● Hash table에서 collision이 발생하면 어떻게 되나? 해결방법엔 뭐가 있을까?

▶ collision이란 서로 다른 key의 해시값이 똑같을 때를 말합니다. collision이 발생할 경우 대표적으로 두 가지 방법으로 해결합니다. 첫 번째, open addressing 방식은 collision이 발생하면 미리 정한 규칙에 따라 hash table의 비어있는 slot을 찾습니다. 빈 slot을 찾는 방법에 따라 크게 Linear Probing, Quadratic Probing, Double Hashing으로 나뉩니다. 두 번째 separate chaining 방식은 linked list를 이용합니다. 만약에 collision이 발생하면, linked list에 노드, 즉 slot를 추가하여 데이터를 저장합니다. 삽입과 검색, 삭제의 시간복잡도는 O(1)이지만, worst case의 경우에 검색과 삭제의 시간복잡도는 O(n)이 될 수 있습니다. worst case의 경우 n개의 모든 key가 동일한 해시값을 갖게 되면 길이 n의 linked list가 생성되기 때문에, 검색의 시간복잡도가 O(n)이 되는 것입니다. worst case의 경우 링크드 리스트 대신 Binary Search Tree(BST) 자료구조인 레드 블랙 트리를 사용함으로써 검색의 worst case 시간복잡도를 O(logn)으로 낮출 수 있습니다.

● Open addressing

▶ open addressing 방식은 collision이 발생하면 미리 정한 규칙에 따라 hash table의 비어있는 slot을 찾습니다. 추가적인 메모리를 사용하지 않으므로 linked list 또는 tree자료구조를 통해 추가로 메모리 할당을 하는 separate chaining방식에 비해 메모리를 적게 사용합니다. open addressing은 빈 slot을 찾는 방법에 따라 크게 Linear Probing, Quadratic Probing, Double Hashing으로 나뉩니다.

▶ Linear Probing(선형 조사법) & Quadratic Probing(이차 조사법): 선형 조사법은 충돌이 발생한 해시값으로부터 일정한 값만큼(+1, +2, +3, ...) 건너 뛰어, 비어 있는 slot에 데이터를 저장합니다. 이차 조사법은 제곱수(+1^2, +2^2, +3^2, ...)로 건너 뛰어, 비어 있는 slot을 찾습니다. 충돌이 여러번 발생하면 여러번 건너 뛰어 빈 slot을 찾습니다. 선형 조사법과 이차 조사법의 경우 충돌 횟수가 많아지면 특정 영역에 데이터가 집중적으로 몰리는 클러스터링(clustering) 현상이 발생하는 단점이 있습니다. 클러스터링 현상이 발생하면, 평균 탐색 시간이 증가하게 됩니다. 충돌할 확률이 더 증가하게 되기 때문입니다.

▶ Double Hashing(이중해시, 중복해시): Open addressing 방식을 통해 collision을 해결할 때, probing 하는 방식 중의 하나입니다. linear probing이나 quadratic probing을 통해 탐사할 경우 탐사이동폭이 같아 발생할 수 있는 클러스터링 문제가 발생하지 않도록, 2개의 해시 함수를 사용하는 방식을 이중 해싱이라고 합니다. 하나는 최초의 해시값을 얻을 때 사용하고, 또 다른 하나는 해시 충돌이 발생할 때 탐사 이동폭을 얻기 위해 사용합니다. 이렇게 하면 충돌이 발생할 때마다 다든 탐사 이동폭을 얻을 수 있기 때문에, 클러스터링 문제가 발생하지 않습니다.

● Separate chaining

▶ Separate chaining 방식은 linked list 또는 Tree를 이용하여 collision을 해결합니다. 충돌이 발생하면 linked list에 노드(slot)를 추가하여 데이터를 저장합니다. 삽입과 검색, 삭제의 시간복잡도는 O(1)이지만, worst case의 경우에 검색과 삭제의 시간복잡도는 O(n)이 될 수 있습니다. worst case의 경우 n개의 모든 key가 동일한 해시값을 갖게 되면 길이 n의 linked list가 생성되기 때문에, 검색의 시간복잡도가 O(n)이 되는 것입니다. seperate chaining은 기본적으로 linked list를 이용하여 데이터를 저장하지만, collision이 많이 발생하여 linked list의 길이가 길어지면, 즉 8개 이상이 되면 Binary Search Tree(BST) 자료구조인 레드 블랙 트리를 이용하여 데이터를 저장하기도 합니다 (6개가 되면 다시 링크드 리스트로 변환). 링크드 리스트는 충돌하는 데이터가 많아질수록 탐색 시간이 O(n)으로 늘어나기 때문에, 레드 블랙 트리를 사용함으로써 검색의 worst case 시간복잡도를 O(logn)으로 낮추기 위함입니다.

- 삽입: 서로 다든 두 key가 같은 해시값을 갖게 되면 linked list에 node를 추가하여 (key, value) 데이터 쌍을 저장합니다. 삽입의 시간복잡도는 O(1)입니다.
- 검색: 기본적으로 O(1)의 시간복잡도 이지만 최악의 경우 O(n)의 시간복잡도를 갖습니다.
- 삭제: 삭제를 하기 위해선 검색을 먼저 해야하므로 검색의 시간복잡도와 동일합니다. 기본적으로 O(1)이지만 최악의 경우 O(n)의 시간복잡도를 갖습니다.

● worst case에 시간복잡도는 O(n)이라고 했는데 어떤 상황인가?

▶ n개의 모든 key가 동일한 해시값을 갖게 되면 길이 n의 linked list가 생성되게 됩니다. 이 때, 특정 key를 찾기 위해서는 길이 n의 linked list를 검색하는 O(n)의 시간복잡도와 동일하게 됩니다.

Resources: inflearn 개발남 노씨