06. 자료구조와 알고리즘

05. 선형 검색과 이진 검색

학습목표

- 선형 검색을 할 수 있다.
- 이진 검색을 할 수 있다.

개요

리스트에서 검색을 하는 방법에는 2가지가 있다. 이번 시간에는 선형 검색과 이진 검색에 대해서 알아보도록 하자.

선형 검색

선형 검색(Linear Search)는 리스트 전체를 하나씩 순회하면서 검색하는 방식이다. 선형 리스트와 연결 리스트 모두에서 사용할 수 있다. 시간 복잡도는 $O(n)$이고, 공간 복잡도는 $O(1)$이다.

```
bool LinearSearch(List<int> list, int item)
{
  // 리스트 전체를 순회하면서
  foreach (int elem in list)
  {
    // 일치하는 것을 찾는다.
    if (elem == item)
    {
      return true;
    }
  }

  // 만약 리스트 전체를 순회했음에도 찾지 못했다면 없는 것이다.
```

```
  return false;
}


// 연결 리스트도 동일하다.
bool LinearSearch(LinkedList<int> list, int item)
{
 foreach (int elem in list)
 {
    if (elem == item)
    {
       return true;
    }
 }

 return false;
}
```

이진 검색

이진 검색(Binary Search)은 검색할 범위를 절반으로 줄여가며 검색하는 방법이다. 이진 검색은 정렬된 선형 리스트에서 사용할 수 있다. 시간 복잡도는 $O(logn)$이며, 공간 복잡도는 $O(1)$이다.

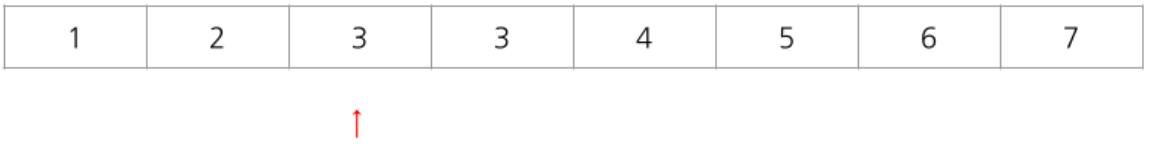
```
bool BinarySearch(List<int> list, int item)
{
  // list는 반드시 정렬되어 있어야 한다.
  // 오름차순으로 정렬되어 있던,
  // 내림차순으로 정렬되어 있던 상관은 없지만
  // 구현이 달라지게 된다.
  // 여기서는 오름차순으로 정렬되어 있다고 가정한다.



  // 1. 주어진 검색 범위에서 중간값을 찾는다.[s, e)
```

```
int s = 0;          // 시작 인덱스
int e = list.Count; // 끝 인덱스



// 범위가 유효한지 확인한다.
while (s < e)
{
    // 오버플로우를 막기 위해 아래처럼 사용할 수 있다.
    // int m = s + (e - s) / 2;
    int m = (s + e) / 2; // 중간 인덱스

    // 2. 중간값과 key를 비교한다.
    // 2-1. key와 중간값이 같은 경우 => 찾았다.
    if (list[m] == item)
    {
        return true;
    }
    // 2-2.찾고자 하는 쪽이 더 크다면
    else if (list[m] < item)
    {
        // 검색 범위를 줄인다. (오른쪽)
        s = m + 1;
    }
    // 2-3.찾고자 하는 쪽이 더 작다면
    else
    {
        // 검색 범위를 줄인다.(왼쪽)
        e = m;
    }
}
```

```
  // 못 찾은 것이다.
  return false;
}
```

LowerBound

이진 검색을 응용해 범위 중 특정 값보다 크거나 같은 첫 번째 원소를 찾아낼 수 있다. 아래와 같은 리스트에 LowerBound(3)을 한다면 빨간색 화살표의 인덱스를 반환한다.



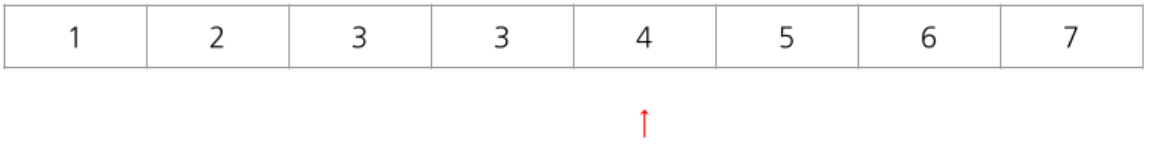
코드로 구현하면 아래와 같다.

```
// value <= element을 만족하는 첫 번째 원소를 찾는다.
// 찾았을 시 해당 원소가 위치한 인덱스를 아니면 -1이다.
int LowerBound(List<int> list, int value)
{
  // list는 오름차순으로 정렬되어 있다고 가정한다.
  int foundIndex = -1;
  int s = 0;
  int e = list.Count;

  while (s < e)
  {
    int m = (s + e) / 2;
    // 같거나 => 검색 범위를 왼쪽으로 줄여야 한다.
    // value가 작거나 => 검색 범위를 왼쪽으로 줄여야 한다.
    if (value <= list[m])
    {
      // 후보군을 저장한다.
      foundIndex = m;
```

```
        // 더 작은 것은 없는지 범위를 옮겨 확인한다.
        e = m;
    }
    // value가 크거나 => 검색 범위를 오른쪽으로
    else
    {
        // 시작 지점을 옮겨 확인한다.
        s = m + 1;
    }
}

return foundIndex;
}
```

UpperBound

이진 검색을 응용해 범위 중 특정 값보다 큰 첫 번째 원소를 찾아낼 수 있다. 아래와 같은 리스트에 UpperBound(3)을 한다면 빨간색 화살표의 인덱스를 반환한다.



코드로 구현하면 아래와 같다.

```
// value < element를 첫 번째 원소를 찾는다.
// 찾았을 시 해당 원소가 위치한 인덱스를 아니면 -1이다.
int UpperBound(List<int> list, int value)
{
// list는 오름차순으로 정렬되어 있다고 가정한다.
int foundIndex = -1;
int s = 0;
int e = list.Count;
```

```
while (s < e)
{
    int m = (s + e) / 2;

    if (value < list[m])
    {
        // 후보군을 저장한다.
        foundIndex = m;

        // 더 작은 것은 없는지 범위를 옮겨 확인한다.
        e = m;
    }
    else
    {
        // 범위를 만족하기 않기 때문에 시작 지점을 옮긴다.
        s = m + 1;
    }
}

return foundIndex;
}
```

라이브러리

선형 검색의 경우 Array.Find() / Array.FindAll() / Array.FindIndex() / Array.FindLast() / Array/FindLastIndex() / Array.IndexOf() / Array.LastIndexof() / List.Contains() / List.Find() / List.FindAll() / List.Exists() / List.FindIndex() / List.FindLast() / List.FindLastIndex() / List.IndexOf() / List.LastIndexOf()로 구현되어 있고, 이진 검색의 경우 Array.BinarySearch() / Array.GetLowerBound() / Array.GetUpperBound() / List.BinarySearch()로 구현되어 있다.

더해보기

1. 아래의 문제를 풀어봅시다.
    a. 1920번: 수 찾기

b. 10816번: 숫자 카드 2
c. 1654번: 랜선 자르기
d. 2805번: 나무 자르기
e. 2110번: 공유기 설치

참고자료

- C로 배우는 쉬운 자료구조
- 누구나 자료 구조와 알고리즘