스터디 2주차 (엄태호)

배열과 리스트, 연결리스트

- 배열 (Array)

- 한개의 값만 가지는 변수와 달리 배열은 여러개의 값을 가지고 있는 데이터 타입이다.
- 배열은 인덱스를 사용해 바로 특정 요소에 바로 접근할 수 있으며, `.length` 속성을 활용해 배열의 길이를 확인할 수 있다.
  또한, 이 외의 다양한 빌트인 메소드(`sort`, `slice`, `filter`, `reduce` 등)들을 지원하고 있다.
- 배열은 정적 배열과 동적 배열로 나누어지며 **JS의 배열은 동적 배열에 해당**한다.

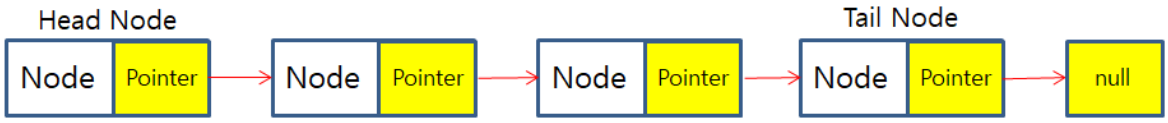
동적 배열과 정적 배열의 차이

	정적 배열 (Static Array)	**동적 배열** (Dynamic Array)
배열의 크기	고정적	유동적
저장 메모리	스택 메모리 (컴파일 과정에 할당)	힙 메모리 (런타임에 할당)
빈 요소 허용 여부	X	O
장점	- 인덱스를 통해 요소에 O(1)의 시간 복잡도로 빠르게 접근이 가능하다. - 필요한만큼의 메모리만 사용한다.	- 배열의 크기가 유동적이므로 크기를 고려해야하는 불편함을 줄여준다. - 인덱스 접근을 통해 요소에 빠르게 접근할 수 있다.
단점	- 배열 내부에서 다루고자 하는 데이터의 상한을 고려해 크기를 고려해야하는 불편함이 있다. - 스택 메모리의 특성상 크기에 제한이 있기 때문에 배열이 너무 큰 경우 스택 오버플로우가 발생할 수 있다. - 고정된 크기로 인해 배열 크기 변경이 제한된다.	- 배열의 크기를 변경한다는것은 메모리상에서 새로운 위치로 옮기는것이기 때문에 이로 인한 비용이 발생한다. - 배열의 크기가 유동적이기 때문에 일반적으로 현재 사용하는 크기의 1.5배 가량의 메모리를 할당하여 불필요한 메모리를 차지한다. - 포인터가 필요해 정적 배열에 비해 메모리가 조금 더 사용된다.

- 리스트 (List)

- 리스트 자료구조는 정적 배열을 사용하는 일부 프로그래밍 언어 (Java, C++ 등)에서 **정적 배열의 한계를 극복하기 위해 주로 사용**된다.
- 프로그래밍 언어에 따라 특징이 조금씩 다를 수 있지만 동적 배열처럼 **배열의 크기가 유동적**이며 인덱스를 통해 요소에 접근할 수 있는 **랜덤액세스 기능을 지원**하고 요소의 **데이터 타입에 대한 제한이 없다.**
- 리스트 자료구조는 Java, C++, C#, Python 등의 언어에서 사용된다.

- 연결리스트 (Linked List)



- 고정 길이를 가진 정적 배열의 단점을 해소하기 위해 등장한 자료구조로 크기가 유동적으로 변하는 특징을 가지고 있다.
- 여러개의 노드가 포인터에 의해 연결되어 있는 구조로 **노드 내부는 데이터와 다음 노드를 가르키는 포인터(= 링크)로 이루어져있다**.
- 가장 앞에 위치한 노드를 Head Node, 마지막에 위치한 노드를 Tail Node라고하며 Tail Node의 포인터는 null을 가르키고 있기 때문에 null을 만나게 되면 순회가 멈추게 된다.

장점

- 데이터 삽입 및 삭제시 해당 노드의 포인터만 연결하거나 끊어주면 되기 때문에 작업이 가볍다.
- 길이가 고정되어 있지 않아 유연하다.

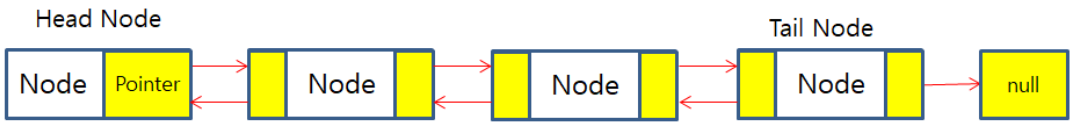
단점

- 모든 노드는 포인터를 가지고 있기 때문에 정적 배열에 비해 메모리 사용량이 많다.

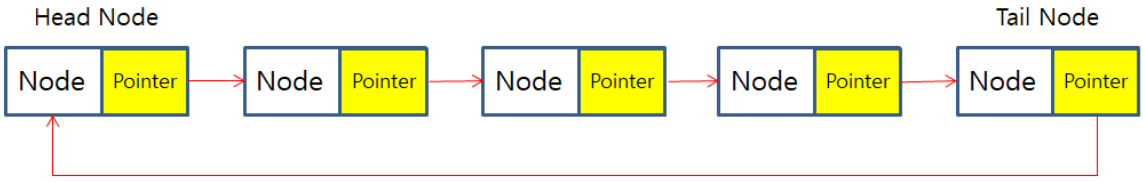
- 랜덤액세스가 불가하므로 노드를 탐색할 때 `O(n)`의 시간복잡도를 가진다.
- 배열의 요소들은 메모리 상 위치가 연속적이지만 링크드리스트는 포인터로 연결되는 형태로 비연속적으로 메모리에 저장되기 때문에 캐싱에 불리하다.

연결리스트의 종류

1. **단일 연결 리스트 (Singly Linked List)**
   일반적으로 연결리스트는 위에서 다뤘던 단일 연결 리스트를 말하며 단방향으로만 요소들을 순회할 수 있는 연결 리스트를 말한다.
2. **이중 연결 리스트 (Doubly Linked List)**
   노드의 전과 후, 양 방향으로 이동이 자유로운 장점이 있지만 각 노드에 포인터가 한개 더 추가되어야해서 메모리를 더 많이 차지한다는 단점이 있다.



3. **원형 연결 리스트 (Circular Linked List)**
   Tail Node가 `null`이 아닌 Head Node를 가르키고 있어 순환 가능한 형태의 연결 리스트를 말한다.



시간 복잡도 (Time Complexity)

- 알고리즘의 성능의 효율성을 판단하기 위해 단순화한 지표이다.

- 코드가 실행될 때의 성능은 단순히 어떤 알고리즘을 적용했는지뿐만 아니라
  다양한 요소의 영향을 받기 때문에 절대적인 수치로 측정하는것은 현실적으로 어렵다.
- 따라서, 불필요한 요소들은 제외하고 **인풋의 크기에 따라 복잡도가 얼만큼 상승하는지에만 초점**을 맞춘 점근적 분석법을 사용해 측정한다.

> 💡 **불필요한 요소란?**
>
> - 코드내의 인풋과 관계없는 상수 (계수)
> - 코드의 라인 수
> - 언어의 특성과 컴파일러의 성능
> - 실행 환경의 하드웨어 성능

- 점근적 복잡도를 표현하는 3가지 (빅 오메가, 빅 오, 빅 세타)

- **빅 오메가 (Big-Ω, Best Case)**
  빅 오메가는 최선의 경우를 말하며 예를 들어, 배열에서 찾으려는 요소가 가장 앞에 위치했을 때의 시간 복잡도를 말한다.
- **빅 오 (Big-O, Worst Case)**
  빅 오는 최악의 경우를 말하며 배열에서 찾으려는 요소가 마지막 인덱스에 위치할 때를 말한다.
- **빅 세타 (Big-θ , Average Case)**
  빅 세타는 빅 오메가와 빅 오의 중간 값인 평균 값을 말한다.

> 💡 **일반적으로 시간복잡도를 말할 때 빅 오를 사용하는 이유**는 알고리즘의 연산수는 정확하지 않기 때문에 **예측 가능한 범위 내에서 제 시간안에 끝나는것을 보장할 수 있는 최악의 경우를 고려하기 위해서**이다.

- 자주 사용되는 시간 복잡도 단위

복잡도 / 인풋 사이즈	1	10	100
O(1)	1	1	1
O(log n)	0	2	5
O(n)	1	10	100
O(n2)	1	100	10000
O(2n)	1	1024	1.2676506002282294e+30
O(n!)	1	3628800	❌ Maximum Callstack!