جامـــــعة المـــــستـقبـل
AL MUSTAQBAL UNIVERSITY

قســـم الامـــــن الـــــــــسيبرانـــــــي

Department of Cyber Security

Subject:

Searching And Sorting Algorithms

Class:

Second

Lecturer:

Asst. Prof. Dr. Ali Kadhum Al-Quraby

Lecture: (1)

Introduction

There are basically ***two aspects*** of computer programming. ***One is data organization*** also commonly called as ***data structures***. Till now we have seen about data structures and the techniques and algorithms used to access them. The ***other part*** of computer programming involves ***choosing the appropriate algorithm*** to solve the problem. Data structures and algorithms are linked each other. After developing programming techniques to represent information, it is logical to proceed to manipulate it. This lecture introduces this important aspect of problem solving.

INTRODUCTION TO SEARCHING AND SORTING ALGORITHMS

تُعتبر خوارزمية البحث ركنا أساسيا من أركان علم الخوارزميات وتتخذ هذه الخوارزمية عدة أشكال، من أبسطها البحث عن عدد في مصفوفة مُحددة الحجم ويزداد الأمر تعقيدا عند الانتقال إلى البحث عن كلمة داخل نص، تماما كما ترى في محررات النصوص العادية والتي تحتوي على خاصية Find & Replace حيث أن أغلب المحررات الحالية تستخدم خوارزميه بحث تسمى Boyer-Moore Searching التي تُعد تقريبا من أسرع الخوارزميات في مجال البحث. هناك أيضا بحث من نوع آخر، فماذا لو كانت لدينا مجموعة حروف ونريد إيجاد جميع الكلمات التي تبدأ بهذه الحروف؟؟ عادة ما يُسمى هذا النوع من الخوارزميات ب prefix searching ولهذا النوع تطبيقات كثيرة خصوصا في محركات البحث والقواميس والمتصفحات التي تستخدم هذه الخوارزمية عند كتابتك لموقع يبدأ بحرف كنت قد زرته سابقا.

لا تقتصر خوارزمية البحث على ما ذكرناه آنفا، فهناك نوع آخر من الخوارزميات يُستخدم لإيجاد نص قريب من النص الذي كنت تبحث عنه، حيث تقوم الخوارزمية بالبحث عن 4 أو 5 كلمات قريبة من الكلمات الخاطئة، تماما كما يفعل Google عند الترجمة أو Office Word عند كتابة نص يحتوي على أخطاء إملائية وتعتمد أغلب هذه الخوارزميات على الوزن الصوتي للحرف ومن أشهرها Soundex Searching

ليس هذا فقط، ***فمضادات الفيروسات*** تستخدم خوارزميات بحث سريعة للبحث عن وجود توقيع مطابق لأحد بيانات الملف المراد فحصه، وبما أن قواعد بيانات التواقيع تكون ضخمة للغاية فالبحث عن كل توقيع

سيكون بطيئا جدا وهناك الأفضل، حيث توجد خوارزميات بإمكانها البحث عن عده تواقيع في نفس اللحظة وتستخدم بنية شجرية للقيام بهذا الأمر، هناك أيضا خوارزميات أخرى تعتمد على مفاهيم مختلفة مثل Hash Table وعدة أمور أخرى، و يُسمى هذا النوع من الخوارزميات ب Multiple pattern searching و هو من أصعب الخوارزميات، وهناك العديد من مضادات الفيروسات التي تستخدم مثل هذه الخوارزميات مثل ClamAV .

Searching is an operation or a technique that helps finds the place of a given element or value in the list. Any search is said to be successful or unsuccessful depending upon whether the element that is being searched is found or not. Some of the standard searching technique that is being followed in data structure is listed below:

1. **Linear or sequential search**
2. **Binary search**

Sorting allows an efficient arrangement of elements within a given data structure. It is a way in which the elements are organized systematically for some purpose. For example, a dictionary in which words is arranged in alphabetical order and telephone director in which the subscriber names are listed in alphabetical order. There are many sorting techniques out of which we study the following.

1. **Bubble sort**
2. **Quick sort**
3. **Selection sort and**
4. **Heap sort**

There are two types of sorting techniques:

1. **Internal sorting**
2. **External sorting**

If all the elements to be sorted are present in the ***main memory***, then such sorting is called ***internal sorting*** on the other hand, if some of the elements to be sorted are kept on the ***secondary storage***, it is called ***external sorting***. Here we study only internal sorting techniques.

1. Linear Search:

This is the ***simplest*** of all searching techniques. In this technique, an ordered or unordered list will be searched ***one by one*** from the beginning until the desired element is found. If the desired element is ***found*** in the list, then the search is ***successful*** otherwise ***unsuccessful***.

Suppose there are "n" elements organized sequentially on a List. The number of comparisons required to retrieve an element from the list, purely depends on where the element is stored in the list. If it is the first element, one comparison will do; if it is second element two comparisons are necessary and so on. On an average you need [(n+1)/2] comparisons to search an element. If search is not successful, you would need "n" comparisons.

The time complexity of linear search is ***O(n)***.

Algorithm

LinearSearch (Array A, Value x)
Step 1: Set i to 1
Step 2: if i > n then go to step 7
Step 3: if A[i] = x then go to step 6
Step 4: Set i to i + 1
Step 5: Go to Step 2
Step 6: Print Element x Found at index i and go to step 8
Step 7: Print element not found
Step 8: Exit

Here is a simple C++ program that demonstrates sequential search (also known as linear search). This search algorithm checks each element in a list (or array) one by one to find the target value.

Sequential Search Example

```cpp
#include <iostream>
using namespace std;

// Function for sequential search
int sequentialSearch(int arr[], int size, int target) {
    for (int i = 0; i < size; i++) {
        if (arr[i] == target) {
            return i; // Return index if target is found
        }
    }
    return -1; // Return -1 if target is not found
}

int main() {
    int arr[] = {10, 20, 30, 40, 50}; // Example array
    int target;

    cout << "Enter the value to search: ";
    cin >> target;

    int size = sizeof(arr) / sizeof(arr[0]); // Calculate the size of the array
    int result = sequentialSearch(arr, size, target);

    if (result != -1) {
        cout << "Element found at index: " << result << endl;
    } else {
        cout << "Element not found in the array." << endl;
    }

    return 0;
}
```

Example 1:

Let us illustrate linear search on the following 9 elements:

Index	*0*	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*
Elements	-15	-6	0	7	9	23	54	82	101

Searching different elements is as follows:

1. Searching for x = 7, Search successful, data found at 3rd position.
2. Searching for x = 82, Search successful, data found at 7th position.
3. Searching for x = 42, Search un-successful, data not found.

Example 2: Let us take an example of an array A[7]={5,2,1,6,3,7,8}. Array A has 7 items. Let us assume we are looking for 7 in the array. Targeted item=7.

Here, we have

Index	**0**	**1**	**2**	**3**	**4**	**5**	**6**
Element	**5**	**2**	**1**	**6**	**3**	**7**	**8**

A[7]={5,2,1,6,3,7,8}

X=7

At first, When i=0 (A[0]=5; X=7) not matched
i++ now, i=1 (A[1]=2; X=7) not matched
i++ now, i=2 (A[2])=1; X=7) not matched
…
….
i++ when, i=5 (A[5]=7; X=7) **Match Found**
Hence, Element X=7 found at index 5.

❖ A Recursive program for linear search

Here is a recursive C++ program to perform linear (sequential) search:

Recursive Linear Search

```cpp
#include <iostream>
using namespace std;

// Recursive function for linear search
int recursiveLinearSearch(int arr[], int size, int target, int index = 0) {
    // Base case: if index exceeds the size of the array, target is not found
    if (index >= size) {
        return -1;
    }
    // Check if the current element matches the target
    if (arr[index] == target) {
        return index;
    }
    // Recursive call to check the next element
    return recursiveLinearSearch(arr, size, target, index + 1);
}

int main() {
    int arr[] = {10, 20, 30, 40, 50}; // Example array
    int target;

    cout << "Enter the value to search: ";
    cin >> target;

    int size = sizeof(arr) / sizeof(arr[0]); // Calculate the size of the array
    int result = recursiveLinearSearch(arr, size, target);

    if (result != -1) {
        cout << "Element found at index: " << result << endl;
    } else {
        cout << "Element not found in the array." << endl;
    }

    return 0;
}
```