해시 테이블은 키와 값 쌍으로 데이터를 저장하고 매우 빠른 검색 속도를 제공하는 비선형 자료 구조이다. 마치 사전을 찾는 것처럼 특정 키를 입력하면 해당하는 값을 거의 순식간에 찾아낼 수 있다.

해시 테이블의 작동 원리

1. 해시 함수 : 임의의 길이의 키를 고정된 길이의 해시 값으로 변환하는 함수이다. 이 해시 값은 배열의 인덱스로 사용되어 데이터가 저장될 위치를 결정한다.
2. 배열(버킷) : 해시 값을 인덱스로 사용하는 배열이다. 각 버킷에는 하나 이상의 데이터가 저장될 수 있다.
3. 저장 : 키를 해시 함수에 넣어 생성된 해시 값을 이용하여 해당 버킷에 데이터를 저장한다.
4. 검색 : 찾고자 하는 키를 해시 함수에 넣어 해시 값을 생성하고 해당 버킷에서 데이터를 검색한다.

해시 테이블 사용 이유

1. 빠른 검색 : 해시 함수를 통해 직접 데이터의 저장 위치를 계산하므로 거의 상수 시간(O(1)) 안에 데이터를 찾을 수 있다.
2. 유연한 크기 조절 : 데이터의 양에 따라 배열의 크기를 동적으로 조절할 수 있다.
3. 다양한 응용 : 캐시, 딕셔너리, 심볼 테이블 등 다양한 분야에서 사용된다.

해시 테이블의 장단점

장점
검색, 삽입, 삭제 연산이 평균적으로 O(1)의 시간 복잡도를 가지므로 매우 빠르다.
다양한 자료 구조를 구현하는 데 활용될 수 있다.
단점
해시 함수의 선택에 따라 성능이 크게 달라질 수 있다.
충돌이 자주 발생하면 성능이 저하될 수 있다.

충돌 해결 기법

충돌이 발생했을 때 데이터를 효율적으로 저장하기 위한 다양한 기법들이 있다.

1. 체이닝(Chaining)

각 버킷에 연결 리스트를 연결하여 충돌이 발생한 데이터를 연결 리스트에 추가하는 방법이다.
장점 : 구현이 간단하고 충돌이 자주 발생하더라도 효율적으로 처리할 수 있다.

단점 : 연결 리스트를 사용하기 때문에 추가적인 메모리 공간이 필요한다.

2. 개방 주소법(Open Addressing)

충돌이 발생하면 다른 버킷에 데이터를 저장하는 방법이다.
선형 탐색 : 충돌이 발생한 버킷의 다음 버킷부터 순차적으로 빈 공간을 찾아 데이터를 저장한다.
제곱 탐색 : 충돌이 발생한 버킷에서 특정 함수(ex: i^2)를 이용하여 이동할 버킷을 계산한다.
이중 해싱 : 두 개의 해시 함수를 사용하여 충돌 시 이동할 버킷을 계산한다.
장점 : 추가적인 메모리 공간이 필요하지 않다.
단점 : 클러스터링 문제가 발생할 수 있으며 삭제 연산이 복잡한다

해시 테이블의 활용 예시

1. 캐시 : 데이터를 빠르게 조회하기 위해 해시 테이블을 사용한다.
2. 연관 배열 : 키를 이용하여 값을 빠르게 찾아야 할 때 사용한다.
3. 집합 : 중복되지 않는 요소들을 저장하고 빠른 멤버십 테스트를 수행해야 할 때 사용한다.