06. 자료구조와 알고리즘

04. 리스트

학습목표

- 리스트가 무엇인지 알 수 있다.
- 리스트를 적재적소에 활용할 수 있다.

들어가며

리스트(list)는 순서를 갖고 있는 자료 구조다. 순차 자료 구조엔 선형 리스트가 있고, 연결 자료 구조엔 단일 연결 리스트, 원형 연결 리스트, 이중 연결 리스트가 있다. 리스트는 다른 자료 구조의 기본이 되기 때문에 꼭 알아둬야 한다. 이번 시간에는 리스트의 추상 자료형과 관련 컬렉션 사용을 알아보도록 하자.

추상 자료형

리스트는 연산에 대해 아래와 같이 동작해야 한다.

- 읽기
    - 인덱스 혹은 반복자를 통해 특정 원소에 접근해야 한다.
- 검색
    - 특정 원소 혹은 특정 범위의 원소들을 검색할 수 있어야 한다.
- 삽입
    - 리스트에서 아무 위치에나 원소가 삽입되어야 한다.
- 삭제
    - 리스트에서 특정 원소를 삭제해야 한다.

선형 리스트

선형 리스트(Linear List)는 순서 리스트(Ordered List)라고도 한다. 순차적으로 구현한 것이기 때문에 임의 접근이 가능하다.

사용법

C#에서는 Array와 List로 구현되어 있다.* Array는 이미 많이 사용했기 때문에 List의 사용법을 확인해보자.

* 둘 다 IList 인터페이스를 구현하고 있다는 것을 알 수 있다.

* Array는 고정 길이의 배열, List는 가변 길이의 배열이다.

```
// 리스트를 생성한다.
List<int> list = new List<int>();


// # 삽입
// Add()는 리스트의 끝에 데이터를 추가한다.
list.Add(5); // [5]
list.Add(4); // [5][4]
list.Add(3); // [5][4][3]
list.Add(2); // [5][4][3][2]
list.Add(1); // [5][4][3][2][1]

// Insert()는 특정 위치에 데이터를 추가한다.
list.Insert(2, 6); // [5][4][6][3][2][1]


// # 읽기
// 인덱서를 통해 특정 위치의 데이터에 접근할 수 있다.
Console.WriteLine(list[3]); // 3
```

```
// # 검색
// Contains()를 통해 데이터의 존재 유무를 확인할 수 있다.
Console.WriteLine(list.Contains(6)); // True
Console.WriteLine(list.Contains(7)); // False

// IndexOf()를 사용하면 데이터가 몇 번째 인덱스에 있는지도 확인할 수 있다.
Console.WriteLine(list.IndexOf(6)); // 2
Console.WriteLine(list.IndexOf(7)); // -1

// # 삭제
// Remove()를 이용하면 특정 데이터를 삭제할 수 있다.
// 중복된 데이터가 있는 경우 첫 번째로 발견된 데이터를 삭제한다.
list.Remove(6); // [5][4][3][2][1]

// RemoveAt()은 특정 위치에 존재하는 데이터를 삭제한다.
list.RemoveAt(3); // [5][4][3][1]
```

성능

각 연산에 대한 성능을 알아보자

- 읽기
  - 선형 리스트는 임의 접근이 가능하기 때문에 $O(1)$의 시간이 걸린다.
- 검색
  - 하나하나 원소를 비교해가야 하므로 $O(n)$의 시간이 걸린다. 정렬되어 있다면 이진 검색*을 사용할 수 있으며 이 경우 $O(logn)$이다.

    * 이진 검색은 추후 배운다.
- 삽입

    - 위치에 따라 시간이 달라진다. 맨 끝일 경우 $O(1)$이지만, 처음이나 중간이라면 모든 데이터를 이동해야 하기에 $O(n)$이 걸린다.
- 삭제
    - 위치에 따라 시간이 달라진다. 맨 끝일 경우 $O(1)$이지만, 처음이나 중간이라면 모든 데이터를 이동해야 하기에 $O(n)$이 걸린다.

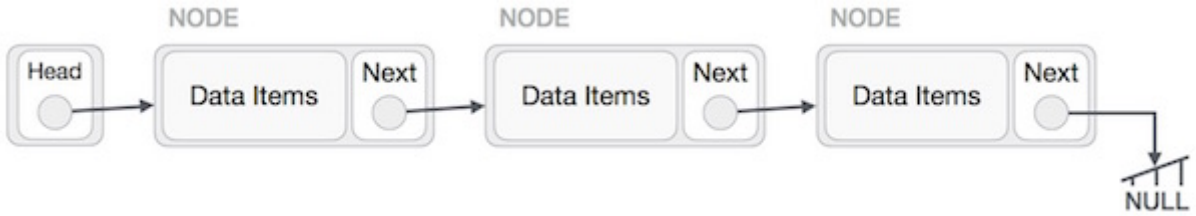
연결 리스트

연결 리스트는 메모리에 연속적으로 배치되어 있지 않아 임의 접근이 불가능하다. 먼저 연결 리스트의 종류에 대해서 알아야 한다.

종류

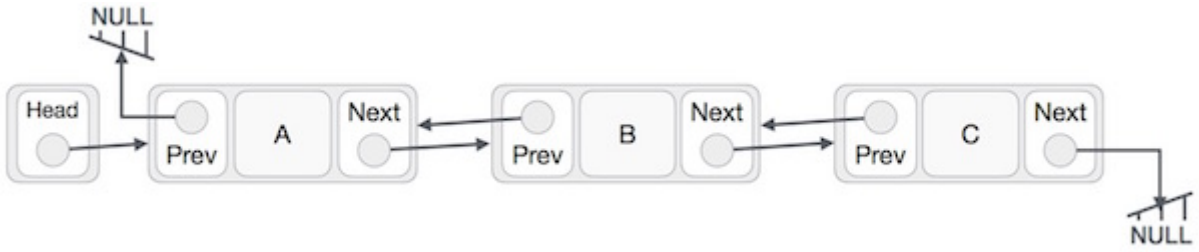
연결 리스트의 종류는 크게 세 가지다

- 단일 연결 리스트(Singly Linked List)
    - 다음 원소로만 이동할 수 있다.



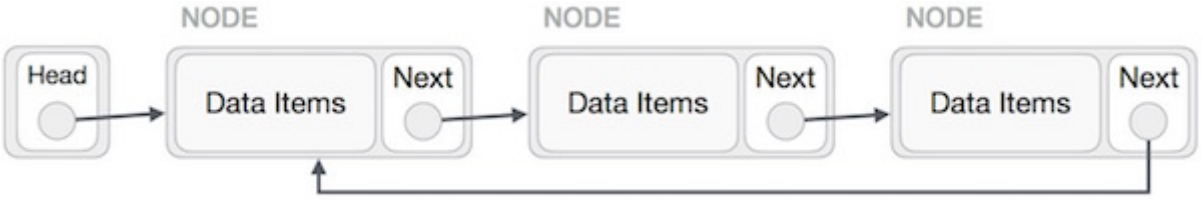
[단일 연결 리스트]

- 이중 연결 리스트(Doubly Linked List)
    - 이전 원소, 다음 원소 모두 이동할 수 있다.

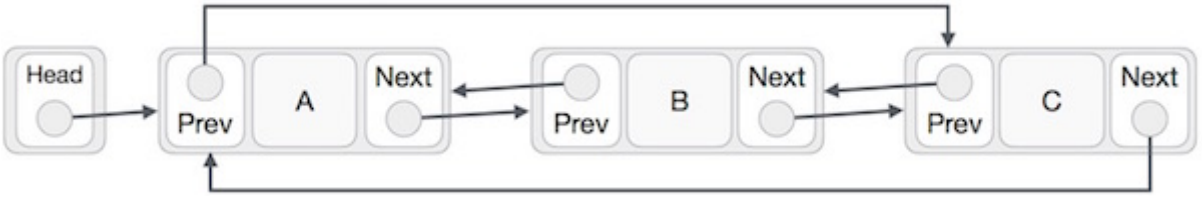


[이중 연결 리스트]

- 원형 연결 리스트(Circular Linked List)
    - 마지막 원소와 첫 번째 원소를 연결한 형태다.



단일 원형 연결 리스트



이중 원형 연결 리스트

사용법

연결 리스트는 이전 혹은 이후 데이터를 가리키기 위한 참조 타입의 변수가 필요하다.

```
// 연결 리스트를 구성하는 단위를 노드(Node)라고 한다.
class Node<T>
{
  public T Data { get; set; }

  // 실제 데이터 말고도 리스트를 구성하기 위한 추가 데이터가 필요하다.
  public Node Next { get; set; }
  public Node Prev { get; set; }
}
```

C#은 LinkedListNode로 구현이 되어 있으며, 연결 리스트는 LinkedList다. LinkedList는 이중 연결 리스트로 구현이 되어 있다.

```
// 리스트를 생성한다.
LinkedList<int> list = new LinkedList<int>();
```

```
// # 삽입
// AddLast()는 리스트의 끝에 데이터를 추가한다.
list.AddLast(5); // [5]
list.AddLast(4); // [5][4]


// AddFirst()는 리스트의 처음에 데이터를 추가한다.
list.AddFirst(1); // [1][5][4]
list.AddFirst(2); // [2][1][5][4]


// # 읽기
// First는 연결 리스트의 첫 번째 노드를 가져온다.
LinkedListNode<int> first = list.First; // [2][1][5][4]
                                        //  ↑


// Last는 연결 리스트의 마지막 노드를 가져온다.
LinkedListNode<int> last = list.Last;  // [2][1][5][4]
                                       //           ↑


// Next 혹은 Previous로 포인터를 이동할 수 있다.
LinkedListNode<int> node = first.Next;  // [2][1][5][4]
                                        //     ↑
```

```
// # 삽입
// AddAfter()를 이용하면 노드 다음에 데이터를 추가할 수 있다.
list.AddAfter(first, 10); // [2][10][1][5][4]

// AddBefore()을 이용하면 노드 이전에 데이터를 추가할 수 있다.
list.AddBefore(node, 8); // [2][10][8][1][5][4]

// # 검색
// Contains()를 통해 데이터의 존재 유무를 확인할 수 있다.
Console.WriteLine(list.Contains(10)); // True
Console.WriteLine(list.Contains(13)); // False

// Find()를 통해 데이터를 저장하고 있는 노드를 가져올 수 있다.
// Null일 수 있다.
Console.WriteLine(list.Find(8)?.Value); // 8

// # 삭제
// Remove()를 이용하면 특정 데이터를 삭제할 수 있다.
list.Remove(10); // [2][8][1][5][4]
list.Remove(6);  // [2][8][1][5][4]

// 삭제할 노드를 넣어도 된다.
```

```
list.Remove(list.Last.Previous); // [2][8][1][4]
```

성능

연결 리스트의 각 연산에 대한 성능은 아래와 같다.

- 읽기
  - 연결 리스트는 임의 접근이 불가능해 요소 하나하나를 탐색해야 하므로 $O(n)$의 시간이 걸린다. 하지만 처음과 끝이라면 구현에 따라 $O(1)$이 될 수 있다.
- 검색
  - 하나하나 원소를 비교해가야 하므로 $O(n)$의 시간이 걸린다. 선형 리스트와 다르게 이진 검색을 사용할 수 없다.
- 삽입
  - 노드의 위치를 정확히 알고 있다면 $O(1)$이지만, 노드의 위치를 모른다면 해당 위치까지 검색해야 하기 때문에 $O(n)$이 걸린다.
- 삭제
  - 노드의 위치를 정확히 알고 있다면 $O(1)$이지만, 노드의 위치를 모른다면 해당 위치까지 검색해야 하기 때문에 $O(n)$이 걸린다.

더 나아가기

1. 선형 리스트에서 맨 끝에 원소를 삽입하는 연산의 시간 복잡도는 O(1)이지만, 중간에 삽입하는 연산은 $O(n)$인 이유를 설명하시오.
2. 만약 데이터를 **자주 읽고(Read)** 특정 인덱스에 **접근**해야 하지만, 삽입/삭제는 드물게 발생하는 환경이라면 어떤 종류의 리스트를 사용하는 것이 성능상 유리하며 그 이유는 무엇인지 설명하시오.
3. 아래의 문제를 풀어보세요.
   a. 1406번: 에디터
   b. 5397번: 키로거
4. 아래의 코드 베이스를 기반으로 선형 리스트를 구현해보세요.

```
class MyLinearList
```

```
{
    // 데이터가 저장되어 있는 배열
    private int[] _container = null;
    // 리스트의 크기
    private int _size = 0;

    // _container[index]에 data를 삽입합니다.
    public void Insert(int index, int data)
    {
        // TODO
        // Insert()를 구현하세요.
        // 크기가 부족한 경우 자동으로 늘어나야 합니다.
    }

    // 리스트에 data가 저장된 인덱스를 반환하세요. 없다면 -1입니다.
    public int IndexOf(int data)
    {
        // TODO
        // IndexOf()를 구현하세요.
    }

    // index에 저장된 data를 삭제하세요.
    public void RemoveAt(int index)
    {
        // TODO
        // RemoveAt()을 구현하세요.
        // index는 유효한 인덱스만 입력된다고 가정합니다.
```

```
  }
}
```

5. 아래의 코드 베이스를 기반으로 이중 연결 리스트를 구현해보세요.

```
class Node
{
  public int Value { get; set; } // 데이터
  public Node Prev { get; set; } // 이전 노드
  public Node Next { get; set; } // 다음 노드
}

class MyLinkedList
{
  // 연결 리스트의 가장 첫 번째 노드
  // 적절히 초기화가 되어 있다고 가정한다.
  private Node _first;

  // 리스트에서 node를 삭제한다.
  public void Remove(Node node)
  {
    // TODO
    // Remove()를 구현하세요.
    // [2][1][3]에서 [1]을 참조하고 있는 node가 들어올 시
    // Remove() 수행 후에는 [2][3]이 되어야 합니다.
  }
```

```
    // node 이전에 data를 추가한다.
    public void AddFirst(Node node, int data)
    {
        // TODO
        // AddFirst()를 구현하세요.
        // [2][1][3]에서 [1]을 참조하고 있는 node와 data가 10이 들어올 때
        // AddFirst() 수행 후에는 [2][10][1][3]이 되어야 합니다.
    }


    // 리스트에서 data가 있다면 true, 없다면 false다.
    public bool Contains(int data)
    {
        // TODO
        // Contains()를 구현하세요.
    }
}
```

참고자료

- C로 배우는 쉬운 자료구조
- 누구나 자료 구조와 알고리즘
- [무료] C로 배우는 자료구조 및 여러가지 예제 실습
- Array Class (System) | Microsoft Learn
- List<T> Class (System.Collections.Generic) | Microsoft Learn
- LinkedListNode<T> Class (System.Collections.Generic) | Microsoft Learn
- LinkedList<T> Class (System.Collections.Generic) | Microsoft Learn