**O(1) 시간 복잡도와 해시 테이블 탐구**

30501 김도현

## **1페이지**

서론 - 탐색 속도의 혁명, O(1) 시간 복잡도  
 알고리즘의 효율성을 평가하는 척도인 시간 복잡도에서 O(1)은 '상수 시간'을 의미하며, 데이터의 크기(n)와 상관없이 언제나 일정한 시간이 걸리는 가장 이상적인 성능을 나타낸다. 리스트(배열)에서 특정 요소를 찾기 위해 처음부터 끝까지 순차적으로 탐색하는 방법의 시간 복잡도는 O(n)이다. 데이터가 1억 개라면 최악의 경우 1억 번의 연산이 필요하다. 이 노트북은 이러한 비효율을 극복하고 O(1)의 탐색 속도를 구현하는 핵심 자료구조인 해시 테이블(Hash Table)의 원리를 깊이 있게 탐구하는 것을 목표로 한다.

## **2페이지**

해시 테이블의 심장, 해시 함수  
 해시 테이블의 핵심 아이디어는 'Key'를 '배열의 인덱스'로 직접 변환하여 원하는 'Value'에 즉시 접근하는 것이다. 이 변환 과정을 수행하는 것이 바로 해시 함수(Hash function)다. 해시 함수는 임의의 길이를 가진 Key(예: 문자열 "apple")를 입력받아, 고정된 길이의 숫자(해시 값)로 변환한다. 노트북에서는 이 해시 값을 배열의 크기 N으로 나눈 나머지(A % N)를 최종 인덱스로 사용하는 가장 기본적인 '나눗셈법'을 소개했다. 예를 들어 배열의 크기가 100이라면, 해시 값이 789이든 123456789이든 나머지 연산을 통해 89라는 인덱스를 얻게 되어, 데이터를 한정된 크기의 배열 안에 효율적으로 매핑할 수 있다.

## **3페이지**

피할 수 없는 문제, 해시 충돌(Hash Collision)  
 해시 함수의 출력값(해시 값)의 범위는 사실상 무한하지만, 이를 저장할 배열의 크기는 유한하다. 이로 인해 서로 다른 Key를 해시 함수에 입력했음에도 불구하고 동일한 인덱스를 결과로 얻게 되는 상황이 발생하는데, 이를 '해시 충돌'이라고 한다. 예를 들어, Key가 "apple"일 때와 "banana"일 때 모두 나머지 연산의 결과가 23이라면, 두 데이터를 23번 인덱스에 동시에 저장할 수 없는 문제가 발생한다. 좋은 해시 함수는 데이터의 분포를 최대한 고르게 하여 충돌 가능성을 줄여주지만, 충돌을 완벽하게 피할 수는 없으므로, 이를 효과적으로 해결하는 메커니즘이 반드시 필요하다.

## **4페이지**

충돌 해결 전략 (1) - 체이닝(Chaining)  
 해시 충돌을 해결하는 가장 대표적인 방법 중 하나는 '체이닝'이다. 이는 해시 테이블의 각 인덱스(슬롯 또는 버킷)에 단일 값이 아닌, 연결 리스트(Linked List)를 할당하는 방식이다. 만약 여러 데이터가 동일한 인덱스로 해싱되면, 이 데이터들은 해당 인덱스의 연결 리스트에 차례대로 추가(chaining)된다. 데이터를 검색할 때는 먼저 Key를 해싱하여 인덱스를 찾고, 해당 인덱스가 가리키는 연결 리스트를 처음부터 순회하며 원하는 Key를 가진 노드(node)를 찾는다. 노트북은 HashTable 클래스 내부에 리스트를 중첩으로 사용하여 이러한 체이닝 구조를 시뮬레이션하고, 이름(문자열)을 Key로 하여 데이터를 저장하고 검색하는 과정을 코드로 구현했다.

## **5페이지**

체이닝의 성능 분석과 부하율(Load Factor)  
 체이닝 방식에서 최악의 경우는 모든 데이터가 하나의 인덱스로만 충돌하는 상황이다. 이 경우 해시 테이블은 사실상 하나의 거대한 연결 리스트와 다를 바 없어지며, 탐색 시간 복잡도는 O(n)으로 저하된다. 하지만 데이터가 해시 테이블 전체에 고르게 분산된다면, 각 연결 리스트의 길이는 매우 짧게 유지될 것이다. 해시 테이블의 성능을 논할 때 중요한 개념은 '부하율(Load Factor)'인데, 이는 (저장된 데이터의 개수 n) / (해시 테이블의 크기 m)으로 정의된다. 부하율이 일정 수준 이하로 유지된다면, 탐색에 걸리는 시간은 평균적으로 O(1)에 수렴한다고 볼 수 있다.

## **6페이지**

충돌 해결 전략 (2) - 개방 주소법(Open Addressing)  
 충돌을 해결하는 또 다른 접근법은 '개방 주소법'이다. 이 방식은 충돌이 발생하면, 미리 정해진 규칙에 따라 해시 테이블 내의 다른 비어있는 슬롯을 찾아 데이터를 저장한다. 가장 간단한 규칙은 '선형 탐사(Linear Probing)'로, 충돌이 발생한 인덱스 바로 다음 인덱스(i+1)를 확인하고, 거기도 차 있다면 그 다음(i+2)을 확인하는 식으로 빈 공간을 찾아 나선다. 검색 시에도 동일한 순서로 탐사를 진행한다. 노트북에서는 HashTable 클래스를 개방 주소법 방식으로 다시 구현하여, 데이터를 저장(set)하고, Key로 값을 가져오는(get) 로직을 보여주었다. 개방 주소법은 체이닝과 달리 추가적인 자료구조(연결 리스트)를 사용하지 않아 메모리 효율성이 좋다는 장점이 있다.

## **7페이지**

결론 - O(1)을 향한 공학적 절충  
 본 노트북은 O(1)이라는 이상적인 시간 복잡도를 현실 세계에서 구현하기 위해 해시 테이블이 어떻게 동작하는지 명확하게 보여주었다. 해시 함수를 통해 빠른 접근을 시도하지만, 필연적으로 발생하는 해시 충돌 문제를 '체이닝'이나 '개방 주소법'과 같은 공학적 절충안으로 해결해야 함을 학습했다. 결국 해시 테이블의 O(1) 성능은 수학적으로 보장된 최악의 경우가 아닌, '잘 설계된 해시 함수와 충돌 해결 전략 하에서 달성되는 평균적인 성능'임을 이해하는 것이 핵심이다. 파이썬의 딕셔너리(dict)와 같은 내장 자료구조는 이러한 원리들을 고도로 최적화하여 구현한 결과물이다.