# Discussion 05/25

## Discussion 11-12

Why is a hash structure not the best choice for a search key on which range queries are likely?

Hash structure 에서는 인접한 값이라도 저장되는 위치가 인접해있다는 보장이 없기 때문에 Range query 를 수행할 때는 결국 해당 range 에 해당하는 모든 값을 찾으려면 저장되어 있는 모든 record 를 다 돌아봐야 한다. 따라서 적합하지 않다.

=〉 Hash function 의 특징이 locality 를 없애는 것이기 때문에 (uniform, random). Locality 를 지원하면서 hash 를 짜면 hash 의 장점이 없어진다.

## Discussion 11-13

Using the hash function *h* defined below, construct an *extendable hash index* on *ID* of the *instructor* table. (bucket size = 2 entries)

- For a decimal number *x*, *h*(*x*) is the binary number obtained by converting each *odd digit* to 1, and each *even digit* to 0.
  - *h*(12345) = 10101
  - *h*(24680) = 00000
  - *h*(13579) = 11111

| *ID* | *name* | *dept_name* | *salary* |
|---|---|---|---|
| 10101 | Srinivasan | Comp. Sci. | 65000 |
| 12121 | Wu | Finance | 90000 |
| 15151 | Mozart | Music | 40000 |
| 22222 | Einstein | Physics | 95000 |
| 32343 | El Said | History | 60000 |
| 33456 | Gold | Physics | 87000 |
| 45565 | Katz | Comp. Sci. | 75000 |
| 58583 | Califieri | History | 62000 |
| 76543 | Singh | Finance | 80000 |
| 76766 | Crick | Biology | 72000 |
| 83821 | Brandt | Comp. Sci. | 92000 |
| 98345 | Kim | Elec. Eng. | 80000 |

처음 3 개 insert =〉 Srinivasan 과 Wu 가 모두 hash function 이 10101 임. Mozart 는 11111. Hash prefix 는 2 이고 address table entry 의 개수는 4 개. Bucket 1 에는 Srinivasan 과 Wu 이고 i_1 = 2. Bucket 2 에는 Mozart 가 있고 i_2 = 2.

**전체 insert 는 각자 해보기!**

## Discussion 11-14

What is the height of a B+-tree index on *ID* of the following table? (degree 3; i.e., at most 3 pointers)
Compare this with the hash index of the previous problem.

| *ID* | *name* | *dept_name* | *salary* |
|---|---|---|---|
| 10101 | Srinivasan | Comp. Sci. | 65000 |
| 12121 | Wu | Finance | 90000 |
| 15151 | Mozart | Music | 40000 |
| 22222 | Einstein | Physics | 95000 |
| 32343 | El Said | History | 60000 |
| 33456 | Gold | Physics | 87000 |
| 45565 | Katz | Comp. Sci. | 75000 |
| 58583 | Califieri | History | 62000 |
| 76543 | Singh | Finance | 80000 |
| 76766 | Crick | Biology | 72000 |
| 83821 | Brandt | Comp. Sci. | 92000 |
| 98345 | Kim | Elec. Eng. | 80000 |

10101 12121 - 15151 22222 - 32343 33456 - 45565 58583 - 76543 76766 - 83821 98345 가 leaf node 이고 그 위에는 12121 22222 - 33456 58583 - 76766 98345. 그 위에는 22222 58583 - 98345. Root 는 58583.

Height 는 4.

Search 할 때, hash 는 평균적으로 address 버켓+ 해당 버켓 해서 2 개인데 얘는 평균적으로 4 개의 block 을 access 해야 함. 개수가 많아질수록 차이가 커짐.

## Discussion 11-15

Does the order of record insertion affect the final state of a hash index …

A. … based on static hashing?
B. … based on expandable hashing?

A. static hashing 은 hash function 이 정해진 bucket address 의 집합에 일정하게 매핑하므로, insert 의 순서는 상관이 없을 것 같다.

B. expandable hashing 은 순서가 상관이 없을 것 같은데 이유를 모르겠음…

=〉 split 되는 정책이 일관성이 있다면, mapping 되는 address 는 순수하게 hash function 에 의해서 결정이 되는 것이고, split 되느냐 마느냐는 테이블의 record 의 분포에 따라 되는 것이기 때문에 **순서가 변해도 바뀌는 것은 없다.**

## Discussion 11-16

Consider the following table. Suppose there are many cases for searching with *lastname* and also with *lastname* and *firstname* combined. Can we use a single index to improve the performance of both types of search queries?

*student(ID, lastname, firstname, dept, year, address)*

lastname 으로 index 를 만들면 두 가지 모두 성능이 향상될 수 있다. 우선 lastname 단독은 자명함. 그리고 lastname + firstname 의 경우 lastname 으로 해당하는 record 들의 범위를 줄이고, 줄어든 범위에서만 firstname 을 기반으로 찾으면 되므로 성능이 향상된다.

=〉 B+ 트리 인덱스는 prefix 에 대한 index 를 support 할 수 있음. [lastname + firstname] 으로 index 를 만들면 prefix 에 대한 search 가 되기 때문에 lastname, lastname + firstname search 의 효율이 모두 향상된다.

Hash 의 경우는 다름. Hash function 은 [lastname + firstname] 에서 lastname 이 같다고 같은 맵핑이 되지가 않음.

실제로 DBA 들이 index 로 search performance 를 올리기 위해서 composite index 를 많이 사용함. 주로 search 가 되는 attribute 를 앞에 두고, specific 하게 search 할 때 이용되는 것들을 뒤에 놓음.