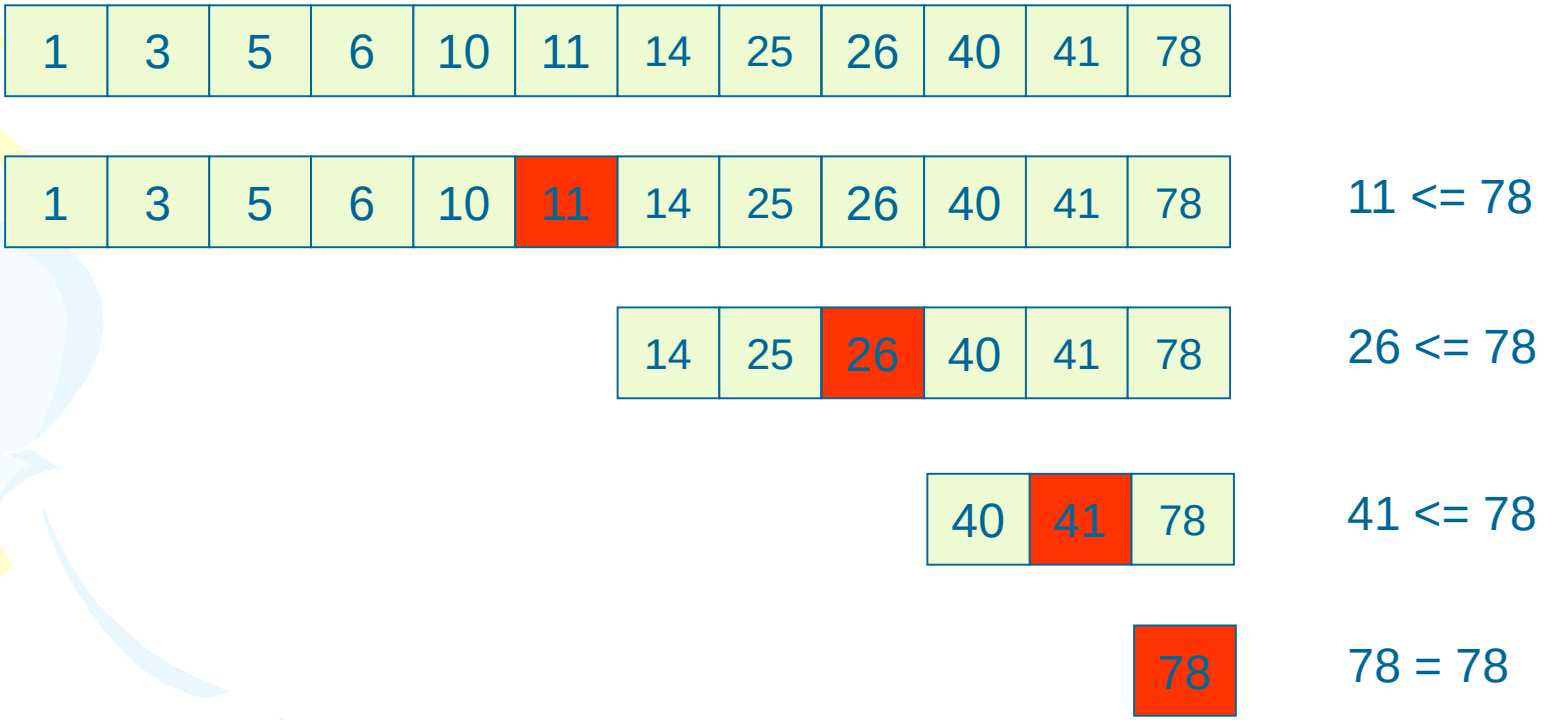
- The binary search algorithm uses a divide-and-conquer technique to search the list.
- First, the search item is compared with the middle element of the list.
- If the search item is less than the middle element of the list, restrict the search to the first half of the list.
- Otherwise, search the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

Solution

```
#define NotFound (-1)
typedef int ElementType;

int  BinarySearch(ElementType A[ ], ElementType X, int Lo, int
    Hi )  {
    if (Lo > High) return NotFound;
    Mid = ( Low + High ) / 2;
    if  (A[ Mid ] < X ) return BinarySearch(A, X, Mid+1, Hi);
    elseif ( A[ Mid ] > X )
                    return BinarySearch(A, X, Lo, Mid – 1);
    else
                   return Mid;  /* Found */
     }
    return NotFound;     /* NotFound is defined as -1 */
}

Usage: BinarySearch(A, X, 0, size -1);
```

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

Solution

```
#include <stdio.h>
#include <string.h>

enum {SUCCESS, FAIL, MAX_ELEMENT = 100};

// the phone book structure
typedef struct phoneaddress_t {
        char name[20];
        char tel[11];
        char email[25];
}phoneaddress;
```

Solution: implement Dictionary order Binary Search

```
int  BinarySearch(phoneaddress A[ ], char name[] , int N )  {
    int Low, Mid, High;
    Low = 0; High = N - 1;
    while( Low <= High ) {
        Mid = ( Low + High ) / 2;
        if( strcmp(A[ Mid ].name, name) < 0 )
                Low = Mid + 1;
            else if(strcmp(A[ Mid ].name, name) > 0)
                High = Mid - 1;
            else
                return Mid;  /* Found */
          }
        return NotFound;     /* NotFound is defined as -1 */
      }
```

Solution

```
int main(void)
{
   FILE *fp, fpout;
   phoneaddress  phonearr[MAX_ELEMENT];
    int i,n, irc; // return code
    char name[20];
   int reval = SUCCESS

   printf("How many contacts do you want to enter
   (<100)?");  scanf("%d", &n);
  if ((fp = fopen("phonebook.dat","rb")) == NULL){
       printf("Can not open %s.\n", "phonebook.dat");
       reval = FAIL;
   }
   irc = fread(phonearr, sizeof(phoneaddress), n, fp);
   printf(" fread return code = %d\n", irc); fclose(fp);
   if (irc <0) {
       printf (" Can not read from file!");
       return -1;
 }
```

Solution (next)

```
printf("Let me know the name you want to search:");
gets(name);
irc = BinarySearch(phonearr, name,n);
if (irc <0) {
    printf (" No contact match the criteria!";
    return -1;
}
// write result to outputfile
if ((fpout = fopen("result.txt","w")) == NULL){
    printf("Can create file to write.\n");
    reval = FAIL;
}
else
 fprintf(fpout, "%s have the email address %s and
telephone number:%s", phonearr[i].name,
phonearr[i].email, phonearr[i].tel);
fclose(fpout);
return reval;
}
```

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.

Solution: Element type and List declaration

- ```
  # define MAX-SIZE 1000/* maximum size of  list plus one */
  typedef struct {
                  int key;
                  /* other fields */
                  } element;
  element list[MAX_SIZE];
  ```
  ```

Binary Search Function

```
int binsearch(element list[ ], int searchnum, int n)
{
/* search list [0], ..., list[n-1]*/
    int left = 0, right = n-1, middle;
    while (left <= right) {
        middle = (left+ right)/2;
    switch (COMPARE(list[middle].key, searchnum)) {
        case -1: left = middle +1;
                    break;
        case  0: return middle;
        case  1:right = middle - 1;
        }
      }
      return -1;
}
```

verifying using a sequential search

```
void verify1(element list1[], element list2[ ], int n, int m)
/* compare two unordered lists list1 and list2 */
{
int i, j;
int marked[MAX_SIZE];

for(i = 0; i<m; i++)
   marked[i] = FALSE;
for (i=0; i<n; i++)
  if ((j = seqsearch(list2, m, list1[i].key)) < 0)
     printf("%d is not in list 2\n ", list1[i].key);
 else
 /* check each of the other fields from list1[i] and list2[j], and
print out any discrepancies */
```

```
        marked[j] = TRUE;
for ( i=0; i<m; i++)
        if (!marked[i])
            printf("%d is not in list1\n", list2[i]key);
}
```

verifying using a binary search

```
void verify2(element list1[ ], element list2 [ ], int n, int m)
/* Same task as verify1, but list1 and list2 are sorted */
{
   int i, j;
   sort(list1, n);
   sort(list2, m);
   i = j = 0;
   while (i < n && j < m)
        if  (list1[i].key < list2[j].key) {
             printf ("%d is not in list 2 \n", list1[i].key);
             i++;
        }
        else if (list1[i].key == list2[j].key) {
        /* compare list1[i] and list2[j] on each of the other field
           and report any discrepancies */
        i++; j++;
        }
```

verifying using a binary search

```
else {
    printf("%d is not in list 1\n", list2[j].key);
    j++;
 }
for(; i < n; i++)
    printf ("%d is not in list 2\n", list1[i].key);
for(; j < m; j++)
    printf("%d is not in list 1\n", list2[j].key);
}
```