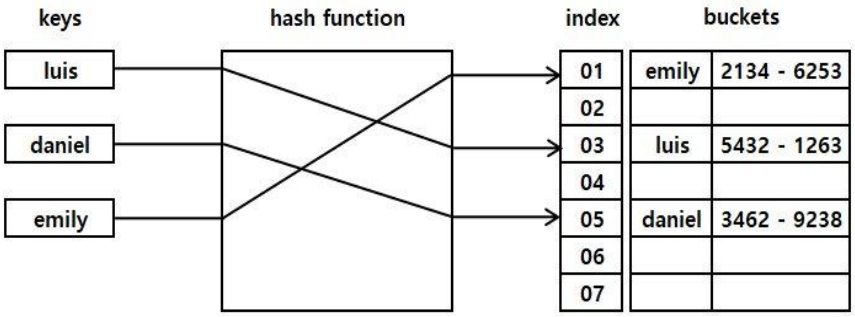
해시테이블

해시함수를 사용하여 키를 해시값으로 매핑하고, 이 해시값을 색인(index) 삼아 데이터의 값(value)을 키와 함께 저장하는 자료구조를 해시테이블(hash table)이라고 합니다. 이 때 데이터가 저장되는 곳을 버킷(bucket) 또는 슬롯(slot)이라고 합니다.

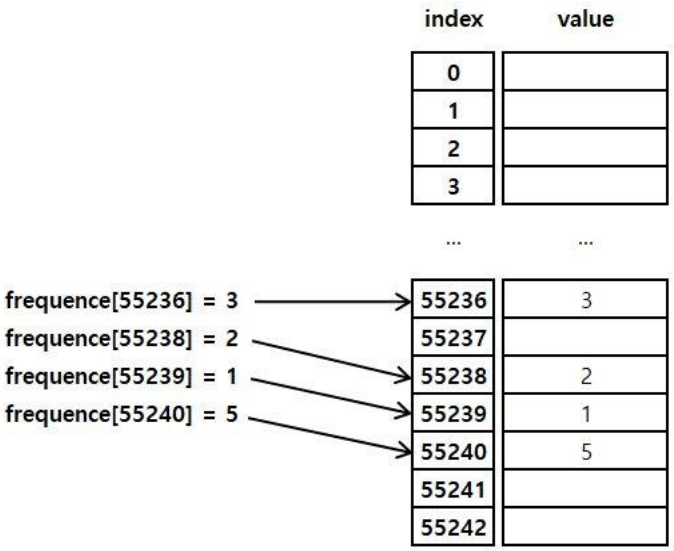


Direct-address table

키의 전체 개수와 동일한 크기의 버킷을 가진 해시테이블을 Direct-address table이라고 합니다. Direct-address table의 장점은 키 개수와 해시테이블 크기가 동일하기 때문에 해시충돌 문제가 발생하지 않는다는 겁니다. 하지만 전체 키 중 실제 사용되는 키의 개수가 적을 경우 메모리 효율성이 크게 떨어집니다. 대표적인것이 배열 인덱스를 키로 사용하는 방법입니다.

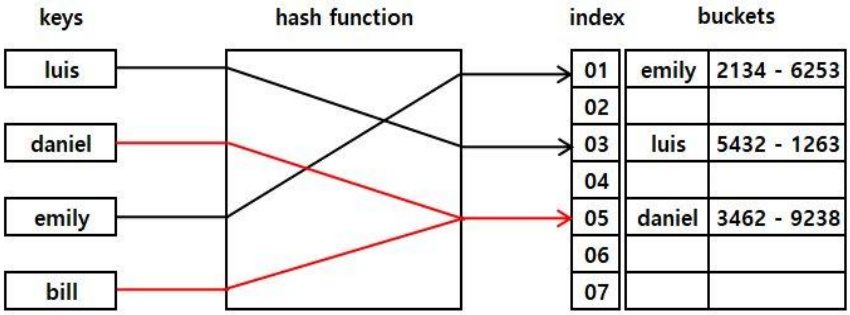
예) 다음 수열에서 각 숫자의 빈도수를 구하세요.

array = [55236, 55238, 55239, 55240, 55236, 55236, 55238, ...55240]



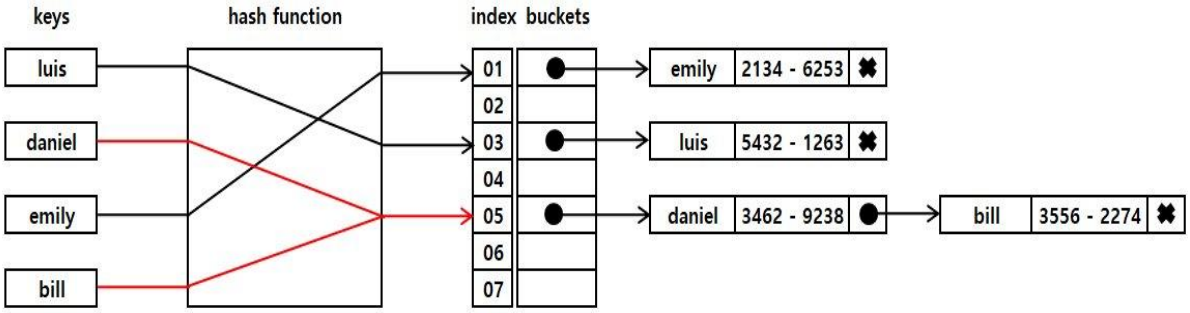
충돌

해시함수는 해쉬값의 개수보다 대개 많은 키값을 해쉬값으로 변환하기 때문에 해시함수가 서로 다른 두 개의 키에 대해 동일한 해시값을 내는 해시충돌(collision)이 발생하게 됩니다. 아래 그림은 이름-전화번호를 매핑하기 위한 해시함수를 개념적으로 나타냈습니다. 예시의 해시함수는 'daniel'과 'bill'를 모두 '05'로 매핑해 해시충돌을 일으키고 있습니다.



충돌해결(체인법 : Separate Chaining)

해시충돌 문제를 해결하기 위한 간단한 아이디어 가운데 하나는 한 버킷당 들어갈 수 있는 엔트리의 수에 제한을 두지 않음으로써 모든 자료를 해시테이블에 담는 것입니다. 해당 버킷에 데이터가 이미 있다면 체인처럼 노드를 추가하여 다음 노드를 가리키는 방식으로 구현(연결리스트)하기 때문에 체인이라고 합니다.

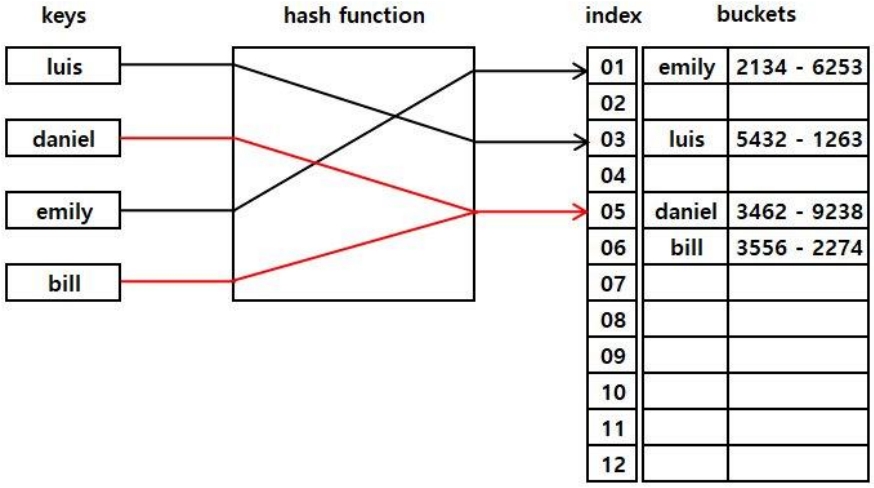


해시테이블의 시간복잡도

"해시테이블의 탐색, 삽입, 삭제의 시간복잡도는 평균적으로 O(1)을 갖지만 충돌이 빈번하게 일어나면 최악의 경우인 O(n)의 시간복잡도로 수렴할 수 있습니다."

충돌해결(개방번지화 : Open Addressing)

open addressing은 chaining과 달리 한 버킷당 들어갈 수 있는 엔트리가 하나뿐인 해시테이블입니다. 해시함수로 얻은 주소가 아닌, 다른 주소에 데이터를 저장할 수 있도록 허용한다는 취지에서 open addressing이라고 합니다. 해시충돌이 빈번히 생길 수 있습니다.



1) 선형탐사(Linear probing)

최초 해시값에 해당하는 버킷에 다른 데이터가 저장돼 있으면 해당 해시값에서 고정 폭(예컨대 1칸)을 옮겨 다음 해시값에 해당하는 버킷에 액세스(삽입, 삭제, 탐색)합니다. 여기에 데이터가 있으면 고정폭으로 또 옮겨 액세스합니다.

2) 제곱탐사(Quadratic probing)

최초 해시값에 해당하는 버킷에 다른 데이터가 저장돼 있으면 해당 해시값에서 그 폭이 제곱수로 늘어난다는 특징이 있습니다. 예컨대 임의의 키값에 해당하는 데이터에 액세스할 때 충돌이 일어나면 1^2칸을 옮깁니다. 여기에서도 충돌이 일어나면 이번엔 2^2칸, 그 다음엔 3^2칸 옮기는 식입니다.

3) 이중해싱(Double hashing)

탐사할 해시값의 규칙성을 없애버려서 clustering을 방지하는 기법입니다. 2개의 해시함수를 준비해서 하나는 최초의 해시값을 얻을 때, 또 다른 하나는 해시충돌이 일어났을 때 탐사 이동폭을 얻기 위해 사용합니다.

dictionary

1. dictionary 객체는 key, value를 한 쌍으로 하는 값을 저장하는 자료구조입니다.
2. dictionary 객체 함수

함수/설명	예제	결과
sH = dict(); dictionary 객체를 생성합니다.	sH = dict();	sH = {}
sH 딕셔너리 key 탐색	sH = {'a': 3, 'b': 5, 'c': 2} for key in sH: print(key)	a, b, c
sH 딕셔너리 value 탐색	sH = {'a': 3, 'b': 5, 'c': 2} for val in sH.values(): print(val)	3, 5, 2
sH 딕셔너리 key, value 동시 탐색	sH = {'a': 3, 'b': 5, 'c': 2} for key, val in sH.items(): print(key, val)	a 3 b 5 c 2

key in sH
sH딕셔너리에 key가 있는지 확인해서 있으면 True, 없으면 False를 반환한다.
시간복잡도는 O(1) 이다.
	sH = {'a': 3, 'b': 5, 'c': 2}
	p1 = 'a' in sH;
	p2 = 'e' in sH;
	print(p1, p2)
		p1 = True
		p2 = False

del(sH[key]) sH의 key를 삭제한다.	sH = {'a': 3, 'b': 5, 'c': 2} del sH['a'] print(sH)	sH = {'b': 5, 'c': 2}
len(sH) sH의 키의 개수를 구해준다. 시간복잡도는 O(1)이다.	sH = {'a': 3, 'b': 5, 'c': 2} print(len(sH))	출력 : 3

빈도수(ver 1)

매개변수 nums에 길이가 n인 수열이 주어지면 수열의 원소중에서 빈도수가 1인 가장 큰 숫자를 반환하는 프로그램을 작성하세요. 빈도수 1인 숫자가 없을 경우 -1를 반환하세요.

입출력 예:

nums	answer
[3, 9, 2, 12, 9, 12, 8, 7, 9, 12]	8
[2, 1, 3, 2, 1, 3, 4, 5, 4, 5, 6, 7, 6 ,7, 8, 8]	-1
[23, 56, 11, 67, 120, 560, 812, 960, 560, 777, 888, 960]	888
[11, 73, 156, 789, 345, 156, 789, 345, 678, 555, 678]	555
[1, 3, 1, 5, 7, 2, 3, 1, 5]	7

제한사항:

- nums의 길이 3 <= n <= 10,000
- 배열 nums의 원소는 자연수입니다. 1 <= nums[i] <= 1,000

빈도수(ver 2)

매개변수 nums에 길이가 n인 수열이 주어지면 수열의 원소중에서 빈도수가 1인 가장 큰 숫자를 반환하는 프로그램을 작성하세요. 빈도수 1인 숫자가 없을 경우 -1를 반환하세요.

입출력 예:

nums	answer
[3, 9, 2, 12, 9, 12, 8, 7, 9, 12]	8
[2, 1, 3, 2, 1, 3, 4, 5, 4, 5, 6, 7, 6 ,7, 8, 8]	-1
[2235253, 5525612, 142561567, 123456789, 2235253, 560, 123456789, 142561567]	5525612
[11, 73, 156, 789, 345, 156, 789, 345, 678, 555, 678]	555
[1, 3, 1, 5, 7, 2, 3, 1, 5]	7

제한사항:

- nums의 길이 3 <= n <= 10,000
- 배열 nums의 원소는 자연수입니다. 1 <= nums[i] <= 1,000,000,000

자기 분열수

자기 분열수란 배열의 원소 중 자기 자신의 숫자만큼 빈도수를 갖는 숫자를 의미합니다.
만약 배열이 [1, 2, 3, 1, 3, 3, 2, 4] 라면
1의 빈도수는 2,
2의 빈도수는 2,
3의 빈도수는 3,
4의 빈도수는 1입니다.
여기서 자기 자신의 숫자와 같은 빈도수를 갖는 자기 분열수는 2와 3입니다.
매개변수 nums에 자연수가 원소인 배열이 주어지면 이 배열에서 자기 분열수 중 가장 작은 수를 찾아 반환하는 프로그램을 작성하세요. 자기 분열수가 존재하지 않으면 -1를 반환하세요.

입출력 예:

nums	answer
[1, 2, 3, 1, 3, 3, 2, 4]	2
[1, 2, 3, 3, 3, 2, 4, 5, 5, 5]	1
[1, 1, 2, 5, 5, 5, 5, 5, 3, 3, 3, 3, 5]	-1
[7, 6, 7, 7, 8, 8, 8, 8, 7, 5, 7, 7, 7, 8, 8]	7
[11, 12, 5, 5, 3, 11, 7, 12, 15, 12, 12, 11, 12, 12, 7, 8, 12, 11, 12, 7, 12, 5, 15, 20, 15, 12, 15, 12, 15, 14, 12]	12

제한사항:
- nums의 길이 3 <= n <= 500,000
- 배열 nums의 원소는 자연수입니다. 1 <= nums[i] <= 1,000,000

팰린드롬 확인

소문자로만 이루어진 문자열이 주어지면 해당 문자열의 문자들의 순서를 재배치해서 팰린드롬(회문)을 만들 수 있는지를 확인하고 싶습니다. 만약 "abbac"같은 문자열은 문자들을 "abcba"로 재 배치하면 팰린드롬을 만들 수 있습니다.
매개변수 s에 문자열이 주어지면 해당 문자열이 재 배치를 통해 팰린드롬을 만들 수 있으면 True를 못 만들면 False를 반환하는 프로그램을 작성하세요.

입출력 예:

s	answer
"abacbaa"	True
"abaaceeffkckbaa"	True
"abcabbcc"	False
"sgsgsgabaaaeceeceekefefkccckbsgaaffsgsg"	True
"aabcefagcefbcabbcc"	False

제한사항:
• s의 길이는 1,000을 넘지 않습니다.

팰린드롬 길이

문자열이 주어지면 해당 문자열의 문자들을 가지고 만들 수 있는 최대길이 팰린드롬을 만들고 그 길이를 구하세요. 문자열은 소문자로만 이루어져 있습니다.
만약 "abcbbbccaaeee" 가 주어진다면 만들 수 있는 가장 긴 팰린드롬은 "ebbcaaacbbe"이고 답은 11입니다.

입출력 예:

s	answer
"abcbbbccaaeee"	11
"aabbccddee"	10
"fgfgabtetaaaetytceefcecekefefkccckbsgaafffg"	41
"aabcefagcefbcabbcc"	17
"abcbbbccaa"	9

제한사항:
• s의 길이는 1,000을 넘지 않습니다.

두 수의 합

정수 수열 안에서 수열의 원소 두 개의 합이 target값이 되는 경우를 찾고 싶습니다.
매개변수 nums에 길이가 n인 수열이 주어지고, 매개변수 target에 자연수 값이 주어지면 이 수열안에서 두 개의 원소의 합이 정수 target값이 되는 두 원소를 구해 배열에 오름차순으로 담아 반환합니다.
두 개의 원소의 합이 target값이 되는 경우는 오직 한가지 뿐인 입력만 주어집니다. 한 원소를 두 번 더하는 것은 안됩니다. nums의 각 원소는 유일값입니다.
답이 없을 경우 [0, 0]을 반환합니다.

입출력 예:

nums	target	answer
[7, 3, 2, 13, 9, 15, 8, 11]	12	[3, 9]
[21, 12, 30, 15, 6, 2, 9, 19, 14]	24	[9, 15]
[12, 18, 5, 8, 21, 27, 22, 25, 16, 2]	28	[12, 16]
[11, 17, 6, 8, 21, 9, 19, 12, 25, 16, 2]	26	[9, 17]
[7, 5, 12, -9, -12, 22, -30, -35, -21]	-14	[-21, 7]
[7, 5, 12, 20]	15	[0, 0]

제한사항:
- nums의 길이 3 <= n <= 200,000
- 배열 nums의 원소는 정수입니다. -10,000 <= nums[i] <= 10,000
- -20,000 <= target <= 20,000