



2021-2 알고리즘

해시 테이블



한남대학교 컴퓨터공학과

검색(Searching)



한남대학교



전체 지도 이미지 뉴스 동영상 더보기

도구

검색결과 약 988,000개 (0.60초)

<http://www.hannam.ac.kr>

한남대학교

대학소개 · 한눈에 보는 한남대학교 · 교육목표, 인재상 및 핵심역량 · 설립배경 · 주요성과 · 한남의 어제와 오늘 · 홍보자료실



한남대

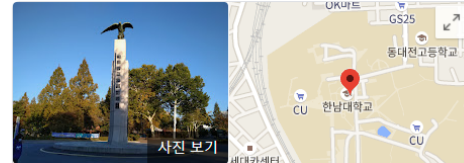
한남대학교 홈페이지 · 외국인 외국인 신입학 모집요강

입학정보

수시모집 2022학년도 수시모집 어학인재전형 면접고사 수험...2021 ...

대학

국어국문·창작학과모집학과 · 영어영문학과모집학과 · 외국어문학부 ...



한남대학교

웹사이트

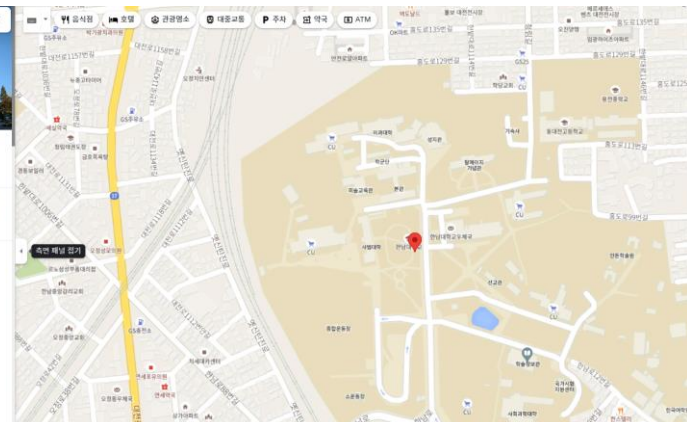
경로

저장

통화

대전광역시 사립대학교

한남대학교는 1956년에 세워진 대한민국의 미국 남장로교회 계열 한국서교회인 서교사들에 의해 세워진 사립 대학이다 대전광역시 대



검색(Searching)



SUMMARY: TOP PRIORITY (RANKED 1ST)

	Manu & N. Res.	Media/Comm	Svcs	Gov.	Edu	Retail	Banking	Insurance	Health-care	Transportation
Enhanced customer experience	20%	37%	20%	15%	-	46%	24%	41%	20%	31%
Process efficiency	27%	11%	15%	23%	42%	8%	3%	11%	40%	38%
New products / new business models	15%	11%	24%	-	25%	8%	3%	11%	10%	31%
More targeted marketing	15%	21%	19%	8%	8%	31%	21%	15%	-	-
Cost reduction	11%	5%	2%	15%	17%	8%	13%	4%	10%	-
Improved risk management	5%	-	7%	8%	8%	-	11%	-	10%	-
Regulatory compliance	4%	5%	3%	8%	-	-	18%	7%	10%	-
Monetize information directly	2%	5%	8%	8%	-	-	3%	4%	-	-
Enhanced security capabilities	-	5%	3%	12%	-	-	5%	4%	-	-
n=	55	19*	114	26*	12*	13*	38	27*	10*	13*

*Base < 30; results are directional

© 2014 Gartner, Inc. and/or its affiliates. All rights reserved.

• 검색(Searching)

- **일상 생활에서의 검색 searching in the real life**
 - 검색 키워드keyword → 키워드와 연관성 있는 결과"들"
- **자료 구조에서의 검색 searching in a data structure**
 - 일상 생활에서의 검색 문제를 구현하려면
 - DB, 웹, 분산 스토리지 등 다양한 영역에 걸쳐 있어 훨씬 더 어려운 주제

검색(Searching)

- 예시)

- 주문 정보를 DB table에 저장
- 각 레코드record는 **유일한** 키(unique key, primary key)를 가짐
- 입력: primary key, 출력: a record
- **Primary key를 저장, 검색하기 위한 자료 구조는?**

Primary key

Foreign key fields

Fields

Records

InvoiceNo	OrderID	CustomerID	EmployeeID	OrderDate	RequiredDate	ShippedDate	ShipVia
100000109	10356	WANDK	6	11/18/2005	12/16/2005	11/27/2005	2
100000110	10357	LILAS	1	11/19/2005	12/17/2005	12/2/2005	3
100000111	10358	LAMAI	5	11/20/2005	12/18/2005	11/27/2005	1
100000112	10359	SEVES	5	11/21/2005	12/19/2005	11/26/2005	3
100000113	10360	BLONP	4	11/22/2005	12/20/2005	12/2/2005	3
100000114	10361	QUICK	1	11/22/2005	12/20/2005	12/3/2005	2
100000115	10362	BONAP	3	11/25/2005	12/23/2005	11/28/2005	1
100000116	10363	DRACD	4	11/26/2005	12/24/2005	12/4/2005	3
100000117	10364	EASTC	1	11/26/2005	1/7/2006	12/4/2005	1
100000118	10365	ANTON	3	11/27/2005	12/25/2005	12/2/2005	2
100000119	10366	GALED	8	11/28/2005	1/9/2006	12/30/2005	2
100000120	10367	VAFFE	7	11/28/2005	12/26/2005	12/2/2005	3
100000121	10368	ERNSH	2	11/29/2005	12/27/2005	12/2/2005	2
100000122	10369	SPLIR	8	12/2/2005	12/30/2005	12/9/2005	2
100000123	10370	CHOPS	6	12/3/2005	12/31/2005	12/27/2005	2
100000124	10371	LAMAI	1	12/3/2005	12/31/2005	12/24/2005	1
100000125	10372	QUEEN	5	12/4/2005	1/1/2006	12/9/2005	2
100000126	10373	HUNGO	4	12/5/2005	1/2/2006	12/11/2005	3
100000127	10374	WOLZA	1	12/5/2005	1/2/2006	12/18/2005	2

Record: 14 of 830

• 검색(Searching)

- CRUD

- Create
- Read
- Update
- Delete

- 9~10장에서 다루는 문제

- **Read (검색)** – Create, Update 까지 포함
- **Delete (삭제)**

검색(Searching)

- 검색 문제

- 입력: a primary key
- 출력: a record
- Primary key를 저장, 검색하기 위한 자료 구조는?

Primary key

Foreign key fields

Fields

Records

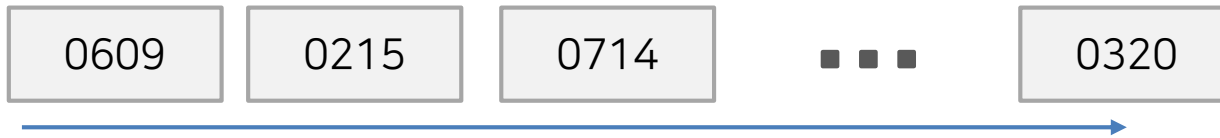
InvoiceNo	OrderID	CustomerID	EmployeeID	OrderDate	RequiredDate	ShippedDate	ShipVia
100000109	10356	WANDK	6	11/18/2005	12/16/2005	11/27/2005	2
100000110	10357	LILAS	1	11/19/2005	12/17/2005	12/2/2005	3
100000111	10358	LAMAI	5	11/20/2005	12/18/2005	11/27/2005	1
100000112	10359	SEVES	5	11/21/2005	12/19/2005	11/26/2005	3
100000113	10360	BLONP	4	11/22/2005	12/20/2005	12/2/2005	3
100000114	10361	QUICK	1	11/22/2005	12/20/2005	12/3/2005	2
100000115	10362	BONAP	3	11/25/2005	12/23/2005	11/28/2005	1
100000116	10363	DRACD	4	11/26/2005	12/24/2005	12/4/2005	3
100000117	10364	EASTC	1	11/26/2005	1/7/2006	12/4/2005	1
100000118	10365	ANTON	3	11/27/2005	12/25/2005	12/2/2005	2
100000119	10366	GALED	8	11/28/2005	1/9/2006	12/30/2005	2
100000120	10367	VAFFE	7	11/28/2005	12/26/2005	12/2/2005	3
100000121	10368	ERNSH	2	11/29/2005	12/27/2005	12/2/2005	2
100000122	10369	SPLIR	8	12/2/2005	12/30/2005	12/9/2005	2
100000123	10370	CHOPS	6	12/3/2005	12/31/2005	12/27/2005	2
100000124	10371	LAMAI	1	12/3/2005	12/31/2005	12/24/2005	1
100000125	10372	QUEEN	5	12/4/2005	1/1/2006	12/9/2005	2
100000126	10373	HUNGO	4	12/5/2005	1/2/2006	12/11/2005	3
100000127	10374	WOLZA	1	12/5/2005	1/2/2006	12/9/2005	3

Record: 109 of 830

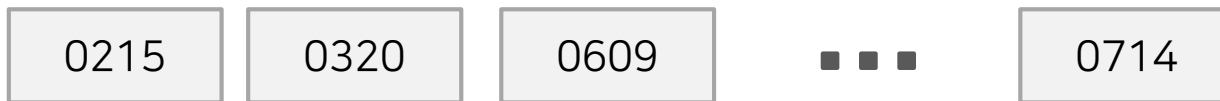
No Filter Search

검색(Searching)

- primary key를 저장하기 위한 선택지들
- **case 1: 배열에 입/출력 순서대로 저장 → sequential search**
 - Insert: $O(1)$
 - Search: avg. $O(n)$



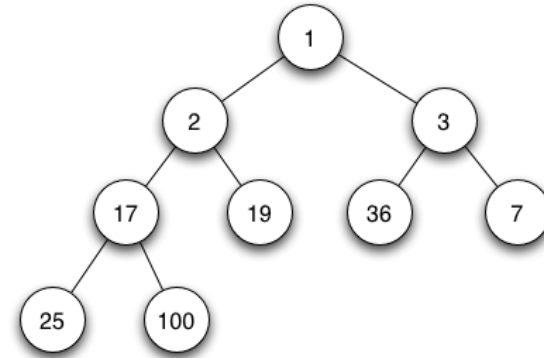
- **case 2: sorted array → binary search**
 - Search: avg. $O(\log n)$
 - Insert: $O(?)$



검색(Searching)

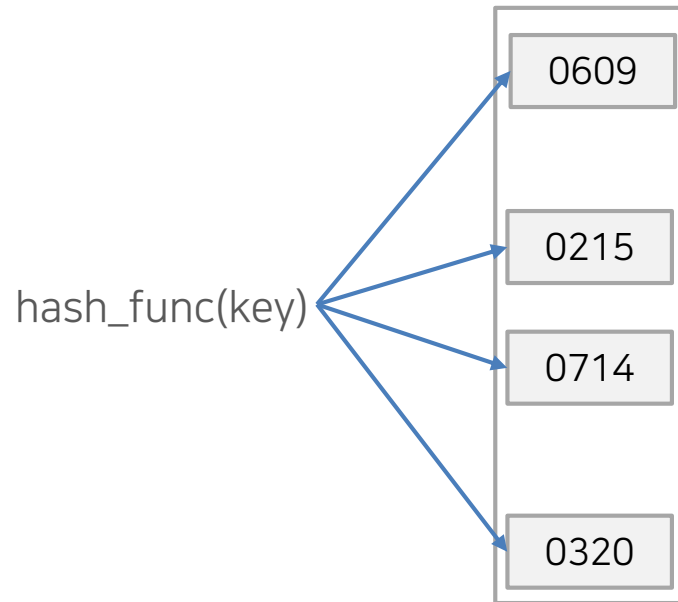
- **case 3: Heap**

- Insert: $O(\log n)$
- Min(Max) Search: $O(1)$
- Search: $O(?)$



- **case 4: Hash table**

- Insert: $O(1)$
- Search: $O(1)$



• 검색(Searching)

- 자료 구조를 선택할 때 고민해야할 부분
 - 각 자료구조를 사용할 때의 장점과 단점은?
 - 이 문제에서는 어떤 자료 구조가 더 적합한가?
- 좀 더 일반적인 문제
 - 입력: 검색 조건
 - 출력: 조건에 맞는 레코드들record's
- 9~10장에서 다루는 문제
 - 입력: 검색 키key
 - 출력: key에 해당되는 레코드record

9~10장 구성

- **Searching** 검색 문제
- **Map**
- **해시테이블 Hash Table**
- **충돌 Collision**
- **충돌 해결 Collision Resolution**
- **Search Tree**
 - (Internal) Binary Search Tree
 - (Internal, Balanced) Red-Black Tree
 - (External) B-tree
 - (Multi-Dimensional) KD-Tree, etc.

MAP



Map, Mapping

- 예시)

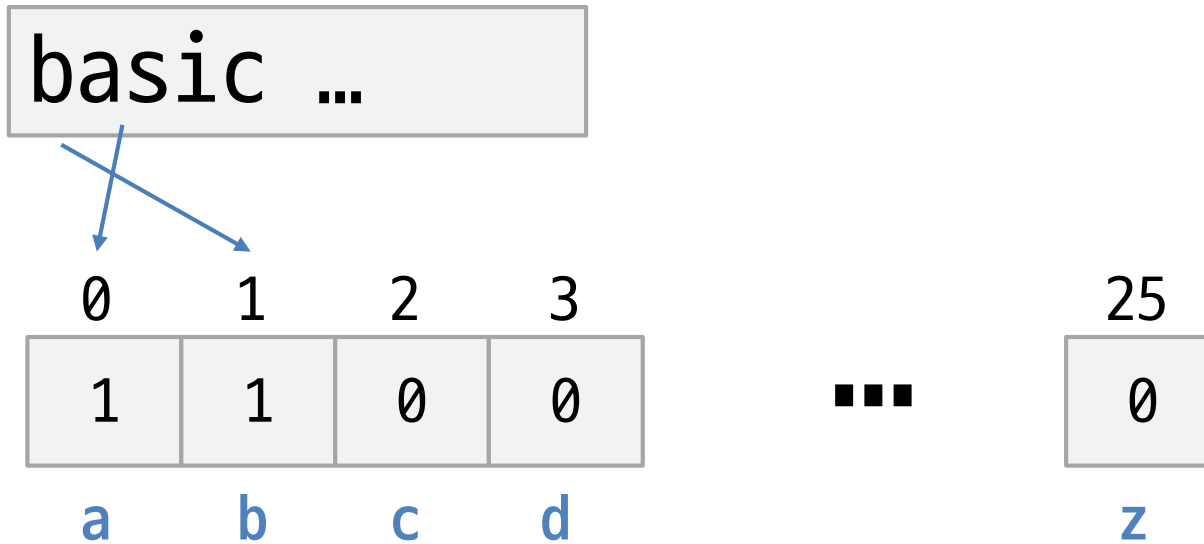
- 아래와 같은 문장을 입력 받고, 사용되지 않은 알파벳을 출력
- 소문자만 입력된다고 가정

the basic outfit of traditional inuit clothing consisted of a parka, pants, mittens, inner footwear, and outer boots, historically made from animal hide and fur. the inuit are a group of culturally related indigenous peoples inhabiting the arctic areas of the united states, canada, and greenland.

Map, Mapping

- Mapping

- 크기 26 배열을 준비, 0으로 초기화
- 'a' ... 'z' \rightarrow 0...26
- 사용된 알파벳에 대해 해당 위치의 값을 1로 바꿔준다.



Map, Mapping

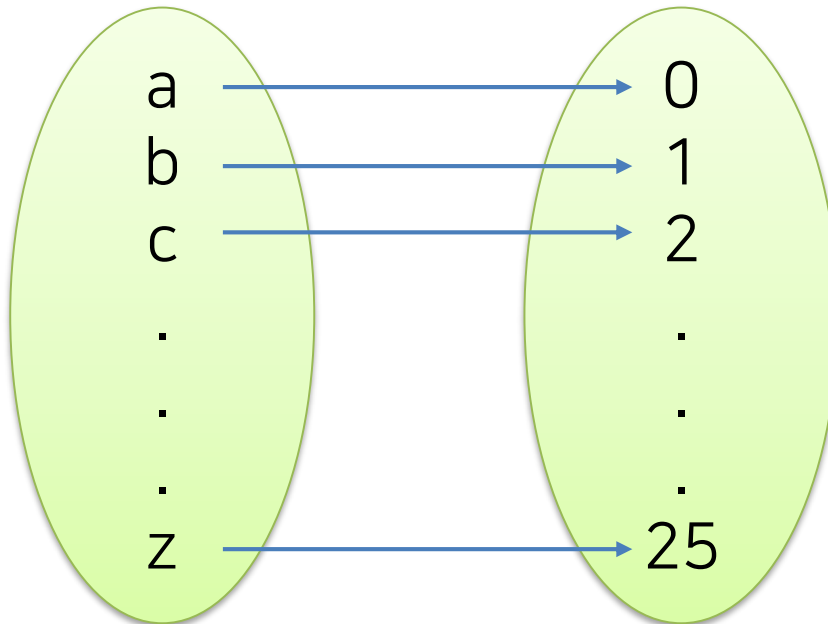
- Solution in C

```
void print_unused_alphabets(char* str, int length) {  
    int used[26] = {0};  
    for (i = 0; i < length; i++) {  
        if (!('a' <= str[i] && str[i] <= 'z'))  
            continue;  
  
        int ascii = str[i];  
        int key = ascii - 'a';  
        used[key] = 1;  
    }  
    for (i = 0; i < 26; i++) {  
        char ch = i + 'a';  
        if (!used[i]) printf("%c\n", ch);  
    }  
}
```

Map, Mapping: 함수Function

```
int ascii = str[i];  
int key = ascii - 'a';
```

$f(\text{char}) : \{\text{'a'}, \dots, \text{'z'}\} \rightarrow \{0, \dots, 25\}$



1:1 대응
1-to-1 correspondence

Map, Mapping

- Python: built-in method `map()`
 - Python의 대표적인 **함수형** 프로그래밍 도구
 - One of the **functional** programming features in Python
- collection의 각 원소에 매핑 함수mapping function을 적용
 - 예) 리스트의 각 원소를 3으로 나눈 나머지

```
def div3_func(x):  
    return x % 3  
  
data = [1, 2, 3, 4, 5]  
print(*map(div3_func, data))
```

1 2 0 1 2

Map, Mapping

- Python: built-in method **map()**

- 예) 앞의 알파벳 문제에서 알파벳을 정수로 변환

```
def map_func(alpha):    알파벳 소문자를 0..25 사이 정수로 매핑하는 함수
    ascii = ord(alpha)
    return ascii - ord('a')
```

```
alphabets = [x.isalpha() for x in str]
keys = list(map(map_func, alphabets))
```

연습문제

- 아래와 같은 리스트(배열)이 주어질 때,
- `map()`을 사용해서 아래 함수를 매핑해 보자.

$$- y = 2x^2 + 5x - 20$$

1	2	3	4	5
---	---	---	---	---

-13 -2 13 32 55

- 참고) `map()`, `filter()` 등을 능숙하게 사용하려면 iterator의 개념을 정확히 알고 있어야 함
- <https://dojang.io/mod/page/view.php?id=2406>

연습문제

- N명의 키(cm)가 아래와 같이 주어진다.
- `map()`을 사용해서 키(m)로 출력해 보자.

179.5	162.0	167.3	191.2	183.5
-------	-------	-------	-------	-------

1.795 1.620 1.673 1.912 1.835

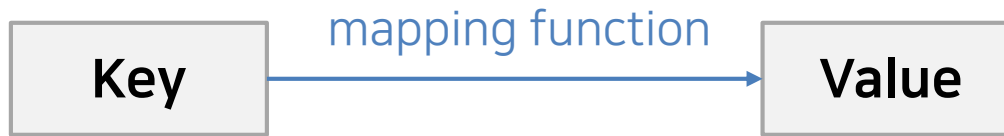
HASH TABLE



Map과 HashMap

- Map

- 종종 **Key** → **Value** 형식으로 표현된다.



- HashMap: OOP 언어에서 **map**을 **해시 함수로 구현**한 클래스(해시테이블)

- hash_map(C++ STL), HashMap(Java, Kotlin), Array(JS, JSON), Dictionary(Python)



해시 함수와 해시 테이블

• Hash Function

- 입력된 키값을 제한된 공간으로 매핑 Map a key into a restricted value space
- Hash Table, Checksum 등에 사용된다.
- 여러 가지 방법이 있으나 가장 대표적인 것은 나누기 방법과 곱하기 방법이다.

- 원시적인 해시 함수의 예 primitive hash function (example)

$\text{hash}(\text{key}) = \text{key} \% \text{TABLE_SIZE}$

소수prime number

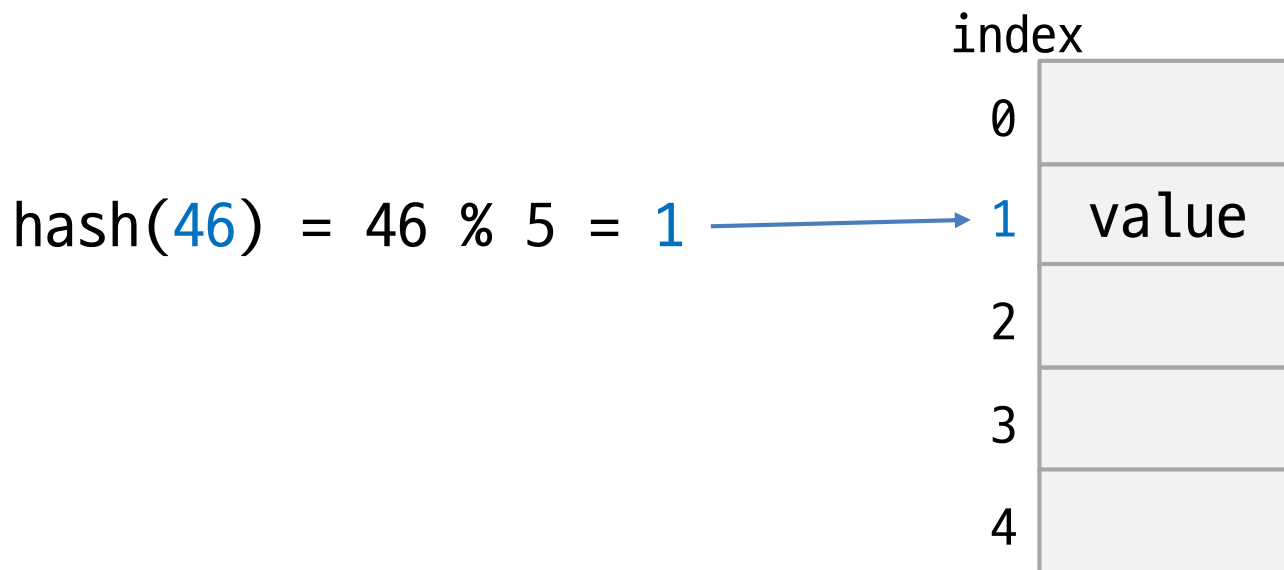
- 해시 함수에 요구되는 특징 A hash func. should be...
 - 입력 원소가 해시 테이블에 고루 저장되어야 한다. even distribution
 - 계산이 간단해야 한다. simple to compute

해시 함수와 해시 테이블

- Hash Table

- 원소가 저장될 위치가 그 값에 의해 결정된다.
- 해시 함수 사용

- 원시적인 해시 테이블의 예 primitive hash table (example)



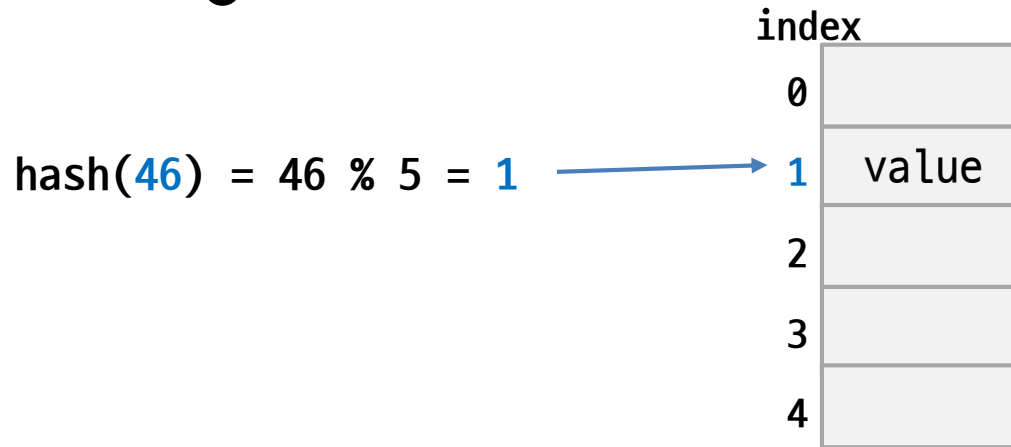
크기 13인 해시 테이블에 5 개의 원소가 저장된 예

입력: 25, 13, 16, 15, 7

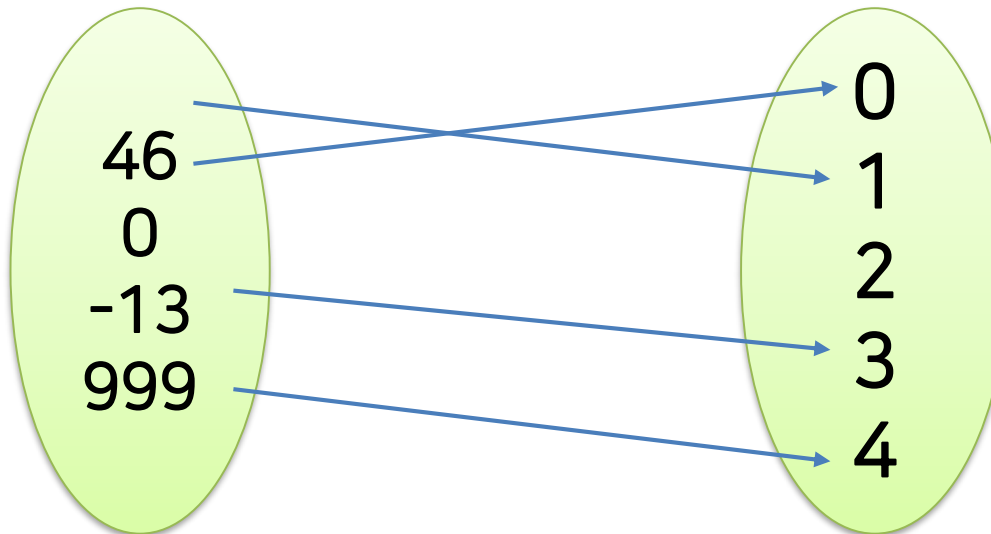
0	13
1	
2	15
3	16
4	
5	
6	
7	7
8	
9	
10	
11	
12	25

해시함수 $h(x) = x \bmod 13$

해시 함수와 해시 테이블

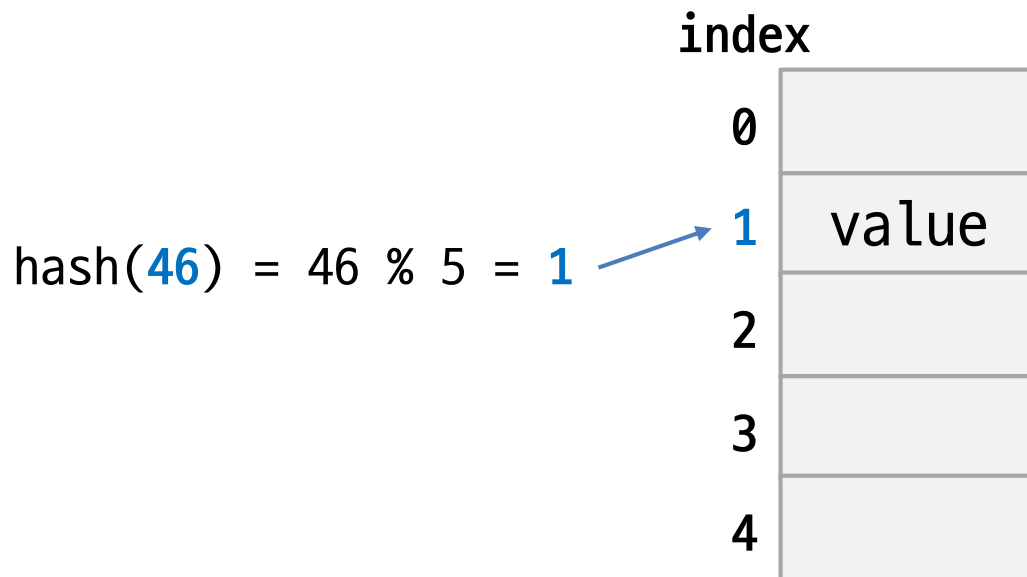


input keys → hash(key) → output values



해시 함수와 해시 테이블

- Hash table의 압도적인 장점
 - 평균 상수 시간 검색(삽입, 삭제) avg. $O(1)$ search



- (수행시간 관점에서) 단점
 - 다른 자료구조와 비교해서 생각해 보기

Dictionary in Python


- 앞의 알파벳 문제를 해시 테이블(dictionary)로 구현

```
import string

def print_unused_alphabets(inpstr):
    used = {'a': False, 'b': False, ..., 'z': False}
    used = {x: False for x in string.ascii_lowercase}
    for ch in inpstr:
        used[ch] = True
    for key in used:
        if not used[key]:
            print(key)
```

used = {'a': False, 'b': False, ..., 'z': False}

"abcdefg...xyz"



연습문제

- Python의 **집합set**은 **사전dictionary**과 마찬가지로 해시 테이블의 일종이다.
- 앞의 문제를 집합을 사용해서 구현해 보자.
- 달라진 점은 무엇인가?
- “set” in python is a sort of hash table as “dictionary”.
- Solve the alphabet problem using “set” instead of “dictionary”.
- What’s the difference?

연습문제

- 해시 테이블과 순서

- 이전 버전의 python에서는 “삽입한 순서를 기억하는 해시 테이블 ” OrderedDict 자료 구조가 따로 있었으며, 현재는 dictionary의 기본 기능으로 통합되었다.
- In previous version of Python, “dictionary” was not a ordered data structure. Currently, “dictionary” is ordered in the latest version of Python.

- 현재 dictionary의 내부 구현이 어떻게 되어 있을지 추정해 보자.

- 해시 테이블에서 값이 삽입된 순서를 기억하기 위해서는 어떤 방법이 필요한가?
- Guess the internal implementation of the current “dictionary”.
- What do you need to implement an **ordered** hash table?

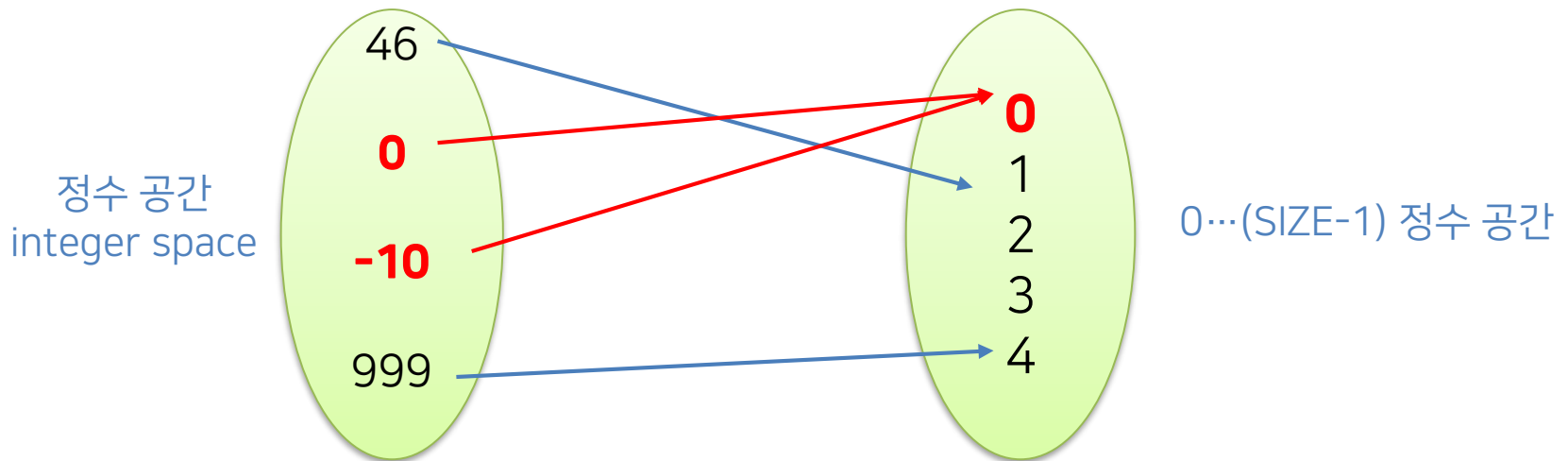
COLLISION



충돌 Collision

- 해시 테이블의 자체의 문제점

– **N:1 대응** N-to-1 correspondence



– 비교) 앞의 알파벳 문제는 1:1 대응

$['a', 'b', 'c', \dots, 'z'] \xrightarrow{\text{ascii}(ch) - \text{ascii}('a')} [0, 1, 2, \dots, 26]$

충돌 Collision

- 해시 테이블의 **충돌collision**

- 서로 키들이 같은 공간slot으로 사상되는 경우
- different keys mapped to a single slot
- 최악의 경우, 테이블 크기보다 더 많은 키가 같은 공간으로 사상될 수도 있음

$$\text{hash}(46) = 46 \% 5 = 1$$

$$\text{hash}(16) = 16 \% 5 = 1$$

$$\text{hash}(21) = 21 \% 5 = 1$$

index

0

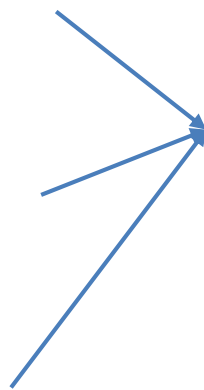
1

2

3

4

value?



충돌의 예

입력: 25, 13, 16, 15, 7

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

← $h(29) = 29 \bmod 13 = 3$

29를 삽입하려 하자
이미 다른 원소가 차지하고 있다!

해시함수 $h(x) = x \bmod 13$

충돌 Collision

- **충돌 회피 collision avoidance**

- 충돌이 발생할 확률을 낮춘다.

- **1) 신중한 해시 함수 설계** design your hash function carefully

- 나머지 연산에 소수를 사용 modulo operation prime number
 - 아주 조금 덜 원시적인 해시 함수의 예

$$\text{hash}(\text{key}) = (\text{key} \% p1) \% p2$$

$p1, p2$: prime numbers
 $p1 \gg p2$

- 잘 알려진 해시 함수들 well-known hash functions
 - MD5, SHA-1, SHA-256, ...

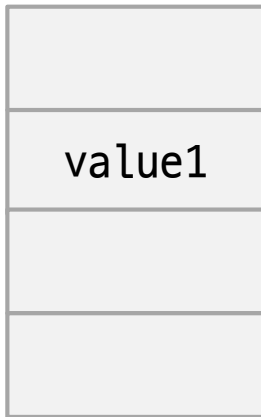
충돌 Collision

- 충돌 회피 collision avoidance

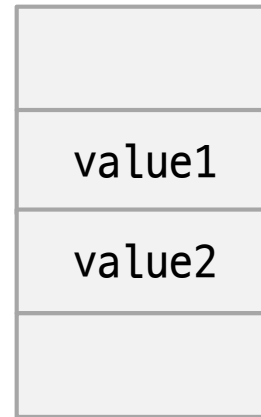
- 2) Rehashing

- 적재율이 임계값을 넘으면 더 큰 해시 테이블로 이전
 - load factor > threshold → migrate stored values to a bigger hash table

- 적재율(load factor) : $\text{occupied slots} / \text{SIZE}$
 - 검색 효율(충돌 확률)과 밀접한 관련이 있음



적재율 25%



적재율 50%

충돌 Collision

- **충돌 해결 Collision Resolution**

- 충돌이 발생했을 때 해시 테이블에 원소를 저장하는 방법

- **체이닝 Chaining**

- 같은 주소로 해싱되는 원소를 모두 하나의 연결 리스트linked list로 관리

- **개방주소 방법 Open Addressing**

- 충돌이 일어나더라도 어떻게든 주어진 테이블 공간에서 해결
 - 추가적인 공간이 필요하지 않다.
 - 선형 조사, 이차원 조사, 더블 해싱

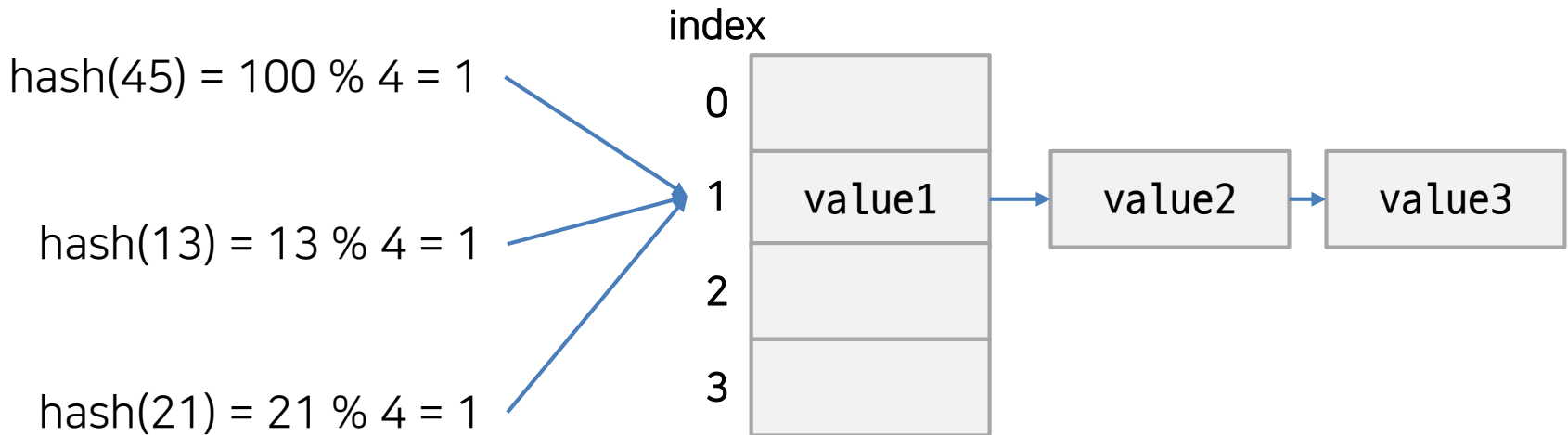
COLLISION RESOLUTION



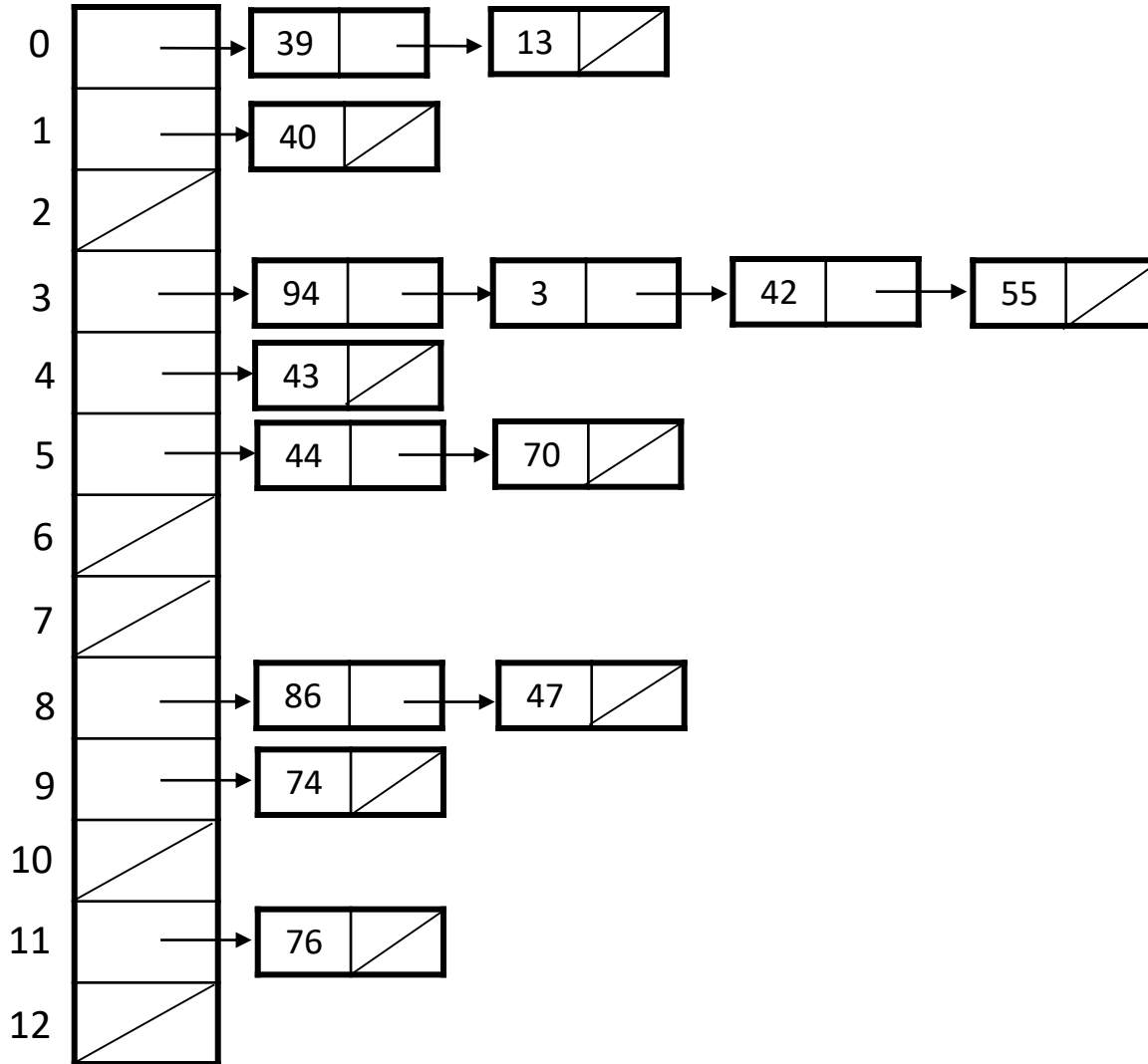
Collision Resolution

- **Chaining**

- 가장 무난한 방법
- slot에 연결 리스트linked list를 만들어 저장
- 추가 공간 필요



체이닝



Collision Resolution

- **Chaining (Array Implementation)**

- Chaining을 $\text{SIZE} \times k$ 배열로 구현
- 공간이 부족하면 rehashing
- 연결리스트 구현보다 속도가 빠르지만 공간의 낭비가 심함

$k = 4$

hash(45) =
 $100 \% 4 = 1$

hash(13) =
 $13 \% 4 = 1$

hash(21) =
 $21 \% 4 = 1$

index

0

1

2

3

value1	value2	value3	

Collision Resolution

- **개방주소 방법 Open Addressing**

- 충돌이 발생하면 빈자리가 생길 때까지 해시값을 계속 만들어 낸다.
- $h_0(x), h_1(x), h_2(x), h_3(x), \dots$

- **개방주소 방법 Open Addressing Methods**

- 선형 조사 Linear Probing
- 이차원 조사 Quadratic Probing
- 더블 해싱 Double Hashing

선형 조사 Linear Probing

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

예: 입력 순서 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

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

선형 조사는 1차군집에 취약하다

1차군집: 특정 영역에 원소가 몰리는 현상

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

← 1차군집의 예

이차원 조사 Quadratic Probing

$$h_i(x) = (h(x) + c_1 i^2 + c_2 i) \bmod m$$

예: 입력 순서 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	

index: $4 + 1^2$

index: $4 + 2^2$

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

이차원 조사는 2차군집에 취약하다

2차군집: 여러 개의 원소가 동일한
초기 해시 함수값을 갖는 현상

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

← 2차군집의 예

더블 해싱 Double Hashing

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

예: 입력 순서 15, 19, 28, 41, 67

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

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

$$h_1(67) = 3$$

$$h_1(28) = 8$$

$$h_1(41) = 10$$

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

$$f(x) = x \bmod 11$$

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

삭제시 조심할 것

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

(a) 원소 10이 삭제된다

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

(b) 38 검색, 문제발생

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

(c) 표식을 해두면 문제없다

생각해 볼 것

- 적재율이 아주 낮으면 각 조사 방법들이 차이가 많이 나는가?
- 성공적인 검색과 삽입의 관계는?