geeksforgeeks.org

Interpolation Search - GeeksforGeeks

5-6 minutes

Given a sorted array of n uniformly distributed values arr[], write a function to search for a particular element x in the array.

Linear Search finds the element in O(n) time, Jump Search takes O(√ n) time and Binary Search take O(Log n) time.
The Interpolation Search is an improvement over Binary Search for instances, where the values in a sorted array are uniformly distributed. Binary Search always goes to middle element to check. On the other hand interpolation search may go to different locations according the value of key being searched. For example if the value of key is closer to the last element, interpolation search is likely to start search toward the end side.

To find the position to be searched, it uses following formula.

```
// The idea of formula is to return higher value
of pos
// when element to be searched is closer to
arr[hi]. And
// smaller value when closer to arr[lo]
pos = lo + [ (x-arr[lo])*(hi-lo) / (arr[hi]-
arr[Lo]) ]

arr[] ==> Array where elements need to be searched
x     ==> Element to be searched
lo    ==> Starting index in arr[]
hi    ==> Ending index in arr[]
```

Algorithm

Rest of the Interpolation algorithm is same except the above partition logic.

Step1: In a loop, calculate the value of “pos” using the probe position formula.
Step2: If it is a match, return the index of the item, and exit.
Step3: If the item is less than arr[pos], calculate the probe position of the left sub-array. Otherwise calculate the same in the right sub-array.
Step4: Repeat until a match is found or the sub-array reduces to zero.

Below is C implementation of algorithm.

- C
- Java
- Python

C

```
#include<stdio.h>

int interpolationSearch(int arr[], int n, int x)

{

    int lo = 0, hi = (n - 1);

    while (lo <= hi && x >= arr[lo] && x <=
arr[hi])

    {

        int pos = lo + (((double)(hi-lo) /

              (arr[hi]-arr[lo]))*(x - arr[lo]));

        if (arr[pos] == x)
```

```
            return pos;
        if (arr[pos] < x)
            lo = pos + 1;
        else
            hi = pos - 1;
    }
    return -1;
}
int main()
{
    int arr[] =  {10, 12, 13, 16, 18, 19, 20, 21,
22, 23,
                 24, 33, 35, 42, 47};
    int n = sizeof(arr)/sizeof(arr[0]);
    int x = 18;
    int index = interpolationSearch(arr, n, x);
    if (index != -1)
        printf("Element found at index %d",
index);
    else
        printf("Element not found.");
    return 0;
}
```

Java

```
class Test
{
    static int arr[] = new int[]{10, 12, 13, 16,
18, 19, 20, 21, 22, 23,
                                 24, 33,
35, 42, 47};
    static int interpolationSearch(int x)
    {
        int lo = 0, hi = (arr.length - 1);
        while (lo <= hi && x >= arr[lo] && x <=
arr[hi])
        {
            int pos = lo + (((hi-lo) /
                  (arr[hi]-arr[lo]))*(x -
arr[lo]));
            if (arr[pos] == x)
                return pos;
            if (arr[pos] < x)
                lo = pos + 1;
            else
                hi = pos - 1;
        }
        return -1;
```

```
    }
    public static void main(String[] args)
    {
        int x = 18;
        int index = interpolationSearch(x);
        if (index != -1)
            System.out.println("Element found at
index " + index);
        else
            System.out.println("Element not
found.");
    }
}
```

Python

```
def interpolationSearch(arr, n, x):
    lo = 0
    hi = (n - 1)
    while lo <= hi and x >= arr[lo] and x <=
arr[hi]:
        pos  = lo + int(((float(hi - lo) /
            ( arr[hi] - arr[lo])) * ( x -
arr[lo])))
        if arr[pos] == x:
            return pos
```

```
            if arr[pos] < x:
                lo = pos + 1;
            else:
                hi = pos - 1;
        return -1
arr = [10, 12, 13, 16, 18, 19, 20, 21, \
                22, 23, 24, 33, 35, 42, 47]
n = len(arr)
x = 18
index = interpolationSearch(arr, n, x)
if index != -1:
    print "Element found at index",index
else:
    print "Element not found"
```

Output :

```
Element found at index 4
```

Time Complexity : If elements are uniformly distributed, then **O (log log n))**. In worst case it can take upto O(n).
Auxiliary Space : O(1)

This article is contributed by **Aayu sachdev**. If you like GeeksforGeeks and would like to contribute, you can also write an article using contribute.geeksforgeeks.org or mail your article to contribute@geeksforgeeks.org. See your article appearing on the GeeksforGeeks main page and help other Geeks.

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.