

Hash Table

2018년 5월 24일 목요일 오후 1:02

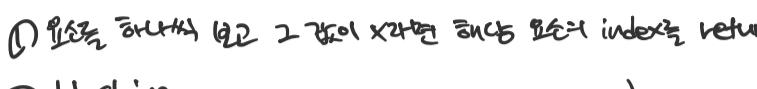
- 자료구조 히든어
- Sequential: list, stack, queue ... 순서가 있는 것들. 특히 linear하다.
- Hierarchy: tree, Priority queue, heap... 무게가 있는 것들.

특정 위치에
대해 CRUD 가능.
Search까지
쉽게呗?

- Searchable:
position을 찾을 수 있는 것들.



list는 Searchable 하기 만들기 쉽지.



Search x. (key)

① 원래 알고 그 값의 위치를 찾는 index를 return. $O(n)$

② Hashing

- x를 hash coding 해서 integer로 변환.
- 인덱스를 Compressing 한다.
- hashing은 $O(1)$ 이 걸리므로 전체 $O(1)$ 으로.

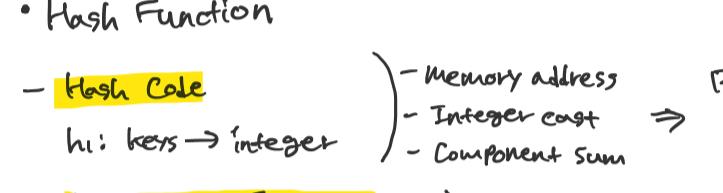
특정 위치에
대해 CRUD 가능.

Search까지
쉽게呗?

• Hash Table

- Hash Function
- Array (list)

특정 위치에
대해 CRUD 가능.



• Hash Function

- Hash Code
 $h_1: \text{key} \rightarrow \text{integer}$
- Memory address
 $h_1: \text{key} \rightarrow \text{integer}$
- Integer cast
 $h_1: \text{key} \rightarrow \text{integer}$
- Component sum
 $h_1: \text{key} \rightarrow \text{integer}$

Polynomial accumulation

• Compression Function

- $h_2: \text{integer} \rightarrow [0, N-1]$

같은 위치에 위치:

Multiple, Add and Divide (MAD): $h_2(y) = (ay+b) \bmod N$

$h_2(y) = (ay+b) \bmod BiPN \bmod N$

정수로

변환

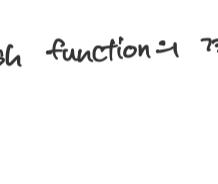
• Hash Function

$$h(x) = h_2(h_1(x))$$

• Collision Handling

- 서로 다른 key를 hashing 했는데 그 hash value가 똑같다면?

L Separate Chaining: 그룹은 자리에 리스트를 만들어 둘다 저장. (단, 메모리를 투자적으로 사용.)



L Linear Probing: 이미 자리에 값이 있다면 옆자리가 비어있는지 체크. (최근 메모리 공간을 사용하기 편리지만 점진적 허용률을.)

L Double Hashing: Collision이 일어난 Secondary hash function을 추가하여 해결 가능.

만들어 (i + jd(k)) mod N, $d_2(k) = q - k \bmod q$. Secondary hash function의 결과값 만큼 뒤에 있는 자리에 넣는다.

만들어 그 자리에도 있다면 ... Primary

$2[h(k) + d_2(k)]$ 만큼 뒤에 넣는다.

• Performance

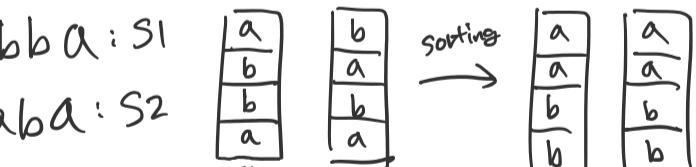
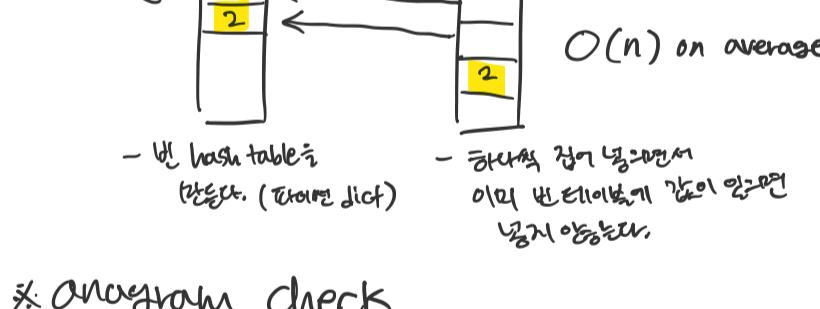
- Worst Case: $O(n)$ 모든 자리에서 Collision이 발생. 거의 일을 수 없는 일.

- 주의 학습적으로 생각: load factor = $\alpha = n/N$, 예상되는 Collision = $1/(1-\alpha)$

$\alpha \geq 100\%$ 가 아니라면 $O(1)$. (75%면 대도 $O(1)$ 이 나온다.)

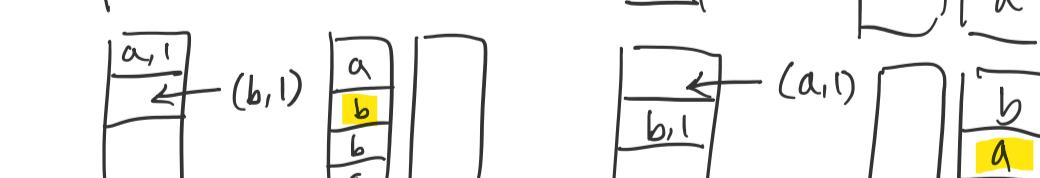
hash function을 잘 만들어서 Collision이 안 일어나게 하는 것의 중요.

*: 같은 elements 처리.



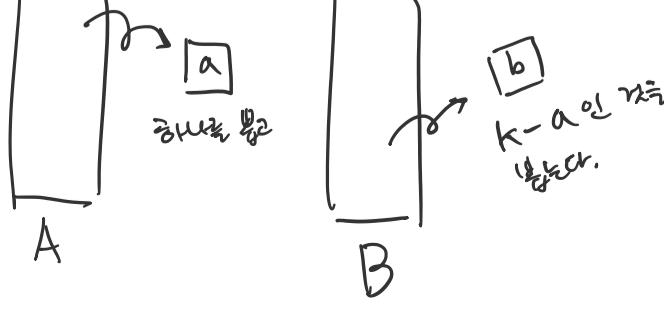
- H_1 hash table을
만들어 (파이썬 dict)
- 하나씩 차례 넘기면서
이미 번트리에 값이 있으면
넘지 않는다.

*: Anagram check



최종적으로 hash table의 모든 value가 0이라면 anagram.

*: A와 B에서 같은 빈도 향이 K가 되는 모든 조합 구하기.



*: 과제: 리스트 안에서 향이 K가 되도록 하는 모든 조합 (Pair) 찾기.