Data structure and algorithms lab

SEARCHING

Lecturers : Cao Tuan Dung
dungct@soict.hust.edu.vn
Dept of Software Engineering
Hanoi University of Science and Technology

Topics of this week

- Search algorithm
  - Sequential searching
  - Sentinel
  - Self organized 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

Sequential search (Linear search)

- Visit all the elements of array from the beginning
- Compare the key with each element of a list (or of an array).
- 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

```
int LinearSearch (T M[], int N,
  T X){
  int k = 0;
  while (M[k] != X && k < N)
  k++;
  if (k < N) return (k);
  return (-1);
}
```

Example

```
#include<stdio.h>

 int sequential_search(char *items, int count, char key)
  {
    register int t;

    for(t=0; t < count; ++t)
      if(key == items[t]) return t;
    return -1; /* no match */
  }

 int main(void){
     char *str = "asdf";

     int index = sequential_search(str, 4, 's');

     printf("%d",index);

 }
```

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 search

```
int LinearSentinelSearch (T M[],
  int N, T X){
  int k = 0; M[N]=X;
  while (M[k] != X)
  k++;
  return k-1;
}
```

Exercise 6-1

- We assume that you write a mobile phone's address book.
- Declare a structure "Address" that can hold at least name, telephone number, and e-mail address, and write a program that can handle about 100 address data.
- Read about 10 address data from the input file, search a name by the linear search, and write the data matched first to the output file.
  - (1) Implement this program using an array of structure.
  - (2) Implement this program using a singly-linked list or a doubly-linked list. Confirm the second search is accelerated by moving data matched to the head of list (self-organizing search).

Exercise 6-2: Searching Arrays by Linear Search

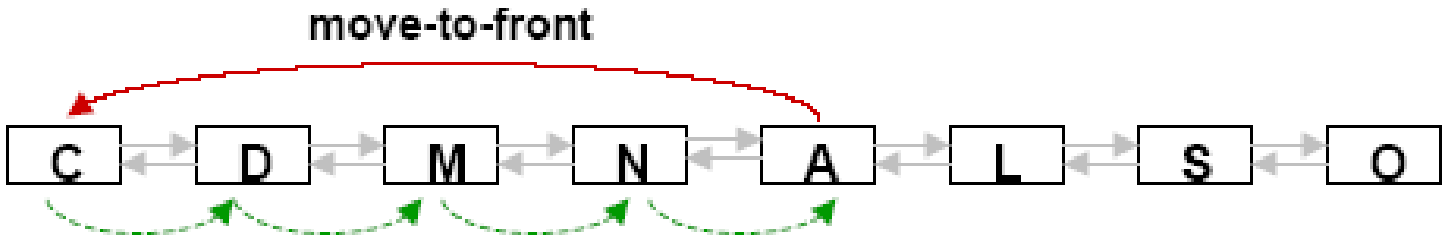
- Read 11 integers from the standard input and assign first ten integers to the array.
- Then if the 11th integer is in the array, output the position of the element (1 – 10). If not, output 0.

Queue (move to front)

- Make a queue that holds integers. The size of the queue is fixed to 10.
- Read integers separated by spaces from the standard input, and add them to the queue. When the program reads the 11th integer, the queue is already full. So the program removes the first integer and adds the 11th integer. Print the removed integer to the standard output.
- Process all the integers in this way.

Self organizing search (move to front)

- Any element searched/requested is moved to the front



Self organizing search (move to front)

```
int search( int key,int r[], int n )
 {
  int i,j;
  int tempr;
  for ( i=0; i<n-1 && r[i] != key; i++ );
  if ( key == r[i] )
  { if ( i>0 ) {
      tempr = r[i];
      for (j=i, j>0; j--) r[j]=r[j-1];
      r[0]=tempr;
  }
  return( i );
  } else return( -1 );
```

Self-organizing (Transpose) sequential search

```
int search( int key,int r[], int n )
 {
  int i;
  int tempr;
  for ( i=0; i<n-1 && r[i] != key; i++ );
  if ( key == r[i] )
  { if ( i>0 ) {
 /*** Transpose with predecessor ***/
  tempr = r[i];
  r[i] = r[i-1];
  r[--i] = tempr;
  }
  return( i );
  } else return( -1 );
```

Exercise: Self Organized List

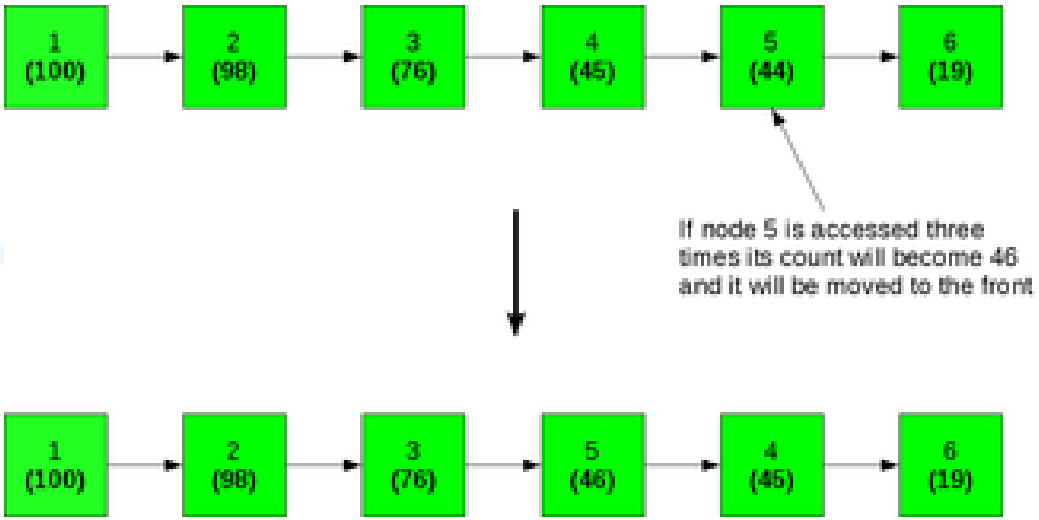
- Modify a list that you have created in previous exercises which support the capacity of self-organizing using "move to front" and transpose strategy.
- Infact, develop the function search an element in a list.

Exercise: Self Organized List

- Implement a Self Organized List using Transpose strategy.

Using counter

- Number of times each element was searched for is counted
- Element are rearranged to decreasing count



Homework

- Extend the program of phone models management (PhoneDB) by implementing movetofront, transpose (and using counter if you can) strategies for feature Search & Update. Users can choose strategies and after the execution of the searching task, go back to the menu to display the Phone models list to check the effect of these strategies.

Homework (advance)

- Write a program that meets the following specifications.

[Format] look character string

[Description] All the words that begin with character strings registered in /user/share/dict/words are displayed.

[Example]

```
% look computer
computer
computerize
computerized
computerizes
computerizing
computers
```

Binary Search

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

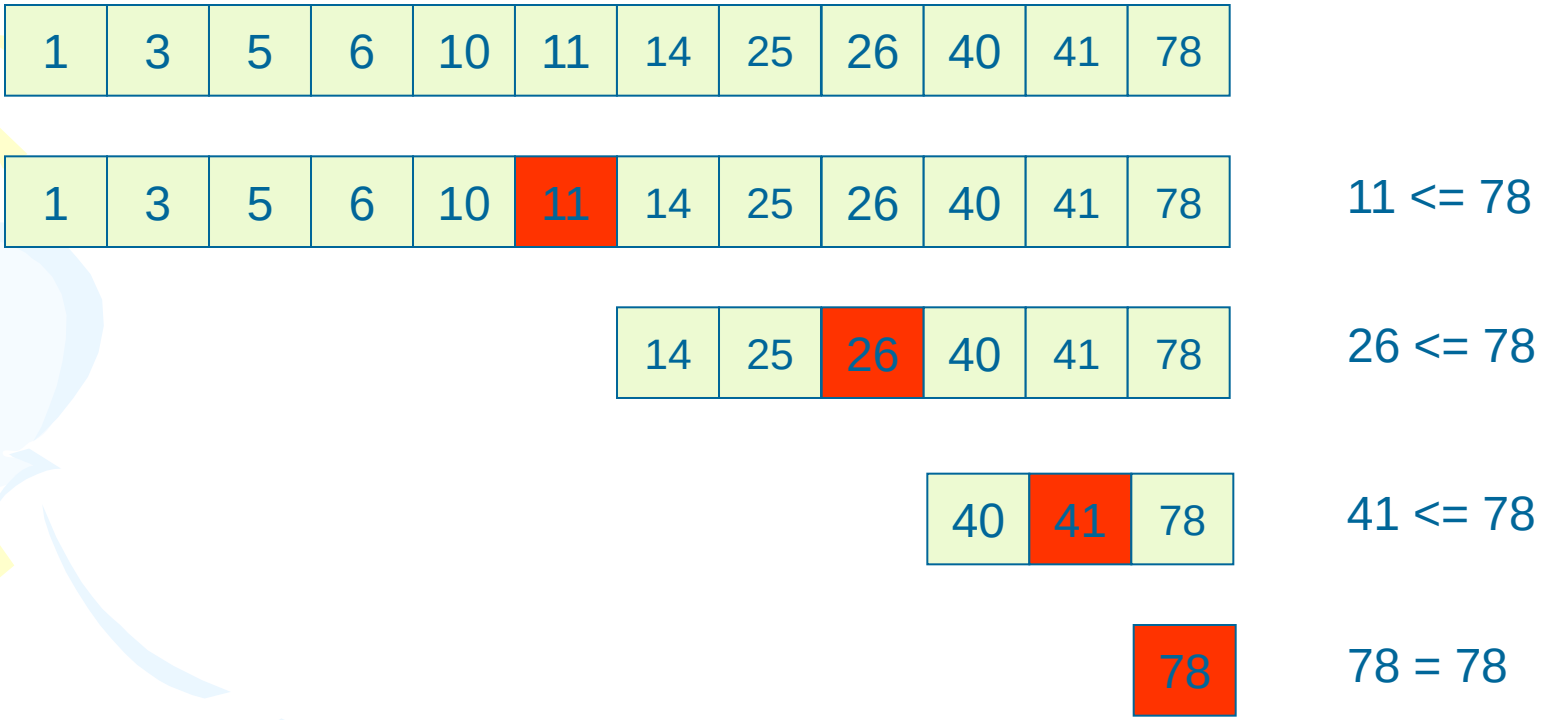
- The binary search algorithm uses a divide-and-conquer technique to search the list.
- First, the search item is compared with the middle element of the list.
- If the search item is less than the middle element of the list, restrict the search to the first half of the list.
- Otherwise, search the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

Homework

- Write a program that declares an array of at least 200000 integers (you can try with 500000 or 1 million integers). Initialize the array elements with ascending values. (For example a[i] = 2*i+3)
- The Program provides a menu for:
  - Sequential search
  - Sequential search using Sentinel
  - Binary search
- For each functionality, display not only the result but the execution time.

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 (sorted by name in alphabetic order), 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[irc].name,
phonearr[irc].email, phonearr[irc].tel);
fclose(fpout);
return reval;
}
```

Homework

- Return to SortedList exercise in Linked List topic (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 sorting technique

```
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 sorting technique

```
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);
}
```

Homework

- Modify the program PhoneDB as follows: The program manages phone models available at Hanoi and Hochiminh city branches which are stored in two files HNPhoneDB.dat and HCMPhoneDB.dat (You can create these files using your previous Labs and homework). Based on the model, the program should :
  - Display the list of phone models available at Hanoi but not at HCM city.
  - Display the list of phone models available at HCMcity but not at Hanoi.
  - Display the list of phones which have the same model but differ in other fields.
- Optional:
  - Search for phones by applying binary search on the model. Display the number of comparisons carried out.