알고리즘

7장 해시테이블(hash table)

학습내용

1. 해시 테이블: 검색 효율의 극단
2. 해시 함수
3. 충돌 해결
4. 해시 테이블에서 검색시간 분석

학습목표

- 해시 테이블의 발생 동기를 이해한다.
- 해시 테이블의 원리를 이해한다.
- 해시 함수 설계 원리를 이해한다.
- 충돌 해결 방법들과 이들의 장단점을 이해한다.
- 해시 테이블의 검색 성능을 분석할 수 있도록 한다.

학습내용

1. **해시 테이블: 검색 효율의 극단**
2. 해시 함수
3. 충돌 해결
4. 해시 테이블에서 검색시간 분석

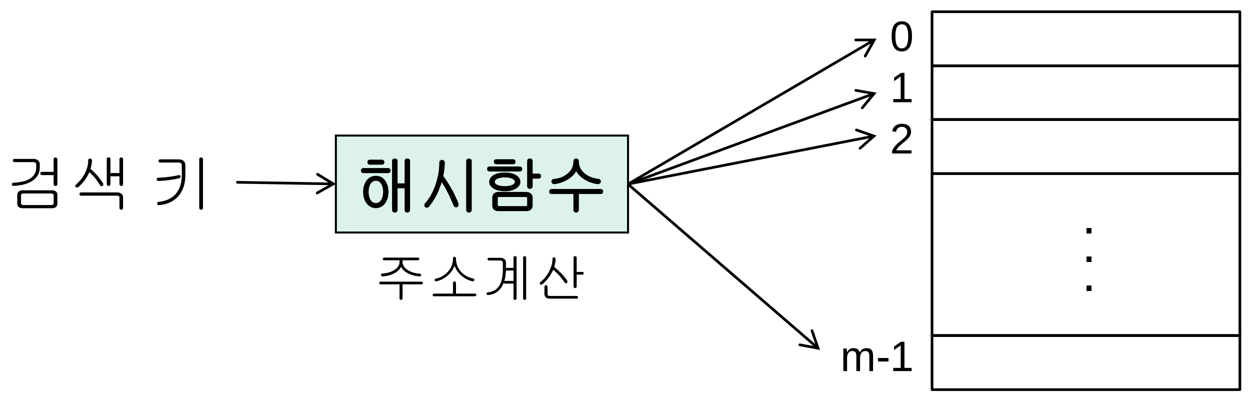
저장/검색의 복잡도

- 원소 하나를 저장하고 검색하는 작업의 시간 복잡도
  - Array
    - $O(n)$
  - Binary search tree
    - 평균 $\Theta(\log n)$, 최악의 경우 $\Theta(n)$
  - Balanced binary search tree(예: red-black tree)
    - 최악의 경우 $\Theta(\log n)$
  - B-tree
    - 최악의 경우 $\Theta(\log n)$
  - Hash table
    - 평균 $\Theta(1)$

검색 효율의 극단 :
저장된 자료의 양에 상관 없이 원소 하나를 저장/검색하는 데 상수 시간이 걸림

해시 테이블(Hash Table)

- 해시 테이블은 원소가 저장될 위치(주소)가 그 원소의 값(검색 키)에 의해 결정되는 자료구조
  - 원소가 저장될 위치는 단번에(상수시간에) 계산됨



배열 형식의 해시 테이블(크기 m)

- 참고) 이진검색트리
  - 다른 원소와 비교하여 해당 원소의 위치를 결정

해시 테이블의 특징

- 매우 빠른 응답을 요하는 응용에 유용하다.
  - 삽입, 삭제, 검색 : 평균 상수 시간. 즉, $\Theta(1)$
- 최소 원소 찾기와 같은 연산은 지원하지 않는다.
  - 참고) 이진검색트리에서 최소 원소 찾기는?

해시 테이블의 예

해시 테이블 크기 m = 13 이고,
해시 함수 $h(x) = x \bmod 13$ 인 경우,

25, 13, 16, 15, 7 를 입력하면

0	
1	
2	
3	
4	
5	
6	
7	
8	
9	
10	
11	
12	

해시 테이블의 예

해시 테이블 크기 m = 13 이고,
해시 함수 $h(x) = x \bmod 13$ 인 경우,

25, 13, 16, 15, 7 를 입력하면
➔오른쪽 그림과 같이 저장된다.

16을 검색하면

17을 검색하면

0	**13**
1	
2	**15**
3	**16**
4	
5	
6	
7	**7**
8	
9	
10	
11	
12	**25**

해시 테이블의 예

해시 테이블 크기 m = 13 이고,
해시 함수 $h(x) = x \bmod 13$ 인 경우,

25, 13, 16, 15, 7 를 입력하면
➔오른쪽 그림과 같이 저장된다.

28을 입력하면
➔15와 충돌이 일어난다. 어떻게 처리?

✓ 충돌은 해시 테이블 연산을 상수시간에 수행하는 데 있어서 주요 장애물임

0	**13**
1	
2	**15**
3	**16**
4	
5	
6	
7	**7**
8	
9	
10	
11	
12	**25**

해시 테이블

- 해시 테이블에서 결정해야 할 요소
  - 해시 함수(hash function)
  - 해시 테이블(hash table)의 크기
  - 충돌(collision) 해결 방법

학습내용

1. 해시 테이블: 검색 효율의 극단
2. **해시 함수**
3. 충돌 해결
4. 해시 테이블에서 검색시간 분석

해시 함수(Hash Function)

- 해시 함수는 검색 키 값을 해시 테이블 주소로 매핑하는 함수
- 해시 함수가 갖추어야 할 요건
  - 입력 원소가 해시 테이블 전체에 고루 흩어지도록 주소를 계산해야 한다. (→ 충돌을 줄이기 위한 조건으로서, 매우 중요한 요건임!)
  - 계산이 간단해야 한다.
- 해시 함수는 검색 키 값을 수치 데이터로 취급
  - 검색 키가 수치 데이터가 아닌 경우, 수치로 변환

해시 함수

- 해시 함수 중에서 대표적인 두 가지 방법을 알아보자.
  - (1) 나누기 방법(division method)
  - (2) 곱하기 방법(multiplication method)

(1) Division Method

- **$h(x) = x \bmod m$**
  - m : 해시 테이블 크기
  - 예) $x = 1025390$, $m = 1601$ 인 경우

    $1025390 \bmod 1601 = 750$ ➔ x는 주소 750에 해시됨
- m의 결정이 중요
  - **m은 2^p 에 가깝지 않은 소수(prime number)**가 바람직
  - $m = 2^p$ 이면 하위 p 비트에 의해 해시값이 결정되므로 원소 분산이 잘 이루어지지 않을 수 있음
    - 예) $m = 2^8$

      $h(x) = x \bmod 2^8$

      $h(\mathrm{BA}) = 65$

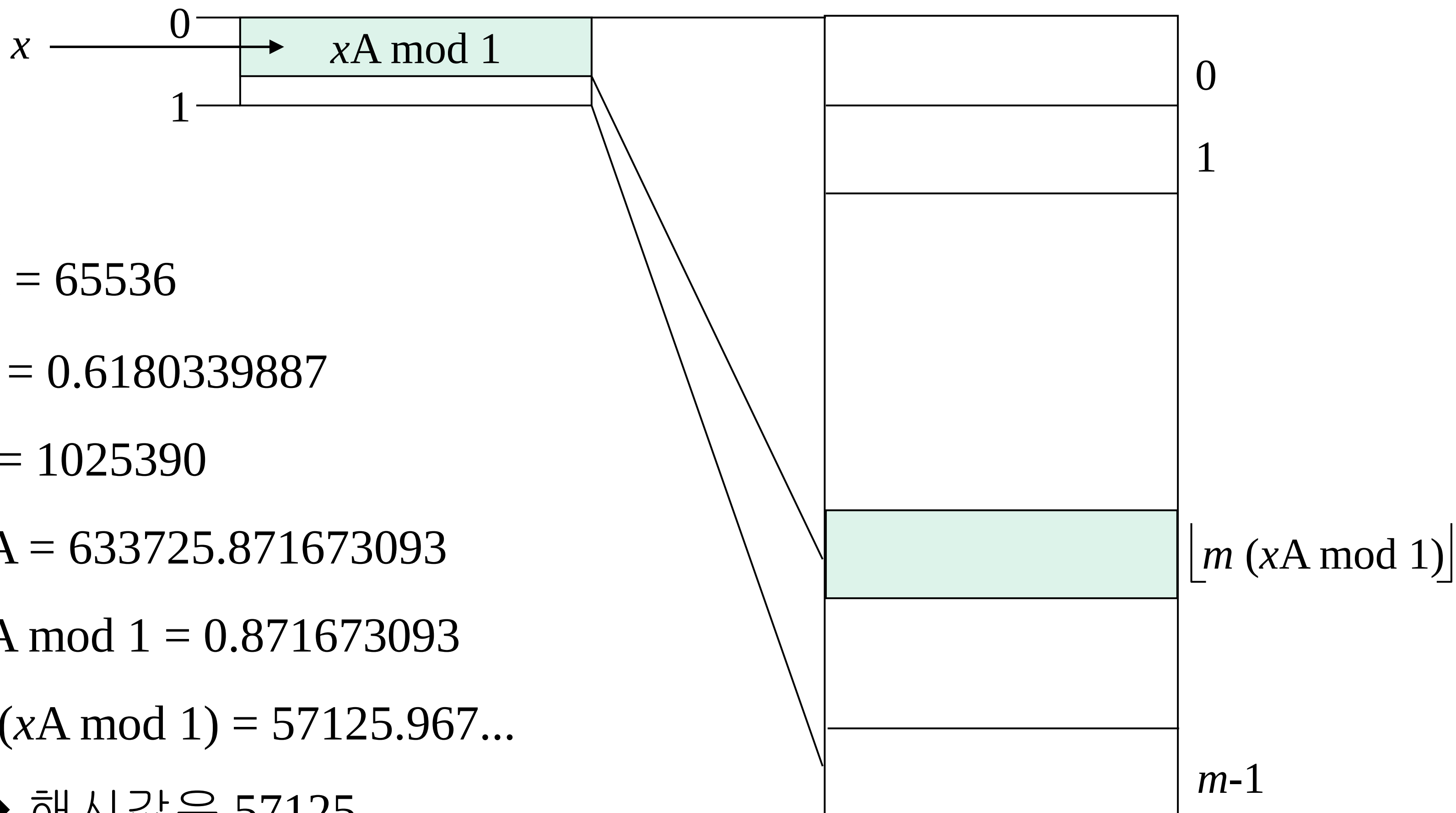
      $h(\mathrm{CA}) = 65$

BA	01000010	01000001
CA	01000011	01000001

(2) Multiplication Method

- **$h(x)$ = floor[$m \cdot (x$A mod 1)]**
  - m : 해시 테이블 크기
  - A : 0 < A < 1 인 상수
  - mod 1 은 어떤 값의 소수부만 취하는 연산
  - floor는 소수부를 버리는 연산
- m은 굳이 소수일 필요 없다.
  - 따라서 보통 2^p 로 잡는다. (p 는 정수)
- 대신, 상수 A 결정이 해시 값 분포에 영향
  - 크누스의 제안 A = $\frac{\sqrt{5}-1}{2}$ = 0.6180339887…

곱하기 방법의 작동 과정



예) m = 65536

A = 0.6180339887

x = 1025390

xA = 633725.871673093

xA mod 1 = 0.871673093

m(xA mod 1) = 57125.967...

➔ 해시값은 57125

학습내용

1. 해시 테이블: 검색 효율의 극단
2. 해시 함수
3. **충돌 해결**
4. 해시 테이블에서 검색시간 분석

충돌(Collision)

- 충돌이란 hash table의 한 주소를 놓고 두 개 이상의 원소가 자리를 다투는 것
  - Hashing을 해서 삽입하려 하니 이미 다른 원소가 자리를 차지하고 있는 상황
- 충돌이 자주 발생할수록 해시 테이블의 성능이 떨어짐
- 해시 함수가 다대일 함수이므로 충돌은 불가피한 현상임
  - 충돌이 적게 일어나도록 해시 함수를 결정하고,
  - 그래도 발생하는 충돌은 적절한 충돌 해결 방법을 이용하여 처리

Collision의 예

해시 테이블 크기 13
해시 함수 $h(x) = x \bmod 13$

입력 x : 25, 13, 16, 15, 7, **20**

$h(20) = 20 \bmod 13 = 7$

20을 주소 7에 삽입하려는데 이미 다른 원소가 차지하고 있다.
충돌(collision)!

0	**13**
1	
2	**15**
3	**16**
4	
5	
6	
7	**7**
8	
9	
10	
11	
12	**25**

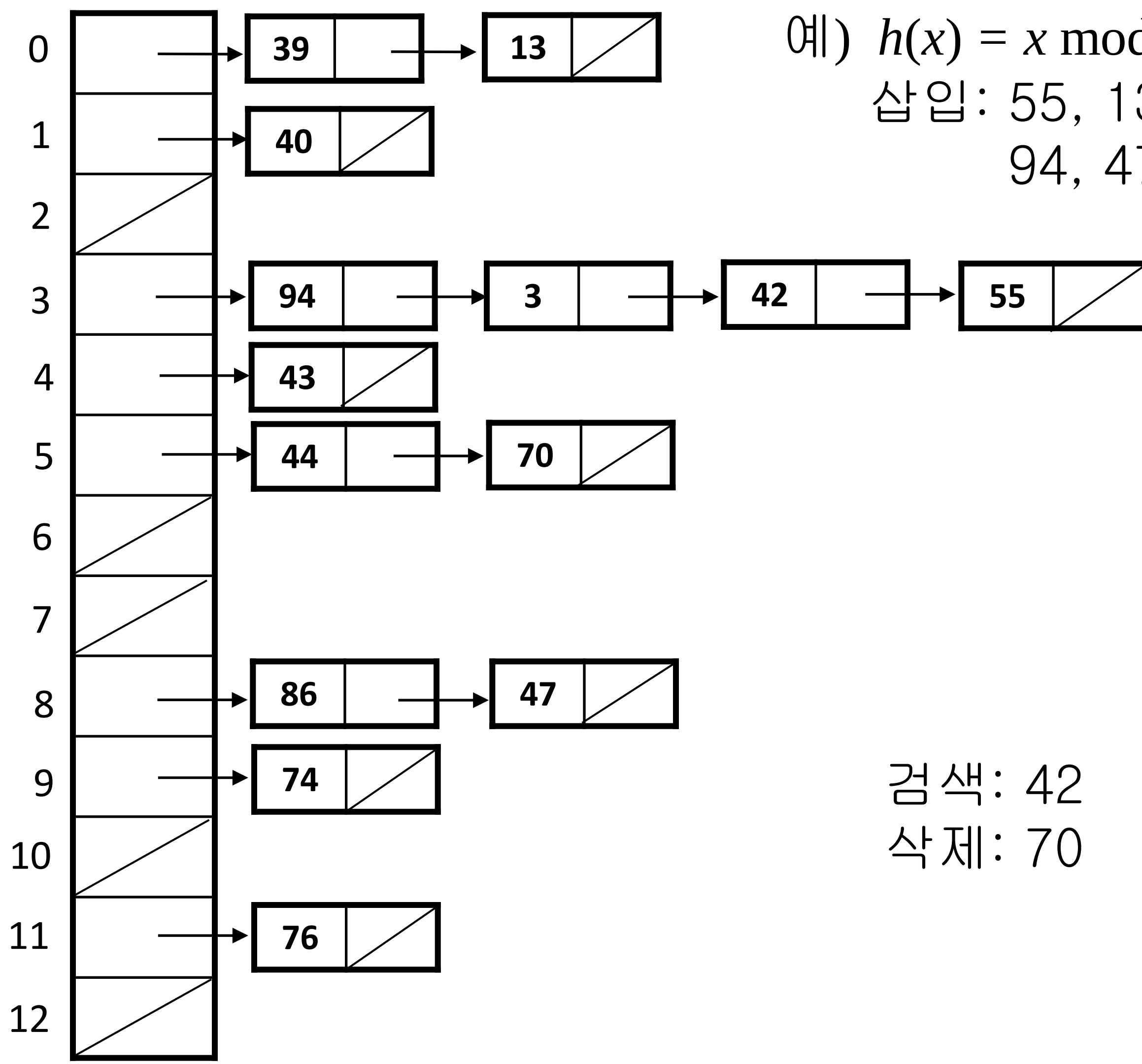
충돌 해결(Collision Resolution)

- 충돌 해결 방법은 크게 체이닝과 개방주소 방법으로 나뉨
- 체이닝(chaining)
  - **하나의 주소로 해싱되는 원소들을 하나의 linked list로 관리**
  - 크기 m인 해시 테이블에 m개 이상의 원소 저장 가능
    (즉, 적재율이 1을 넘어도 사용할 수 있다)
  - 추가적인 연결 리스트 공간이 필요
- 개방주소 방법(open addressing)
  - **충돌이 일어나더라도 어떻게든 주어진 테이블 공간에서 해결**
  - 추가적인 공간 불필요
  - 모든 원소가 자신의 해시값 주소에 저장된다는 보장은 하지 못함
  - 삭제가 잦은 경우 비효율적
  - linear probing, quadratic probing, double hashing

학습내용

1. 해시 테이블: 검색 효율의 극단
2. 해시 함수
3. 충돌 해결
   - **체이닝 방법**
   - 개방주소 방법
4. 해시 테이블에서 검색시간 분석

충돌 해결 – Chaining



예) $h(x) = x \bmod 13$

삽입: 55, 13, 42, 70, 43, 44, 3, 94, 47, 74, 39, 86, 76, 40

검색: 42

삭제: 70

삽입/검색/삭제 알고리즘(체이닝 사용시)

- 삽입

```
chainedHashInsert(T, x)     ▷ T: 해시테이블, x: 삽입원소
{
    리스트 T[h(x)]의 맨 앞에 x를 삽입
}
```

- 검색

```
chainedHashSearch(T, x)     ▷ T: 해시테이블, x: 검색원소
{
    리스트 T[h(x)]에서 x 값을 가지는 원소를 검색
}
```

- 삭제

```
chainedHashDelete(T, x)     ▷ T: 해시테이블, x: 삭제원소
{
    리스트 T[h(x)]에서 x의 노드를 삭제
}
```

학습내용

1. 해시 테이블: 검색 효율의 극단
2. 해시 함수
3. 충돌 해결
   - 체이닝 방법
   - **개방주소 방법**
4. 해시 테이블에서 검색시간 분석

충돌 해결 – Open Addressing

- 충돌이 일어나면 빈 자리를 만날 때까지 다음 주소(해시값)를 계속 만들어 낸다.
  - 하나의 x에 대해 $h_0(x), h_1(x), h_2(x), h_3(x), \ldots, h_{m-1}(x)$
  - $h_0(x)$는 $h(x)$임
- 세가지 open addressing 방법

  (1) 선형 조사(linear probing)

  (2) 이차원 조사(quadratic probing)

  (3) 더블 해싱(double hashing)

(1) Linear Probing

- open addressing 중 가장 간단
- 충돌이 일어난 바로 뒷자리를 조사(probe)함
  - $h(x)$, $h(x)+1$, $h(x)+2$, $h(x)+3$, ... ➔ ***i*** 번째 해시 함수는 $\boldsymbol{h(x)}$로부터 ***i*** 만큼 떨어진 자리
- ***i*** 번째 해시 함수의 모양

$$h_i(x) = (h(x) + i) \bmod m, \qquad i = 0, 1, 2, \ldots$$

  - 충돌이 일어나면 1씩 점프함

i 번째 해시 함수 $h_i(x)$란? :
제일 먼저 0번째 해시 함수 $h_0(x)$를 사용하여 충돌이 일어나면 1번째 해시 함수 $h_1(x)$를 사용하고, 그래도 충돌이 일어나면 2번째 해시 함수 $h_2(x)$를 사용하고, ...

Linear Probing 예

$h(x) = x \bmod 13$

삽입: 25,13,16,15,7, 28,31,20,1,38

0	13
1	
2	15
3	16
4	28
5	
6	
7	7
8	
9	
10	
11	
12	25

0	13
1	
2	15
3	16
4	28
5	31
6	
7	7
8	20
9	
10	
11	
12	25

0	13
1	1
2	15
3	16
4	28
5	31
6	38
7	7
8	20
9	
10	
11	
12	25

$h_i(x) = (h(x) + i) \bmod 13$

다시 말해

$h_0(x) = (h(x) + 0) \bmod 13$

$h_1(x) = (h(x) + 1) \bmod 13$

$h_2(x) = (h(x) + 2) \bmod 13$

$h_3(x) = (h(x) + 3) \bmod 13$

...

- Linear probing은 1차 군집(primary clustering) 문제 발생 가능
  - 1차 군집: 특정 영역에 원소가 몰리기 시작하면 점점 더 몰려 더 큰 군집을 형성하여 성능이 치명적으로 떨어짐

$$h(x) = x \bmod 13$$

입력: 37, 15, 16, 28, 31, 44

0	
1	
2	**15**
3	**16**
4	**28**
5	**31**
6	**44**
7	
8	
9	
10	
11	**37**
12	

29가 입력되어 3에 해시되면?

이곳에 원소가 해시될 확률은?

이곳에 원소가 해시될 확률은?

(2) Quadratic Probing

- 충돌이 일어난 바로 뒷자리를 조사하지 않고, 보폭을 이차함수에 의해 넓혀가며 조사함
  - 예를 들어 $h(x), h(x)+1, h(x)+4, h(x)+9, \ldots$ ➔ $\boldsymbol{i}$ 번째 해시 함수는 $\boldsymbol{h(x)}$로부터 $\boldsymbol{i^2}$ 만큼 떨어진 자리로 점프함
  - 특정 영역에 원소가 몰리더라도 그 영역을 빨리 벗어날 수 있음
- $\boldsymbol{i}$ 번째 해시 함수의 모양

$$\boldsymbol{h_i(x) = (h(x) + c_1 i^2 + c_2 i) \bmod m, \qquad i = 0, 1, 2, \ldots}$$

  - $c_1 \neq 0$
  - 위의 예에서는 $c_1 = 1,\ c_2 = 0$

Quadratic Probing 예

$h(x) = x \bmod 13$

삽입: 15, 18, 43, 37, 45, 30

0	
1	
2	15
3	
4	43
5	18
6	45
7	
8	30
9	
10	
11	37
12	

$\boldsymbol{h_i(x) = (h(x) + c_1 i^2 + c_2 i) \bmod 13}$

에서 $c_1 = 1$, $c_2 = 0$ 인 경우를 예로 들면

$h_i(x) = (h(x) + i^2) \bmod 13$

다시 말해

$h_0(x) = (h(x) + 0) \bmod 13$

$h_1(x) = (h(x) + 1) \bmod 13$

$h_2(x) = (h(x) + 4) \bmod 13$

$h_3(x) = (h(x) + 9) \bmod 13$

…

- Quadratic probing은 2차 군집(secondary clustering) 문제 발생 가능
  - 2차 군집: 여러 원소가 초기 해시 함수값이 동일하여 한 번 충돌이 일어나면 그 이후로도 모두 같은 곳을 같은 순서로 조사하게 됨

$$h(x) = x \bmod 13$$

15, 28, 41, 67, 54 는 초기에 모두 2로 해시됨

0	
1	
2	15
3	28
4	
5	54
6	41
7	
8	21
9	
10	
11	67
12	

$$h_0(15) = h_0(28) = h_0(41) = h_0(67) = h_0(54) = 2$$

$$h_1(28) = h_1(41) = h_1(67) = h_1(54) = 3$$

$$h_4(54) = 5$$

$$h_2(41) = h_2(67) = h_2(54) = 6$$

$$h_3(67) = h_3(54) = 11$$

- Quadratic probing은 2차 군집(secondary clustering) 문제 발생 가능
  - 2차 군집: 여러 원소가 초기 해시 함수값이 동일하여 한 번 충돌이 일어나면 그 이후로도 모두 같은 곳을 같은 순서로 조사하게 됨

0	13
1	
2	15
3	28
4	
5	54
6	41
7	7
8	21
9	
10	10
11	67
12	

➡ 더블 해싱

(3) Double Hashing

- 두 개의 함수를 사용
- ***i*** 번째 해시 함수의 모양

$$\mathbf{h_i(x) = (h(x) + i \cdot f(x)) \bmod m}, \quad \mathbf{i = 0, 1, 2, \ldots}$$

  – $h(x)$ 와 $f(x)$는 서로 다른 함수
  – 충돌이 일어나면 $f(x)$만큼씩 점프함
    - 두 원소의 첫번째 해시값이 같더라도 두번째 함수값까지도 같을 확률은 매우 작으므로 대부분 서로 다른 보폭으로 점프
  – 권장하는 방법
    - $h(x) = x \bmod m$으로 잡고,
    - m보다 조금 작은 소수 m'에 대해 $f(x) = 1+(x \bmod m')$ 로 잡음

Double Hashing 예

$h(x) = x \bmod 13$ $f(x) = 1+(x \bmod 11)$

삽입: 15, 19, 28, 41, 67

0	
1	
2	15
3	
4	67
5	
6	19
7	
8	
9	28
10	
11	41
12	

$h_0(15) = h_0(28) = h_0(41) = h_0(67) = 2$

$h_1(67) = (2+2) \bmod 13 = 4$

$h_1(28) = (2+7) \bmod 13 = 9$

$h_1(41) = (2+9) \bmod 13 = 11$

$h_i(x) = (h(x)+i \cdot f(x)) \bmod 13$

다시 말해

$h_0(x) = (h(x) + 0 \cdot f(x)) \bmod 13$

$h_1(x) = (h(x) + 1 \cdot f(x)) \bmod 13$

$h_2(x) = (h(x) + 2 \cdot f(x)) \bmod 13$

$h_3(x) = (h(x) + 3 \cdot f(x)) \bmod 13$

...

삽입 알고리즘(개방주소방법 사용시)

```
hashInsert(T[], x)      ▷ 해시테이블 T에 원소 x를 삽입하고
                          x가 저장된 위치를 리턴
{
    i ← 0;
    repeat {
        j ← h_i(x);
        if (T[j] = NIL)
            then { T[j] ← x;  return  j ; }
            else i++;
    } until (i = m);        ▷ m 은 해시테이블 크기

    error “테이블 오버플로우”;
}
```

검색 알고리즘(개방주소방법 사용시)

```
hashSearch(T[], x)      ▷ 해시테이블 T에서 원소 x를 찾아
                          x가 저장된 위치를 리턴
{
    i ← 0;
    repeat {
        j ← h_i(x);
        if (T[j] = x)
            then return j ;
            else i++;
    } until (T[j] = NIL or i = m);   ▷ m 은 해시테이블 크기

    return NIL;
}
```

삭제시 주의할 점(개방주소방법 사용시)

예) linear probing의 경우
$h(x) = x \bmod 13$

0	13
1	1
2	15
3	16
4	28
5	31
6	38
7	7
8	20
9	
10	
11	
12	25

원소 1을
그냥 삭제

0	13
1	
2	15
3	16
4	28
5	31
6	38
7	7
8	20
9	
10	
11	
12	25

잘못된
삭제 방법임!

차후 38 검색시 문제발생

삭제 방법(개방주소방법 사용시)

- 자료를 삭제한 자리에 **DELETED**라는 상수 값을 저장해 둠
  - 원래 자료가 있던 자리로 인식하므로 충돌로 밀려나 다른 곳에 저장된 자료를 찾아가는 데 지장 없다.
  - 실제로는 비어있으므로 새로운 원소를 저장할 수 있다.
  - 단, 삭제가 빈번한 경우 효율이 떨어질 수 있다.

올바른 삭제 예(개방주소방법 사용시)

예) linear probing의 경우
$h(x) = x \bmod 13$

0	13
1	1
2	15
3	16
4	28
5	31
6	38
7	7
8	20
9	
10	
11	
12	25

원소 1 삭제시
DELETED 표시
를 해둠

0	13
1	DELETED
2	15
3	16
4	28
5	31
6	38
7	7
8	20
9	
10	
11	
12	25

차후 38 검색시 성공

자료 삭제 예(개방주소방법 사용시)

linear probing의 경우; $h(x) = x \bmod 13$
26 검색, 1 삭제, 38 검색, 14 삽입

0	13
1	1
2	15
3	16
4	28
5	31
6	38
7	7
8	20
9	
10	
11	
12	25

학습내용

1. 해시 테이블: 검색 효율의 극단
2. 해시 함수
3. 충돌 해결
4. **해시 테이블에서 검색시간 분석**

해시 테이블의 검색 시간 분석

- 적재율(load factor) α
  - Hash table 전체에서 얼마나 원소가 차 있는지를 나타내는 수치
  - 크기가 m인 Hash table에 n개의 원소가 저장되어 있다면 적재율 $\alpha = n/m$
- 적재율의 범위
  - open addressing의 경우 $\alpha \leq 1$
  - chaining의 경우는 α 값 제한 없음
- Hash table에서의 검색 효율은 load factor와 밀접한 관련이 있다.

Chaining 사용시 검색 시간

- 정리 1
  - Chaining을 이용하는 hashing에서 load factor가 α 일 때, 실패하는 검색에서 조사 횟수의 기대치는 α 이다.
- 정리 2
  - Chaining을 이용하는 hashing에서 load factor가 α 일 때, 성공하는 검색에서 조사 횟수의 기대치는 $1 + \alpha/2 - \alpha/2n$ 이다.

Open Addressing 사용시 검색 시간

- 가정
  - 조사 순서 $h_0(x), h_1(x), \ldots, h_{m\text{-}1}(x)$가 0부터 m-1 사이의 수로 이루어진 순열을 이루고, 모든 순열은 같은 확률로 일어난다. (조사 순서가 임의의 순서이다.)
- 정리 3
  - open addressing hashing에서 load factor가 α일 때 (α <1), 실패하는 검색에서 조사 횟수의 기대치는 최대 $1/(1- \alpha)$ 이다.
- 정리 4
  - open addressing hashing에서 load factor가 α일 때 (α <1), 성공하는 검색에서 조사 횟수의 기대치는 최대 $(1/ \alpha)\log(1/(1- \alpha))$ 이다.

체이닝과 개방주소방법의 성능

- 앞의 정리들로 보아 이론적으로 개방주소방법보다 체이닝이 평균 조사 횟수가 적음
  - 체이닝: 충돌을 일으키지 않은 원소끼리는 서로 검색을 방해하지 않음
  - 개방주소 방법: 직접 충돌을 일으키지 않은 원소라도 검색을 방해할 수 있음
- 적재율이 그리 높지 않을 때(예를 들어 0.5 이하)는 개방주소방법이 나은 경우가 많음
  - 체이닝은 연결리스트마다 헤드를 두어야 하고 저장되는 원소마다 연결을 위한 공간 필요

적재율이 지나치게 높아지는 경우

- 적재율(load factor) α가 높아지면 일반적으로 hash table의 효율이 떨어진다.
- 적절하게 낮은 적재율을 유지해야 한다.
  - 일반적으로, 적재율의 임계값(threshold, 예를 들어 0.75)을 미리 설정해 둔다.
  - 적재율이 임계값에 이르면 해시테이블의 크기를 두 배로 늘인 다음 기존 해시테이블에 저장되어 있는 모든 원소를 다시 hashing하여 새 테이블에 저장함으로써 임계값 이하의 적재율을 유지한다.

요약

- 해시 테이블은 검색 효율의 극단을 추구하는 자료구조
  - 저장된 원소의 양에 관계 없이 상수 시간 검색이 가능
- 해시 함수는 원소들을 해시 테이블에 고루 분산시켜야 함
- 충돌(collision)은 동일한 주소에 두개 이상의 원소가 해싱되는 것으로서, 충돌을 해결하는 방법은 해시 테이블에서 가장 중요한 문제
  - 체이닝: 충돌이 일어난 원소들을 하나의 연결 리스트로 관리하는 방법
  - 개방 주소 방법: 추가적인 공간을 사용하지 않고 해시 테이블 내에서 해결하는 방법

문제

★ m은 해시 테이블의 크기이며, n은 해시 테이블에 저장된 원소 수이다.

★ 문제에서 사용한 수치는 문제를 간단하게 하기 위해 지정한 것이므로 바람직한 수치가 아닐 수 있다.

1. 해시 함수에 관한 설명으로 올바른 것을 모두 고르세요.

   (1) 해시 함수는 키값을 해시 테이블 주소(인덱스)로 변환한다.

   (2) 키값이 다르면 해시 함수 값이 다르다.

   (3) 해시 함수로 나누기 방법을 사용하는 경우 해시 테이블의 크기를 보통 2^p 로 한다.

   (4) 해시 함수로 곱하기 방법을 사용하는 경우 해시 테이블의 크기를 보통 2^p 로 한다.

   (5) 해시 함수를 고를 때, 키 값의 특성을 살려서 비슷한 키 값들이 비슷한 위치에 저장되도록 하는 해시 함수를 고르는 것이 좋다.

2. 해시 테이블 성능에 관한 설명으로 올바른 것을 모두 고르세요.

(1) 해시 테이블의 삽입 시간은 평균 O(1)이다.

(2) 해시 테이블의 삭제 시간은 평균 O(1)이다.

(3) 해시 테이블의 검색 시간은 평균 O(1)이다.

(4) 해시 테이블의 최소값 검색 시간은 평균 O(n)이다.

(5) 충돌 해결을 위해 open addressing 중 linear probing을 이용하는 경우 1차 군집 문제가 발생할 수 있다.

(6) 충돌 해결을 위해 open addressing 중 double hashing을 이용하는 경우 2차 군집 문제가 발생할 수 있다.

3. 해시 테이블의 적재율에 관한 설명으로 올바른 것을 모두 고르세요.

(1) 크기가 100인 해시 테이블에 30개의 키 값이 저장된 경우 load factor는 0.3이다.

(2) 충돌 해결을 위해 개방주소 방법을 이용하는 경우 load factor가 1보다 큰 값일 수 있다.

(3) 충돌 해결을 위해 체이닝 방법을 이용하는 경우 load factor가 1보다 큰 값일 수 있다.

(4) 해시 테이블의 load factor가 클 수록 해시 테이블의 성능이 향상된다.

4. 다음과 같은 비어있는 해시 테이블에 아래 연산을 차례대로 모두 수행한 후 최종 해시 테이블 상태를 그림으로 나타내시오.

- 해시 함수: 나누기 방법 $h(x) = x \bmod m$
- 충돌해결 방법: open addressing 중에서 <u>linear probing</u>

연산: 10 삽입
12 삽입
14 삽입
20 삽입
30 삽입

답:

0	
1	
2	
3	
4	
5	
6	
7	
8	
9	

5. 다음과 같은 비어있는 해시 테이블에 아래 연산을 차례대로 모두 수행한 후 최종 해시 테이블 상태를 그림으로 나타내시오.

- 해시 함수: 나누기 방법 $h(x) = x \bmod m$
- 충돌해결: open addressing 중에서 <u>quadratic probing</u>

$$h_i(x) = (h(x) + i^2) \bmod m, \quad i = 0, 1, 2, \ldots$$

연산: 10 삽입
12 삽입
14 삽입
20 삽입
30 삽입

답:

0	
1	
2	
3	
4	
5	
6	
7	
8	
9	

6. 다음과 같은 비어있는 해시 테이블에 아래 연산을 차례대로 모두 수행한 후 최종 해시 테이블 상태를 그림으로 나타내시오.

- 해시 함수: 나누기 방법 $h(x) = x \bmod m$
- 충돌해결: open addressing 중에서 <u>double hashing</u>

$$h_i(x) = (h(x) + i \cdot f(x)) \bmod m, \quad i = 0, 1, 2, \ldots$$

$$f(x) = 1 + x \bmod 9$$

연산: 44 삽입
21 삽입
31 삽입
10 삽입
20 삽입
41 삽입
30 삽입

답:

0	
1	
2	
3	
4	
5	
6	
7	
8	
9	

7. 다음과 같은 비어있는 해시 테이블에 아래 연산을 차례대로 모두 수행한 후 최종 해시 테이블 상태를 그림으로 나타내시오.

- 해시 함수: 곱하기 방법 $h(x) = \text{floor}[m \cdot (x\text{A mod } 1)]$

  단, $\text{A} = 0.13$

- 충돌해결 방법: chaining
  * 삽입시 연결 리스트의 맨 앞 또는 맨 뒤에 삽입
  * 해시 테이블의 각 칸은 노드에 대한 포인터(참조 변수)로서, 데이터 삽입시 테이블 외부에 노드를 할당 받아 저장

연산:
44 삽입
21 삽입
31 삽입
10 삽입
20 삽입
41 삽입
30 삽입

초기:

0	null
1	null
2	null
3	null
4	null
5	null
6	null
7	null
8	null
9	null

답:

0	
1	
2	
3	
4	
5	
6	
7	
8	
9	

8. 다음과 같은 해시 테이블에 대해 물음에 답하시오.

- 해시 함수: 나누기 방법 $h(x) = x \bmod m$
- 충돌해결 방법: open addressing 중에서 linear probing

(1) 비어있는 해시 테이블에 아래 연산을 차례대로 모두 수행한 후 최종 해시 테이블 상태를 그림으로 나타내시오.

연산:
29 삽입
19 삽입
11 삽입
39 삽입
22 삽입
39 **삭제**
41 삽입
11 **삭제**

답:

0	
1	
2	
3	
4	
5	
6	
7	
8	
9	

(2) 위의 연산을 모두 수행한 후, 다음 각 키값을 검색할 때(검색 성공 또는 실패) 조사하는 **해시 테이블 인덱스**를 조사 순서대로 모두 적으시오.

16 검색 :

30 검색 :

41 검색 :