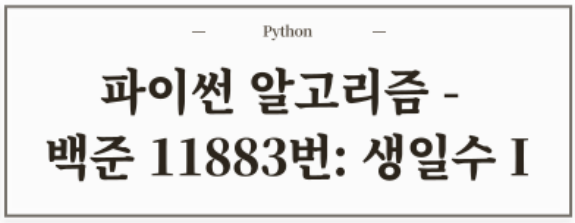

생일수 I

골드 III

시간 제한	메모리 제한	제출	정답	맞힌 사람	정답 비율
1 초	256 MB	485	146	114	30.978%

문제

승현이는 1998년 3월 5일에 태어났습니다. 그래서 승현이는 자신의 생년월일에 있는 수들 중에서 **3**, **5**, **8**을 굉장히 좋아합니다. 이에 승현이는 10진수로 표현했을 때 3, 5, 8로만 구성되어 있는 자연수를 '**생일수**'로 부르기로 했습니다. 예로 들어 3, 533, 858553은 생일수이지만, 3518, 1642, 88808은 생일수가 아닙니다.

여러분은 임의의 자연수 N이 주어졌을 때, 10진법으로 나타냈을 때 각 자리수의 합이 N이 되는 가장 작은 생일수를 찾아서 출력하는 프로그램을 작성해야 합니다. 문제가 너무 간단한가요?ㅋㅋ

입력

첫째 줄에 테스트 케이스의 수 T (1 ≤ T ≤ 100)가 주어집니다. 이후 T개 줄에 자연수 N이 주어집니다. (1 ≤ N ≤ 1,000,000)

출력

각 테스트 케이스에 대해, 한 줄에 하나씩 각 자리수의 합이 N이 되는 가장 작은 생일수 하나를 출력합니다. 만약 존재하지 않는다면, -1을 출력합니다.

다이나믹 프로그래밍, 탐욕법 관련 문제입니다. 요새 다이나믹하고 탐욕스럽게 먹고 자느라 알고리즘 머리가 초기화된 상태라서, 문제 풀기가 너무 힘들었습니다.

이런 알고리즘 문제를 풀 때마다 수능 수학 생각이 납니다. 둘 다 평소 비슷한 유형의 고난도 문제를 꾸준히 풀고 정리해두면 수월하게 풀립니다. 원래도 못하다가 갑자기 도전하려고 하면 벽에 주먹치기 하는 느낌이 들고요.

이론은 겉핥기 식으로 배우고 관련하여 적당히 쉬운 문제만 풀어버릇하면 나중에 실전에서 어찌어찌 문제에 손 대는 것까지는 가능해도 특정 지점을 넘기는 게 잘 안됩니다. 답이나 결정적인 힌트를 알고 나면 허무하리만치 쉽다 보니, 스스로 실력을 가늠하기도 어려워지고요.

여하튼 오랜만에 열심히 짱구 굴려 문제를 풀었고, 아래 풀이 과정을 남겨보려 합니다.

문제를 세 줄 요약해 봅시다.

1. 10진수로 표현했을 때 3, 5, 8로만 구성되어 있는 자연수를 '생일수'라고 하자.
2. N을 입력받으면 각 자릿수의 합이 N이 되는 가장 작은 생일수를 출력하는 프로그램을 작성하라.
3. 존재하지 않으면 -1을 출력하라.

처음에는 무작정 풀어봤습니다. N을 입력받으면, 숫자별로 그 숫자를 만들기 위해 사용할 수 있는 최대 개수를 구합니다. 예를 들어 100을 만들기 위해 3은 최대 33개, 5는 20개, 8은 12개까지 사용 가능합니다. 이제 0~33개의 3, 0~20개의 5, 0~12개의 8을 더하여 100이 나올 때까지 반복하고, 구한 숫자로 생일수를 조합하면 됩니다.

```
def get_min_birthnum(n):
    if n < 3:
        return 0

    max_3 = n // 3
    max_5 = n // 5
    max_8 = n // 8

    cnt_358 = []

    for i in range(max_3 + 1):
        for j in range(max_5 + 1):
            for k in range(max_8 + 1):
                if 8*i + 5*j + 3*k == n:
                    cnt_358 = [k, j, i]

    if cnt_358 is None:
        return -1
    else:
        return '3'*cnt_358[0] + '5'*cnt_358[1] + '8'*cnt_358[2]

t = int(input())
for _ in range(t):
    n = int(input())
    print(get_min_birthnum(n))
```

이런 궁금증이 생길 수 있습니다. N이 100일 때 5를 20개 사용하여 생일수를 만들 수도 있고, 5 4개와 8 10개로 만들 수도 있는데 더 짧은 수를 선택하는 로직을 따로 넣지 않아도 되는지 말이죠.

그 부분은 반복문 배치로 자연스레 해결할 수 있습니다. 위 코드에서는 중첩 반복문의 안쪽으로 갈수록 큰 수 반복문이 있도록 배치했습니다. 따라서 큰 수를 많이 쓰는 경우를 우선적으로 순회하게 됩니다. 큰 수를 최대한 많이 쓸수록 자릿수가 짧아지고, 가장 작은 생일수에 가까워집니다.

생일수에 사용될 3, 5, 8의 개수는 각각 cnt_358[0], cnt_358[1], cnt_358[2]에 담깁니다. 앞자리로 갈수록 작은 수가 배치되어야 가장 작은 수를 만들 수 있습니다. 따라서 코드와 같이 3, 5, 8을 각각 개수만큼 줄지어 놓은 수가 가장 작은 생일수가 됩니다.

시간제한이 없다면 썩 나쁘지 않은 접근인데, 연산 횟수가 너무 많습니다. 문제의 제한 시간은 1초이고, 파이썬은 1초에 2천만 번의 연산이 가능한 것으로 알려져 있습니다. N이 최댓값인 1000000일 때를 생각해 봅시다. 3은 333334, 5는 200001, 8은 125001가지 케이스가 나오고, 이들을 모두 곱하면 최대 연산 횟수가 됩니다. 대충 계산해 봐도 2천만은 훌쩍 넘길 것 같습니다.

그래서 1차원 DP를 떠올립니다.

```python
def get_min_birthnum(digit_sum):
    dp = [float('inf')] * (digit_sum + 1)
    dp[0] = 0

    for i in range(3, digit_sum + 1):
        dp[i] = min(dp[i], dp[i - 3] + 1)
    for i in range(5, digit_sum + 1):
        dp[i] = min(dp[i], dp[i - 5] + 1)
    for i in range(8, digit_sum + 1):
        dp[i] = min(dp[i], dp[i - 8] + 1)

    if dp[digit_sum] == float('inf'):
        return -1

    number = ''
    current_sum = digit_sum

    while current_sum > 0:
        if current_sum >= 3 and dp[current_sum] == dp[current_sum - 3] + 1:
            number += '3'
            current_sum -= 3
        elif current_sum >= 5 and dp[current_sum] == dp[current_sum - 5] + 1:
            number += '5'
            current_sum -= 5
        else:
            number += '8'
            current_sum -= 8

    return int(number)

t = int(input())
for _ in range(t):
    n = int(input())
    print(get_min_birthnum(n))
```

dp[i]는 각 자릿수의 합이 i가 되는 가장 작은 생일수의 자릿수를 뜻합니다. 1차원 배열을 N 만큼의 길이로 생성하고, 여기에 무한을 채워 넣습니다.

이제 반복문을 통해 dp의 값을 갱신합니다. min(dp[i], dp[i - 3] + 1)의 경우, 현재 저장된 값과 3을 넣기 전 생일수의 자릿수에 1을 더한 값 중 더 작은 값을 비교하는 부분입니다. 처음에는 dp가 무한으로만 채워져 있으므로, dp[3] = 1, dp[6] = 2, dp[9] = 3, ... , dp[3*x] = x (x는 자연수)와 같은 형태로 dp가 채워지게 될 것입니다. 이제 보니 첫 반복문은 간격을 3으로 두는 게 낫겠네요. 3의 배수가 아닌 인덱스는 어차피 무한과 무한의 비교일 테니까요.

이후 5, 8 만큼 차이가 나는 인덱스와 비교하는 반복문을 거치고 나면, dp가 모두 채워집니다. 이 과정을 거친 이후에도 무한이 담겨있다면 생일수가 존재하지 않는 것이니 -1을 리턴하고요.

이제 dp[n]으로 가장 작은 생일수의 자릿수를 알 수 있습니다. 마지막으로 역산하여 생일수의 앞자리부터 채워나갑니다. 앞자리는 가장 작은 숫자부터 채워야 하니 3, 5, 8 순서대로 비교해 줍니다.

current_sum >= 3 and dp[current_sum] == dp[current_sum - 3] + 1이라는 조건을 예로 설명해 보겠습니다. current_sum이 9일 때, dp[9]는 3입니다. 해당 생일수는 '333'이고요. dp[6]은 2로 '33'이고 여기에 1을 더하면 dp[9]와 같으니 number의 끝에는 '3'이 추가됩니다. number에 떼어 붙인 숫자인 3만큼 current_sum은 감소하고요.

current_sum이 20일 때, dp[20]은 4입니다. 해당 생일수는 '5555'이고요. dp[17]은 4, 해당 생일수는 '3338'입니다. dp[17]에 해당하는 4에 1을 더한 값은 dp[20]에 해당하는 4와 일치하지 않습니다. 따라서 해당 조건문은 통과하고, 아래 조건문에 걸려 해당 생일수의 앞자리 '5'가 number의 끝에 붙게 되겠죠.

풀어서 설명해보니 복잡하네요. current_sum을 줄여 나가며 dp[current_sum]에 해당하는 생일수의 앞자리를 number의 끝에 떼어 붙여 나가는 과정이라고 이해하시면 됩니다. 저는 문자열로 처리했지만, 숫자로도 가능합니다. number를 0으로 초기화하고, 문자열을 더하는 대신 10을 곱하고 해당 숫자를 더해주면 됩니다.

응~ 시간 초과~
앞서 코드를 보면 알겠지만, 매번 dp 배열을 새로 만드는 것이 가장 잘못됐습니다. dp 배열을 미리 N 최대 범위까지 만들어 놓고, 입력할 때마다 리턴해주는 게 훨씬 좋아 보입니다. 앞쪽 반복문도 상당히 비효율적으로 보이고요.

여기까지는 할 만했는데 이다음부터가 어렵더라고요. 3, 5, 8을 2, 3의 합으로 쪼개고 합치는 방법으로 접근해 봤지만 잘 안됐고요. 좀 막막했는데, 백준 질문 게시판에서 힌트를 얻었습니다.

1차원으로 풀면 안되는 이유..

```
20
WA:3388
AC:5555
```

참고로 AC는 ACcepted, 정답을 뜻하는 말이고 WA는 Wrong Answer, 오답을 뜻하는 말입니다. 제 코드에 해당하지 않는 반례보다 '1차원'이라는 말에서 힌트를 얻었습니다.

어차피 공간 복잡도, 메모리 제한은 충분하니 dp에 최소 자릿수뿐만 아니라 해당 생일수 맨 앞자리 숫자도 저장하기로 했습니다. 반복문도 줄여보고요. dp는 미리 만들어 놓은 뒤 입력이 들어올 때마다 숫자 역산해서 만들고 출력하기

로 했습니다.

그렇게 완성된 코드입니다.

```
N_MAX = 1000000
BIG_NUM = 100

dp = [[None] * 2 for _ in range(N_MAX + 1)]

dp[1][0] = BIG_NUM
dp[2][0] = BIG_NUM
dp[4][0] = BIG_NUM
dp[7][0] = BIG_NUM

dp[0][0] = 0
dp[3][0] = 1
dp[3][1] = 3
dp[5][0] = 1
dp[5][1] = 5
dp[6][0] = 2
dp[6][1] = 3

for i in range(8, N_MAX + 1):
    min_num = min(dp[i - 8][0], dp[i - 5][0], dp[i - 3][0])
    if min_num == dp[i - 3][0]:
        dp[i][0] = dp[i - 3][0] + 1
        dp[i][1] = 3
    elif min_num == dp[i - 5][0]:
        dp[i][0] = dp[i - 5][0] + 1
        dp[i][1] = 5
    else:
        dp[i][0] = dp[i - 8][0] + 1
        dp[i][1] = 8

def get_min_birthnum(n):
    if n in [1, 2, 4, 7]:
        return -1

    ret = ''
    while n > 0:
        ret += str(dp[n][1])
        n -= dp[n][1]
    return ret

t = int(input())
for _ in range(t):
    n = int(input())
    print(get_min_birthnum(n))
```

dp는 dp[i][0] = j, dp[i][1] = k에서 자릿수의 합이 i인 생일수의 최소 자릿수 j, 앞 자릿수 k라고 정의했습니다.

BIG_NUM은 float('inf') 대신 써봤습니다. 어차피 비교했을 때 최솟값으로 선택되지 않게 2보다 크기만 하면 되니까, 그냥 적당히 큰 수 넣고 상수처럼 썼습니다.

이전에 작성했던 3, 5, 8 각각의 반복문을 묶기 위해 0~7사이의 dp 값을 별도로 초기화해주었습니다. 세련된 방식은 아니지만 일단 풀자는 생각으로...

반복문은 8에서부터 시작하는 반복문 하나로 합쳤습니다. 16을 예로 들어보겠습니다. 16에서 3, 5, 8을 뺀 값인 13, 11, 8일 때의 값 중 최솟값을 고릅니다. 8일 때 1로 가장 작으므로 dp[16][0]은 dp[8][0]에 1을 더한 값인 2가 되고, dp[8][1]은 8이 됩니다. 나중에 역산하여 n=16일 때 생일수를 구한다면 str(dp[16][1]) + str(dp[8][1])이 될 테고요. 앞자리를 구하는 과정이니, 자릿수가 같은 경우 더 작은 수가 선택되어야 합니다. 그래서 조건문은 3, 5, 8 순입니다.

풀이를 위해 dp 표를 그려봤는데, 여기까지 와서야 1, 2, 4, 7을 제외한 모든 자연수 N에 대하여 자릿수 총합이 N인 생일수가 존재한다는 것을 알게 되었습니다. 증명이라고 하기까진 뭐하지만 아래 그림을 보시면 이해가 되실 겁니다.

dp[i][0] = j
dp[i][1] = k
자릿수의 합이 i인 생일수의 최소 자릿수 j, 앞 자릿수 k

i	j	k
0	X	X
1	X	X
2	X	X
3	1	3
4	X	X
5	1	5
6	2	3
7	X	X
8	1	8
9	3	3
10	2	5
11	2	3
12	4	3
13	2	5
14	3	3
15	3	5
16	2	8

dp[i][0] = j
dp[i][1] = k
자릿수의 합이 i인 생일수의 최소 자릿수 j, 앞 자릿수 k

i	j	k
0	X	X
1	X	X
2	X	X
3	1	3
4	X	X
5	1	5
6	2	3
7	X	X
8	1	8
9	3	3
10	2	5
11	2	3
12	4	3
13	2	5
14	3	3
15	3	5
16	2	8

자릿수의 총합이 N인 생일수는 자릿수의 총합이 N-3, N-5, N-8인 생일수가 하나라도 있다면 존재합니다. 예를 들어 만약 총합이 N-3인 생일수가 존재한다면 그 앞에 3만 붙여주면 되니까요. 그런데 8~16까지는 생일수가 모두 존재하므로, 이후의 모든 수는 생일수가 존재할 것입니다.

여하든 많이 바뀐 것 같지만 사실 이전 코드와 달라진 건 dp를 함수 밖으로 뺀 것, 반복문을 세 개에서 하나로 줄인 것 정도입니다. 덕분에 시간 초과 문제를 해결하고 통과했습니다.

그러나 좋은 정답이라 보긴 어렵습니다. 풀고 나서 다른 분 풀이를 찾아보니, dp에 숫자를 만들기 위한 3, 5, 8의 개수를 메모해두면 제 코드에서 get_min_birthnum()에 해당하는 역산 과정을 단축시킬 수 있더라고요. 제 엉성한 삽질 과정을 참고하셔서 더 나은 풀이에 도전하셨으면 좋겠습니다.

혹시 replit 비롯한 클라우드 IDE를 알고리즘 문제 풀이에 이용하시는 분이 있다면, 여기서의 시간 측정은 신뢰하지 않으시길 바랍니다. 애초에 속도가 상당히 느린 편이라 정확하지 않습니다. 조금 귀찮더라도 비교적 빠른 로컬 환경에서 시간 측정을 하거나 파이썬 위키의 시간 복잡도 문서를 참고하여 어림 계산하는 편이 낫겠습니다.