

Linked List VS Array List

Data structure 의 미션

-> 메모리의 효율적 사용

메모리의 특징

- 메모리의 각 위치는 주소를 갖는다.
- 메모리의 각 위치에 데이터가 저장된다.

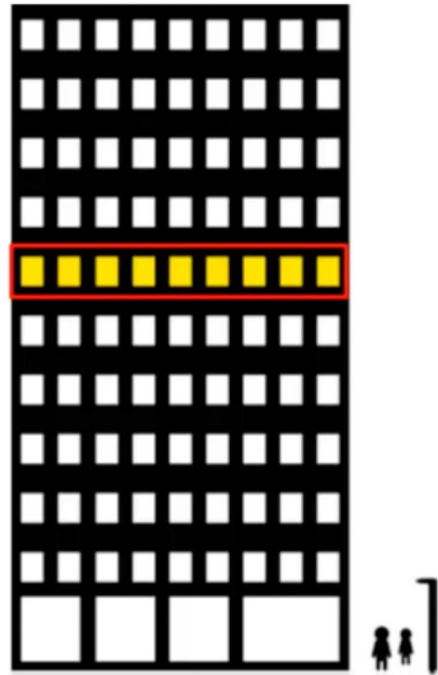
- Random Access Memory

각 위치(주소) 접속하는 시간이 동일하다

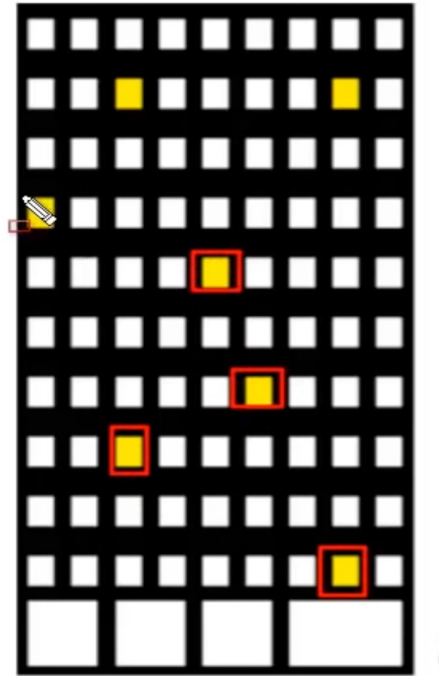
각각의 위치에 접속하는 시간이 동일 하기에 주소를 미리 알고 있다면, 바로 검색할 수 있다.

검색 시간이 단축된다

Array List와 Linked List의 메모리 할당 방식 비교



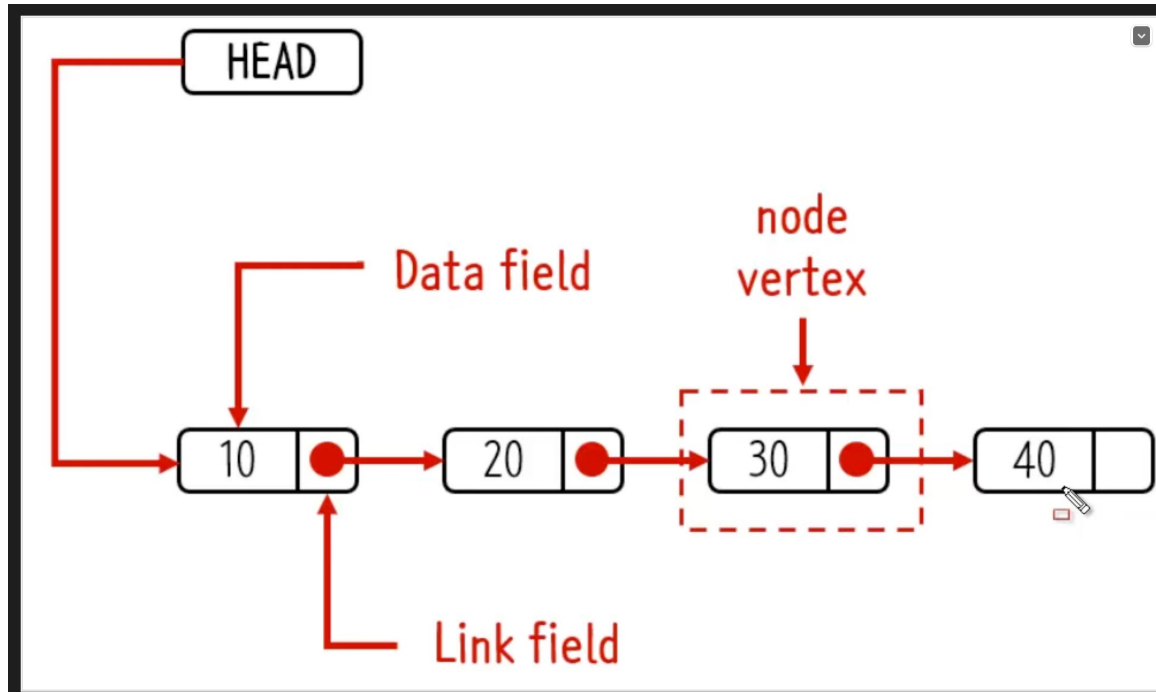
Array List



Linked List

- Array List는 물리적인 위치가 붙어있다.
: Array가 가진 크기를 넘어서면 오류가 난다.
100만 개의 공간을 정의 했는데 한 두개의 공간만 사용하면 낭비다!
- Linked List는 물리적으로 위치가 떨어져있다.
: 램 메모리가 허용하는 한 확정적인 크기를 갖지 않는다.

Linked List 구조



- 하나의 Vertex에는 두 개의 field(변수)가 존재한다.

(1) Data field : 저장되는 실제 값

(2) Link field : 다음 노드에 대한 정보

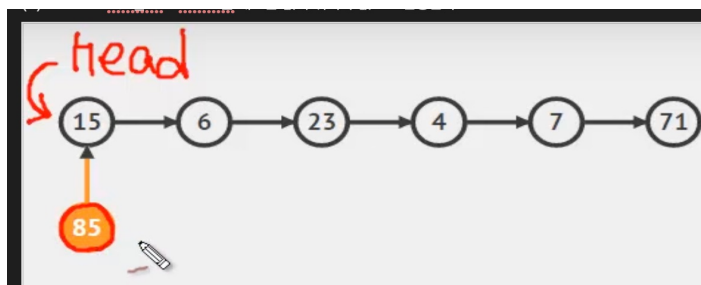
- Head는 Linked List의 시작점을 나타낸다

Linked List의 맨 앞에 데이터를 추가하는 시나리오



1. 주어진 Linked List 형태

- Head의 값으로 15가 저장됨

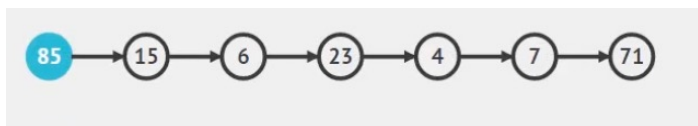


2. 맨 앞에 85를 추가하고 싶음

- Vertex에 Temp 변수 생성하고 85를 저장
- Temp의 다음 Vertex로 head로 지정
- Head를 Temp로 변환

```
Vertex temp = new Vertex(input)
temp.next = head
head = temp
```

85



3. 끝

- 기존 Linked List 원소가 가지는 정보는 변하지 않았음.
- Head의 정보만 새로운 값으로 변경

Linked List의 중간에 데이터를 삽입하는 시나리오



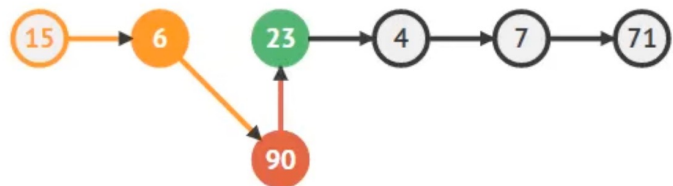
1. 주어진 Linked List 형태

- Head의 값으로 15가 저장됨



2. 2번 노드에 90을 삽입하고 싶음.

- Head로부터 1번 노드까지 이동

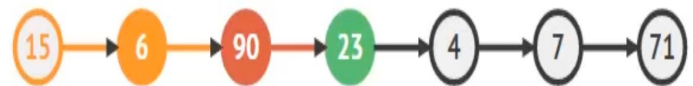


3. 데이터 교환

- 1번 노드 뒤에 있는 정보를 temp2에 저장
- 새 값(90)을 1번 노드 뒤에 저장
- 새 값 뒤에 temp2를 저장

```
Vertex temp1 = head
while (--k!=0)
    temp1 = temp1.next
```

```
Vertex temp2 = temp1.next
Vertex newVertex = new Vertex(input)
temp1.next = newVertex
newVertex.next = temp2
```



4. 끝

Linked List의 중간에 데이터가 삭제되는 시나리오



1. 주어진 Linked List 형태

- Head의 값으로 15가 저장됨



2. 2번 노드에 90을 삭제하고 싶음.

- Head로부터 1번 노드까지 이동

```
Vertex cur = head  
while (--k!=0)
```

```
cur = cur.next
```



3. 데이터 교환

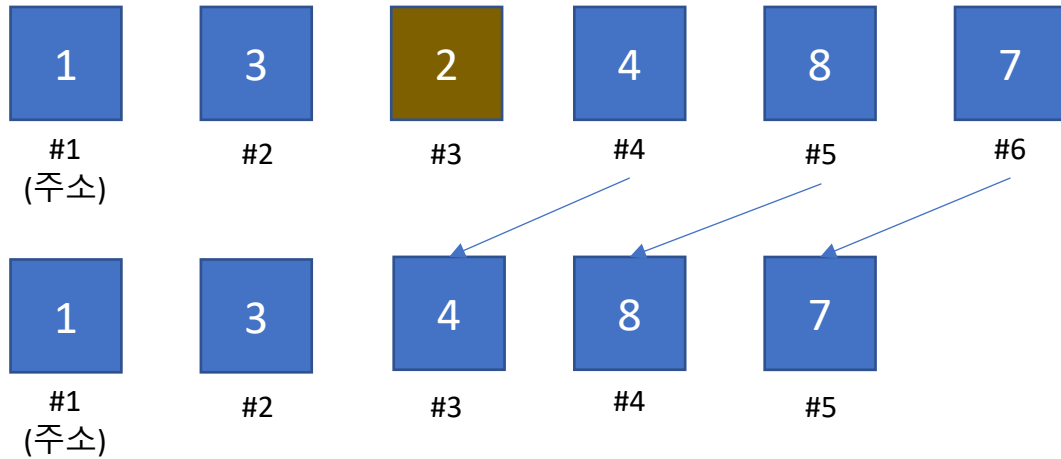
- 1번 노드 뒤에 있는 정보를 tobedeleted에 저장함
- 1번 노드 뒤+뒤에 있는 정보를 뒤로 바꿈
- 삭제!

```
Vertex tobedeleted = cur.next  
cur.next = cur.next.next  
delete tobedeleted
```



4. 끝

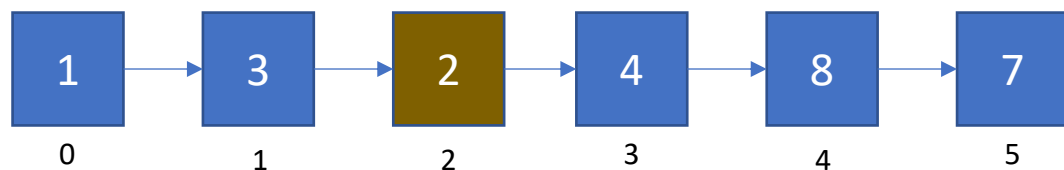
Array List의 삭제 방식



메모리가 확정적이기 때문에 값이
입력된 주소가 바뀜

그래서 Linked List와 대조적으로 느림!

Linked List의 조회 방식



2번 인덱스를 조회 했을 때, HEAD 부터 차근차근 찾아간다
그래서 느리다.

Array List의 조회 방식




2번 인덱스의 값을 찾는다고 했을 때

Array List는 2번의 주소를 얻어 내부적으로 계산해서 바로
찾아낸다 (주소값을 알고 바로 접근한다)

즉!

Array List는 인덱스를 이용해서 조회할 때 **Random Access**를
이용하기 때문에 빠르다.

결론

	추가/삭제	인덱스 조회
Array List	 느리다	 빠르다
Linked List	 느리다	 빠르다