Date
20/10/2020
30 Oct

Assignment No. 04

Searching Operations

2118

Name: Shubham Chemate

Class: SE-1 (E-1)

Problem statement:

a) Write a python program to store roll numbers of students in array who attended training program in random order. Write function for searching wheather particular student attended training program or not, using linear search & sentinel search.

b) Write a python program to store roll numbers of students in an array who attended training program in sorted order. Write a function for searching whether particular students attended training program or not using binary search & fibonacci search.

Objective: To learn & implement the following searching algorithms using python language 1) linear search 2) Sentinel search 3) Binary search 4) fibonacci Search.

Outcome: learn how the searching algorithm works & get knowledge of space & time complexity of that algorithms.

Software Requirements:

OS: Windows 10

Python version: 3.8.5

Vs code (text editor): Version 1.49.1 (Aug. 2020 version)

Hardware Requirement:

Manufactures : Acer

Processor: Intel(R) i.5 . 8265U CPU @ 1.80 GHz

Installed memory(RAM): 8GB

System Type. 64-bit OS , x 64-based architecture

Theory:

1) Linear search:

It is a simplest searching algorithm.

In this algorithm, we check array element by element sequentially from first element to the last element.

If we get the match, stop checking array & return the index we have found. but if after checking whole array we don't find element that means element is not present in the array.

Although, it is simplest algorithm, the running time of algorithm is O(n) because we have to check every element in worst case.

We don't need extra space for the algorithm.

2) ~~Sequential~~ Sentinel Search:

This algorithm is a modification of previous searching algorithm i.e. Linear Search. In linear search there were some unneccessary comparisons which are avoided in sentinel search.

In this algorithm the element to find (let x) is appended at the end of the list. we check list element by element. After getting the element we check that index. If index is equal to last element that means element is not present in the list. Otherwise the element is present in the list.

The time complexity of this algorithm is O(n) but due to less comparisons it is efficient than linear search.

3) Binary Search:

This is most used searching algorithm in practical world do to it's efficient time complexity.

This algorithm only works for sorted array. In this algorithm we check x ~~is~~th middle element of array if they matched we return middle index. else if x is less than middle element we continue our search in left part else we continue our search in right part of array.

As in each iteration we eliminate half part of array (taking advantage of sorted list) this algorithm runs very fast. The running time of algorithm is O(logn) which is very huge improvement over previous two searching algorithms.

4) Fibonacci Search:

It is a method of searching a sorted array using divide & conquer algorithm that narrows down possible location of element using generated fibonacci sequence. Some properties ~~are~~ of fibonacci sequence are used to reduce search area. Compared to binary search fibonacci search also gives same asymptotic time complexity i.e. O(logn). The advantage of fibonacci search over binary search is that it doesn't uses complex ~~bin~~ operations like ~~~~ multiplication & division. It only used addition & substraction operation which gives this algorithm little edge.

Algorithms (Pseudo Code)

1) Linear Search

```
1. Algorithm LinearSearch (list A, element x):   // returns index of x if present
2.     for i ← 0 to n:     // n is size of list
3         if A(i) == x:
4.            return i
5. return -1
```

classmate
Date
Page

2> Sentinel Search

1. Algorithm SentinelSearch (list A, element x): // return index if x is present
2. { A.append(x)
3. for i ← 0 to n+1 // n is size of i/p list
4. if A[i] == x:
5. break.
6. if (i <= n): return i
7. else return -1

3> Binary Search.

1. Algorithm BinarySearch (list A, element x): // returns index of if present
2. left ← 0 , right ← n // n is size of i/p list.
3. while (left <= right):
4. if (x == middle element of list)
5. return index of middle element.
6. else if (x is less than middle element)
7. right ← mid-1 // searching in left half.
8. else if (x is greater than middle element)
9. left ← mid+1 // searching in right half.
10. return -1 // if element not found return -1

4> fibonacci Search.

1. Algorithm fiboSearch (list A, element x): // returns index of x if present
2. find smallest fibonacci number greater than or equal to n. (let it be f_m)
f_{m-1} , f_{m-2} → numbers preceding f_m.
3. while (if there is an element in array):
4. compare x with range covered by f_{m-2}.
5. if x matches return index.
6. else if x is less, move f_m, f_{m-1}, f_{m-2} two steps down
7. else. if x is larger, move f_m, f_{m-1}, f_{m-2} one step down.

8. check for last remaining single element.

class Declaration:

```
Class Search6
    def __init__(self):
        self.students = []
        self.n = 0.
    def. readData(self):
        // reads data in self students
    def. sort(self):
        // sorts self.students.
    def linear search(self, x):
        // linear search implementation
    def Sentinel Search(self, x)
        // Sentinel search algorithm implementation
    def. BinarySearch(self, x):
        // Binary search algorithm implementation
    def fibonacciSearch(self, x)
        // fibonacci search algorithm implementation.
```

Analysis of Algorithms

Algorithm	Time Complexity	Space Complexity
1) Linear Search	searching the list element by element takes the time proportional to number of element in list. In asymptotic notation, time complexity of linear search is O(n).	The algorithm doesn't require creation of any additional data structure. It works on original list hence in asymptotic notation it require const. space.

4> Sentinel Search	This algorithm also search the list element by element same as linear search hence gives asymptotic time complexity as O(n)	Same as linear Search algorithm doesn't require any additional space hence it is a constant space algorithm
5. Binary Search	In each iteration the algorithm eliminate half elements of list & operate on half. $\therefore$ $T(n) = T(n/2) + c$ $= 2c + T(n/2^2)$ $= kc + T(n/2^k)$ at $n=1$, $T(1) = c$ $\therefore \frac{n}{2^k} = 1 \rightarrow n = 2^k$ $k = \log_2 n$ $\therefore T(n) = kc + T(1)$ $= kc + c$ $= \log_2 n \, (c+1)$ $= O(\log n)$ $\therefore$ The asymptotic time complexity of binary search is logarithmic i.e. O(log n)	The algorithm doesn't require any additional space, hence it is constant space algorithm.

classmate
Date ____
Page ____

4) fibonacci search	Same as binary search this algorithm eliminate ~1/3 or ~2/3 part of list in each iteration hence gives logarithmic time complexity. Asymptotic time complexity of algorithm is $O(\log n)$	If we consider no extra memory for fibonacci sequence the algorithm is constant space algorithm.

Test cases:

Test case No.	Input given	expected output	Actual output.
1.	Linear search on [2, 3, 5, 1] for $x = 3$	Index : 1	index : 1
2)	Linear search on [2, 3, 6, 4, 1] for $x = 5$	Not present	index : -1
3)	Sentinel search on [2, 3, 5, 1] for $x = 3$	index : 1	index : 1
3)	Sentinel search on [2, 3, 6, 4, 1] for $x = 5$	Not present	index : -1
4)	Binary search on [1, 3, 5, 7, 9] for $x = 7$.	index : 3	index : 3
5)	Binary search on [1, 3, 5, 7, 9] for $x = 6$	Not present	index : -1

6) fibonacci search on [1, 3, 7, 11, 15] for x = 12.	Not present	index : -1
7) fibonacci search on [1, 3, 7, 11, 15] for x = 11	index = 3	index : 3

Application:

The searching algorithms are one of the most used algorithms in computer science as well as in real life. They have wide application in every field of industry from banking to management.

Conclusion:

At the end of this assignment I'm able to use & program general searching ~~assign~~ algorithms for various applications in my day-to-day life.