Searching

Mais Saad Safoq
Computer Science Department
Kerbala University

2016

Searching

- Information retrieval is one of the most important application of computers.
- Searching for the keys that locate records is the most time consuming action in a program

- **Sequential Search :**
  - Start at the first piece of data and look at it and if its not then keep going until you find what you are looking for or until you have reached the last.

Sequential Searching Algorithm

The algorithm search for the required key and returns the position i of the structure k(i) where the “key” is found

```
int key , n=5;
int found = 0, i =0 ;
cin>>key;
 while( i<n && found==0){
if( k[i]==key)
found =1 ;
i++ ;                }
if (found=1)
cout<<"record found at position i"<<i-1;
else
cout<<"no match found";
```

Sequential search

- Example:
  - Looking for a Name by giving the telephone number, depending on structures.
  - We give one piece of information (KEY) and we are asked to find a record that contains other information associated with the key.

Records in c++ (Structures)

- **Record** - a linear, direct-access data structure.
- **Field** or **member** - component of the record; each field has its own type
- **Example:**

```
struct StudentType {
int idNumber;
char name[30];
float credits;};
StudentType   student1;
```

- The **dot (.) operator** is used to select members from a record (struct).

Sequential search

- Find the marks of the students from the Data structures and algorithms course using an array. (Key_ID)

```
 array name is ‘ a ‘       // create

For ( i =0; i < size; i++)

{ if  a[i].id == ‘123’

cout << a[i].id<<a[i].mark1;

found = true }

if (found !=true) cout << “Not in the List”
```

Sequential Search

- efficiency

  Two Kinds of result

  1)Successful

  Best Case – element at position 1

  (number of comparison=1)

  2) UnSuccessful

  Worst Case – element not in record

  (number of comparison=n+1)

Sequential search

Advantages:

- Easy to implement
- Can be used on small data sets
- Not practical for searching large collections
- Dose not require the data to be ordered
- Dose not require any additional data structure

Binary Search

- Binary- List is broken down into two parts and searched
- Working : Compare the key with the one in the center of the list and then restrict our attention to only the first or second half depending on whether the key comes before or after the central one. (we reduce the length to be searched by half)

Binary Search (Items sorted in accending order)

- Rule – Elements must be in sorted order
- Steps : 1. l = Low (first index)
  2. h = High (last index)
  3. m = Mid ((l + h)/ 2)

- 1. mark l & h
- 2. Find Mid

Binary search

3. Check whether mid = x if so return index
4. If not , check whether x > k[mid]

   bring low to mid+1 and carry on the same process
5. If < then bring high to mid-1 and carry on the same process
6. Until you find the value or if low and high cross each other

Binary search

EG : 10 20 30 40 50 60 70

Find X = 20 (search value)

10 20 30 40 50 60 **70**

low **Mid** **high**

Mid = l+h/2 = 1+7/2 = 4

10 20 **30** 40 50 60 70

low **high** high = mid -1

Binary search

10 20 **30** 40 50 60 70

low **high**

mid = l +h/2 = 1+3/2 =2

10 20 **30** 40 50 60 70

low mid **high**

check **x = a[mid]** , yes matching so return index - ie **2**

Binary search

```
beg=1;
end=n;
mid=(beg+end)/2;                    // Find Mid Location of Array
while(beg<=end && a[mid]!=item)     // Compare Item and Value of
    Mid
{
if(item >a[mid])
beg=mid+1;
else
end=mid-1;
mid=(beg+end)/2;
}
if(a[mid]==item)
cout<<"\nData is Found at Location : "<<mid;
else
cout<<"Data is Not Found";
```