Linear search algorithm

Source: https://github.com/rusevrosen/codehub.github.io/tree/main/Algorithms

1. What is a Linear search algorithm?

A linear search algorithm is a simple search algorithm that involves iterating through a list or array of elements one by one, starting at the beginning, and checking each element to see if it is the one you are looking for. If the element is found, the search stops, and the index of the element is returned. If the element is not found, the search returns **`-1`** or some other error value.

2. Code example.

```
#include <iostream>

// Function to perform linear search

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

{

    // Iterate through the array

    for (int i = 0; i < n; i++)

    {

        // Check if the current element is the one we are looking for

        if (arr[i] == x)

        {

            // If it is, return the index

            return i;

        }

    }

    // If the element is not found, return -1

    return -1;

}

int main()

{

    // Array to search

    int arr[] = {4, 2, 6, 1, 3};

    int n = sizeof(arr) / sizeof(arr[0]);

    // Element to search for

    int x = 6;

    // Perform the search

    int result = linear_search(arr, n, x);

    // Print the result

    if (result == -1)

    {

        std::cout << "Element not found" << std::endl;

    }

    else

    {

        std::cout << "Element found at index " << result << std::endl;

    }

    return 0;}
```

Output: `Element found at index 2`

This is because the element we are searching for (**6**) is located at **`index 2`** in the array. When the linear_search function is called, it iterates through the elements of the array and returns the index of the first element it finds that is equal to **6**. In this case, that element is located at **`index 2`**, so the function returns 2, and the code in the main function prints **`"Element found at index 2".`**

If the element we are searching for is not present in the array, the linear_search function will return **`-1.`** In this case, the code in the main function will print **`"Element not found".`**

3. *Code explanation.*

The code includes the necessary header file iostream, which provides input and output streams (e.g., cin and cout) that we will use later to output the result of the search.

Next, the code defines a function called linear_search that takes three arguments:

arr: an array of integers that we want to search

n: the length of the array

x: the element we are looking for

The function returns an integer, which will be the index of the element if it is found, or -1 if it is not found.

The function begins by iterating through the elements of the array using a **`for`** loop. For each element, it checks to see if the element is equal to the value we are looking for **`(x).`** If it is, the function returns the index of the element (i). If the element is not found after iterating through the entire array, the function returns -1.

In the main function, the code declares an array of integers called **`arr`** and initializes it with the values **`{4, 2, 6, 1, 3}`**. It then calculates the length of the array **`(n)`** and assigns it to the variable **`n`**.

Next, the code declares a variable called **`x`** and initializes it with the value **`6`**, which is the element we want to search for in the array.

The code then calls the linear_search function, passing it the array, the length of the array, and the element we are looking for as arguments. It assigns the return value of the function to a variable called the **`result`**.

Finally, the code uses an if statement to check the value of the **`result`**. If it is -1, the element was not found, so the code prints a message saying **`"Element not found".`** If it is not -1, the element was found, so the code prints a message saying **`"Element found at index [result]"`**.

4. Time complexity.

The time complexity of the **linear search** algorithm implemented above is **O(n),** where n is the length of the array being searched. This means that the time it takes for the algorithm to execute will increase linearly with the size of the input.

In other words, if the array has 10 elements, the algorithm will take about 10 times longer to execute than if the array had only 1 element. If the array has 100 elements, the algorithm will take about 100 times longer to execute, and so on.

This makes **linear search** less efficient than some other search algorithms when working with large datasets. However, it has the advantage of being simple to implement and easy to understand, which can make it a good choice for certain types of problems.

If you need to perform searches on large datasets and need a more efficient search algorithm, you might want to consider using a **binary search** or a **hash table**, both of which have a time complexity of **O(log n).**

I wish you happy learning,

Your Rosen Rusev