CS 3EA3 Sheet 5 Optional Assignment

Yunfeng Li

April 26, 2017

1 Abstract

A report on linear search and binary search algorithms is presented here. Searching algorithm plays a significant role in modern software engineering development, however, it is very difficult to implement it correctly and sometimes, it is even harder to implement it for a certain data structure. In this report, I will discuss my implementation of the linear search on C linked list and the binary search implementation on C normal array.

2 Introduction

The report first introduces the implementation of the algorithms and then discusses the technical details and correctness. Following that is the comparison of the two searching algorithms. It includes some of the implementation difficulties in the end.

3 Implementation

First, it is a brief introduction. To implement a C linked list, the general way that people adopt is to use C `struct`. The underlying node of linked list is consist of a node value and a pointer to the next node:

```
struct List {
    int  value;
    struct List *next;
};
```

In the C language, it is always more efficient to use pointers rather than direct access to the next element, because it could potentially copy the whole memory content of the next element and that is very expensive.

The linear search implmentation on C linked list:

```
#include <stdio.h>

struct List {
    int  value;
    struct List *next;
};

/*@
  @ requires \valid(p_list);
```

```
*/
int linear_search(struct List *p_list, int key)
{
    while (p_list != NULL) {
        if (p_list->value == key)
            return 1;
        p_list = p_list->next;
    }

    return 0;
}
```

We need to pass two variables into the searching function. The first one is the address of the first node of the linked list, and the second is the value that we are looking for. We achieve it by checking each node in linked order, if the current node is not `NULL`, we check the next node connected to it.

The binary search implementation on C normal array:

```
/*@
  @ requires len >= 1;
  @ requires \forall int i; 0 <= i <= len - 1 ==> arr[i] <= 9999;
  @ requires \valid(arr+(0..len-1));
  @ requires \exists int e; 0 <= e <= len - 1 ==> arr[e] == key;
  @ ensures \exists int e; 0 <= e <= len - 1 ==> arr[\result] == arr[e];
*/

int binary_search(int *arr, int len, int key)
{
    int low, high, mid;
    //@ assert len >= 1;
    low = 0;
    high = len;

    /*@ loop invariant 0 <= low < high <= len;
      @ loop variant high - low;
    */
    while (low + 1 != high) {
        mid = (low + high) / 2;
        if (arr[mid] <= key)
            low = mid;
        if (arr[mid] > key)
            high = mid;
    }
    return low;
}
```

From a research blog that is accessible from the course web page, the above implementation is not bug-free. According to the blog, the bug is caused by `mid = (low + high) / 2`. The computing of the sum of `low` and `high` could potentially cause an overflow. For the particular case here, I assume the value of each array

element is not too large to cause the problem.

4 Comparison

The general difference between the two introduced algorithms is that for a C array, we can have the array length, and the list length is not available in the linked list approach. Therefore, we cannot find the middle point of the linked list and then divide the linked list into two partitions. It is also noticed that we use the array index to locate each array element, and its time complexity is $O(1)$, so it is feasible to compare each element directly with the `key` value. However, in linked list, each node only keeps the information of the next node, it is impossible to jump over several nodes to find the corresponding value.

5 Conclusion

During the implementation, one of the hurdles I found is that it is very challenging to extract post-condition for a problem if it is not arithmetic counting. In the implementation that I provide, there is no any information on the position of each node in the whole linked list. As I understand, it could be almost impossible to write the post-condition for the linear search without writing some helper functions to memorize the location. We could include location information on each node, and it virtually becomes an array, so we can use it to locate the next node. However, it could be very problematic because whenever it needs to concatenate two linked list, we need to modify each node in one of the two lists, it is expensive and error-prone.