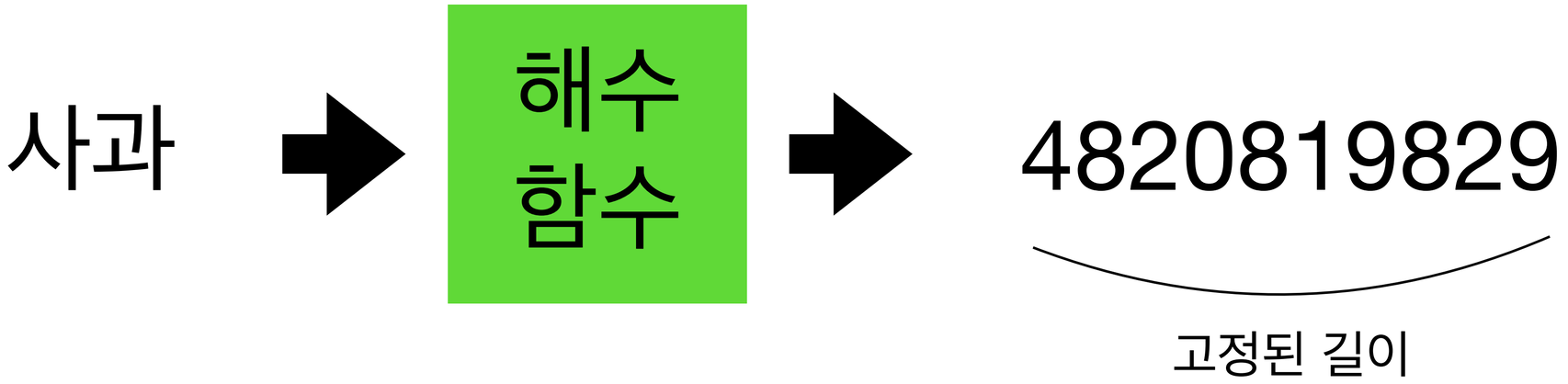
해시

해시 기본 개념
관련 문제 풀이 (문제34,36,37)

23.11.20 송이

9.1 해시란?

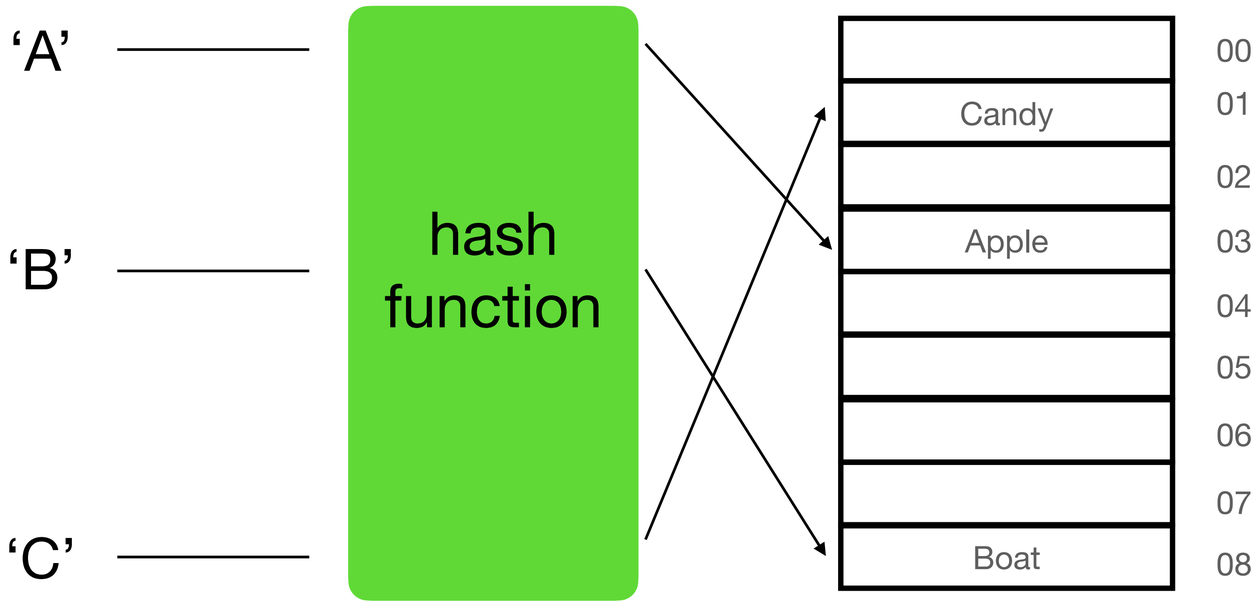
- 해시는 임의의 데이터(값)를 변환 함수를 사용해 고정된 크기의 데이터로 변환한 값.
- 특정 값을 입력받으면 길이에 상관없이 항상 일정한 결과를 만들어낸다.



보안 때문에 다른 입력을 넣으면 고정된 길이의 다른 값이 나온다.

해시의 작동원리를 사용해 만든 자료형이 해시 테이블
파이썬에서는 딕셔너리를 이용한다.
알고리즘 대회에 출전할 정도가 아니라면 그냥 딕셔너리를 사용해 풀면 된다.

9.1.1 해시 테이블이란?



딕셔너리는 key, value 두개의 인자를 갖고 있다.
선언할 때는 {key:value}로 한다.

다만 key는 문자열, 숫자, boolean 형식만 사용 가능

value의 자료형은 형식에 구애받지 않는다.

9.1.1 해시 테이블이란?

사과 4개, 바나나 11개 , 체리 7개 **딕셔너리** book = {‘Apple’:4,’Banana’:11,’Cherry’:7}

배열 [‘Apple’,’Banana’,’Cherry’] [4,11,7]

딕셔너리를 사용했을 때의 장점?

1. 탐색에 걸리는 시간 비용이 크게 줄어든다.
2. 특수한 상황이 아니라면 각 요소가 어떻게 나열되어 있는지에 대해 정할 필요가 없다.

하지만 배열을 사용해야 하는 경우도 있다.

정렬이 잦거나 전체 탐색이 필요한 경우, 최소값/최대값을 찾아야 하는 경우에는 배열이 더 낫다.

9.1.2 해시의 시간 복잡도

딕셔너리를 사용했을 때의 장점?

1. 탐색에 걸리는 시간 비용이 크게 줄어든다. O(1) **하지만 딕셔너리의 탐색 시간이 항상 O(1)을 보장하지 않는다.** 충돌 때문에

딕셔너리에서 사용되는 키는 해시 함수를 기반으로 만들어지는데

이때 해시함수는 낮은 확률로 동일한 값을 출력할 수 있다.

이것을 바로 충돌(collision)이라고 한다.

이 충돌을 해결하는 방법은 크게 개별연결과 공개 주소로 나뉜다.

개별 연결 방식

공개 주소 방식

9.1.2 해시의 시간 복잡도

충돌을 해결하기 (1) 개별 연결 방식

개별 연결 방식

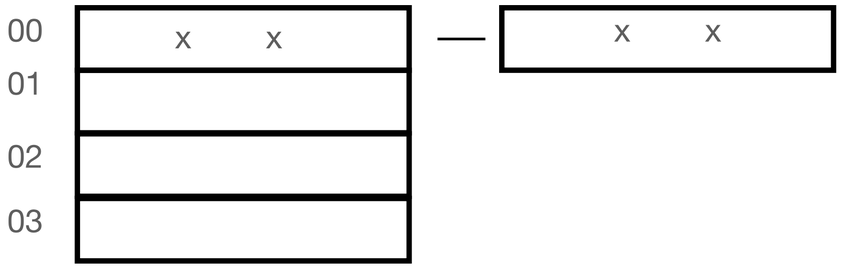
개별 연결 방식은 동일한 해시가 생성됐을 경우 연결리스트(배열)를 만들고 데이터를 집어넣어 한 해시에 연속적으로 연결된 형태를 취하는 방식

장점 만들기 쉽고 직관적이다.

단점 연결이 많아지면 해당 데이터를 찾기 위해 순차적으로 탐색해야 하니 사실상 2차원 배열이 된다.

보완 연결리스트가 아니라 **트리**로 만들어 탐색 속도를 끌어올릴 수 있으나 탐색을 위한 추가 비용이 발생하는 것은 변하지 않는다.

(C++, 자바에서는 이 방식을 사용한다.)



9.1.2 해시의 시간 복잡도

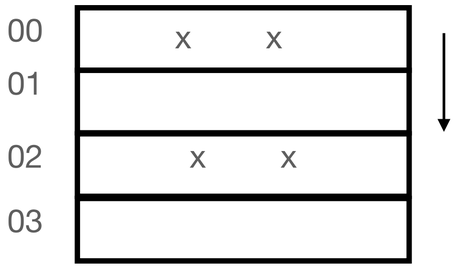
충돌을 해결하기 (1) 공개 주소 방식

공개 주소 방식

인근의 빈 공간을 검색하여 값을 넣는 방식으로 충돌 시 주어진 계산식에 따라 위치를 다시 계산하여 배정한다.

장점 이미 생성된 리스트 내에서 해결한다. 즉 추가적인 메모리를 차지하지 않고 성능이 의외로 괜찮다. 파이썬과 루비에서는 이 방식을 사용한다.

단점 원래 있어야 할 공간에 들어가는게 아니므로 다른 값과 다시 충돌하는 **'재충돌 현상'**이 발생하기 쉽다.



파이썬에서는 연결 리스트를 새로 생성하면서 메모리를 생성하는 비용이 생각보다 크기 때문에 공개 주소 방식을 사용한다.
주로 인터프리터 언어들이 이 방식들을 사용한다.

9.1.2 해시의 시간 복잡도

비교

두 방식 모두 일정 비율(대략 70%)을 넘어서면 충돌 비율이 높아지고 결국 성능이 저하되기 때문에
(1) 메모리를 늘리거나
(2) 해시 함수를 다시 만드는 조치를 취해야 한다.

결론

자세히 보면 가장 빠르다는 해시도 모든 상황에서 O(1) 시간이 보장되지는 않는다.
하지만 일반적으로 사용하는 범위라면 충돌이 발생할 일이 거의 없으므로 사실상 O(1)로 봐도 무방하다.

문제 34

문제 이해하기

전화번호부에 적힌 전화번호 중, 한 번호가 다른 번호의 접두어인 경우가 있는지 확인하려 합니다.
전화번호가 다음과 같을 경우, 구조대 전화번호는 영석이의 전화번호의 접두사입니다.

- 구조대 : 119
- 박준영 : 97 674 223
- 지영석 : 11 9552 4421

전화번호부에 적힌 전화번호를 담은 배열 phone_book 이 solution 함수의 매개변수로 주어질 때, 어떤 번호가 다른 번호의 접두어인 경우가 있으면 false를 그렇지 않으면 true를 return 하도록 solution 함수를 작성해주세요.

제한 사항

- phone_book의 길이는 1 이상 1,000,000 이하입니다.
  - 각 전화번호의 길이는 1 이상 20 이하입니다.
  - 같은 전화번호가 중복해서 들어있지 않습니다.

입출력 예제

phone_book	return
["119", "97674223", "1195524421"]	false
["123","456","789"]	true
["12","123","1235","567","88"]	false

문제 34

제가 푼 방식 소개

solution.py

```python
def solution(phone_book):
    phone_book.sort()
    for i in range(len(phone_book)-1):
        limit = len(phone_book[i])
        if phone_book[i+1][:limit] == phone_book[i]:
            return False
    return True
```

```
["119", "97674223", "1195524421"]
false
테스트를 통과하였습니다.
['119', '1195524421', '97674223']
```

문제 34
책의 접근방식

접두사

낱말의 앞에 붙어 의미를 첨가하여 다른 낱말을 이루는 말

119 접두사 + 다른 낱말의 구조 1 + 19 11 + 9 ‘119’ + “”

글자 수와 상관없이 일치하는 일부 숫자가 왼쪽끝부터 연속적으로 등장할 경우 접두사로 취급할 수 있다.

숫자별로 접두사 여부를 확인하도록 만들면 바로 문제를 해결할 수 있다.

step1 각 숫자에서 가능한 접두사 조합을 만든다.

step2 숫자를 하나씩 살펴보면서 접두사인지 판단한다.

배열을 사용하면 비교 하면서 O(n)만큼 연산하지만 이번에는 딕셔너리를 사용해 이 연산을 O(1)로 줄이기

문제 34

step1 각 숫자에서 가능한 접두사 조합을 만들기

1-1 자기 자신을 딕셔너리에 등록하기

```
def solution(phone_book):
    header = dict()

    for phone_number in phone_book:
        header[phone_number] = 1
```

```
{'12': 1, '123': 1, '1235': 1, '567': 1, '88': 1}
```

1-2 문자열을 모두 쪼개 한 글자씩 합쳐나가면서 나오는 결과를 딕셔너리에 등록

```
for phone_number in phone_book:
    header = ''
    for number in phone_number:
        header += number
```

```
["12", "123", "1235", "567", "88"]
false
테스트를 통과하였습니다.
1
12
1
12
```

```
1
11
119
9
97
976
9767
97674
976742
9767422
97674223
1
11
119
```

문제 34

step2 숫자를 하나씩 살펴보면서 접두사인지 판단하기

2 문자열을 하나씩 살펴보면서 접두사인지 판단하기

```
{'12': 1, '123': 1, '1235': 1, '567': 1, '88': 1}
```

```
for phone_number in phone_book:
    header = ''
    for number in phone_number:
        header += number

        if header in headers and header != phone_number:
            return False


        return True
```

```
["12", "123", "1235", "567", "88"]
false
테스트를 통과하였습니다.
1
12
1
12
```

문제 36

문제 이해하기

"[닉네임]님이 들어왔습니다."

채팅방에서 누군가 나가면 다음 메시지가 출력된다.

"[닉네임]님이 나갔습니다."

채팅방에서 닉네임을 변경하는 방법은 다음과 같이 두 가지이다.

- 채팅방을 나간 후, 새로운 닉네임으로 다시 들어간다.
- 채팅방에서 닉네임을 변경한다.

닉네임을 변경할 때는 기존에 채팅방에 출력되어 있던 메시지의 닉네임도 전부 변경된다.

예를 들어, 채팅방에 "Muzi"와 "Prodo"라는 닉네임을 사용하는 사람이 순서대로 들어오면 채팅방에는 다음과 같이 메시지가 출력된다.

입출력 예

record	result
`["Enter uid1234 Muzi", "Enter uid4567 Prodo","Leave uid1234","Enter uid1234 Prodo","Change uid4567 Ryan"]`	`["Prodo님이 들어왔습니다.", "Ryan님이 들어왔습니다.", "Prodo님이 나갔습니다.", "Prodo님이 들어왔습니다."]`

예를 들어, 채팅방에 "Muzi"와 "Prodo"라는 닉네임을 사용하는 사람이 순서대로 들어오면 채팅방에는 다음과 같이 메시지가 출력된다.

"Muzi님이 들어왔습니다."
"Prodo님이 들어왔습니다."

채팅방에 있던 사람이 나가면 채팅방에는 다음과 같이 메시지가 남는다.

"Muzi님이 들어왔습니다."
"Prodo님이 들어왔습니다."
"Muzi님이 나갔습니다."

Muzi가 나간후 다시 들어올 때, Prodo 라는 닉네임으로 들어올 경우 기존에 채팅방에 남아있던 Muzi도 Prodo로 다음과 같이 변경된다.

"Prodo님이 들어왔습니다."
"Prodo님이 들어왔습니다."
"Prodo님이 나갔습니다."
"Prodo님이 들어왔습니다."

문제 36

제가 푼 방식 소개

```python
def solution(record):
    dictionary = {}
    answer = []
    for re in record:
        msg = re.split()
        if msg[0] =='Enter' or msg[0]=='Change':
            dictionary[msg[1]] = msg[2]

    for re in record:
        msg = re.split()
        if msg[0] =='Enter':
            a = dictionary[msg[1]]+"님이 들어왔습니다."
            answer.append(a)
        elif msg[0] =='Leave':
            a = dictionary[msg[1]]+"님이 나갔습니다."
            answer.append(a)
    return answer
```

record

```
["Enter uid1234 Muzi", "Enter uid4567 Prodo","Leave uid1234","Enter uid1234 Prodo","Change uid4567 Ryan"]
```

```
{'uid1234': 'Prodo', 'uid4567': 'Ryan'}
```

문제 36

책의 접근방식

step1 정답 배열, 과정 배열, 유저 데이터를 저장할 딕셔너리를 정의한다.

step2 명령줄을 하나씩 따라가면서 유저의 닉네임 정보를 계속 변경한다.

step3 한 번 더 명령줄을 따라가면서 출입 기록을 저장한다.

문제 36

step1 정답 배열, 과정 배열, 유저 데이터를 저장할 딕셔너리를 정의한다.

step1 정답 배열, 과정 배열, 유저 데이터를 저장할 딕셔너리를 정의한다.

```
def solution(record):
    answer = []
    actions = []
    user = {}
```

유저 정보는 반복해서 탐색과 수정이 이루어져야 하므로 딕셔너리를 사용한다.

문제 36

step2 명령줄을 하나씩 따라가면서 유저의 닉네임 정보를 계속 변경한다.

step2 명령줄을 하나씩 따라가면서 유저의 닉네임 정보를 계속 변경한다.

```
for event in record:
    info = event.split()
    cmd,uid = info[0], info[1]
    if cmd in ('Enter','Change'):
        nick = info[2]
        user[uid] = nick

    actions.append((cmd,uid))
```

문제 36

step3 한 번 더 명령줄을 따라가면서 출입 기록을 저장한다.

step3 한 번 더 명령줄을 따라가면서 출입 기록을 저장한다.

```
for action in actions:
    cmd, did = action
    if cmd=='Enter':
        answer.append(f'{user[uid]}님이 들어왔습니다.')

    elif cmd=='Leave':
        answer.append(f'{user[uid]}님이 나갔습니다.')
```

문제 37

문제 이해하기

스트리밍 사이트에서 장르 별로 가장 많이 재생된 노래를 두 개씩 모아 베스트 앨범을 출시하려 합니다. 노래는 고유 번호로 구분하며, 노래를 수록하는 기준은 다음과 같습니다.

1. 속한 노래가 많이 재생된 장르를 먼저 수록합니다.
2. 장르 내에서 많이 재생된 노래를 먼저 수록합니다.
3. 장르 내에서 재생 횟수가 같은 노래 중에서는 고유 번호가 낮은 노래를 먼저 수록합니다.

노래의 장르를 나타내는 문자열 배열 genres와 노래별 재생 횟수를 나타내는 정수 배열 plays가 주어질 때, 베스트 앨범에 들어갈 노래의 고유 번호를 순서대로 return 하도록 solution 함수를 완성하세요.

입출력 예

genres	plays	return
["classic", "pop", "classic", "classic", "pop"]	[500, 600, 150, 800, 2500]	[4, 1, 3, 0]

입출력 예 설명

classic 장르는 1,450회 재생되었으며, classic 노래는 다음과 같습니다.

- 고유 번호 3: 800회 재생
- 고유 번호 0: 500회 재생
- 고유 번호 2: 150회 재생

pop 장르는 3,100회 재생되었으며, pop 노래는 다음과 같습니다.

- 고유 번호 4: 2,500회 재생
- 고유 번호 1: 600회 재생

따라서 pop 장르의 [4, 1]번 노래를 먼저, classic 장르의 [3, 0]번 노래를 그다음에 수록합니다.

- 장르 별로 가장 많이 재생된 노래를 최대 두 개까지 모아 베스트 앨범을 출시하므로 2번 노래는 수록되지 않습니다.

문제 37

제가 푼 방식 소개

```
{'classic': {'total_count': 1450, 'songs_list': [[0, 500], [2, 150], [3, 800]]}, 'pop': {'total_count': 3100, 'songs_list': [[1, 600], [4, 2500]]}}
```

```python
def solution(genres, plays):
    genre_dict = {}
    answer = []
    for i in range(len(genres)):
        if genres[i] not in genre_dict:
            genre_dict[genres[i]] = {'total_count': plays[i], 'songs_list': [[i,plays[i]]]}
        else:
            genre_dict[genres[i]]['total_count'] += plays[i]
            genre_dict[genres[i]]['songs_list'].append([i,plays[i]])
```

```python
genre_dict = dict(sorted(genre_dict.items(),key = lambda x : x[1]['total_count'],reverse=True))
for genre in genre_dict:
    genre_dict[genre]['songs_list'] = sorted(genre_dict[genre]['songs_list'],key=lambda x : x[1],reverse=True)
```

```python
for genre in genre_dict:
    count = 0
    for song in genre_dict[genre]['songs_list']:
        count +=1
        if count >2:
            continue
        else:
            answer.append(song[0])
return answer
```

문제 37
책의 접근방식

step1 데이터를 분할하여 조건에 맞게 정리한다.

step2 데이터를 정렬하여 문제의 조건에 맞게 탐색한다.

문제 37

step1 데이터를 분할하여 조건에 맞게 정리한다.

```
def solution(genres, plays):
    answer = []
    info = {}
    gens = {}

    for idx, (gen,play),in enumerate(zip(genres, plays)):
        if gen not in info:
            info[gen] = [(idx, play)]
        else:
            info[gen].append((idx, play))

        gens[gen] = gens.get(gen, 0) + play
```

```
{'classic': [(0, 500), (2, 150), (3, 800)], 'pop': [(1, 600), (4, 2500)]}
{'classic': 1450, 'pop': 3100}
```

(0,('classic',500)

```
a = {"banana": 1500, "watermelon": 900}
```

```
print(a["grape"])  # KeyError: 'grape'
```

```
a = {"banana": 1500, "watermelon": 900}

print(a.get("banana"))  # 1500
print(a.get("grape"))  # None
```

```
print(a.get("banana", 0))  # 1500
print(a.get("grape", 0))  # 0
```

문제 37

step2 데이터를 정렬하여 문제의 조건에 맞게 탐색한다.

```
answer  = []
for (gen,_) in sorted(gens.items(),key=lambda x : x[1], reverse=True):
    for (idx,_) in sorted(info[gen],key=lambda x : x[1], reverse=True)[:2]:
        answer.append(idx)
```

↑

한 장르에 최대 2개의 노래만 수록해야 하므로 잘라준다.

gens 딕셔너리

[('classic',1450),('pop',3100)]