geeksforgeeks.org

Jump Search - GeeksforGeeks

5-6 minutes

Like Binary Search, Jump Search is a searching algorithm for sorted arrays. The basic idea is to check fewer elements (than linear search) by jumping ahead by fixed steps or skipping some elements in place of searching all elements.

For example, suppose we have an array arr[] of size n and block (to be jumped) size m. Then we search at the indexes arr[0], arr[m], arr[2m]…..arr[km] and so on. Once we find the interval (arr[km] < x < arr[(k+1)m]), we perform a linear search operation from the index km to find the element x.

Let's consider the following array: (0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610). Length of the array is 16. Jump search will find the value of 55 with the following steps assuming that the block size to be jumped is 4.
STEP 1: Jump from index 0 to index 4;
STEP 2: Jump from index 4 to index 8;
STEP 3: Jump from index 8 to index 16;
STEP 4: Since the element at index 16 is greater than 55 we will jump back a step to come to index 9.
STEP 5: Perform linear search from index 9 to get the element 55.

What is the optimal block size to be skipped?
In the worst case, we have to do n/m jumps and if the last checked value is greater than the element to be searched for, we perform m-1 comparisons more for linear search. Therefore the total number of comparisons in the worst case will be ((n/m) + m-1). The

value of the function ((n/m) + m-1) will be minimum when m = √n. Therefore, the best step size is m = √**n.**

- C++
- Java
- Python3

C++

```
#include <bits/stdc++.h>
using namespace std;
int jumpSearch(int arr[], int x, int n)
{
    int step = sqrt(n);
    int prev = 0;
    while (arr[min(step, n)-1] < x)
    {
        prev = step;
        step += sqrt(n);
        if (prev >= n)
            return -1;
    }
    while (arr[prev] < x)
    {
        prev++;
        if (prev == min(step, n))
```

```
            return -1;
    }
    if (arr[prev] == x)
        return prev;
    return -1;
}
int main()
{
    int arr[] = { 0, 1, 1, 2, 3, 5, 8, 13, 21,
                  34, 55, 89, 144, 233, 377, 610
};
    int x = 55;
    int n = sizeof(arr) / sizeof(arr[0]);
    int index = jumpSearch(arr, x, n);
    cout << "\nNumber " << x << " is at index " <<
index;
    return 0;
}
```

Java

```
public class JumpSearch
{
    public static int jumpSearch(int[] arr, int x)
    {
```

```
        int n = arr.length;
        int step = (int)Math.floor(Math.sqrt(n));
        int prev = 0;
        while (arr[Math.min(step, n)-1] < x)
        {
            prev = step;
            step +=
(int)Math.floor(Math.sqrt(n));
            if (prev >= n)
                return -1;
        }
        while (arr[prev] < x)
        {
            prev++;
            if (prev == Math.min(step, n))
                return -1;
        }
        if (arr[prev] == x)
            return prev;
        return -1;
    }
    public static void main(String [ ] args)
    {
        int arr[] = { 0, 1, 1, 2, 3, 5, 8, 13, 21,
```

```
                        34, 55, 89, 144, 233, 377,
610};
        int x = 55;
        int index = jumpSearch(arr, x);
        System.out.println("\nNumber " + x +
                                " is at index " +
index);
    }
}
```

Python3

```
import math
def jumpSearch( arr , x , n ):
    step = math.sqrt(n)
    prev = 0
    while arr[int(min(step, n)-1)] < x:
        prev = step
        step += math.sqrt(n)
        if prev >= n:
            return -1
    while arr[int(prev)] < x:
        prev += 1
        if prev == min(step, n):
            return -1
```

```
    if arr[int(prev)] == x:
        return prev
    return -1
arr = [ 0, 1, 1, 2, 3, 5, 8, 13, 21,
       34, 55, 89, 144, 233, 377, 610 ]
x = 55
n = len(arr)
index = jumpSearch(arr, x, n)
print("Number" , x, "is at index" ,"%.0f"%index)
```

Output:

```
Number 55 is at index 10
```

Time Complexity : O(√n)
Auxiliary Space : O(1)

Important points:

- Works only sorted arrays.
- The optimal size of a block to be jumped is O(√ n). This makes the time complexity of Jump Search O(√ n).
- The time complexity of Jump Search is between Linear Search ((O(n)) and Binary Search (O (Log n)).
- Binary Search is better than Jump Search, but Jump search has an advantage that we traverse back only once (Binary Search may require up to O(Log n) jumps, consider a situation where the element to be search is the smallest element or smaller than the smallest). So in a systems where jumping back is costly, we use Jump Search.

References:

https://en.wikipedia.org/wiki/Jump_search

This article is contributed by **Harsh Agarwal**. 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.