

Data Structure

자료구조



해싱



한국기술교육대학교
온라인평생교육원

학습내용

- 해싱의 개념
- 선형 조사법에 의한 오버플로 처리
- 체이닝에 의한 오버플로 처리

학습목표

- 해싱의 개념과 해시맵의 구조를 설명할 수 있다.
- 선형 조사법에 의한 오버플로 처리 방법을 설명할 수 있다.
- 체이닝에 의한 오버플로 처리 방법을 설명할 수 있다.



해싱의 개념

해싱

해싱의 개념

1 해싱이란?

◉ 해싱(Hashing)

예 아파트의 공동 우편함과 세대별 우편함



해싱

해싱의 개념

1 해싱이란?

해싱(Hashing)

▪ 고급 탐색 구조

- 탐색 연산의 시간 복잡도 $O(1)$
- 큰 테이블이 필요 → 해시 테이블

▪ **키값에 대한 산술적 연산**을 통해 해시 테이블에서의 엔트리가 있을 위치(주소)를 바로 계산하는 방법

- 키값을 비교하는 방식(순차 탐색, 이진 탐색 등)이 아님

해싱의 개념



해싱

해싱의 개념

1 해싱이란?

해시 테이블(Hash Table)

- 키값의 연산에 의해 계산된 주소를 직접 접근할 수 있어야 함
 - 어떤 주소의 엔트리를 $O(1)$ 에 접근(배열 구조)



해싱

해싱의 개념

1 해싱이란?

◎ 해싱의 구조

해시 테이블, 버킷, 슬롯

- 하나의 슬롯에 하나의 엔트리가 저장

해시 함수(Hash Function)

- 탐색 키를 입력 받아 **해시 주소(Hash Address)** 생성

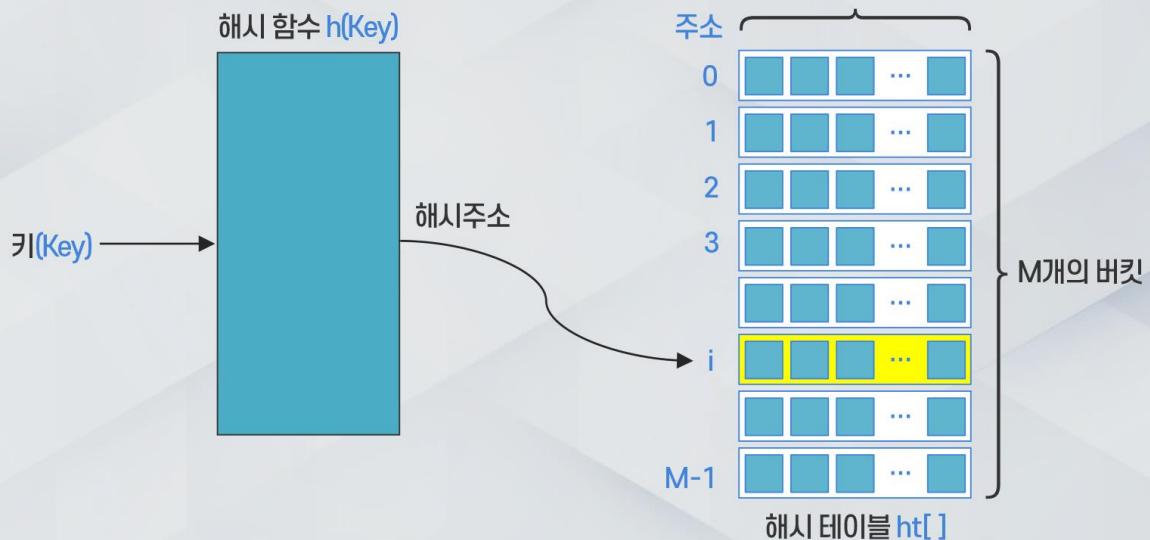


해싱의 개념

해싱 해싱의 개념

1 해싱이란?

○ 해싱의 구조



해싱 해싱의 개념

2 충돌과 오버플로

충돌

서로 다른 키가
해시 함수에 의해
같은 주소로 계산되는 상황

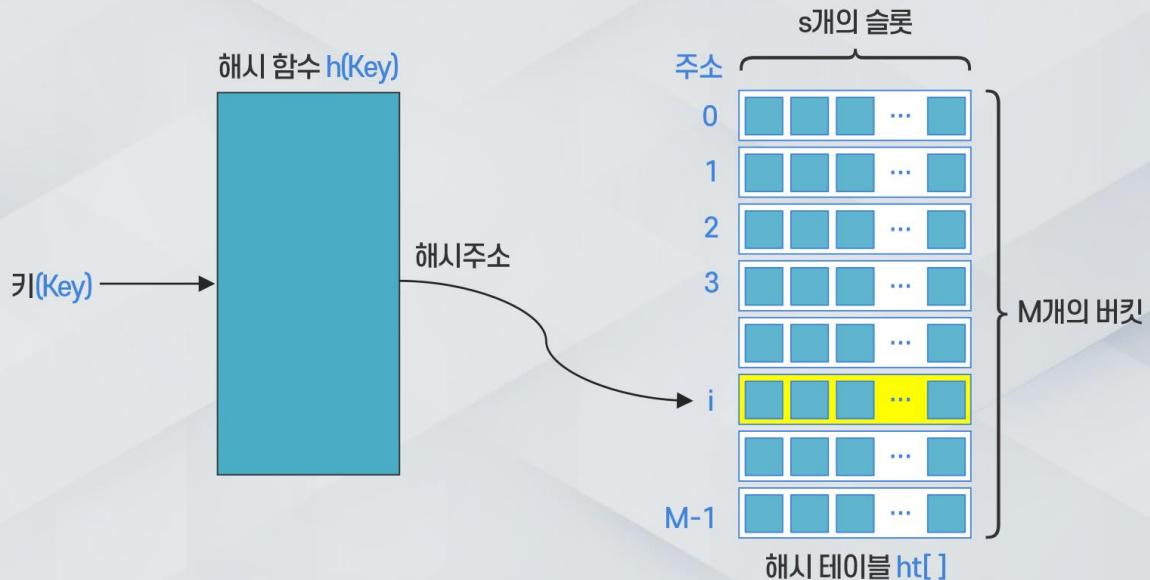


해싱의 개념

해싱

해싱의 개념

2 충돌과 오버플로



해싱

해싱의 개념

2 충돌과 오버플로

충돌

서로 다른 키가
해시 함수에 의해
같은 주소로 계산되는 상황

오버플로(Overflow)

충돌이 슬롯 수보다 많이
발생하는 것

예

$h(\text{홍길동}) \Rightarrow 3, h(\text{이순신}) \Rightarrow 2,$
 $h(\text{장영실}) \Rightarrow 5, h(\text{임꺽정}) \Rightarrow 3$

해싱의 개념

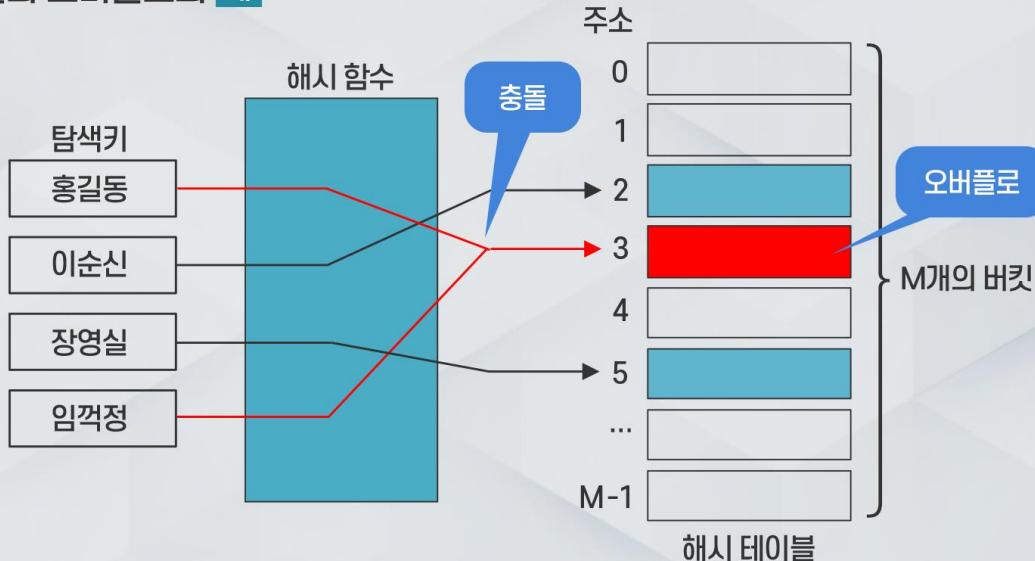


해싱

해싱의 개념

2 충돌과 오버플로

○ 충돌과 오버플로의 예



해싱

해싱의 개념

2 충돌과 오버플로

○ 이상적인 해싱과 실제의 해싱

이상적인 해싱
<ul style="list-style-type: none"> ▪ 오버플로가 일어나지 않는 경우 ▪ 충분히 테이블을 크게 하면 됨 → 메모리 문제 ▪ 탐색의 시간 복잡도: $O(1)$





해싱의 개념

해싱

해싱의 개념

2 충돌과 오버플로

● 이상적인 해싱과 실제의 해싱

실제의 해싱

- 테이블을 무한히 늘릴 수 없음
- 적절한 해시 함수를 사용해 주소 계산
- 충돌과 오버플로가 빈번하게 발생
- 오버플로 처리 방안이 제시되어야 함
 - 예 선형 조사법, 체이닝 등**
- 탐색의 시간 복잡도: $O(1)$ 보다는 좋지 않음

해싱

해싱의 개념

2 충돌과 오버플로

● 오버플로 처리방법

개방 주소법(Open Addressing)

오버플로가 일어나면 항목을
해시 테이블의 다른 위치에 저장

예

선형 조사법, 이차 조사법,
이중 해싱법, 임의 조사법 등

체이닝(Chaining)

해시 테이블의 하나의 위치에
여러 개의 항목을
저장할 수 있도록 하는 방법

해싱의 개념



해싱

해싱의 개념

3 해시 함수

● 좋은 해시 함수의 조건

1 충돌이 적어야 함

2 함수 값이 테이블의 주소 영역 내에서 고르게 분포되어야 함

3 계산이 빨라야 함



해싱

해싱의 개념

3 해시 함수

● 해시 함수의 종류

제산 함수

- $h(k) = k \bmod M$
- 해시 테이블의 크기 M 은 소수 (Prime Number) 선택

풀딩 함수

- 탐색키

123	203	241	112	20
-----	-----	-----	-----	----
- 이동풀딩
$$[123] + [203] + [241] + [112] + [20] = 699$$
- 경계풀딩
$$[123] + [302] + [241] + [211] + [20] = 897$$





해싱의 개념

해싱

해싱의 개념

3 해시 함수

○ 해시 함수의 종류

중간 제곱 함수	비트 추출 함수	숫자 분석 방법
탐색키를 제곱한 다음, 중간의 몇 비트를 취해서 해시 주소 생성	키를 이진수로 간주. 임의의 위치의 k개의 비트를 사용	키에서 편중되지 않는 수들을 테이블의 크기에 적합하게 조합

해싱

해싱의 개념

3 해시 함수

○ 해시 함수의 종류

▶ 탐색 키가 문자열인 경우

- 각 문자의 아스키 코드 값을 활용

```
def hashFn(self, key) :          # key가 문자열인 경우의 해시 함수
    sum = 0
    for c in key :               # 문자열의 모든 문자 c에 대해
        sum = sum + ord(c)       # 그 문자의 아스키 코드 값을 sum에 더함
    return sum % self.M          # 아스키 코드 합을 제산 함수로 처리
```



선형 조사법에 의한 오버플로 처리

해설

선형 조사법에 의한 오버플로 처리

1 선형 조사법

선형 조사법(Linear Probing)

- 충돌이 일어나면 해시 테이블의 다음 위치에서 비어 있는 버킷을 찾는 방법
- 빈 버킷을 찾는 위치: $h(k)+1, h(k)+2, \dots$
- 빈 곳이 있으면 저장

예 선형조사법의 예: $M=13, h(k)=k\%M$ 인 경우

키값이 45, 27, 88, 9, 71, 60, 46, 38, 24 저장 과정

해설

선형 조사법에 의한 오버플로 처리

1 선형 조사법

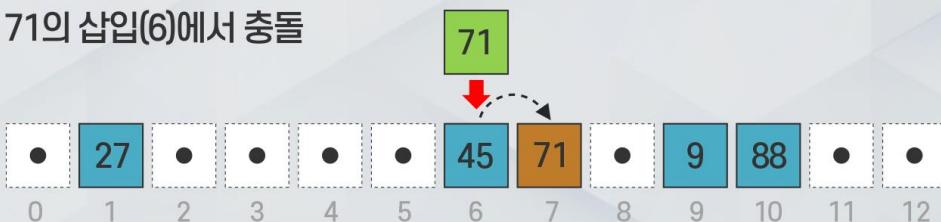
◎ 삽입 연산

key	45	27	88	9	71	60	46	38	24
$h(key)$	6	1	10	9	6	8	7	12	11

1 45, 27, 88, 9 까지의 삽입 → 문제 없음



2 71의 삽입(6)에서 충돌





선형 조사법에 의한 오버플로 처리

해설

선형 조사법에 의한 오버플로 처리

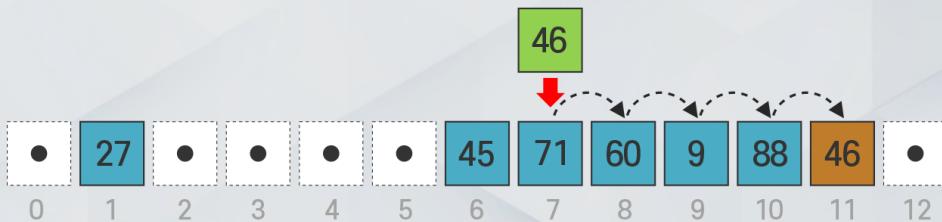
1 선형 조사법

● 삽입 연산

key	45	27	88	9	71	60	46	38	24
h(key)	6	1	10	9	6	8	7	12	11

3 60의 삽입(8)

4 46의 삽입(7)에서 다시 충돌



해설

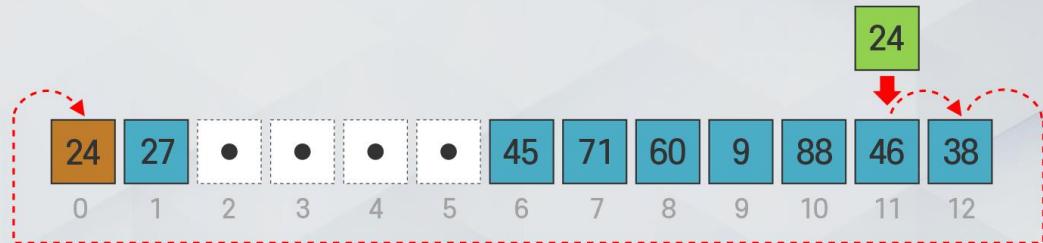
선형 조사법에 의한 오버플로 처리

1 선형 조사법

● 삽입 연산

key	45	27	88	9	71	60	46	38	24
h(key)	6	1	10	9	6	8	7	12	11

5 38은(12) 충돌 없이, 24는(11) 다시 충돌



선형 조사법에 의한 오버플로 처리



해설

선형 조사법에 의한 오버플로 처리

1 선형 조사법

● 삽입 연산



군집화 현상

한 번 충돌이 난 위치에서
계속 충돌이 발생

해설

선형 조사법에 의한 오버플로 처리

1 선형 조사법

● 탐색 연산

▶ 46 탐색(7)



▶ 39 탐색(0)



선형 조사법에 의한 오버플로 처리



해설

선형 조사법에 의한 오버플로 처리

1 선형 조사법

◎ 탐색 연산

탐색 종료 상황

- 레코드를 찾은 경우
- 레코드가 없는 버킷을 만나는 경우
- 모든 버킷을 다 검사한 경우



해설

선형 조사법에 의한 오버플로 처리

1 선형 조사법

◎ 삭제 연산

▶ 60을(8) 먼저 삭제한 후 46 삭제하는 (7) 경우 → 오류





선형 조사법에 의한 오버플로 처리

해설

선형 조사법에 의한 오버플로 처리

1 선형 조사법

● 삭제 연산

▶ 60을(8) 먼저 삭제한 후 46 삭제하는 (7) 경우 → 오류



▶ 빈 버킷을 두 가지로 분류해야 함

선형 조사법에 의한 오버플로 처리



```

1  # LinearProbMap.
2  class Entry:
3      def __init__( self, key, value ):
4          self.key = key
5          self.value = value
6
7      def __str__( self ):
8          return str("%s:%s"%(self.key, self.value) )
9
10
11  class LinearProbMap:
12      def __init__( self, M ):
13          self.table = [None]*M
14          self.M = M
15
16      def hashFn(self, key) :
17          return key % self.M
18
19      def insert(self, key, value) :
20          id = self.hashFn(key)
21          count = self.M
22          while count>0 and (self.table[id] != None and self.table[id] != -1) :

```

실습 단계

Entry 클래스는 이진탐색을 위한 맵과 같은 클래스

삽입 연산은 Key와 value를 레코드로 하는 것을 삽입

0은 원래 빈 버킷, -1은 한 번 사용되었던 버킷을 표현

None으로 삭제하는 것이 아니라 -1을 넣음 -> 기존에 값이 있다가 삭제를 의미

테스트

'x' 한번도 사용되지 않은 버킷, 'o' 이미 한 번 레코드가 있다가 삭제된 버킷



선형 조사법에 의한 오버플로 처리

해설

선형 조사법에 의한 오버플로 처리

2 군집화 완화 방법

이차 조사법(Quadratic Probing)

충돌이 발생하면, 바로 다음 위치가 아니라 제곱 이용

$$(h(k) + i * i) \% M \text{ for } i = 0, 1, \dots, M - 1$$

$\rightarrow +1, +4, +9, +16, \dots$

해설

선형 조사법에 의한 오버플로 처리

2 군집화 완화 방법

이중 해싱법(Double Hashing)

- 재해싱(Rehashing)
- 충돌이 발생하면, 다른 해시 함수를 이용해 다음 위치 계산





체이닝에 의한 오버플로 처리

해설

체이닝에 의한 오버플로 처리

1 체이닝에 의한 오버플로 처리



체이닝 (Chaining)

해시 테이블의 하나의 위치에
여러 개의 항목을 저장할 수 있도록 하는 방법

연결된 구조

연결 리스트

배열 구조

파이썬의 리스트

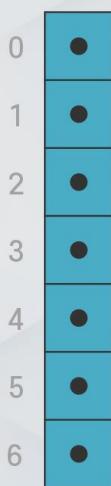
해설

체이닝에 의한 오버플로 처리

1 체이닝에 의한 오버플로 처리

● 체이닝 예

- ▶ 테이블의 크기 = 7
- ▶ 해시 함수: $h(k)=k \% 7$
- ▶ 삽입할 데이터: 8, 1, 9, 6, 13





체이닝에 의한 오버플로 처리

해설

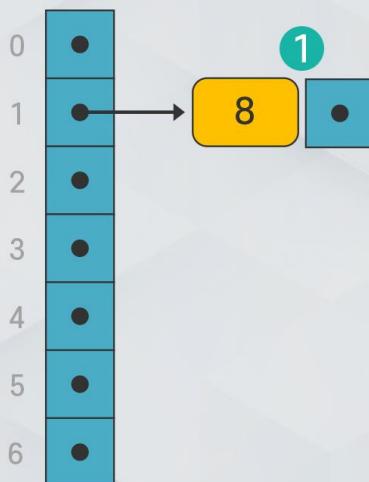
체이닝에 의한 오버플로 처리

1 체이닝에 의한 오버플로 처리

◎ 8 삽입

» $H(8) = 8 \% 7 = 1$

» 저장



해설

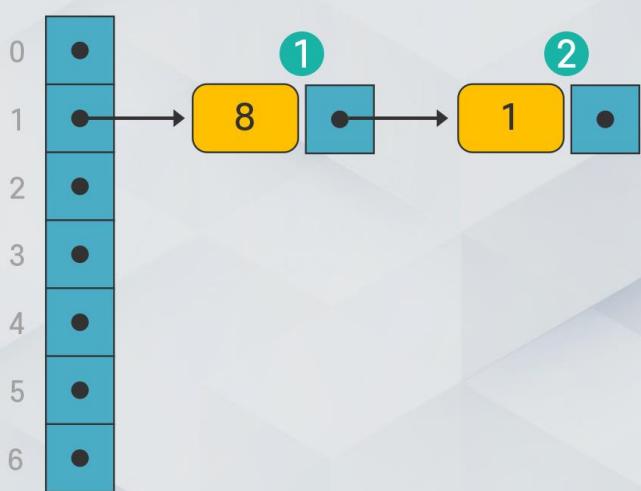
체이닝에 의한 오버플로 처리

1 체이닝에 의한 오버플로 처리

◎ 1 삽입

» $H(1) = 1 \% 7 = 1$

» 충돌: 새로운 노드 추가 저장





체이닝에 의한 오버플로 처리

해설

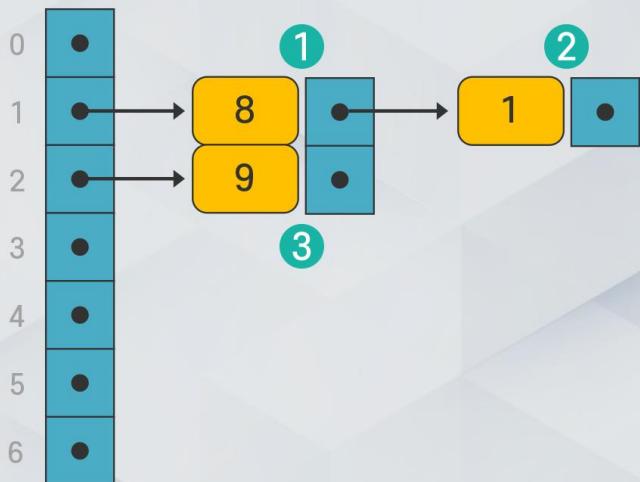
체이닝에 의한 오버플로 처리

1 체이닝에 의한 오버플로 처리

◎ 9 삽입

$$\rightarrow H(9) = 9 \% 7 = 2$$

→ 저장



해설

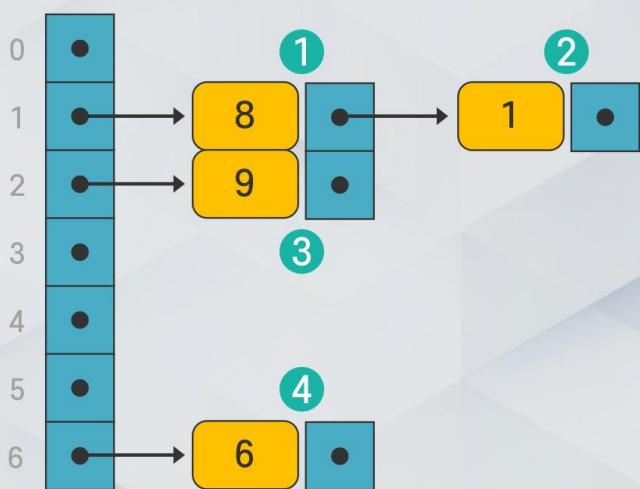
체이닝에 의한 오버플로 처리

1 체이닝에 의한 오버플로 처리

◎ 6 삽입

$$\rightarrow H(6) = 6 \% 7 = 6$$

→ 저장





체이닝에 의한 오버플로 처리

해설

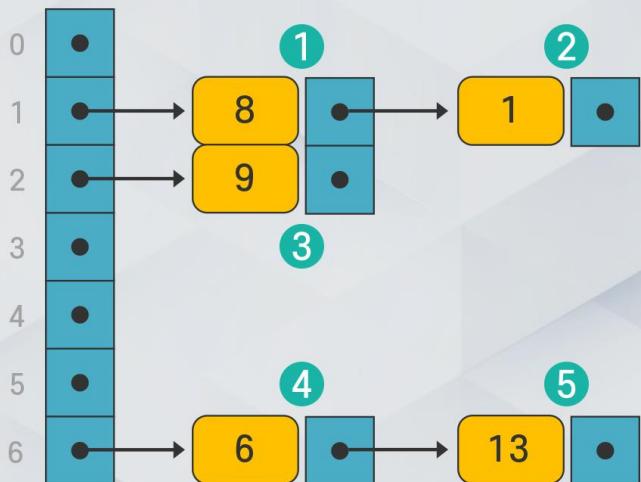
체이닝에 의한 오버플로 처리

1 체이닝에 의한 오버플로 처리

◎ 13 삽입

► $H(13) = 13 \% 7 = 6$

► 충돌: 새로운 노드 추가 저장



해설

체이닝에 의한 오버플로 처리

2 연결된 구조의 체이닝을 이용한 해시 맵

◎ “나의 단어장”을 위한 해시 맵 구현

- 엔트리: (영어 단어(Key) : 단어의 의미(Value))
- 연결된 구조로 체이닝을 구현

- Node 사용: 엔트리 + 링크
- 해시 테이블의 각 항목은 시작 노드를 가리킴



체이닝에 의한 오버플로 처리

```

64
65 =====
66 map = HashChainMap(13)
67 map.insert('data', '자료')
68 map.insert('structure', '구조')
69 map.insert('sequential search', '선형 탐색')
70 map.insert('game', '게임')
71 map.insert('binary search', '이진 탐색')
72 map.display("나의 단어장: ")
73
74 print(" 탐색:game --> ", map.search('game'))
75 print(" 탐색:over --> ", map.search('over'))
76 print(" 탐색:data --> ", map.search('data'))
77
78 map.delete('game')
79 map.display("나의 단어장: ")

```

실습 단계

입력 예시 설명

Entry

파이썬의 리스트로 구현

Key값을 이용해서 해시 주소(idx) 계산

새로운 노드를 연결리스트에 맨 앞에 연속적으로 추가하도록 구현

-> 현재 노드의 링크를 연결리스트의 head항목을 가리키게 함, 즉, 현재 노드가 head

엔트리만 반환, 링크는 반환할 필요 없음-> 연결된 구조를 나타내기 위함

삭제를 할 때는 그 이전 노드가 필요 -> 'before' 변수

레코드를 저장하지 않은 열은 출력에서 제외



체이닝에 의한 오버플로 처리

해싱

체이닝에 의한 오버플로 처리

2 연결된 구조의 체이닝을 이용한 해시 맵

● 해싱의 성능

▶ 해싱의 적재 밀도(Loading Density) 또는 적재 비율

저장되는 항목의 개수 n과 해시 테이블의 크기 M의 비율

$$\alpha = \frac{\text{저장된 항목의 개수}}{\text{해싱테이블의 버킷의 개수}} = \frac{n}{M}$$

해싱

체이닝에 의한 오버플로 처리

2 연결된 구조의 체이닝을 이용한 해시 맵

● 해싱의 성능

선형 조사법
0 ~ 1
<ul style="list-style-type: none"> ▪ 선형 조사법은 0.5 이하를 유지하는 것이 좋음 ▪ 이차 조사법과 이중 해싱은 0.7이하로 유지하는 것이 좋음

체이닝
0 ~ 무한대
<ul style="list-style-type: none"> ▪ 적재 밀도에 비례하는 성능을 보임 ▪ 항목의 수가 균등하게 분포되는 것이 유리



체이닝에 의한 오버플로 처리

해싱

체이닝에 의한 오버플로 처리

2 연결된 구조의 체이닝을 이용한 해시 맵

● 탐색 방법들의 성능 비교

탐색 방법		탐색	삽입	삭제
순차 탐색		$O(n)$	$O(1)$	$O(n)$
이진 탐색		$O(\log_2 n)$	$O(n)$	$O(n)$
이진 탐색 트리	균형 트리	$O(\log_2 n)$	$O(\log_2 n)$	$O(\log_2 n)$
	경사 트리	$O(n)$	$O(n)$	$O(n)$
해싱	최선의 경우	$O(1)$	$O(1)$	$O(1)$
	최악의 경우	$O(n)$	$O(n)$	$O(n)$