Module 6: Searching

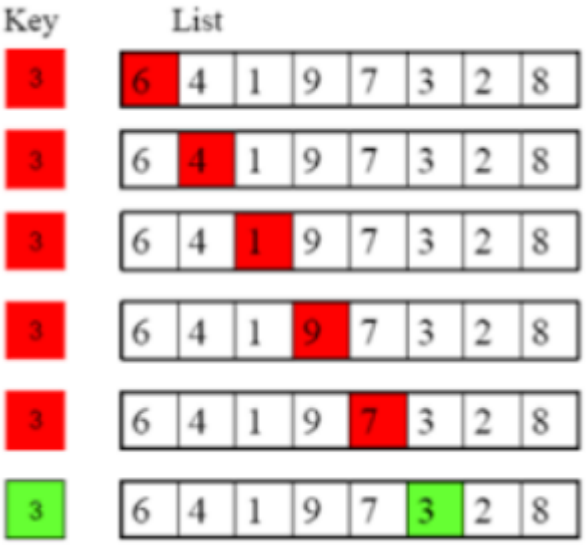
- Searching Algorithms
- Compare String
- Searching in Struct

Searching Algorithms

Searching is an algorithm used to find a specific data element within a dataset. In this module, we will discuss two common searching methods: Linear Search and Binary Search.

Linear Search

The simplest and most commonly used searching algorithm is the sequential or linear search. The way this algorithm works is by comparing the search key with each element in the list sequentially until the desired data is found or until all elements have been checked.



Below is an illustration of the linear search process. The search key is the number 3, and it is compared one by one until the number 3 is found.

Example Program using Linear Search:

```
#include <stdio.h>
#define SIZE 10

int main() {
    int arr[SIZE] = {15, 7, 2, 10, 27, 77, 32, 43, 69, 56};
    int i, input;
```

```
    printf("Enter the number to search: ");
    scanf("%d", &input);

    for (i = 0; i < SIZE; i++) {
        if (arr[i] == input) {
            printf("\n%d is located at position %d\n", input, i+1);
            break;
        }
    }

    if (i == SIZE) {
        printf("\nNot found in the list!\n");
    }

    return 0;
}
```

Binary Search

Binary search is an algorithm that searches for a data element within a sorted list. This method is faster than Linear Search but requires the list to be sorted in ascending order beforehand. The algorithm repeatedly divides the search range into two halves. If the search key is smaller than the middle element, the search continues in the left half. If the search key is larger, the search continues in the right half.

Below is an illustration of the binary search process. The input key is 8, so the search range is reduced to the right half of the list.

Example Program using Binary Search:

```c
#include <stdio.h>
#define SIZE 7

int main() {
    int arr[SIZE] = {3, 8, 16, 29, 32, 47, 66};
    int left, right, middle, input;

    printf("Enter the number to search: ");
    scanf("%d", &input);

    left = 0;
    right = SIZE - 1;

    while (left <= right) {
        middle = (left + right) / 2;

        // Check if the middle element is the target
        if (arr[middle] == input) {
            printf("\n%d is located at position %d\n", input, middle+1);
            break;
        }

        // Adjust search range
        if (arr[middle] < input)
            left = middle + 1;
        else
            right = middle - 1;
    }

    if (left > right)
        printf("\nNot found in the list!\n");

    return 0;
}
```

Compare String

Introduction

The `strcmp` function is part of the standard C library and is used to compare two strings. This function determines the lexicographical order of the given strings.

Syntax

```
#include <string.h>


int strcmp(char str1[], char str2[]);
```

- **Parameters:**
  - `str1`: Array representing the first string.
  - `str2`: Array representing the second string.
- **Return Value:**
  - Returns a value less than 0 if `str1` is less than `str2`.
  - Returns 0 if `str1` is equal to `str2`.
  - Returns a value greater than 0 if `str1` is greater than `str2`.

Explanation

The `strcmp` function compares two strings character by character in a lexicographical manner. The comparison stops when a difference is found or when the end of one of the strings is reached.

Example Usage

Below is a simple C program demonstrating the use of `strcmp`:

```
#include <stdio.h>
#include <string.h>
```

```
int main() {
    char str1[] = "Geeks";
    char str2[] = "Geeks";
    char str3[] = "GEEKS";

    // Comparing str1 and str2
    int result = strcmp(str1, str2);
    if (result == 0) {
        printf("str1 and str2 are equal.\n");
    } else {
        printf("str1 and str2 are different.\n");
    }

    // Comparing str1 and str3
    result = strcmp(str1, str3);
    if (result == 0) {
        printf("str1 and str3 are equal.\n");
    } else {
        printf("str1 and str3 are different.\n");
    }

    return 0;
}
```

Expected Output:

```
str1 and str2 are equal.
str1 and str3 are different.
```

In the example above, `strcmp(str1, str2)` returns 0 because both strings are identical. However, `strcmp(str1, str3)` returns a nonzero value because of the difference in letter casing.

Using `strcmp` in Searching

The `strcmp` function can also be used in searching within an array of strings. Below is an example demonstrating how to search for a string in an array using `strcmp`:

```c
#include <stdio.h>
#include <string.h>
#define SIZE 5

int main() {
    char names[SIZE][20] = {"Alice", "Bob", "Charlie", "David", "Eve"};
    char search[20];
    int found = 0;

    printf("Enter a name to search: ");
    scanf("%s", search);

    for (int i = 0; i < SIZE; i++) {
        if (strcmp(names[i], search) == 0) {
            printf("%s found at position %d\n", search, i + 1);
            found = 1;
            break;
        }
    }

    if (!found) {
        printf("%s not found in the list.\n", search);
    }

    return 0;
}
```

Expected Output:

```
Enter a name to search: Bob
Bob found at position 2
```

This program takes user input for a name and searches for it in the predefined list using `strcmp`.

Important Notes

- The `strcmp` function is **case-sensitive**. To perform a case-insensitive comparison, you can use `strcasecmp` (available on some platforms) or convert both strings to lowercase or

uppercase before comparing.
- Ensure both string arrays are valid and contain null-terminated strings (`'\0'`) to avoid undefined behavior.

Searching in Struct

Introduction

In C, structures (`struct`) allow grouping different types of data together. Sometimes, we need to search for specific records inside an array of structures. This module explains how to perform searching operations on an array of structures in C using both **Linear Search** and **Binary Search**.

Defining a Structure

Before searching, let's define a simple structure to store student information:

```
#include <stdio.h>
#include <string.h>

#define SIZE 5

struct Student {
    int id;
    char name[20];
    float grade;
};
```

Here, the `Student` struct contains three fields:

- `id`: an integer representing the student ID.
- `name`: a character array storing the student's name.
- `grade`: a floating-point number representing the student's grade.

Linear Search on Struct Array

Linear search iterates through each element in the struct array until a match is found.

Example: Searching for a Student by Name

```
int main() {
    struct Student students[SIZE] = {
        {101, "Alice", 85.5},
        {102, "Bob", 78.0},
        {103, "Charlie", 92.0},
        {104, "David", 88.5},
        {105, "Eve", 90.0}
    };

    char searchName[20];
    int found = 0;

    printf("Enter student name to search: ");
    scanf("%s", searchName);

    for (int i = 0; i < SIZE; i++) {
        if (strcmp(students[i].name, searchName) == 0) {
            printf("Student found: ID=%d, Name=%s, Grade=%.2f\n", students[i].id, students[i].name,
students[i].grade);
            found = 1;
            break;
        }
    }

    if (!found) {
        printf("Student not found!\n");
    }

    return 0;
}
```

Output Example:

```
Enter student name to search: Bob
Student found: ID=102, Name=Bob, Grade=78.00
```

Binary Search on Struct Array

Binary search is faster than linear search but requires the array to be sorted. It repeatedly divides the array into halves to locate the desired element.

Example: Searching for a Student by ID (Binary Search)

```
int binarySearch(struct Student students[], int left, int right, int key) {
    while (left <= right) {
        int mid = (left + right) / 2;
        if (students[mid].id == key)
            return mid;
        if (students[mid].id < key)
            left = mid + 1;
        else
            right = mid - 1;
    }
    return -1;
}

int main() {
    struct Student students[SIZE] = {
        {101, "Alice", 85.5},
        {102, "Bob", 78.0},
        {103, "Charlie", 92.0},
        {104, "David", 88.5},
        {105, "Eve", 90.0}
    };

    int searchID;
    printf("Enter student ID to search: ");
    scanf("%d", &searchID);
```

```
    int result = binarySearch(students, 0, SIZE - 1, searchID);
    if (result != -1)
        printf("Student found: ID=%d, Name=%s, Grade=%.2f\n", students[result].id, students[result].name,
students[result].grade);
    else
        printf("Student not found!\n");

    return 0;
}
```

Output Example:

```
Enter student ID to search: 103
Student found: ID=103, Name=Charlie, Grade=92.00
```

Conclusion

- **Linear Search** is simple and works on unsorted data but is slower for large datasets.
- **Binary Search** is much faster but requires the array to be sorted.
- The `strcmp` function is useful for searching strings within structures.