**프로그래머스 lv 2. 2018,2019 KAKAO BLIND RECRUITMENT**

#[1차] 캐시

입력된 도시이름 배열을 순서대로 처리할 때, 총 실행시간을 출력한다.

캐시 교체 알고리즘은 LRU(Least Recently Used)를 사용한다. cache hit일 경우 실행시간은 1이다. cache miss일 경우 실행시간은 5이다.

이런 조건들에 맞추어서 페이지 알고리즘의 한 종류인 LRU 알고리즘을 구현해 보았다. 이 알고리즘은 사용한지 오래된 데이터일수록 사용할 확률도 낮을 것이라는 기본 지식을 바탕으로 진행한다.

1. 저장 용량이 0일 때

1. 그냥 cache miss으로 모든 상황이 발생할 것이기 때문에 전체 데이터 X 5를 한다.

2. 저장 용량이 0보다 클 때

1. 만약에 데이터가 기존에 있는 데이터가 아니라면

1. 데이터의 용량이 꽉 찼다면 제일 앞의 데이터 삭제 후 새로운 데이터를 맨 뒤에 삽입 answer += 5

2. 자리가 남았다면 그냥 삽입 후 answer += 5, v+=1

2. 기존에 있는 데이터라면

1. index를 이용해서 해당 데이터를 지우고(제일 오래된 데이터를 지워야 한다) 맨 뒤에 새로운 데이터를 갱신한다. Answer += 1

def **check**(cache, cities, cacheSize):

    global answer

    v = 0

    if cacheSize == 0:

        answer = **len**(cities)\*5

    else:

        for i in **range**(**len**(cities)):

            if cities[i] in cache:

                cache.pop(cache.index(cities[i]))

                cache.append(cities[i])

                answer += 1

            else:

                if v < cacheSize:

                    cache.append(cities[i])

                    answer += 5

                    v += 1

                elif v == cacheSize:

                    cache.pop(0)

                    cache.append(cities[i])

                    answer += 5

    return answer

#오픈 채팅방

채팅방에 들어오고 나가거나, 닉네임을 변경한 기록이 담긴 문자열 배열 record가 매개변수로 주어질 때, 모든 기록이 처리된 후, 최종적으로 방을 개설한 사람이 보게 되는 메시지를 문자열 배열 형태로 return 하도록 solution 함수를 완성하라.

이 문제를 처음에 잘못 이해해서 그때 그때, 즉 변경 할 떄 마다 변경된 닉네임 값으로 출력이 되도록 하면 되는 것인 줄 알았다. 그러나 그게 아니라 미리 다 변경 해놓고 최종적으로 변경 된 상태로 출력이 되도록 하는 것이었다. 마지막에 70%의 정확도만 나왔는데, 그것은 중간에 record[i][1]에서 아이디의 uid를 없앴기 때문인데, 그 알파벳도 아이디의 일부이므로 지우면 안되는 것이었다 사소한 조건이라도 꼼꼼히 보도록 하자.

def **change**(record,name):

    for i in **range**(**len**(record)):

        if record[i][0] == 'Enter':

            name[record[i][1]] = record[i][2]

        elif record[i][0] == 'Change':

            name[record[i][1]] = record[i][2]

        elif record[i][0] == 'Leave':

            del name[record[i][1]]

def **solution**(record):

    for i in **range**(**len**(record)):

        record[i] = list(**map**(str, record[i].split(' ')))

    ans, res = [], ''

    name = dict()

    change(record, name)

    for i in **range**(**len**(record)):

        if record[i][0] == 'Enter':

            res = '%s님이 들어왔습니다.' %(name[record[i][1]])

            ans.append(res)

        elif record[i][0] == 'Leave':

            res= '%s님이 나갔습니다.' %(name[record[i][1]])

            ans.append(res)

    return ans

**#후보키**

관계 데이터베이스에서 릴레이션(Relation)의 튜플(Tuple)을 유일하게 식별할 수 있는 속성(Attribute) 또는 속성의 집합 중, 다음 두 성질을 만족하는 것을 후보 키(Candidate Key)라고 한다.

유일성(uniqueness) : 릴레이션에 있는 모든 튜플에 대해 유일하게 식별되어야 한다.

최소성(minimality) : 유일성을 가진 키를 구성하는 속성(Attribute) 중 하나라도 제외하는 경우 유일성이 깨지는 것을 의미한다. 즉, 릴레이션의 모든 튜플을 유일하게 식별하는 데 꼭 필요한 속성들로만 구성되어야 한다.

제이지를 위해, 아래와 같은 학생들의 인적사항이 주어졌을 때, 후보 키의 최대 개수를 구하라.

이런 조건을 갖고 문제를 풀려 했는데, 처음에는 모든 조합을 슬라이싱등을 이용해서 구하는 등 조건의 조합의 개수별로 매번 set()를 검사해 주어야 하는 줄 알았다. 그런데 그게 아니었고 오히려 처음에 모든 조건의 번호로 조합을 구해서 답을 찾는게 효과적이었다.

DFS를 이용해서 모든 번호의 조합을 구해 주었고, 그 이후에 set()을 이용해서 이 조건들로 모든 사람이 구분이 되는지 확인 해준 뒤에 check()함수를 이용해서 해당 조건의 조합에 들어있는 모든 조건이 포함된 다른 조합을 제거해 주었다.

import copy

def **check**(j,i):

    a = 0

    for k in **range**(**len**(i)):

        if i[k] in j:

            a += 1

    if a == **len**(i):

        return False

    else:

        return True

def **DFS**(v, ans,before,b):

    if v == b:

        if ans:

            before.append(ans)

    elif v < b:

        DFS(v+1, ans, before,b)

        DFS(v+1, ans + str(v),before,b)

    return before

def **solution**(relation):

    a,b = **len**(relation), **len**(relation[0])

    before = []

    DFS(0,'',before, b)

    before = **sorted**(before, key = lambda x: (**len**(x),x))

    answer = 0

    while before:

        i = before.pop(0)

        file = set()

        for j in **range**(a):

            res = ''

            for l in **range**(**len**(i)):

                res += relation[j][int(i[l])]

            file.add(res)

        if **len**(file) == a:

            answer += 1

            ans = []

            for j in before:

                if check(j, i):

                    ans.append(j)

            before = copy.deepcopy(ans)

**#방금 그곡**

방금그곡 서비스에서는 음악 제목, 재생이 시작되고 끝난 시각, 악보를 제공한다. 네오가 기억한 멜로디와 악보에 사용되는 음은 C, C#, D, D#, E, F, F#, G, G#, A, A#, B 12개이다. 각 음은 1분에 1개씩 재생된다. 음악은 반드시 처음부터 재생되며 음악 길이보다 재생된 시간이 길 때는 음악이 끊김 없이 처음부터 반복해서 재생된다. 음악 길이보다 재생된 시간이 짧을 때는 처음부터 재생 시간만큼만 재생된다. 음악이 00:00를 넘겨서까지 재생되는 일은 없다. 조건이 일치하는 음악이 여러 개일 때에는 라디오에서 재생된 시간이 제일 긴 음악 제목을 반환한다. 재생된 시간도 같을 경우 먼저 입력된 음악 제목을 반환한다. 조건이 일치하는 음악이 없을 때에는 `(None)`을 반환한다. 조건에 맞는 음악 제목을 출력하시오

처음에는 #이 있는 것을 잊고 그냥 m과 일치하는 부분이 있으면 출력하는 것으로 풀었었다. 그러나 C#과 C등 #이 붙은 것이 분명히 다르기 때문에 #이 붙은 음들은 소문자로 바꿔서 비교를 해 주었고, 재생시간에 맞춰서 문자열도 변경을 해주었다. 마지막에는 “(None)”으로 조건에 일치하는 음악이 없는 경우에 출력을 하도도록 함으로서 코드 작성을 마무리 하였다. 이 문제는 solution()함수와 #을 변형해 주는 2개의 함수로 이루어졌다.

def **changesharp**(a):

    change = []

    for i in **range**(**len**(a)):

        if a[i] == '#':

            change[-1] = change[-1].lower()

        else:

            change.append(a[i])

    return (''.join(change))

def **solution**(m, musicinfos):

    for i in **range**(**len**(musicinfos)):

        musicinfos[i] = list(**map**(str, musicinfos[i].split(',')))

    runningtime = []

    for i in **range**(**len**(musicinfos)):

        a1,a2,b1,b2 = int(musicinfos[i][0][:2]), int(musicinfos[i][0][3:]),int(musicinfos[i][1][:2]), int(musicinfos[i][1][3:])

        runningtime.append((b2-a2)+60\*(b1-a1))

    m = changesharp(m)

    for i in **range**(**len**(musicinfos)):

        musicinfos[i][3] = changesharp(musicinfos[i][3])

    for i in **range**(**len**(runningtime)):

        a,b = runningtime[i], **len**(musicinfos[i][3])

        if a < b:

            musicinfos[i][3] = (musicinfos[i][3])[:a]

        else:

            c,d = a//b,a%b

            musicinfos[i][3] = musicinfos[i][3]\*c + (musicinfos[i][3])[:d]

    answer,idx = 0,-1

    for i in **range**(**len**(runningtime)):

        if m in musicinfos[i][3]:

            if **len**(m) == runningtime[i]:

                return musicinfos[i][2]

            if answer < runningtime[i]:

                answer = runningtime[i]

                idx = i

    if idx == -1:

        return "(None)"

    else:

        return musicinfos[idx][2]

**#[3차]n진수 게임**

자신이 말해야 하는 숫자를 스마트폰에 미리 출력해주는 프로그램을 만들려고 한다. 튜브의 프로그램을 구현하라.

진법 n, 미리 구할 숫자의 갯수 t, 게임에 참가하는 인원 m, 튜브의 순서 p 가 주어진다.

여기서 10~15까지의 자연수는 각각 대문자 A~F로 표현한다고 했는데, 처음에는 이게 16진법에만 해당하는 줄 알았다. 진법 표현을 할 떄는 나머지를 바꾼 문자열에 붙이고 계속 몫으로 값을 갱신해야 한다. 그렇기 때문에 T = ‘123456789ABCDEF’이라는 문자열을 만들어서 숫자를 n으로 나누어서 나온 나머지를 index로서 T에서의 값을 더하고 나눌 때의 몫으로 갱신할 수 있도록 했다. 이렇게 했더니 답이 나왔고, 16진법에만 알파벳이 적용되는 것이 아니었다는 점이 중요하다.

몫, 나머지로 갱신 할 때에 divmod의 내장 함수를 이용할 수 있는데, divmod(a,b)를 하면 (몫, 나머지)가 순서대로 출력 된다.

def **change**(n,k):

    T = '0123456789ABCDEF'

    res = ''

    if k == 0:

        return '0'

    while True:

        if k == 0:

            break

        res = T[k%n]+ res

        k = k//n

    return res

def **solution**(n,t,m,p):

    length = p+(t-1)\*m

    k,ans = 0,''

    while True:

        if **len**(ans) >= length:

            break

        else:

            ans += change(n,k)

            k += 1

    res = ''

    for x in **range**(p-1, length,m):

        res += ans[x]

    return res

**#[3차]파일명 정렬**

* 파일명은 우선 HEAD 부분을 기준으로 사전 순으로 정렬한다. 이때, 문자열 비교 시 대소문자 구분을 하지 않는다. MUZI와 muzi, MuZi는 정렬 시에 같은 순서로 취급된다. 파일명의 HEAD 부분이 대소문자 차이 외에는 같을 경우, NUMBER의 숫자 순으로 정렬한다. 9 < 10 < 0011 < 012 < 13 < 014 순으로 정렬된다. 숫자 앞의 0은 무시되며, 012와 12는 정렬 시에 같은 같은 값으로 처리된다. 두 파일의 HEAD 부분과, NUMBER의 숫자도 같을 경우, 원래 입력에 주어진 순서를 유지한다. MUZI01.zip과 muzi1.png가 입력으로 들어오면, 정렬 후에도 입력 시 주어진 두 파일의 순서가 바뀌어서는 안 된다.

이 문제를 풀기 위해서는 너무나도 쉽게 알 수 있던 점이 우선 분리를 하기 위한 함수를 만들어야 하고, 그 기준이 숫자 리스트에 해당 값이 있는지의 여부였다. 그리고 함수를 이용해서 문자열을 분리 한 후에 lambda x함수를 이용하여 우선은 .lower()로 바꾸어 알파벳 순으로, 이후에는 0을 무시한 숫자의 오름차순이므로 int()를 붙여서 정렬을 했다. 마지막으로 ‘’.join()을 이용해 문자열을 합쳐서 리스트에 넣어 출력했다. 이문제의 경우 slicing()이 제일 까다로웠는데, 무엇보다도 마지막TAIL이 없을 경우가 까다로웠는데, 끝까지 num에 속하지 않는 값이 없으면 TAIL이 없으므로 len(file) == 1이므로 이때 전체 남은 문자열을 모두 리스트에 넣어 주었다.

def **divide**(string):

    num = '0123456789'

    file = []

    k = 0

    for i in **range**(**len**(string)):

        if string[i] in num:

            file.append(string[:i])

            k = i+1

            break

    for j in **range**(k, **len**(string)):

        if string[j] not in num:

            file.append(string[k-1:j])

            file.append(string[j:])

            break

    if **len**(file)==1:

        file.append(string[k-1:**len**(string)])

    return file

def **solution**(files):

    file = []

    ans = []

    for i in **range**(**len**(files)):

        file.append(divide(files[i]))

    file = **sorted**(file, key = lambda x: (x[0].lower(), int(x[1])))

    for i in **range**(**len**(file)):

        ans.append(''.join(file[i]))

    return file