땅울림
자구 스터디

Made by 규정

10주차

1

hash

2

dictionary

3

dynamic programming

1. hash

1 hash

hash
(잘게 썰다)

· hashing : 데이터를 잘게 썰어서 작은 크기의 공간에 저장하는 것

1 hash

첫째줄에 N이 주어지고 둘째 줄부터 N개의 데이터 a가 주어진다.
그 다음 줄에 M이 주어지고 M개의 질의 b가 들어온다.
N개의 데이터를 입력받고 각 질의에 대해서,
주어진 숫자가 입력받은 데이터에 존재하는 숫자인지 검사해라.

1 <= N <= 100,000, 1 <= a, b <= 10,000, 1 <= M <= 100,000

예제 입력

```
5
8000
3853
1020
4140
3120
3
1020
2535
8001
```

예제 출력

```
YES
NO
NO
```

1 hash

· 일반적인 방식은 처음부터 끝까지 보면서 그 숫자가 존재하는지 검사해야함 => 한번 검사하는데 O(N)

0	1	2	3	4
8000	3853	1020	4140	3120

· b : {1020, 2535, 8001}

입력	O(N)
검사	O(N * M)

=> O(N * M)

1 hash

· 관찰)
a의 범위가 1~10000이다.
=〉 10000개짜리 배열을 만들자

· b : {1020, 2535, 8001}

arr[b]가 true인지만 보면 됨 =〉 한번 검사하는데 O(1)

입력	O(N)
검사	O(M)

=> O(max(N, M))

1 hash table

hash table : 공간을 팔아 시간을 사다

- 탐색 key값을 통해 배열에 직접 접근하자!
- 이론적으로는 O(1)의 시간에 탐색 가능

데이터의 범위가 커진다면..?

1 hash table

첫째줄에 N이 주어지고 둘째 줄부터 N개의 데이터 a가 주어진다.
그 다음 줄에 M이 주어지고 M개의 질의 b가 들어온다.
N개의 데이터를 입력받고 각 질의에 대해서,
주어진 숫자가 입력받은 데이터에 존재하는 숫자인지 검사해라.

1 <= N <= 100,000, 1 <= a, b <= 100,000,000, 1 <= M <= 100,000

예제 입력

```
5
99773423
3853
1020
41403421
3120
3
1020
2535
80017783
```

예제 출력

```
YES
NO
NO
```

1 hash table

데이터의 범위가 배열로 표현할 수 있는 크기를 넘어간다면 직접 참조가 불가능

=〉 데이터를 해시(잘게 부수고 다시 뭉쳐서)해서 테이블에 들어갈 수 있게 압축시키자

1 hash function

데이터를 해시함수를 통해 변환 => 변환된 값으로 테이블에 접근 (arr[4756])

1 hash function

hash function(해시 함수)

- 탐색키를 입력으로 받아 해시 주소를 리턴하고, 그 리턴한 주소가 해시 테이블의 인덱스가 된다.
- 문자열도 해시 함수를 통해 변환한 후 인덱스로 참조가 가능해진다.
- 해시 함수는 다양한 방법으로 구현 가능

1 hash function

나눗셈법

- hash(x) = x mod N;

```
int hashFunction(int x)
{
    return x % 65599;
}
```

- 테이블의 크기는 소수(prime number)로 잡는 것이 좋다.
- 특히 2의 제곱수와 가까운 소수가 좋다고함 ex)65599

1 hash function

MAD(Multiply, Add and Divide)

- hash(x) = (ax + b) mod N;

```
int hashFunction(int x)
{
    int a = 31;
    int b = 132;
    return (a*x + b) % 65599;
}
```

1 hash function

문자열 해시

- 각 문자를 숫자로 변환해서 해시

```
int hashFunction(string s)
{
    int x = 0;
    for (auto i : s)
        x += (int) (i - 'a');
    return x % 65599;
}
```

1 hash function

이외에도 제곱법, 폴딩법, 무작위법 등
수많은 해시 함수가 존재함

1 hash collision

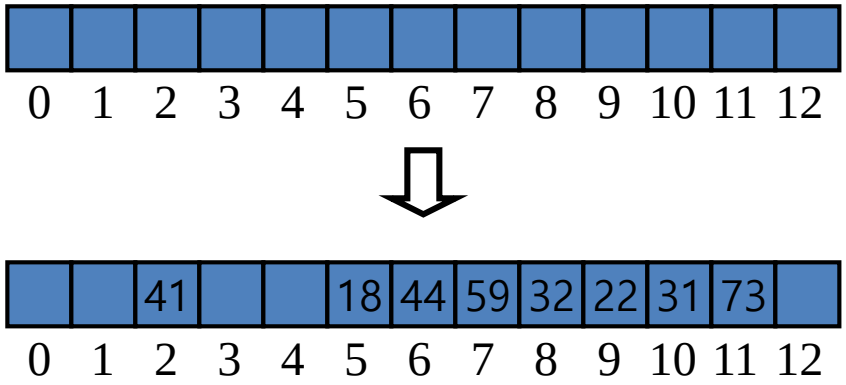
충돌이 일어날 수 있다!!

- 변환된 인덱스를 참조하는 값이 이미 있었다면 충돌 발생
- 충돌 : 서로 다른 두 개의 키 k1과 k2에 대해서, hash(k1) == hash(k2) 인 경우

1 hash collision

- 선형 탐사
  - 충돌이 발생하면 빈 칸을 찾을 때까지 뒤로 한칸씩 이동

- Example:
  - $h(x) = x \bmod 13$
  - keys : 18, 41, 22, 44, 59, 32, 31, 73



1 hash collision

- 제곱 탐사
  - 처음 충돌이 발생하면 1칸 뒤로 이동,
    또 발생하면 4칸, 또 발생하면 9칸, … 이런 식

1 hash collision

- 더블 해싱
  - 해시 함수를 두개 사용
  - 충돌이 발생하면 다른 해시 함수로 위치를 탐색
  - $h(k) = k \bmod N$
    $d(k) = q - k \bmod q$

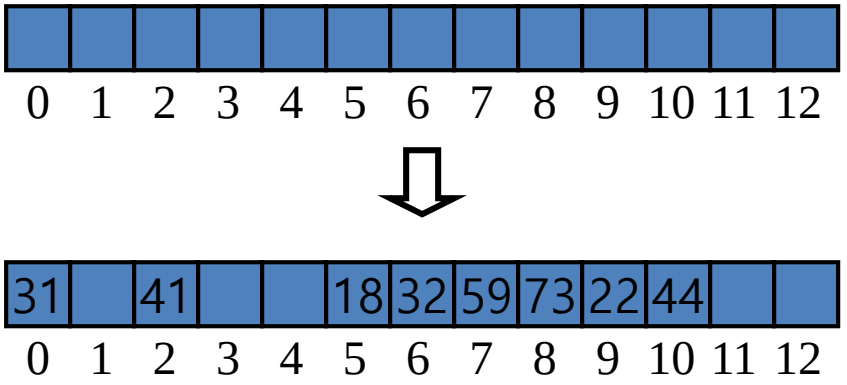
q는 N보다 작은 소수

1 hash collision

· 더블 해싱

N = 13
h(k) = k mod 13
d(k) = 7 − k mod 7

k	$h(k)$	$d(k)$	Probes		
18	5	3	5		
41	2	1	2		
22	9	6	9		
44	5	5	5	10	
59	7	4	7		
32	6	3	6		
31	5	4	5	9	0
73	8	4	8		

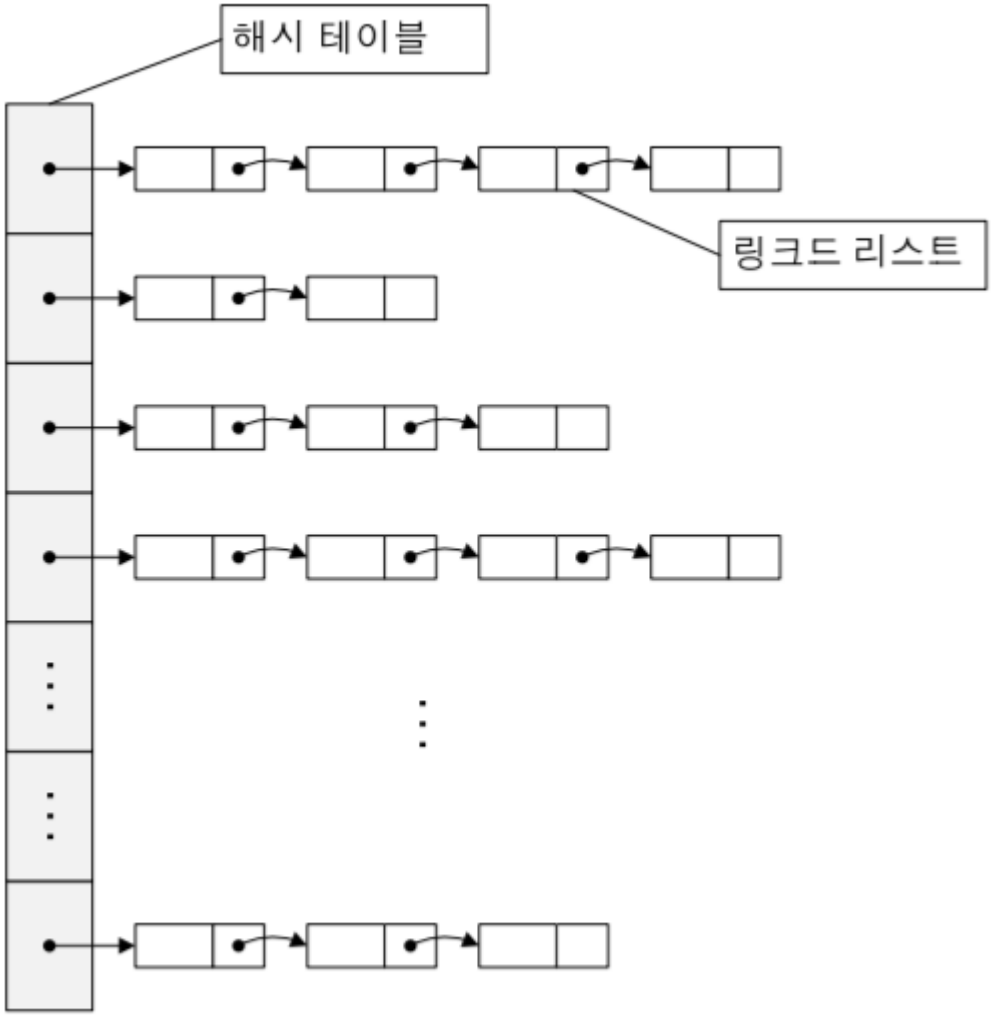


1 hash collision

- 체이닝
  - 해시 테이블을 연결리스트로 구현
  - 충돌이 발생하면 그 버킷에 연결되어있는 리스트를 탐색

1 hash collision

- 체이닝



2. dictionary

2 dictionary

dictionary

- 사전에서 단어를 찾듯이 데이터를 빠르게 찾기 위한 자료구조
- key와 element로 구성되어있음

list-based dictionary

- 배열이나 더블 링크드리스트로 구현
- put() : O(1) 맨 앞이나 맨 뒤에 집어넣으면 됨
  find(), erase() : O(n) key와 일치하는 걸 찾는데 O(n)

2 dictionary

search table

- 배열을 정렬시키면서 관리해서 빠르게 탐색하는 방법

- find() : O(log n) binary search 이용
  put(), erase() : O(n) 어디에 넣을지 찾기 위해 배열을 다 봐야함

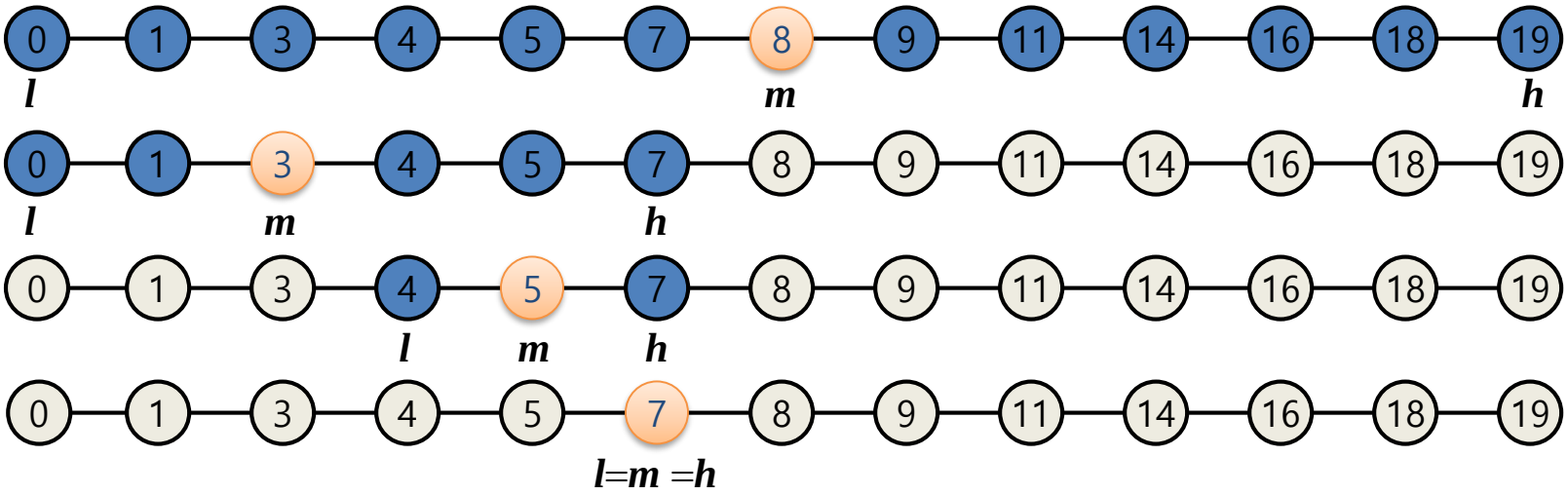
binary search

- 정렬된 배열에서 데이터를 빠르게 찾기 위한 방법
- 배열의 가운데 값과 key를 비교하고,
  key가 더 크다면 오른쪽을 보고
  key가 더 작다면 왼쪽을 본다.

2 dictionary

binary search

- find(7)



3. dynamic programming

3 dp

Dynamic Programming

- 복잡한 문제를 작은 부분 문제들로 나누고 작은 문제들을 먼저 해결해서 최종 문제를 해결하는 방법

- 작은 문제의 답을 저장하고 활용하는것이 중요

- 수학적으로 점화식을 세울 필요가 있음

3 dp

2747_피보나치 수

- 점화식 : d[n] = d[n−1] + d[n−2]
- 초기값 : d[0] = 0, d[1] = 1

3 dp

2747_피보나치 수

- 재귀

```cpp
#include <iostream>
using namespace std;
int fibonacci(int n) {
    if (n <= 1) {
        return n;
    } else {
        return fibonacci(n-1) + fibonacci(n-2);
    }
}
int main() {
    int n;
    cin >> n;
    cout << fibonacci(n) << '\n';
    return 0;
}
```

시간복잡도 : O(2^N)

3 dp

2747_피보나치 수

- 재귀
(메모이제이션)

```cpp
#include <iostream>
using namespace std;
int memo[50];
int fibonacci(int n) {
    if (n <= 1) {
        return n;
    } else if (memo[n] != 0) {
        return memo[n];
    } else {
        return memo[n] = fibonacci(n-1) + fibonacci(n-2);
    }
}
int main() {
    int n;
    cin >> n;
    cout << fibonacci(n) << '\n';
    return 0;
}
```

한번 구한 값은 다시 구하지 않음

시간복잡도 : O(N)

3 dp

2747_피보나치 수

- 반복문

```
#include <iostream>
using namespace std;
long long fibo[100] = {0,1};
int main() {
    int n;
    cin >> n;
    for (int i=2; i<=n; i++) {
        fibo[i] = fibo[i-1] + fibo[i-2];
    }
    cout << fibo[n] << '\n';
    return 0;
}
```

시간복잡도 : O(N)

3 dp

11726_2xn 타일링

- 점화식 : d[n] = d[n−1] + d[n−2]
- 초기값 : d[1] = 1, d[2] = 2

3 dp

11727_2xn 타일링 2

- 점화식 : d[n] = d[n−1] + d[n−2] * 2
- 초기값 : d[1] = 1, d[2] = 3

감사합니다.

Made by 규정