구디 아카데미
알고리즘 스터디
중급반 - 3주차

[멘토 유준혁]

스터디 주제

- 문제 접근 방식에 대한 고찰
- 스택과 큐의 차이점 (LIFO, FIFO)
- LinkedList 간단한 구현

알고리즘 문제 접근 방식을 어떻게 해야할까?

대부분의 사람들이 알고리즘이 어렵다고 하는 이유는 아마 다음과 같을 것이다.

[추가적인 이유가 있다면 말해주세요]

1. 도대체가 무슨 말을 하는 건지 모르겠다. 문제 이해 자체가 안된다.
2. 이해는 가나, 어떻게 풀어나가야 할 지 감이 안잡힌다.
3. 분명히 머릿속으로는 그림이 그려지는데 실제로 코딩하기가 어렵다.
4. 풀 수는 있으나 온갖 경우의 수를 다 해보는 바람에 라인이 100줄을 넘긴다.
5. 완벽히 작성했다고 생각했으나 제출하면 항상 틀렸다는 메세지가 뜬다.

문제의 이해

‘이게 대체 무슨 말이야?’

어떻게 보면 가장 어려운 케이스다. 문제를 이해하지 못하면 당연하게도 풀 수 없으니 말이다.

이와 같은 경우에는 문제를 찬찬히 읽어보면서 자신만의 문장으로 재구성 해 볼 필요가 있다.

백준이나 프로그래머스에서는 문제에 자잘한 조건들도 세세하게 넣어놓기 때문에

쓸 데 없이 문제 설명이 길어지는 경우가 종종 있다. (게다가 대부분 상식 내의 조건들이다.)

이런 조건들을 제외시키면서 최대한 간단하고 명료한 문장들로 문제를 재구축하는 연습을 해보자.

음이 아닌 양의 정수 N이 주어진다.

이때 N은 10^{10} - 1의 크기까지 주어지며 조건을 제외한 입력은 주어지지 않는다.

N이 입력으로 주어질 때 1부터 N까지 1의 등차를 가지는 수열의 합을 return하는 메서드를 작성하시오.

이런 거지같은 내용을 한 문장으로 줄이면 **1부터 N까지의 자연수들의 합을 구해라.** 가 된다.

문제에서 저런 세세한 조건까지 들이미는 이유는 테스트케이스를 명확하게 하기 위해서다.

이런 입력들을 줄거니까 이 외의 조건들은 생각하지 않으셔도 됩니다. 라는 말을 하려면

어쩔 수 없이 저런 문장들이 추가되기 때문에 우린 부가적인 조건들을 잠시 제외시킬 필요가 있다.

이렇게 새로이 문제를 정리하게 되면 자연스럽게 문제가 의도하는 바를 이해하게 될 것이다.

정보의 중요성

‘이걸 어떻게 풀어?’

문제는 충분히 이해했다.

입력과 출력 사이의 중간 과정들이 어떻게 진행되는지 파악했다.

그러나 그걸 실제로 풀어낼 방법을 모르겠다. 하는 사람들도 있을 것이다.

어쩌면 모르는 게 당연하기도 하다. 알려주는 사람도, 따로 배운 적도 없으니까.

‘이런 문제 분류에 속하니 이러한 자료구조와 알고리즘을 사용해야겠다.’

라는 일련의 의식 자체가 정립되지 않으면 어려울 수 밖에 없다.

여기에 속하는 사람은 알고리즘 문제에 사용되는 기본적인 정보를 얻고, 활용해보는 시간이 필요하다.

이것이 우리가 스터디를 진행하는 주 목적이다.

(멘토 개인적으로는 꾸준히 나와주었으면 하는 이유이기도 하다.)

기업들이 내는 문제들도 어려운 알고리즘이 사용되지는 않는다.

대부분 기본적인 자료구조를 알고 사용할 줄 안다면 충분히 풀 만한 문제들이다.

그럼에도 직접 사용해본적이 없다면 코드 구현을 제대로 하지 못할 확률이 크다.

이건 여담이지만 스터디에서 공지하는 과제를 반드시 풀어야만 하는 건 아니다.

만약 본인에게 과제가 압박으로 느껴진다면, 풀지 않아도 된다. (그래도 시도는 해주세용)

그렇지만 스터디 시간은 웬만한 일이 아니라면 필참해주었으면 한다.

문제 접근에 대한 새로운 방식을 알고 모르고는 굉장한 차이가 있다. 정말로 어마어마한 차이가 있다.

게다가 초반의 자료구조들이 후에 나오는 알고리즘들에서 계속해서 쓰이기 때문에

초반에 빠지게 되면 점점 더 스터디 나오기가 벅차게 될 수도 있다.

활용과 응용

‘어떻게 쓰는거지?’

스택이나 큐 같이 정보처리 자격증에서도 충분히 배우는 자료구조들을 실제로 사용해볼 기회는 적다.

이 단계에 속했다면 해당 자료구조를 직접 사용해본 적이 없거나 적을 것이다.

그리고 아마도 해당 자료구조를 직접 다루어 보는 건 이번이 처음일 것이다.

해답은 예상했다시피 다양한 문제를 접하는 데 있다.

시작은 연습에 가까운 문제로 시작하여 난이도를 높여가면서 데이터의 흐름을 익혀가면 된다.

처음엔 일일이 출력을 해가면서 현재 데이터가 어떻게 이동하고 축적되었는지 확인하던 게

점점 머릿속으로 그리기만 해도 충분해질 것이다.

더 나아가 같은 문제를 다른 자료구조로 푸는 방법들도 생각이 나게 될 것이다.

알고리즘 문제들은 중/고등학교 때 치르던 시험과는 다르게 반복은 그다지 중요하지 않다.

비슷한 문제들을 하나 더 풀 바에는 같은 문제를 다르게 푼 사람의 풀이를 보는 게 더 좋다.

규칙 찾기

‘이렇게 하는 거 맞나?’

가끔 문제를 풀다 보면 풀이가 쓸 데 없이 길어지는 경우가 있다.

보통 **처음에 설계를 제대로 잡고 코딩을 시작하지 않았거나 규칙을 파악하지 못한 경우로,**

이를 해결하기 위해서 온갖 경우의 수를 catch 하다보니 라인이 점점 길어지게 되는 것이다.

(여기서 라인이 길다의 기준은 100~150 라인 정도로 생각하면 됩니다. 어려울 경우 ++)

해결하기 위해선 문제를 분석하는 시간을 가질 필요가 있다.

바로 코딩에 들어가서 조건들을 하나하나 맞춰가며 작성하는 것도 하나의 방법이겠지만,

문제에서 유추 가능한 규칙이 있는 지 찾아보는 것도 좋은 방법 중 하나다.

공책을 펴 가능한 경우의 수 몇가지를 해보다보면 대부분 규칙이 발견 될 텐데,

그를 토대로 코딩을 하면 훨씬 수월하게 풀리게 된다.

규칙을 발견했다면 다음과 같이 코딩을 시작해보자.

1. 문제가 원하는 중간 과정의 밑바탕을 그리고
2. 자료구조와 알고리즘을 선택한 뒤
3. 특이한 경우의 입력이 주어지는지 체크하고
4. 설계대로 구현하며 오차를 잡아간다.

여기서 제일 중요한 건 TODO의 작성이다.

앞의 1부터 3단계의 모든 내용들을 TODO로 작성해두면 코딩테스트의

주요 출제 문제 중 하나인 “알고리즘 구현 문제”와 다르지 않게 풀 수 있다.

기초 되짚기

'이거 맞는데..?'

분명 문제에서 요구하는 알고리즘 대로 작성했다고 하더라도,

기본적인 수칙들을 깜빡했다면 오답처리가 떠버린다.

기본적인 수칙이라면 위에 소개한 4가지와 추가적으로 자료형의 범위, 또 출력 스트림 설정이 있다.

길이 막혔다면 잠시 멈춰서 위의 4가지 중 소홀히 한 게 있는지 먼저 살펴보자.

그래도 문제가 없다면 자료형을 제대로 파악했는지, 또 마지막으로 입출력 스트림을 확인해보자.

지난 시간 두 번에 걸쳐 Scanner와 BufferedReader 차이점을 소개한적이 있다.

온라인 코딩 테스트에서는 주로 매개변수가 주어지는 메서드를 완성하는 문제들이 대부분이지만,

백준 온라인 저지는 메인에서 결과값을 출력해야 하기 때문에 입출력 스트림도 은근 문제가 된다.

백준 2447번 : 별 찍기 - 10번 문제

위의 문제는 출력의 양이 기하급수적으로 커지기 때문에 자칫하면 시간초과에 걸리는 문제이다.

우리가 아무리 잘 풀었다고 생각해도 결국엔 그게 맞는지 눈으로 확인을 해야 마음이 편하다.

입출력 스트림까지 확인을 했는데도 통과가 안된다면, 앞의 4가지를 다시 체크해보길 바란다.

**** 실제 기업코딩테스트에선 입출력 스트림을 신경 쓸 필요가 전혀 없습니다.**

오로지 연습과정에서 불필요한 과정을 겪지 않길 바라는 마음으로 써내려간 글이니

단순히 참고사항으로만 생각하셔도 무방합니다.

실제로 대부분의 문제는 Scanner로도 충분히 풀립니다.

정리해보기

알고리즘 문제를 풀기위해 거치는 몇가지 단계를 소개해봤다.

이를 학원과정의 개인프로젝트 / 팀프로젝트에 비유하자면 다음과 같다.

- 문제의 조건들을 추려서 나만의 문장으로 만들 수 있을까?
  - **요구사항 분석**
    고객 (기업) 이 원하는 바를 충분히 캐치하여 문서화 시킬 수 있는가?

- 기본 프로세스 흐름 설계와 사용할 알고리즘 / 자료구조 분류
  - **화면설계**
    전체 프로세스의 흐름을 파악할 줄 아는지, 기본적인 정보들을 알고 있는지

- 점화식의 도출과 반복되는 함수들의 메서드화
  - **구조설계**
    반복적인 작업을 보다 단순화 시킬 수 있는 역량이 있는지

- 주어지는 알고리즘 그대로 작성
  - **화면구현**
    정리된 문서를 토대로 코드를 작성해 원문이 원하는 바를 만들 수 있는지

주절주절 잡소리가 길었지만 결국 기업이 테스트를 통해 얻는 정보는 대략 위와 같다.

따라서 난이도가 높아 참가자들이 풀 지 못하는 문제들 보단

충분히 풀 수 있는 난이도의 문제를 내며

원하는 정보를 캐기 위한 밑작업에 불과하다고 생각한다.

알고리즘에 재능이 없다고 한탄할 필요는 없다. 어차피 실무에서 쓰이지 않을 확률이 높으며,

개인적으로는 노력으로 충분히 따라잡을 수 있다고 생각하기 때문이다.

거듭 강조하지만 기본적인 정보들을 숙지하고 있는지가 훨씬 중요하다.

스택 (Stack / LIFO)

드디어 뻔한 소리가 끝나고 다시 자료구조로 들어왔다.

이번 주는 스택과 큐의 자료구조에 대한 이론과 구현, 응용과 문제 풀이를 해보려고 한다.

정보처리 자격증 공부를 통해 알고 있겠지만 스택과 큐의 가장 큰 차이는 자료의 입출력 순서에 있다.

스택은 Last In First Out (LIFO) 의 특징을 가지며

책을 바닥에서부터 위로 하나씩 겹쳐올리는 모양새라고 생각하면 된다.

JAVA에서는 스택 클래스가 기본적으로 구현되어 있지만 내부 소스를 살펴보면 아래와 같다.

```
48 public
49 class Stack<E> extends Vector<E> {
50     /**
51      * Creates an empty Stack.
52      */
```

Vector를 상속한 Stack

```
90 public class Vector<E>
91     extends AbstractList<E>
92     implements List<E>, RandomAccess, Cloneable, java.io.Serializable
93 {
94     /**
95      * The array buffer into which the components of the vector are
96      * stored. The capacity of the vector is the length of this array buffer,
97      * and is at least large enough to contain all the vector's elements.
98      *
99      * <p>Any array elements following the last element in the Vector are null.
100     *
101     * @serial
102     */
103    protected Object[] elementData;
```

배열로 구현되어 있는 Vector 클래스

크기가 정해져있다는 가정 하에 따르면 배열로 구현하여도 전혀 문제가 없다.

(혹시 모르겠다면 스터디 시간에 같이 구현해보았던 소스를 다시 살펴보길 바란다.)

주로 사용되는 메서드는 Push (자료 추가), Peek (top 값 가져오기), Pop (top 값 가져오며 삭제)

비어있는 상태에서 Pop을 할 시 UnderFlow, size가 꽉 찼을 때 Push 한다면 OverFlow가 발생한다.

개발자 포럼 중 하나인 StackOverFlow (스택오버플로우) 가 여기서 유래했다.

큐 (Queue / FIFO)

다음으론 스택과 같이 선형 자료구조에서 항상 언급되는 자료구조인 큐에 대해 알아보자.

큐는 먼저 들어온 자료가 먼저 나가는 First In First Out (FIFO) 의 특징을 가지고 있다.

JAVA에선 특이하게도 클래스가 아닌 Interface 형태로 구현되어있다.

```
Queue<Integer> q = new LinkedList<>();
```

실제 객체화 모습

```
138 public interface Queue<E> extends Collection<E> {
```

큐의 구현

이는 큐가 가지는 FIFO 특징 때문에 그러하다.

선입선출의 특징을 가지는 큐를 배열로 구현을 했다고 가정해보자.

최대 사이즈가 6인 큐에 1, 2, 3을 차례대로 Add 해보자.

1	2	3	-1	-1	-1

↑최대 사이즈가 6인 Queue

이 상태에서 맨 처음 자료를 꺼내보자.

-1	2	3	-1	-1	-1

가장 처음의 자료는 꺼내어지고, 대신 공백을 의미하는 -1로 채워졌다.

이 상태에서 {4, 5, 6} 3개의 자료를 추가적으로 넣어보자.

- 1	2	3	4	5	6

여기서 한 개의 자료를 더 넣고 싶다고 하더라도 더 이상 들어갈 자리가 없다. (FIFO 특성)

따라서 맥시멈 사이즈를 유지하면서 자료를 더 넣고 싶다면 자리를 한 칸씩 당겨줄 필요가 있다.

바로 그 점이 문제가 된다.

배열의 자리를 한 칸씩 당긴다는 건, O(N)의 시간복잡도를 가지는 함수를 실행한다는 말과 동일하다.

데이터의 입출력이 잦다면, 이는 엄청난 시간을 잡아먹게 될 것이다.

따라서 스택과 같이 배열로 구현하는 게 아니라

동적인 메모리 할당이 가능하며 Iterable한 Collection 인터페이스를 구현하게 되는 것이다.

(주로 List 인터페이스를 구현한 LinkedList가 사용된다.)

주로 사용하는 메서드는 Add, Peek, Poll 이 있다. (Stack과 메서드 순서 동일)

참고로 온라인 게임에서 주로 사용되는 매칭 시스템에 사용되는 자료구조 또한 큐 이다.

“큐 잡혔다.” 라고 하는 말이 여기서 유래됐다.

Linked List의 간단한 구현

큐에 대한 이론을 알아보면서 Linked List로 객체화를 시킨다는 건 알게되었다.

그럼 Linked List는 무엇일까?

Linked List의 사전적인 정의는 다음과 같다.

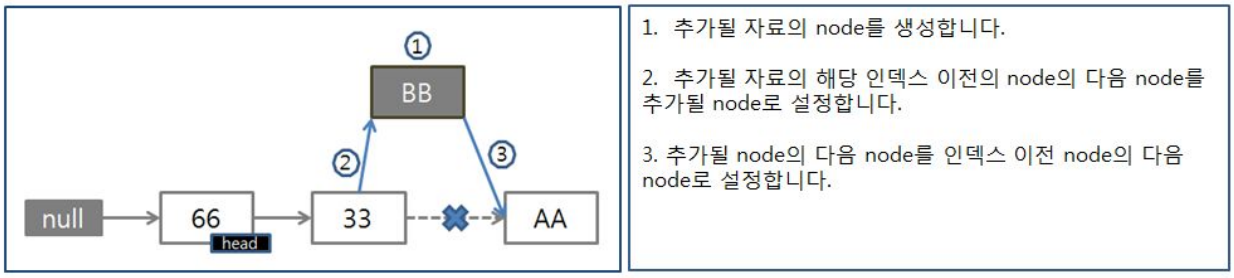
각 노드가 데이터와 포인터를 가지고 한 줄로 연결되어 있는 방식으로 데이터를 저장하는 자료 구조

비유하자면 사람들이 **(노드)** 서로 손을 잡고 **(포인터)** 서있는 모습을 생각하면 된다.

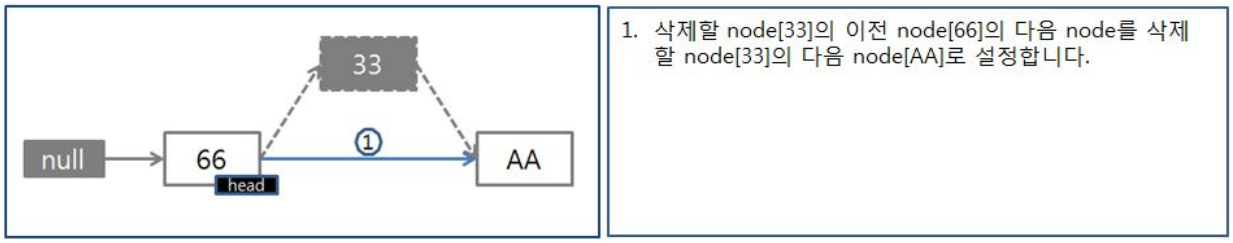
만약 여기서 화장실 다녀온 사람이 중간에 끼게 됐다면 끼는 자리의 양 옆 사람과 손을 잡으면 된다.

이는 큐가 가지던 데이터 입출력의 문제점을 해결해주는 방식이기에

큐가 LinkedList로 객체화를 시키는 이유가 되기도 한다.



Linked List의 데이터 삽입 과정



Linked List의 데이터 삭제 과정

큐에서 데이터를 Poll 할 때는 해당 노드의 이전 노드가 다음 노드를 가리키도록 하고,

원래 노드를 빼주기만 하면 된다. 이로써 데이터의 불필요한 이동이 줄어들었다.

이러한 이점을 가지고 있지만 단점 또한 존재한다.

Index로 바로 접근이 안되기 때문에 원하는 Index까지 접근하기 위해선 O(N)번의 연산을 거친다.

또한 포인터의 사용으로 인해 저장 공간의 낭비가 있으며, 좀 더 복잡한 알고리즘을 사용한다.

추가적으로 Linked List에는 3가지 종류가 존재하는데

1. Head가 Tail과 연결되어 있는 형태를 띄는 원형 링크드 리스트 (Circular Linked List)
2. 노드에 링크가 2개 존재하는 양방향 리스트 (Double Linked List)
3. 마지막으로 제일 구현이 간단하고 자주 쓰이는 단방향 리스트 (Single Linked List)

이렇게 3가지의 종류가 있다.

스터디에서 구현해본 종류는 마지막 3번째 단방향 리스트 이며 제일 간단한 형태를 띄고있다.

스택과 큐는 실제 코딩 테스트에서도 자주 출제되는 유형의 자료구조이니

기본적인 개념을 토대로 다양한 문제를 접해보기를 바라며

마지막으로 단방향 리스트 구현 소스를 공유하며 마치도록 하겠다.

```java
package workspace;

/*
1. 단일 링크드 리스트(Single Linked List)
노드에 링크가 1개
단방향으로 진행

2. 이중(양방향) 링크드 리스트(Double Linked List)
노드에 링크가 2개
양방향으로 진행

3. 환형(원형) 링크드 리스트(Circular Linked List)
마지막 노드가 첫번째 노드를 가리킴
원형으로 진행 (회전)

 */
public class LinkedList{
	public static void main(String [] args) {
		SingleLinkedList<Integer> sl = new SingleLinkedList<>();
		sl.add(new Node<Integer>(1));
		System.out.println("sl.add(1) ->" + sl);
		sl.remove();
		System.out.println("sl.remove() ->" + sl);
		sl.add(new Node<Integer>(1));
		sl.add(new Node<Integer>(2));
		sl.add(new Node<Integer>(3));
		System.out.println(sl.get(2));
		System.out.println("head:" + sl.getHead() + ", tail:" +
sl.getTail());
		System.out.println(sl.size());
		System.out.println(sl);
		sl.remove();
		System.out.println("=============remove==============");
		System.out.println("head:" + sl.getHead() + ", tail:" +
sl.getTail());
		System.out.println(sl.size());
		System.out.println(sl);
		sl.add(new Node<Integer>(4));
		System.out.println("head:" + sl.getHead() + ", tail:" +
sl.getTail());
		System.out.println(sl.size());
		sl.remove(0);
```

```
            System.out.println("head:" + sl.getHead() + ", tail:" +
sl.getTail());
            System.out.println(sl.size());
            System.out.println(sl);

System.out.println("=============setHead,setTail==============");
            sl.setHead(new Node<Integer>(1));
            System.out.println("setHead(1) ->" + sl);
            sl.setTail(new Node<Integer>(4));
            System.out.println("setTail(4) ->" + sl);
      }
}

class Node<T> {
      private Node<T> next;
      private T data;

      // constructor
      public Node(T t) {
            this.data = t;
      }

      // getter,setter,toString
      public Node<T> getNext() {
            return next;
      }

      public void setNext(Node<T> next) {
            this.next = next;
      }

      public T getData() {
            return data;
      }

      public void setData(T data) {
            this.data = data;
      }

      @Override
      public String toString() {
            return data.toString();
```

```
	}
}

class SingleLinkedList<T> {
	private int size = 0;
	private Node<T> head;
	private Node<T> tail;

	public SingleLinkedList() {
	}

	public void add(Node<T> node) {
		if (head == null) {
			head = node;
			tail = node;
		} else {
			this.tail.setNext(node);
			this.tail = node;
		}
		this.size++;
	}

	public void setHead(Node<T> node) {
		node.setNext(this.head);
		this.head = node;
	}

	public void setTail(Node<T> node) {
		this.tail.setNext(node);
		this.tail = node;
	}

	public Node<T> getHead() {
		return this.head;
	}

	public Node<T> getTail() {
		return this.tail;
	}

	public Node<T> get(int index) {
		if (index >= this.size || index < 0)
```

```
                throw new IndexOutOfBoundsException(index);
        Node<T> node = this.head;
        for (int i = 0; i < index; i++)
                node = node.getNext();
        return node;
    }

    public void remove() {
        if (this.size == 1) {
                this.head = null;
                this.tail = null;
                this.size--;
                return;
        }
        Node<T> node = this.head;
        for (int i = 0; i < size - 2; i++)
                node = node.getNext();
        node.setNext(null);
        this.tail = node;
        this.size--;
    }

    public void remove(int index) {
        if (index >= this.size || index < 0)
                throw new IndexOutOfBoundsException(index);
        if (index == 0) {
                try {
                        this.head = this.head.getNext();
                        this.size--;
                } catch (IndexOutOfBoundsException e) {
                        this.head = null;
                        this.tail = null;
                }
        } else if (index == this.size - 1)
                remove();
        else {
                get(index - 1).setNext(get(index + 1));
                this.size--;
        }
    }

    public int size() {
```

```
            return this.size;
      }

      @Override
      public String toString() {
            StringBuffer sb = new StringBuffer();
            Node<T> node = this.head;
            sb.append("[");
            while (node != null) {
                  sb.append(node.toString());
                  node = node.getNext();
                  if (node != null)
                        sb.append(", ");
            }
            sb.append("]");
            return new String(sb);
      }
}
```