Fast campus Copyright FASTCAMPUS Corp.All Rights Reserved

JavaScript 핵심 자료구조 알아보기

2) 배열(Array)과 리스트(List)

배열과 리스트 | 다양한 알고리즘의 기본이 되는 자료구조 이해하기

강사 나동빈

fast campus Copyright FASTCAMPUS Corp.All Rights Reserved

JavaScript
핵심 자료구조 알아보기

2) 배열(Array)과 리스트(List)

fast campus Copyright FASTCAMPUS Corp.All Rights Reserved

JavaScript 자료구조
배열과 리스트

배열(Array)

- **가장 기본적인 자료구조**다.
- 여러 개의 변수를 담는 공간으로 이해할 수 있다.
- 배열은 **인덱스(index)**가 존재하며, 인덱스는 0부터 시작한다.
- 특정한 인덱스에 직접적으로 접근 가능 → 수행 시간: $O(1)$

인덱스	0	1	2	3	4	5	6	7	8
값	25	75	83	59	24	72	55	17	42

fast campus Copyright FASTCAMPUS Corp.All Rights Reserved

배열의 특징

JavaScript
자료구조
배열과 리스트

- 컴퓨터의 메인 메모리에서 배열의 공간은 연속적으로 할당된다.

장점: **캐시(cache)** 히트 가능성이 높으며, 조회가 빠르다.

단점: 배열의 크기를 미리 지정해야 하는 것이 일반적이므로, 데이터의 추가 및 삭제에 한계가 있다.

메모리 주소	0x0000	0x0004	0x0008
인덱스	0	1	2
값	25	75	83

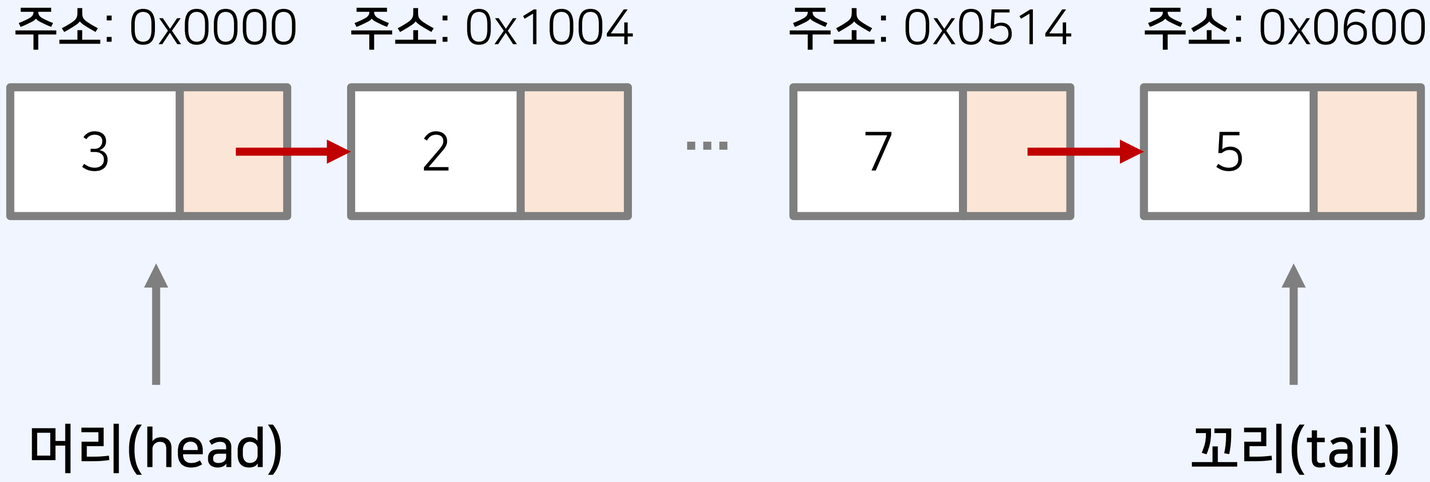
fast campus Copyright FASTCAMPUS Corp.All Rights Reserved

연결 리스트(Linked List)

- **연결 리스트**는 컴퓨터의 메인 메모리상에서 주소가 연속적이지 않다.
- 배열과 다르게 크기가 정해져 있지 않고, 리스트의 크기는 동적으로 변경 가능하다.

장점: **포인터(pointer)**를 통해 다음 데이터의 위치를 가리킨다는 점에서 **삽입과 삭제가 간편**하다.

단점: 특정 번째의 원소를 검색할 때는 앞에서부터 원소를 찾아야 하므로, 데이터 검색 속도가 느리다.



fast campus Copyright FASTCAMPUS Corp.All Rights Reserved

코딩 테스트를 위한 JavaScript의 배열

- 본 강의에서는 직접 연결 리스트를 구현하는 방법에 대해서는 다루지 않을 예정이다.
- **연결 리스트를 직접 구현하지 않아도** 대부분의 알고리즘 문제를 해결하기에는 어려움이 없다.

fast campus Copyright FASTCAMPUS Corp.All Rights Reserved

JavaScript의 배열

JavaScript
자료구조
배열과 리스트

- JavaScript에서는 **배열** 기능을 제공한다.

[알아 둘 점]

- 일반적인 프로그래밍 언어에서의 **배열**로 이해할 수 있다.
- JavaScript의 배열은 일반 배열처럼 임의의 인덱스를 이용해 직접적인 접근이 가능하다.
- JavaScript의 배열은 **동적 배열**의 기능을 제공하여, 맨 뒤의 위치에 원소 추가가 가능하다.

fast campus Copyright FASTCAMPUS Corp.All Rights Reserved

JavaScript의 배열

JavaScript
자료구조
배열과 리스트

- JavaScript의 배열 자료형은 **동적 배열**이다.
- 배열의 용량이 가득 차면, 자동으로 크기를 증가시킨다.
- 내부적으로 포인터(pointer)를 사용하여, 연결 리스트의 장점도 가지고 있다.
- **배열(array) 혹은 스택(stack)**의 기능이 필요할 때 사용할 수 있다.

[참고] 큐(queue)의 기능을 제공하지 못한다. (비효율적)

fast campus Copyright FASTCAMPUS Corp.All Rights Reserved

JavaScript 배열 초기화 1) 대괄호 사용하기

- JavaScript에서는 대괄호를 이용해 간단히 배열을 생성할 수 있다.

```
// 빈 배열 생성
let arr = [];

arr.push(7);
arr.push(8);
arr.push(9);

for (let i = 0; i < arr.length; i++) {
  console.log(arr[i]);
}
```

[실행 결과]

```
7
8
9
```

fast campus Copyright FASTCAMPUS Corp.All Rights Reserved

JavaScript 배열 초기화 2) Array() 사용하기

- JavaScript에서는 Array()를 이용해 간단히 배열을 생성할 수 있다.

```
let arr = new Array();

arr.push(7);
arr.push(8);
arr.push(9);

for (let i = 0; i < arr.length; i++) {
  console.log(arr[i]);
}
```

[실행 결과]

```
7
8
9
```

fast campus Copyright FASTCAMPUS Corp.All Rights Reserved

JavaScript 배열 초기화

JavaScript
자료구조
배열과 리스트

- JavaScript의 배열은 일반적인 변수 외에도 객체를 담을 수도 있다.

```
let arr = ["Hello", 777, true];

console.log(arr);
```

[실행 결과]

```
[ 'Hello', 777, true ]
```

fast campus Copyright FASTCAMPUS Corp.All Rights Reserved

JavaScript 배열 초기화

JavaScript
자료구조
배열과 리스트

- JavaScript에서는 임의의 크기를 가지는 배열을 만들 수 있다.
- 원하는 값을 직접 입력하여 초기화 할 수 있다.
- 크기가 N인 **1차원 배열을 만드는 방법**은 다음과 같다.

```
// 원하는 값을 직접 입력하여 초기화
let arr1 = [0, 1, 2, 3, 4];
console.log(arr1);

// 하나의 값으로 초기화
let arr2 = Array.from({ length: 5 }, () => 7);
console.log(arr2);
```

[실행 결과]

```
[ 0, 1, 2, 3, 4 ]
[ 7, 7, 7, 7, 7 ]
```

fast campus Copyright FASTCAMPUS Corp.All Rights Reserved

크기가 N X M인 2차원 리스트(배열) 만들기 ①

- 2차원 배열이 필요할 때는 다음과 같이 원하는 값을 직접 넣어 초기화할 수 있다.

```
// 원하는 값을 직접 입력하여 초기화
let arr1 = [
  [0, 1, 2, 3],
  [4, 5, 6, 7],
  [8, 9, 10, 11]
];
console.log(arr1);
```

[실행 결과]

```
[
  [ 0, 1, 2, 3 ],
  [ 4, 5, 6, 7 ],
  [ 8, 9, 10, 11 ]
]
```

fast campus Copyright FASTCAMPUS Corp.All Rights Reserved

크기가 N X M인 2차원 리스트(배열) 만들기 ②

- 최신 JavaScript 환경(ES6 이상)에서 사용할 수 있는 문법이다.
- 한 줄로 2차원 배열을 초기화할 수 있다.
- 배열의 각 원소에 크기가 5인 배열을 할당한다.

```
let arr = Array.from(Array(4), () => new Array(5))

console.log(arr);
```

[실행 결과]

```
[
  [ <5 empty items> ],
  [ <5 empty items> ],
  [ <5 empty items> ],
  [ <5 empty items> ]
]
```

fast campus Copyright FASTCAMPUS Corp.All Rights Reserved

크기가 N X M인 2차원 리스트(배열) 만들기 ③

- 다음과 같이 반복문을 이용해 2차원 배열을 초기화할 수 있다.

```
// 반복문을 이용해 배열 초기화
let arr2 = new Array(3);
for (let i = 0; i < arr2.length; i++) {
  arr2[i] = Array.from(
    { length: 4 },
    (undefined, j) => i * 4 + j
  );
}
console.log(arr2);
```

[실행 결과]

```
[
  [ 0, 1, 2, 3 ],
  [ 4, 5, 6, 7 ],
  [ 8, 9, 10, 11 ]
]
```

fast campus Copyright FASTCAMPUS Corp.All Rights Reserved

JavaScript 배열 다루기

JavaScript
자료구조
배열과 리스트

- JavaScript의 배열은 **동적 배열**이다.
- 배열이 생성된 이후에도 배열의 크기를 임의로 변경할 수 있다.
- push( ) 메서드를 통해 배열의 가장 뒤쪽에 새로운 원소를 추가할 수 있다.

```
let arr = [5, 6, 7, 8, 9];
arr.length = 8;
arr[7] = 3;
arr.push(1);

for (let x of arr) {
  console.log(x);
}
```

[실행 결과]

```
5
6
7
8
9
undefined
undefined
3
1
```

fast campus Copyright FASTCAMPUS Corp.All Rights Reserved

JavaScript 배열의 대표적인 메서드

- concat(): 여러 개의 배열을 이어 붙여서 합친 결과를 반환한다. $O(N)$

```
let arr1 = [1, 2, 3, 4, 5];
let arr2 = [6, 7, 8, 9, 10];
let arr = arr1.concat(arr2, [11, 12], [13]);

console.log(arr);
```

[실행 결과]

```
[
  1, 2, 3,  4,  5,  6,
  7, 8, 9, 10, 11, 12, 13
]
```

fast campus Copyright FASTCAMPUS Corp.All Rights Reserved

JavaScript 배열의 대표적인 메서드

- slice(left, right): 특정 구간의 원소를 꺼낸 배열을 반환한다. $O(N)$

```
let arr = [1, 2, 3, 4, 5];
let result = arr.slice(2, 4);

console.log(result);
```

[실행 결과]

```
[ 3, 4 ]
```

fast campus Copyright FASTCAMPUS Corp.All Rights Reserved

JavaScript 배열의 대표적인 메서드

- indexOf(): 특정한 값을 가지는 원소의 첫째 인덱스를 반환한다. $O(N)$
- 만약, 해당하는 원소가 없는 경우 -1을 반환한다.

```
let arr = [7, 3, 5, 6, 6, 2, 1];

console.log(arr.indexOf(5));
console.log(arr.indexOf(6));
console.log(arr.indexOf(8));
```

[실행 결과]

```
2
3
-1
```

fast campus Copyright FASTCAMPUS Corp.All Rights Reserved

연결 리스트(Linked List)

JavaScript
자료구조
배열과 리스트

- 연결 리스트는 각 노드가 한 줄로 연결되어 있는 자료 구조다.
- 각 노드는 (데이터, 포인터) 형태를 가진다.
- **포인터**: 다음 노드의 메모리 주소를 가리키는 목적으로 사용된다.

fast campus Copyright FASTCAMPUS Corp.All Rights Reserved

연결 리스트(Linked List)

JavaScript
자료구조
배열과 리스트

- 연결 리스트는 각 노드가 한 줄로 연결되어 있는 자료 구조다.
- 각 노드는 (데이터, 포인터) 형태를 가진다.
- **연결성**: 각 노드의 포인터는 다음 혹은 이전 노드를 가리킨다.

fast campus Copyright FASTCAMPUS Corp.All Rights Reserved

연결 리스트(Linked List)

JavaScript
자료구조
배열과 리스트

- 연결 리스트를 이용하면 다양한 자료구조를 구현할 수 있다.
- **예시)** 스택, 큐 등
- JavaScript는 연결 리스트를 활용하는 자료구조를 제공한다.
- 그래서 연결 리스트를 실제 구현해야 하는 경우는 적지만, 그 원리에 대해서 이해해 보자.

fast campus Copyright FASTCAMPUS Corp.All Rights Reserved

연결 리스트(Linked List) vs. 배열(Array)

- 연결 리스트와 배열(array)을 비교하여 장단점을 이해할 필요가 있다.
- 특정 위치의 데이터를 삭제할 때, 일반적인 **배열**에서는 $O(N)$만큼의 시간이 소요된다.
- 하지만, **연결 리스트**를 이용하면 단순히 연결만 끊어주면 된다.
- 따라서 삭제할 위치를 정확히 알고 있는 경우 $O(1)$의 시간이 소요된다.

fast campus Copyright FASTCAMPUS Corp.All Rights Reserved

삽입할 때: 배열

JavaScript
자료구조
배열과 리스트

- 배열에 새로운 원소를 삽입할 때, 최악의 경우 시간 복잡도를 계산하세요.
- **예시)** 아래 배열에서 인덱스 3에 원소 “59”를 삽입할 경우

삽입할 위치

인덱스	0	1	2	3	4	5	6	7	8
값	25	75	83	24	72	55	17	42	

fast campus Copyright FASTCAMPUS Corp.All Rights Reserved

삽입할 때: 배열

JavaScript
자료구조
배열과 리스트

- 배열에 새로운 원소를 삽입할 때, 최악의 경우 시간 복잡도를 계산하세요.
- **예시)** 아래 배열에서 인덱스 3에 원소 “59”를 삽입할 경우

한 칸씩 밀기 →

인덱스	0	1	2	3	4	5	6	7	8
값	25	75	83		24	72	55	17	42

fast campus Copyright FASTCAMPUS Corp.All Rights Reserved

삽입할 때: 배열

JavaScript
자료구조
배열과 리스트

- 배열에 새로운 원소를 삽입할 때, 최악의 경우 시간 복잡도를 계산하세요.
- **예시)** 아래 배열에서 인덱스 3에 원소 “59”를 삽입할 경우

삽입 완료

인덱스	0	1	2	3	4	5	6	7	8
값	25	75	83	59	24	72	55	17	42

fast campus Copyright FASTCAMPUS Corp.All Rights Reserved

삭제할 때: 배열

JavaScript
자료구조
배열과 리스트

- 배열에 존재하는 원소를 삭제할 때, 최악의 경우 시간 복잡도를 계산하세요.
- **예시)** 아래 배열에서 인덱스 3에 해당하는 원소를 삭제하는 경우

삭제할 원소

인덱스	0	1	2	3	4	5	6	7	8
값	25	75	83	59	24	72	55	17	42

fast campus Copyright FASTCAMPUS Corp.All Rights Reserved

삭제할 때: 배열

JavaScript
자료구조
배열과 리스트

- 배열에 존재하는 원소를 삭제할 때, 최악의 경우 시간 복잡도를 계산하세요.
- **예시)** 아래 배열에서 인덱스 3에 해당하는 원소를 삭제하는 경우

삭제 완료

인덱스	0	1	2	3	4	5	6	7	8
값	25	75	83		24	72	55	17	42

fast campus Copyright FASTCAMPUS Corp.All Rights Reserved

삭제할 때: 배열

JavaScript
자료구조
배열과 리스트

- 배열에 존재하는 원소를 삭제할 때, 최악의 경우 시간 복잡도를 계산하세요.
- **예시)** 아래 배열에서 인덱스 3에 해당하는 원소를 삭제하는 경우

한 칸씩 당기기

인덱스	0	1	2	3	4	5	6	7	8
값	25	75	83	24	72	55	17	42	

fast campus Copyright FASTCAMPUS Corp.All Rights Reserved

JavaScript
자료구조
배열과 리스트

삭제할 때: 배열

- 배열에 존재하는 원소를 삭제할 때, 최악의 경우 시간 복잡도를 계산하세요.
- **예시)** 아래 배열에서 인덱스 3에 해당하는 원소를 삭제하는 경우

한 칸씩 당기기

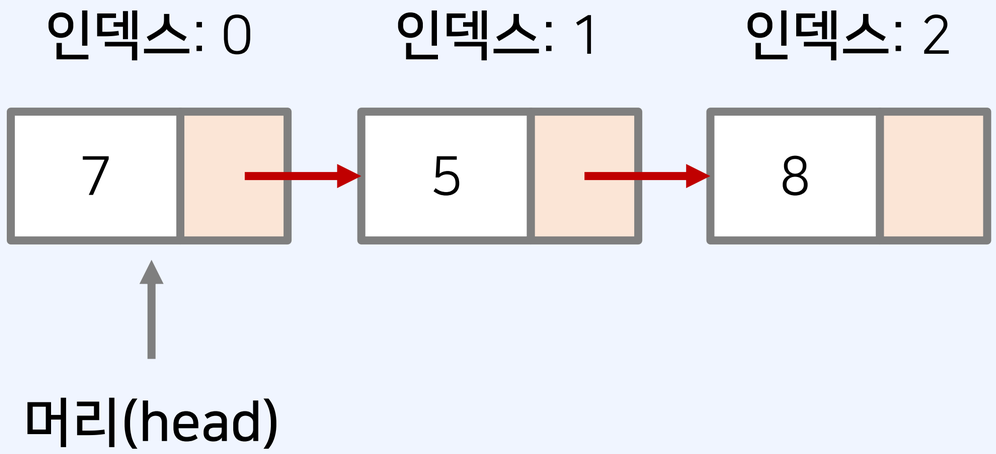
인덱스	0	1	2	3	4	5	6	7	8
값	25	75	83	24	72	55	17	42	

- 따라서, 최악의 경우 시간 복잡도는 $O(N)$이다.

fast campus Copyright FASTCAMPUS Corp.All Rights Reserved

연결 리스트(Linked List): 삽입(Insert) 연산

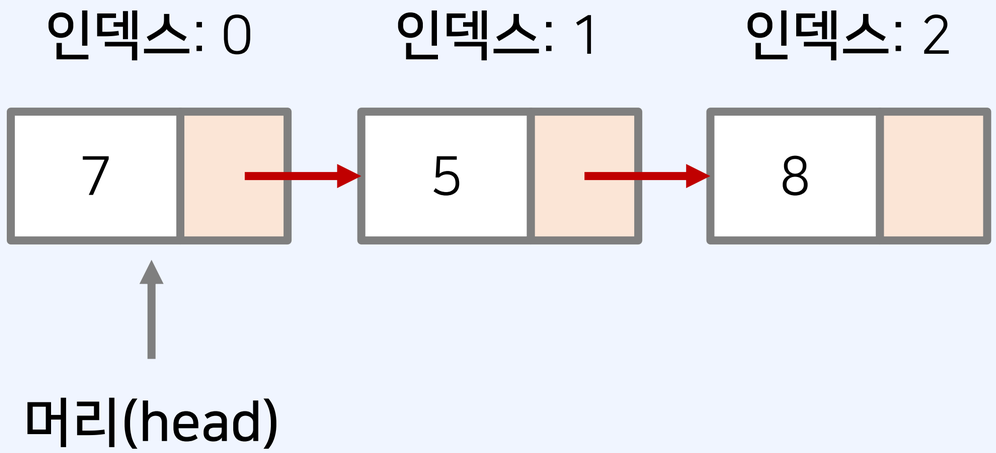
- 삽입할 위치를 알고 있다면, 물리적인 위치를 한 칸씩 옮기지 않아도 삽입할 수 있다.



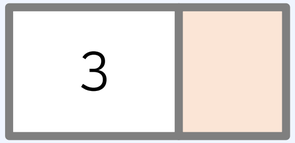
fast campus Copyright FASTCAMPUS Corp.All Rights Reserved

연결 리스트(Linked List): 삽입(Insert) 연산

- 삽입할 위치를 알고 있다면, 물리적인 위치를 한 칸씩 옮기지 않아도 삽입할 수 있다.



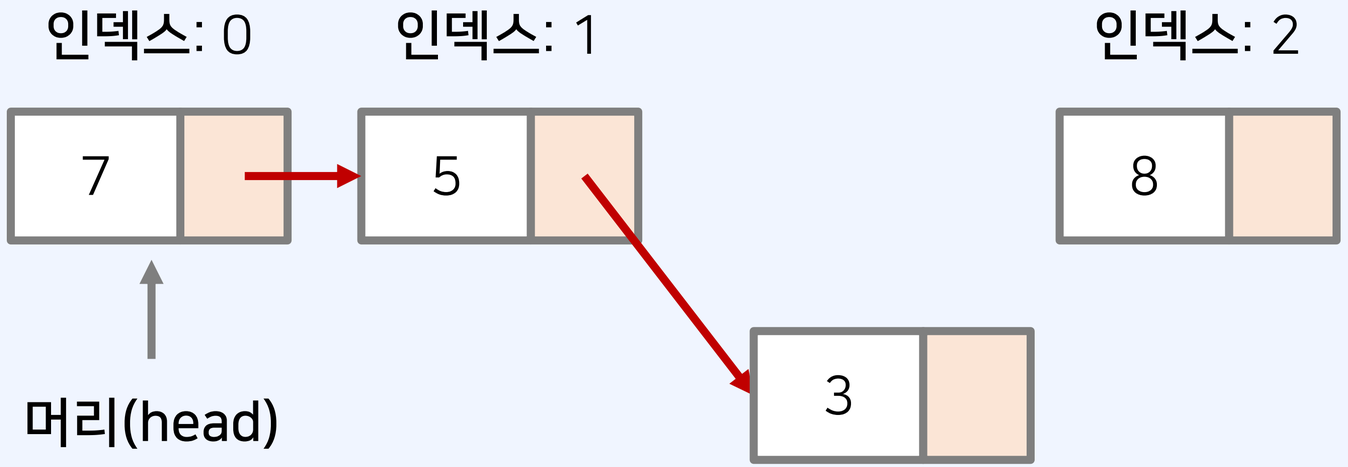
- 인덱스 2의 위치에 원소를 삽입



fast campus Copyright FASTCAMPUS Corp.All Rights Reserved

연결 리스트(Linked List): 삽입(Insert) 연산

- 삽입할 위치를 알고 있다면, 물리적인 위치를 한 칸씩 옮기지 않아도 삽입할 수 있다.

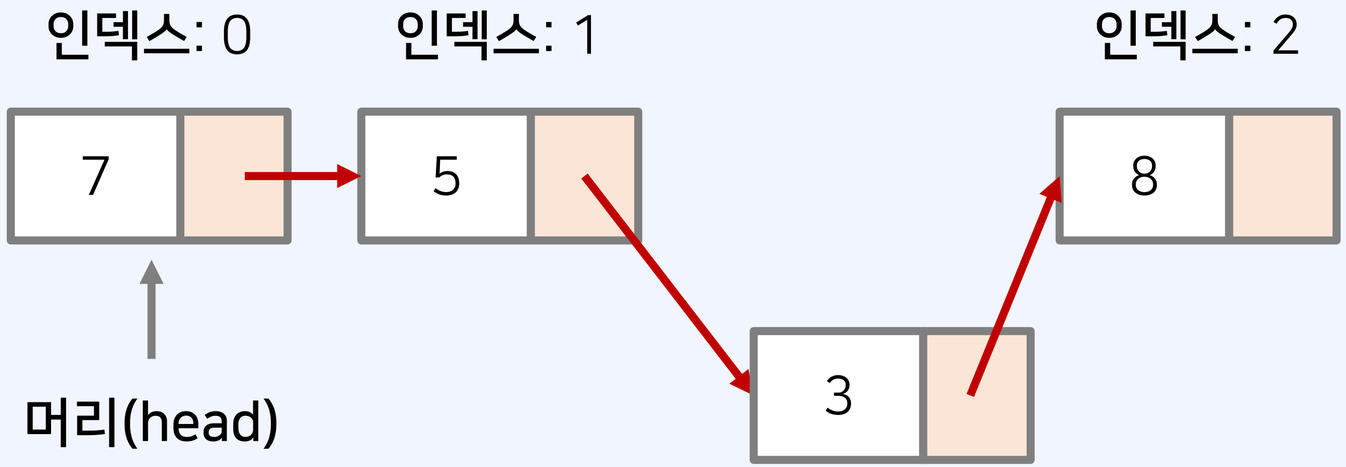


- 인덱스 2의 위치에 원소를 삽입

fast campus Copyright FASTCAMPUS Corp.All Rights Reserved

연결 리스트(Linked List): 삽입(Insert) 연산

- 삽입할 위치를 알고 있다면, 물리적인 위치를 한 칸씩 옮기지 않아도 삽입할 수 있다.

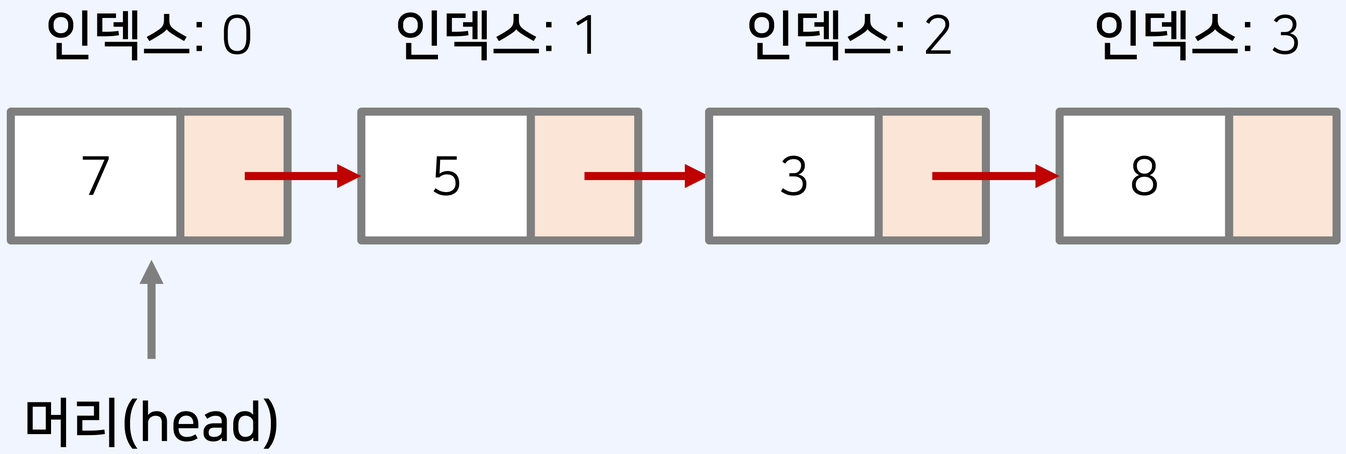


- 인덱스 2의 위치에 원소를 삽입

fast campus Copyright FASTCAMPUS Corp.All Rights Reserved

연결 리스트(Linked List): 삽입(Insert) 연산

- 삽입할 위치를 알고 있다면, 물리적인 위치를 한 칸씩 옮기지 않아도 삽입할 수 있다.

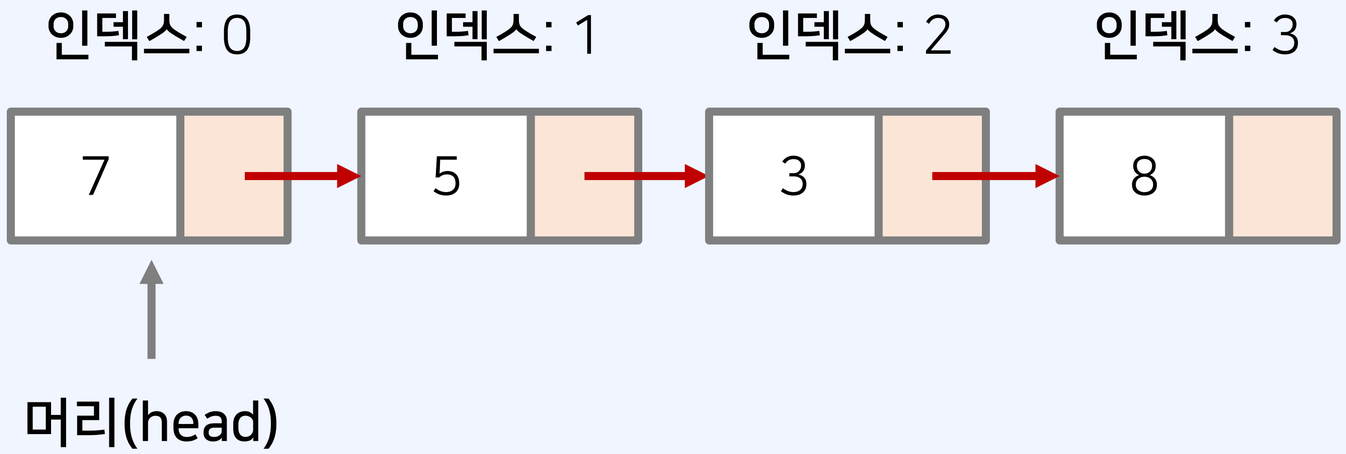


- 인덱스 2의 위치에 원소를 삽입

fast campus Copyright FASTCAMPUS Corp.All Rights Reserved

연결 리스트(Linked List): 삭제(Remove) 연산

- 삭제할 위치를 알고 있다면, 물리적인 위치를 한 칸씩 옮기지 않아도 삽입할 수 있다.

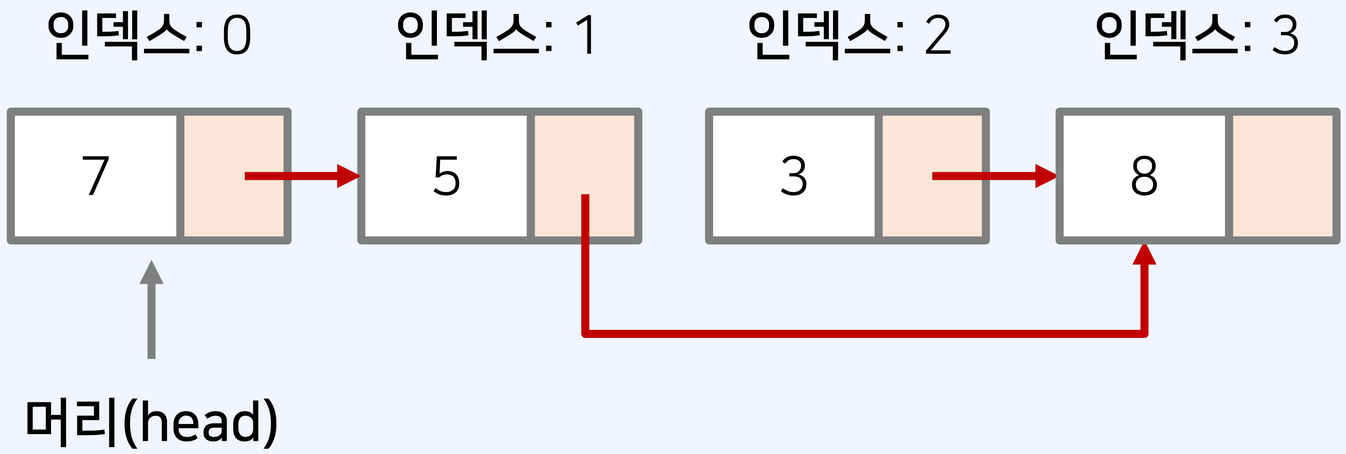


- 인덱스 2의 위치의 원소를 삭제

fast campus Copyright FASTCAMPUS Corp.All Rights Reserved

연결 리스트(Linked List): 삭제(Remove) 연산

- 삭제할 위치를 알고 있다면, 물리적인 위치를 한 칸씩 옮기지 않아도 삽입할 수 있다.

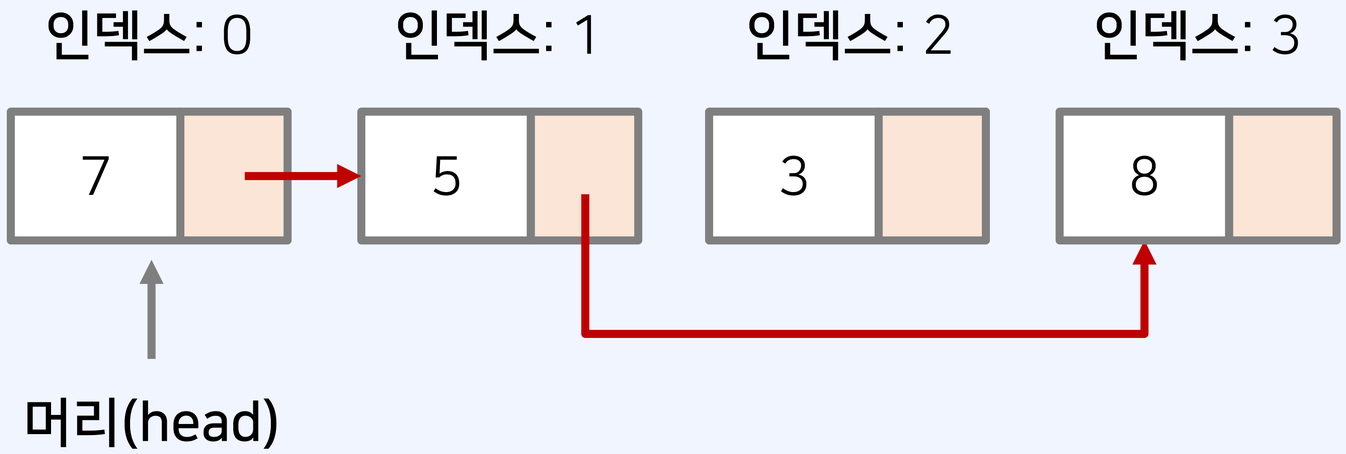


- 인덱스 2의 위치의 원소를 삭제

fast campus Copyright FASTCAMPUS Corp.All Rights Reserved

연결 리스트(Linked List): 삭제(Remove) 연산

- 삭제할 위치를 알고 있다면, 물리적인 위치를 한 칸씩 옮기지 않아도 삽입할 수 있다.

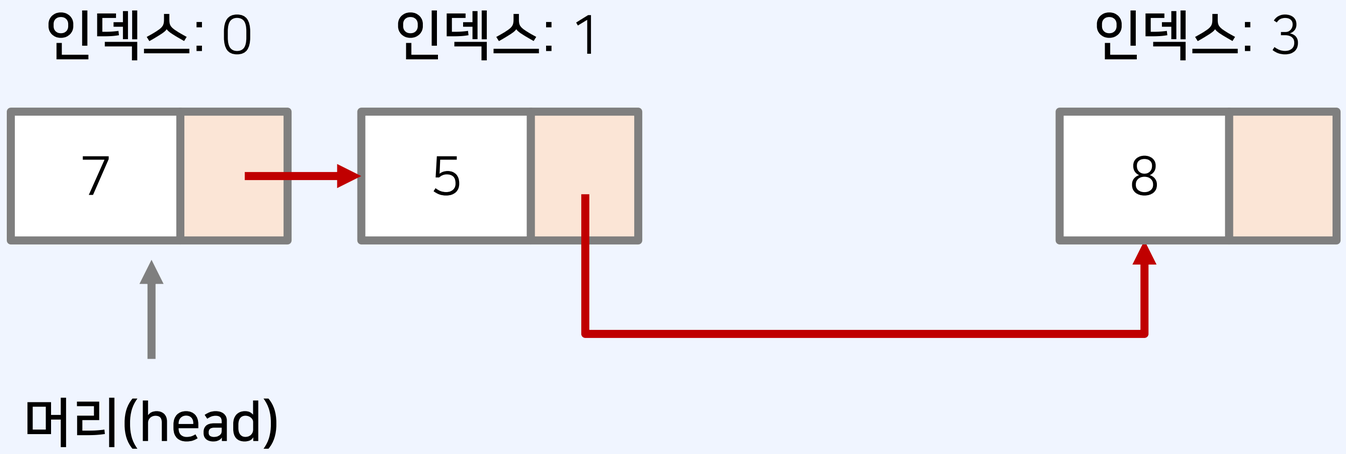


- 인덱스 2의 위치의 원소를 삭제

fast campus Copyright FASTCAMPUS Corp.All Rights Reserved

연결 리스트(Linked List): 삭제(Remove) 연산

- 삭제할 위치를 알고 있다면, 물리적인 위치를 한 칸씩 옮기지 않아도 삽입할 수 있다.

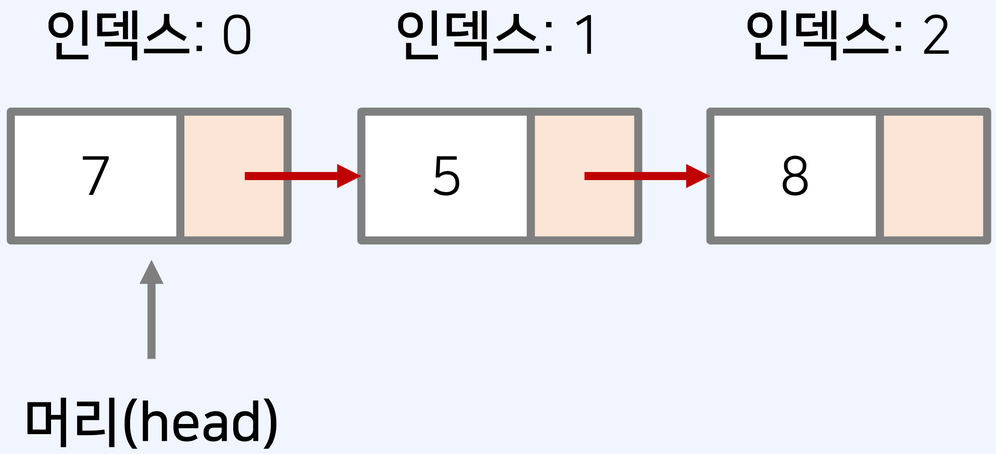


- 인덱스 2의 위치의 원소를 삭제

fast campus Copyright FASTCAMPUS Corp.All Rights Reserved

연결 리스트(Linked List): 삭제(Remove) 연산

- 삭제할 위치를 알고 있다면, 물리적인 위치를 한 칸씩 옮기지 않아도 삽입할 수 있다.

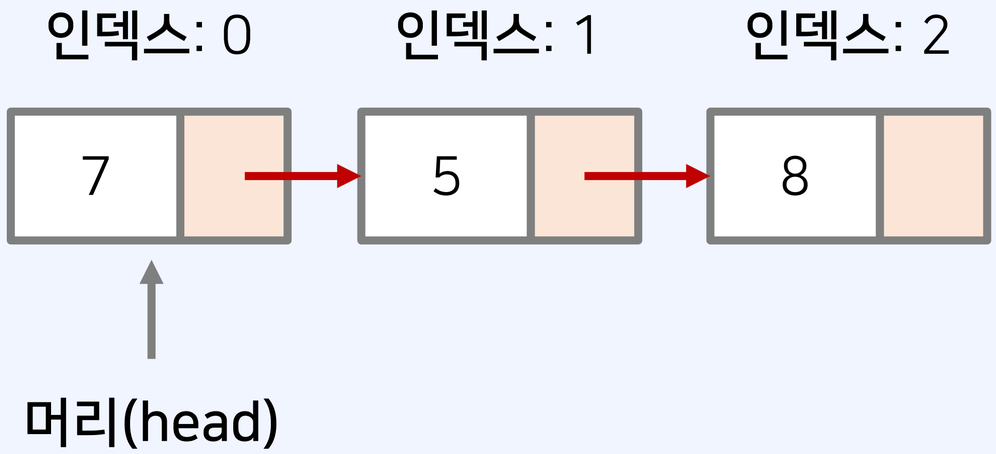


- 인덱스 2의 위치의 원소를 삭제

fast campus Copyright FASTCAMPUS Corp.All Rights Reserved

연결 리스트(Linked List): 뒤에 붙이기(Append) 연산

- 뒤에 붙일 때는 마지막 노드의 다음 위치에 원소를 넣으면 된다.

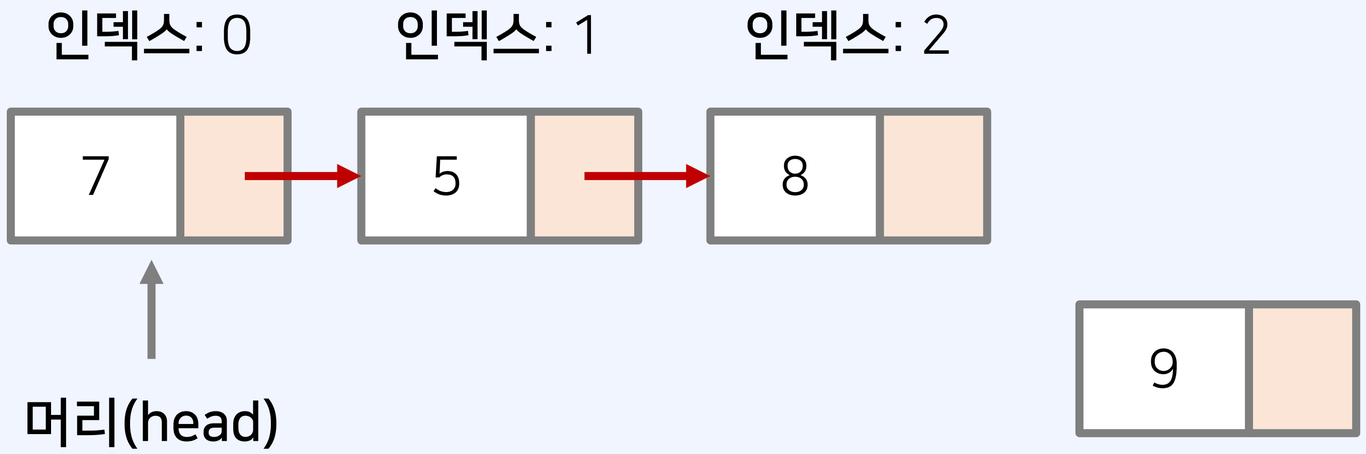


- **마지막 위치**에 새로운 원소를 추가

fast campus Copyright FASTCAMPUS Corp.All Rights Reserved

연결 리스트(Linked List): 뒤에 붙이기(Append) 연산

- 뒤에 붙일 때는 마지막 노드의 다음 위치에 원소를 넣으면 된다.

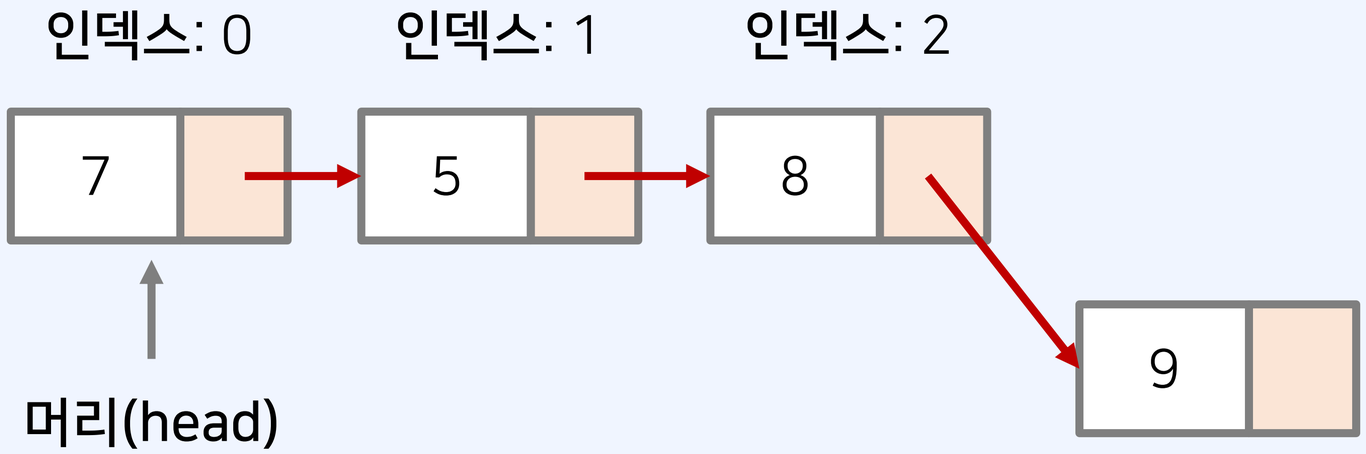


- **마지막 위치**에 새로운 원소를 추가

fast campus Copyright FASTCAMPUS Corp.All Rights Reserved

연결 리스트(Linked List): 뒤에 붙이기(Append) 연산

- 뒤에 붙일 때는 마지막 노드의 다음 위치에 원소를 넣으면 된다.

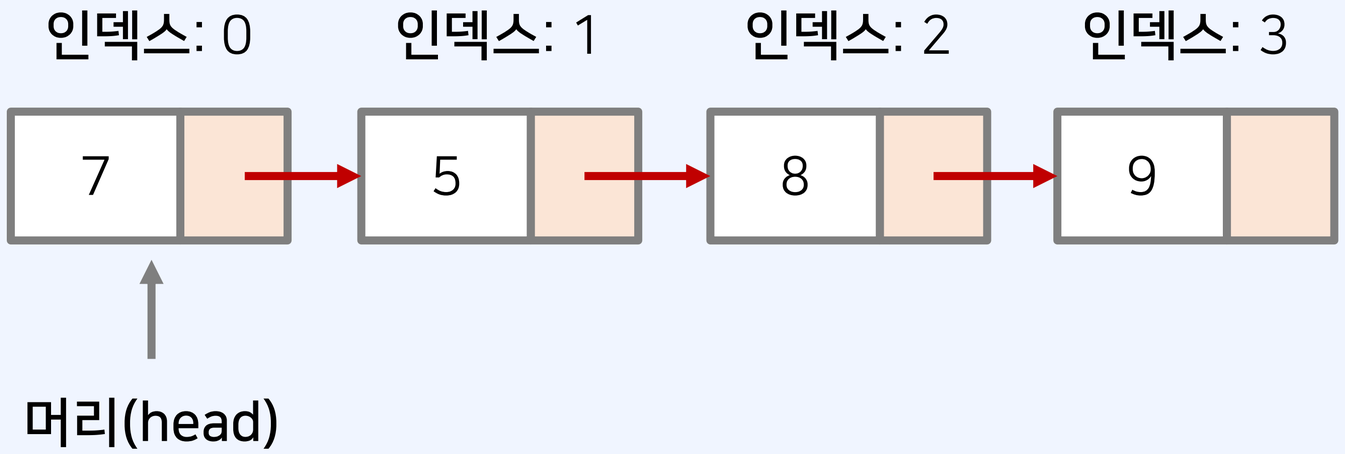


- **마지막 위치**에 새로운 원소를 추가

fast campus Copyright FASTCAMPUS Corp.All Rights Reserved

연결 리스트(Linked List): 뒤에 붙이기(Append) 연산

- 뒤에 붙일 때는 마지막 노드의 다음 위치에 원소를 넣으면 된다.



- **마지막 위치**에 새로운 원소를 추가