기술면접 – LinkedList (연결리스트), Array vs LinkedList

연결 리스트(Linked List)

- 연결 리스트는 동적 자료구조라고 불립니다. 그러므로 크기를 정할 필요가 없습니다. 또한 배열처럼 **연속된 메모리 주소를 할당 받지 않습니다.**
- 대신 **노드(Node)라는게 존재**하며, 그 노드 안에 데이터가 있고, 다음 데이터를 가리키는 주소를 가지고 있습니다. 그런 식으로 연결되어 이어진 형태인 것입니다.
- 연결 리스트는 위에서 말한 것처럼 크기를 정해 놓지 않았기 때문에 크기의 제한이 없습니다. 그로 인해 **데이터 추가, 삭제가 자유롭다는 장점이 있습니다.**
- 반면에 배열처럼 연속적인 메모리 주소를 할당 받지 않았기 때문에 **임의로 접근하는 것이 불가능**합니다. 그 말은 즉 슨 **데이터를 탐색할 때 순차적으로 접근해야 한다는 것입니다. 단점**이라고 한다면 단점일 수 있는 거죠.

배열(Array)

- 배열은 정적 자료구조라고 불립니다. 그래서 배열을 만들기 위해서는 미리 크기를 정해 놓게 되는데요. 그렇게 되면 해당 크기만큼의 **연속된 메모리 주소를 할당** 받게 됩니다.
- 연속된 메모리 주소를 할당 받고 있기 때문에 데이터가 **인덱스(index)라는 것을 갖게 됩니다.**
- index를 갖게 된다는 것은 즉 **임의 접근이 가능하다는 장점이 있는 거죠. 그러므로 접근과 탐색에 용이합니다.**
- 하지만 **크기**를 미리 정해놓았기 때문에 **수정하는 것이 불가능합니다. 또한 이미 크기를 정해 놓은 터라 해당 배열 크기 이상의 데이터를 저장할 수 없다는 단점**이 있습니다.

Array와 LinkedList 차이점

< 특징 차이점 >

- **Array는 연속된 메모리 공간을 사용하고, Linked List는 연속되지 않은 메모리 공간**을 사용합니다.
- 그래서 **Array의 경우, index**를 통해 각 요소에 접근할 수 있고, **탐색에 필요한 시간복잡도는 O(1)**입니다. 하지만 **삽입이나 삭제**의 경우에는 shift에 필요한 연산이 발생하여 **O(n)의 시간 복잡도**를 가집니다.

- 반면 **Linked List의 경우**, shift가 필요 없고, **노드** 간의 연결을 이어주고 끊어주는 과정을 통해 **삽입과 삭제를 O(1)** 시간복잡도로 수행합니다. **탐색은 O(n)** 시간복잡도로 수행합니다.
- **정리하면, Array는 추가나 삭제보다 빠른 접근의 이점을 필요로 할 때 사용하고, Linked List는 탐색보다는 추가나 삭제에서의 이점을 필요로 할 때 사용한다.**

< 저장 순서 차이점 >

- Array의 경우 논리적 저장 순서와 물리적 저장 순서가 일치한다. 이 덕분에 index를 통해 해당 원소로 접근할 수 있다.

- 반면, Linked List는 논리적 저장 순서와 물리적 저장 순서가 일치하지 않는다. 이 때문에, 탐색을 위해서는 첫 번째 원소부터 일일이 전부 노드를 타며 확인을 해야 하는 것이다

< 표 정리 >

배열(Array)	연결 리스트(Linked List)
정적 자료구조	동적 자료구조
미리 크기를 정해 놓음	크기를 정할 필요가 없음
연속된 메모리 주소를 할당 받음	연속된 메모리 주소를 할당 받지 않음
접근, 탐색 용이	추가, 삭제 용이
index 존재	Node 존재

< 시간 복잡도 >

연결리스트와 배열이 가지는 시간복잡도를 크게 데이터 탐색, 데이터 추가, 데이터 삭제 3가지로 나누어서 설명

1) 데이터 탐색

- **연결리스트:** 순차 접근 방식을 사용하기 때문에 어떤 한 데이터를 찾기 위해서는 처음부터 **순차적으로 탐색**해야 한다. 따라서 이 때 **연결리스트는 O(n)의 시간복잡도**를 가진다.

- **배열:** 임의 접근 방식을 사용하기 때문에 데이터를 탐색할 때 처음부터 찾을 필요가 없다. 인덱스 번호가 있기 때문에 **인덱스를 통해서 매우 빠르게 탐색**할 수 있다. 이 때 **배열은 O(1)의 시간복잡도**를 가진다.

2) 데이터 추가

- **연결리스트**
- 1. **추가하려는 데이터의 위치가 맨 앞**이라면, 노드가 가지고 있는 메모리 주소 값만 갈아끼워주면 되기 때문에 **O(1)의 시간복잡도**를 가진다.
- 2. **추가하려는 데이터의 위치가 맨 앞 그 이후**라면, 탐색하면서 해당 위치까지 가야하기 때문에 **O(n)의 시간복잡도**를 가진다.

- **배열**
- 1. **추가하려는 데이터가 맨 뒤가 아니라면**, 추가되는 데이터 위치 이후에 있는 모든 데이터들을 한 칸씩 미뤄야 하므로 **O(n)의 시간복잡도**를 가진다.
- 2. **추가하려는 데이터의 위치가 맨 뒤이고 배열에 공간이 남아있다면**, **O(1)의 시간복잡도**를 가진다.

3) 데이터 삭제

- **연결리스트**
- 1. **삭제하려는 데이터의 위치가 맨 앞**이라면, **O(1)의 시간복잡도**를 가진다.
- 2. **삭제하려는 데이터의 위치가 맨 앞 그 이후**라면, **O(n)의 시간복잡도**를 가진다.

- **배열**
- 1. **삭제하려는 데이터의 위치가 맨 뒤가 아니라면**, **O(n)의 시간복잡도**를 가진다.
- 2. **삭제하려는 데이터의 위치가 맨 뒤이고 배열에 공간이 남아있다면**, **O(1)의 시간복잡도**를 가진다.

Array와 LinkedList 활용

- **Array**과 같은 경우는 일반적으로 **불규칙적인 정보를 저장하거나, 이후에 자주 접근할 정보를 저장할 때 사용**할 수 있다. 예를 들어 메뉴판 데이터를 저장한다고 하면, Linked List보다 Array가 더 효율적일 것이다. 랜덤으로 메뉴를 뽑을 수도 있고, 각 메뉴가 추가나 삭제되는 경우보다 접근되는 경우가 훨씬 많이 발생 할 테니, **추가나 삭제 연산보다 탐색이 효과적인 Array**를 쓰는 것이 좋다.

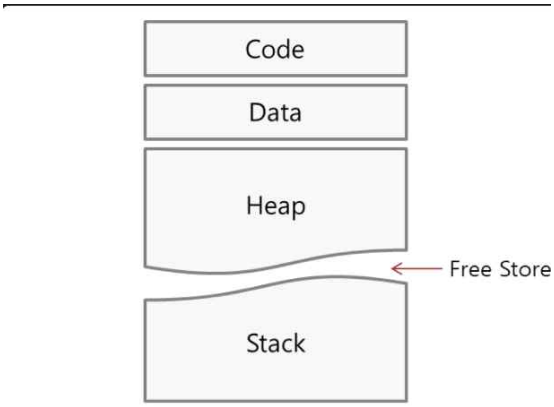
- Linked List와 같은 경우는 **history 데이터를 저장할 때 유용하게 사용**할 수 있다. 문서 편집기를 사용할 때, 사용자 기록을 doubly linked list를 활용하여 저장하면, 사용자가 이전 기록으로 돌아가거나 다시 이후 기록으로 이동할 때 유리하게 사용할 수 있을 것이다. 이 경우는 history 데이터 중간에 **추가나 삭제 연산**이 발생하더라도, Array보다 **빠른 시간복잡도로 연산을 수행**할 수 있다. 음악 플레이어 앱에 이전 곡과 다음 곡이 연결되어 있듯 이런 기능이 연결 리스트로 많이 활용된다고 합니다.

Array와 ArrayList 차이점

- **Array는 고정 길이 자료 구조인 반면 ArrayList는 가변 길이 자료 구조이다.**
- Array는 정해진 길이의 배열이 모두 사용되면, 새로운 데이터를 추가하고 싶을 때는 새로운 배열을 만들어주어야 한다. 반면, **ArrayList는 내부적으로는 배열로 구성되어있으나, capacity를 넘어가면 일반적으로 배열의 크기를 2배 증가시키면서 배열의 크기를 늘린다**.

< 동적 할당, 정적 할당 >

- 정적 할당은 컴파일 단계에서 필요한 메모리 공간을 할당하고, 동적 할당은 실행 단계에서 메모리 공간을 할당해주는 것이다.
- Array가 선언되면, 컴파일 과정에서 메모리가 Stack 영역에 할당된다 (정적 메모리 할당).
- 반면, ArrayList, LinkedList가 선언되면, 런타임 환경에서 메모리가 Heap 영역에 할당된다 (동적 메모리 할당).


Code
Data
Heap
Free Store
Stack