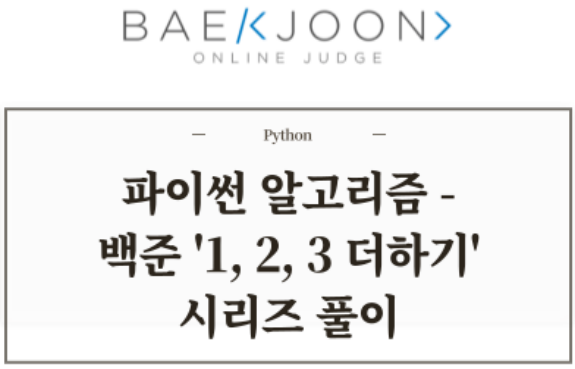
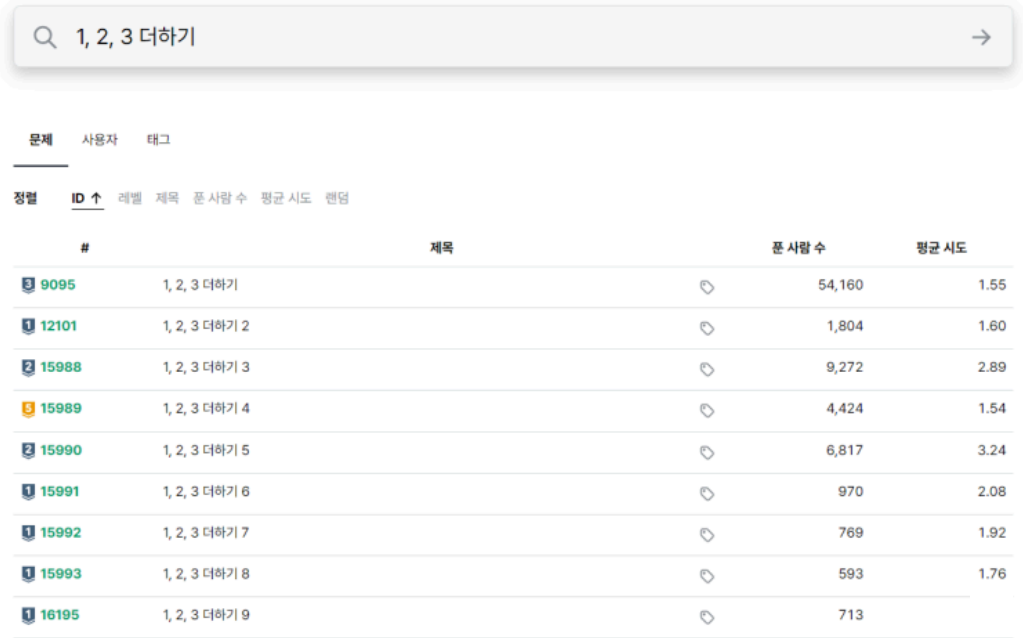


근래 다이나믹 프로그래밍 위주로 알고리즘 문제 풀이를 하고 있습니다. 규칙성을 발견하고 점화식만 잘 세우면 어려운 문제도 별다른 배경지식 없이 풀이 가능하기 때문입니다. 조금 더 DP 관련 문제 풀이에 익숙해진 후 정렬, 그래프 관련 알고리즘 공부를 시작할 예정입니다.

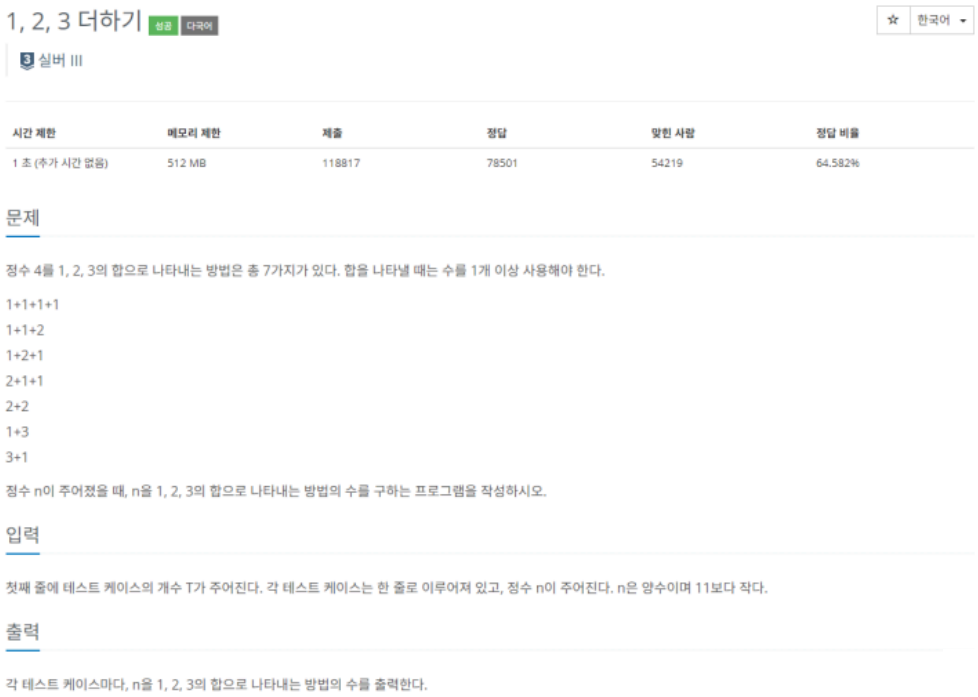


#	제목	푼 사람 수	평균 시도
9095	1, 2, 3 더하기	54,160	1.55
12101	1, 2, 3 더하기 2	1,804	1.60
15988	1, 2, 3 더하기 3	9,272	2.89
15989	1, 2, 3 더하기 4	4,424	1.54
15990	1, 2, 3 더하기 5	6,817	3.24
15991	1, 2, 3 더하기 6	970	2.08
15992	1, 2, 3 더하기 7	769	1.92
15993	1, 2, 3 더하기 8	593	1.76
16195	1, 2, 3 더하기 9	713	

오늘은 총 10개의 '1, 2, 3 더하기' 시리즈 문제를 풀이해 보려 합니다. 대표적인 DP 문제들인데, 한 번 풀이할 때 같이 푸는 것을 추천드립니다. 서로 비슷한 문제가 많다 보니 코드 재활용하기 좋거든요.

10문제 풀이를 한 번에 적다 보니 본문 길이가 좀 깁니다. 그래도 순서대로 작성했으니 필요한 문제 풀이만 찾아보시는 데 큰 어려움은 없으리라 생각합니다. 각설하고 시작해 보겠습니다!

▶ 9095번: 1, 2, 3 더하기



1, 2, 3 더하기 성공 다국어

☆ 한국어

실버 III

시간 제한	메모리 제한	제출	정답	맞힌 사람	정답 비율
1 초 (추가 시간 없음)	512 MB	118817	78501	54219	64.582%

문제

정수 4를 1, 2, 3의 합으로 나타내는 방법은 총 7가지가 있다. 합을 나타낼 때는 수를 1개 이상 사용해야 한다.

- 1+1+1+1
- 1+1+2
- 1+2+1
- 2+1+1
- 2+2
- 1+3
- 3+1

정수 n이 주어졌을 때, n을 1, 2, 3의 합으로 나타내는 방법의 수를 구하는 프로그램을 작성하시오.

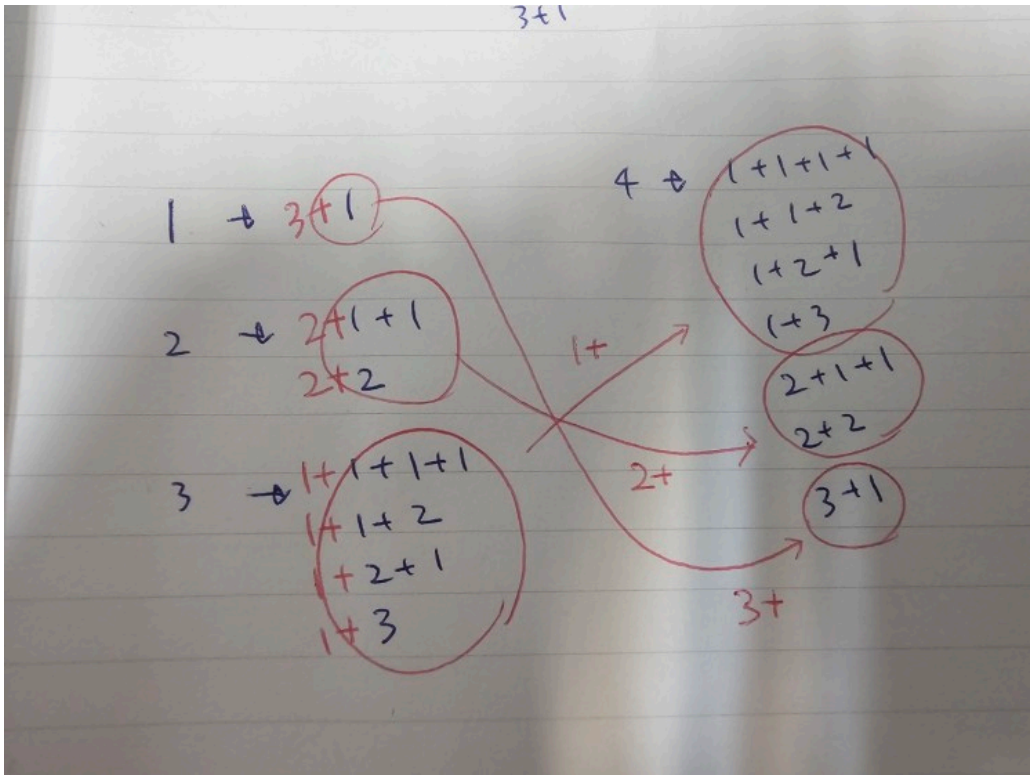
입력

첫째 줄에 테스트 케이스의 개수 T가 주어진다. 각 테스트 케이스는 한 줄로 이루어져 있고, 정수 n이 주어진다. n은 양수이며 11보다 작다.

출력

각 테스트 케이스마다, n을 1, 2, 3의 합으로 나타내는 방법의 수를 출력한다.

DP는 코드 작성보다 규칙성 발견에 오랜 시간을 들여야 합니다. 감이 좋다면 1부터 4까지 1, 2, 3의 합으로 나타냈을 때 규칙성이 보일 것입니다. 저는 감각이 둔해서 그 이상까지 나타내보고 나서야 감이 잡히더라고요. 다만 규칙성을 찾더라도 검증해 볼 필요가 있기 때문에, 너무 가진 않더라도 5, 6 정도까지는 직접 그려서 확인해 보는 것이 좋습니다.



위 그림에 보이는 것과 같이, 4를 1, 2, 3의 합으로 나타내는 방법은 1을 나타내는 방법에 3을 더한 것, 2를 나타내는 방법에 2를 더한 것, 3을 나타내는 방법에 1을 더한 것과 같습니다. 따라서 **4 이상인 자연수 n은 1, 2, 3의 합으로 나타내는 방법의 수는 n-1, n-2, n-3은 1, 2, 3의 합으로 나타내는 방법의 수를 모두 합한 것**과 같습니다.

코드는 다음과 같습니다. 3년 전 풀이라 C로 작성되어 있네요. n의 범위가 11 이하의 자연수로 비교적 좁기 때문에 재귀나 기타 다소 비효율적인 알고리즘으로도 충분히 풀 수 있는 문제입니다.

```c
#include <stdio.h>
#include <stdlib.h>

int ft_sum_123(int n)
{
    int ret, *sum_123 = (int *)malloc(sizeof(int) * (n + 1));

    sum_123[1] = 1;
    sum_123[2] = 2;
    sum_123[3] = 4;
    for (int i = 4; i < n + 1; i++)
        sum_123[i] = sum_123[i - 3] + sum_123[i - 2] + sum_123[i - 1];
    ret = sum_123[n];
    free(sum_123);

    return ret;
    /* 재귀함수로 만들어 똑같은 기능을 하지만 느림
    if (n == 1)
        return 1;
    else if (n == 2)
        return 2;
    else if (n == 3)
        return 4;
    else if (n > 3)
        return ft_sum_123(n - 3) + ft_sum_123(n - 2) + ft_sum_123(n - 1);
    */
}

int main(void)
{
    int T, n;

    scanf("%d", &T);
    for (int i = 0; i < T; i++)
    {
        scanf("%d", &n);
        printf("%d\n", ft_sum_123(n));
    }

    return 0;
}
```

▶ 12101번: 1, 2, 3 더하기 2

1, 2, 3 더하기 2

성공

실버 I

시간 제한	메모리 제한	제출	정답	맞힌 사람	정답 비율
1 초 (추가 시간 없음)	512 MB	3523	2183	1804	62.487%

문제

정수 4를 1, 2, 3의 합으로 나타내는 방법은 총 7가지가 있다. 합을 나타낼 때는 수를 1개 이상 사용해야 한다.

- 1+1+1+1
- 1+1+2
- 1+2+1
- 2+1+1
- 2+2
- 1+3
- 3+1

이를 사전순으로 정렬하면 다음과 같이 된다.

1. 1+1+1+1
2. 1+1+2
3. 1+2+1
4. 1+3
5. 2+1+1
6. 2+2
7. 3+1

정수 n과 k가 주어졌을 때, n을 1, 2, 3의 합으로 나타내는 방법 중에서 k번째로 오는 식을 구하는 프로그램을 작성하시오.

입력

첫째 줄에 정수 n과 k가 주어진다. n은 양수이며 11보다 작고, k는 2^{31}-1보다 작거나 같은 자연수이다.

출력

n을 1, 2, 3의 합으로 나타내는 방법 중에서 사전 순으로 k번째에 오는 것을 출력한다. k번째 오는 식이 없는 경우에는 -1을 출력한다.

n의 범위는 그대로이나, 일반적인 4byte int 양수 범위(2^31 - 1)를 넘지 않는 k가 입력으로 함께 들어옵니다. n을 1, 2, 3의 합으로 나타내는 방법 중에서 사전 순으로 k번째에 오는 것을 출력하고, 없다면 -1을 출력해야 합니다.

앞서 그림으로 나타낸 것을 문자열로 표현하면 됩니다. 경우의 수만 저장하면 됐던 이전 문제보다 조금 복잡해지긴 했지만, 첫 세 항 기반으로 그 이후의 배열을 채워나가는 방식은 동일합니다.

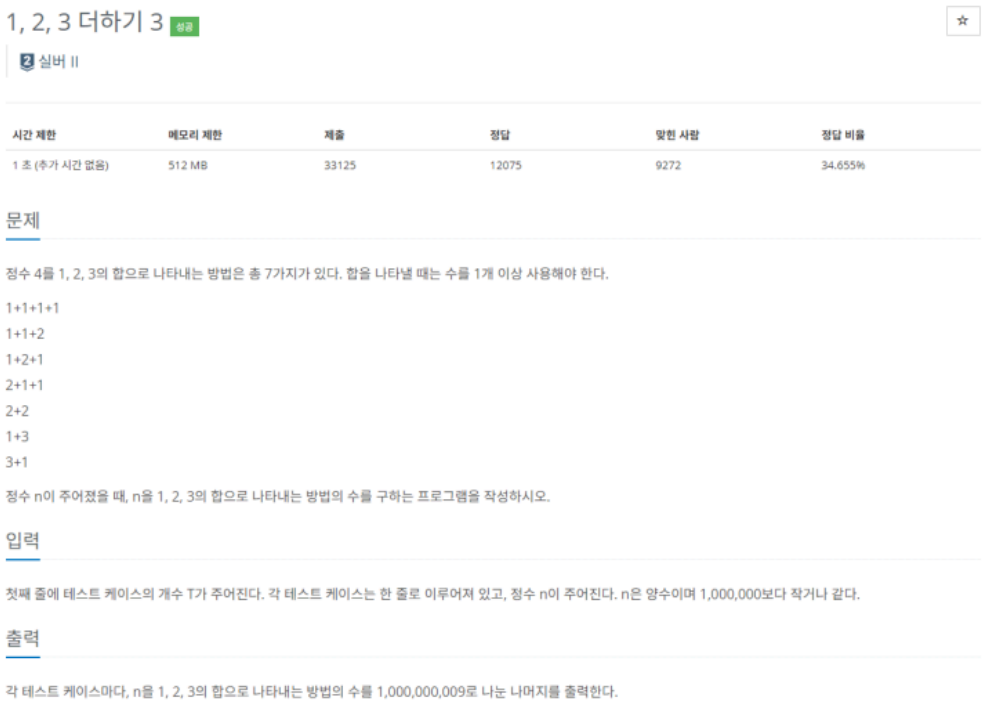
n이 최댓값인 10일 때 274개의 조합이 등장하기 때문에, 배열은 274*274 크기로 만들고 dp[i][j]에는 i+1을 1, 2, 3의 합으로 나타내는 방법 중에서 사전 순으로 k번째에 오는 것이 저장되도록 코드를 작성하였습니다. 나머지 공간에는 방법이 존재하지 않는다는 의미의 -1을 채워 넣고, k로 274 초과되는 값이 입력되면 -1을 리턴하도록 하였습니다. 3개의 while True 문이 조금 더럽긴 하지만... 별도로 함수로 구현하자니 그게 더 별로인 것 같더라고요.

파이썬 코드는 다음과 같습니다.

```
N_MAX = 10**1
K_MAX = 274
dp = [[-1]*K_MAX for _ in range(N_MAX)]
dp[0] = ["1"] + [-1]*(K_MAX - 1)
dp[1] = ["1+1", "2"] + [-1]*(K_MAX - 2)
dp[2] = ["1+1+1", "1+2", "2+1", "3"] + [-1]*(K_MAX - 4)
for i in range(3, N_MAX):
    j = 0
    while True:
        if dp[i-1][j] == -1:
            break
        dp[i][j] = "1+" + dp[i-1][j]
        j += 1
    l = 0
    while True:
        if dp[i-2][l] == -1:
            break
        dp[i][j] = "2+" + dp[i-2][l]
        j += 1
        l += 1
    l = 0
    while True:
        if dp[i-3][l] == -1:
            break
        dp[i][j] = "3+" + dp[i-3][l]
        j += 1
        l += 1

n, m = map(int, input().split())
if m > K_MAX:
    print(-1)
else:
    print(dp[n-1][m-1])
```

▶ 15988번: 1, 2, 3 더하기 3



1, 2, 3 더하기 3

성공

실버 II

시간 제한	메모리 제한	제출	정답	맞힌 사람	정답 비율
1 초 (추가 시간 없음)	512 MB	33125	12075	9272	34.655%

문제

정수 4를 1, 2, 3의 합으로 나타내는 방법은 총 7가지가 있다. 합을 나타낼 때는 수를 1개 이상 사용해야 한다.

- 1+1+1+1
- 1+1+2
- 1+2+1
- 2+1+1
- 2+2
- 1+3
- 3+1

정수 n이 주어졌을 때, n을 1, 2, 3의 합으로 나타내는 방법의 수를 구하는 프로그램을 작성하시오.

입력

첫째 줄에 테스트 케이스의 개수 T가 주어진다. 각 테스트 케이스는 한 줄로 이루어져 있고, 정수 n이 주어진다. n은 양수이며 1,000,000보다 작거나 같다.

출력

각 테스트 케이스마다, n을 1, 2, 3의 합으로 나타내는 방법의 수를 1,000,000,009로 나눈 나머지를 출력한다.

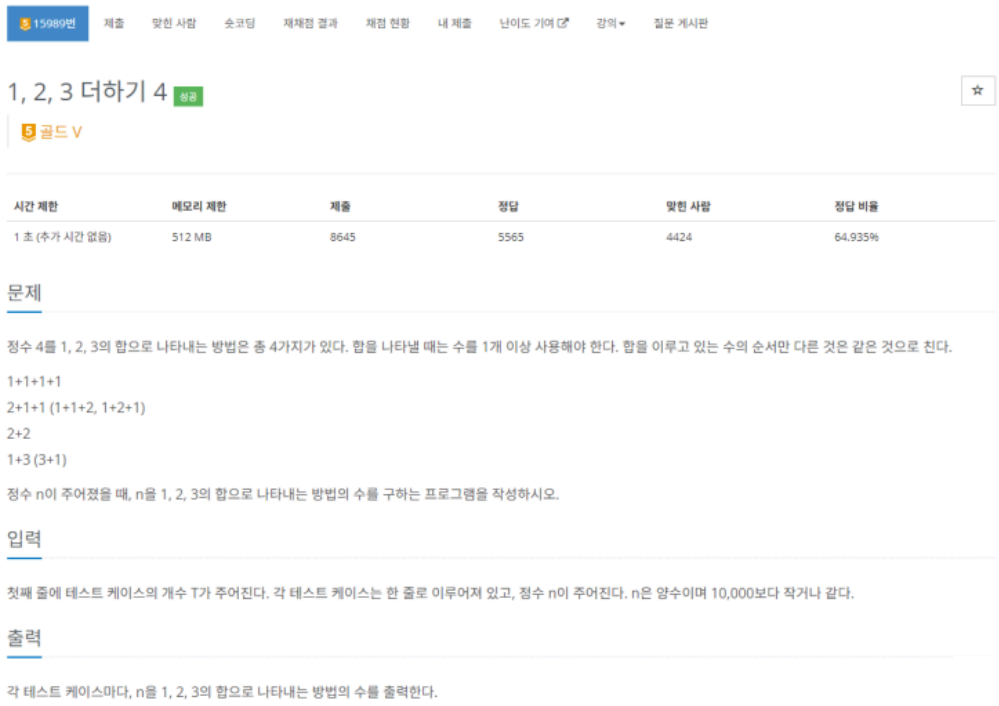
n의 범위가 10^6으로 크게 늘어나고, 테스트 케이스의 개수를 먼저 입력받은 후 n을 그만큼 입력받게 됨에 따라 입/출력 형식이 수정됩니다. 또한 n이 커짐에 따라 방법의 수도 크게 증가하므로 방법의 수는 10^9+9로 나눈 나머지를 출력하도록 수정됩니다.

처음 9095번 문제 풀이의 시간 복잡도를 O(N^2)으로 설정한 것이 아니라면, 그대로 사용하고 값을 저장하는 부분에 나머지 연산만 추가해 주면 됩니다. 참고로 제가 작성한 코드의 시간 복잡도는 O(N)입니다. 코드는 다음과 같습니다.

```
import sys
input = lambda: sys.stdin.readline().rstrip()

N_MAX = 10**6
dp = [1, 2, 4] + [0] * (N_MAX - 3)
for i in range(3, N_MAX):
    dp[i] = (dp[i-3] + dp[i-2] + dp[i-1]) % (10**9 + 9)
t = int(input())
for _ in range(t):
    n = int(input())
    print(dp[n-1])
```

▶ 15989번: 1, 2, 3 더하기 4



제출 맞힌 사람 숏코딩 재채점 결과 채점 현황 내 제출 난이도 기여 강의 질문 게시판

1, 2, 3 더하기 4 성공

골드 V

시간 제한	메모리 제한	제출	정답	맞힌 사람	정답 비율
1 초 (추가 시간 없음)	512 MB	8645	5565	4424	64.935%

문제

정수 4를 1, 2, 3의 합으로 나타내는 방법은 총 4가지가 있다. 합을 나타낼 때는 수를 1개 이상 사용해야 한다. 합을 이루고 있는 수의 순서만 다른 것은 같은 것으로 친다.

- 1+1+1+1
- 2+1+1 (1+1+2, 1+2+1)
- 2+2
- 1+3 (3+1)

정수 n이 주어졌을 때, n을 1, 2, 3의 합으로 나타내는 방법의 수를 구하는 프로그램을 작성하시오.

입력

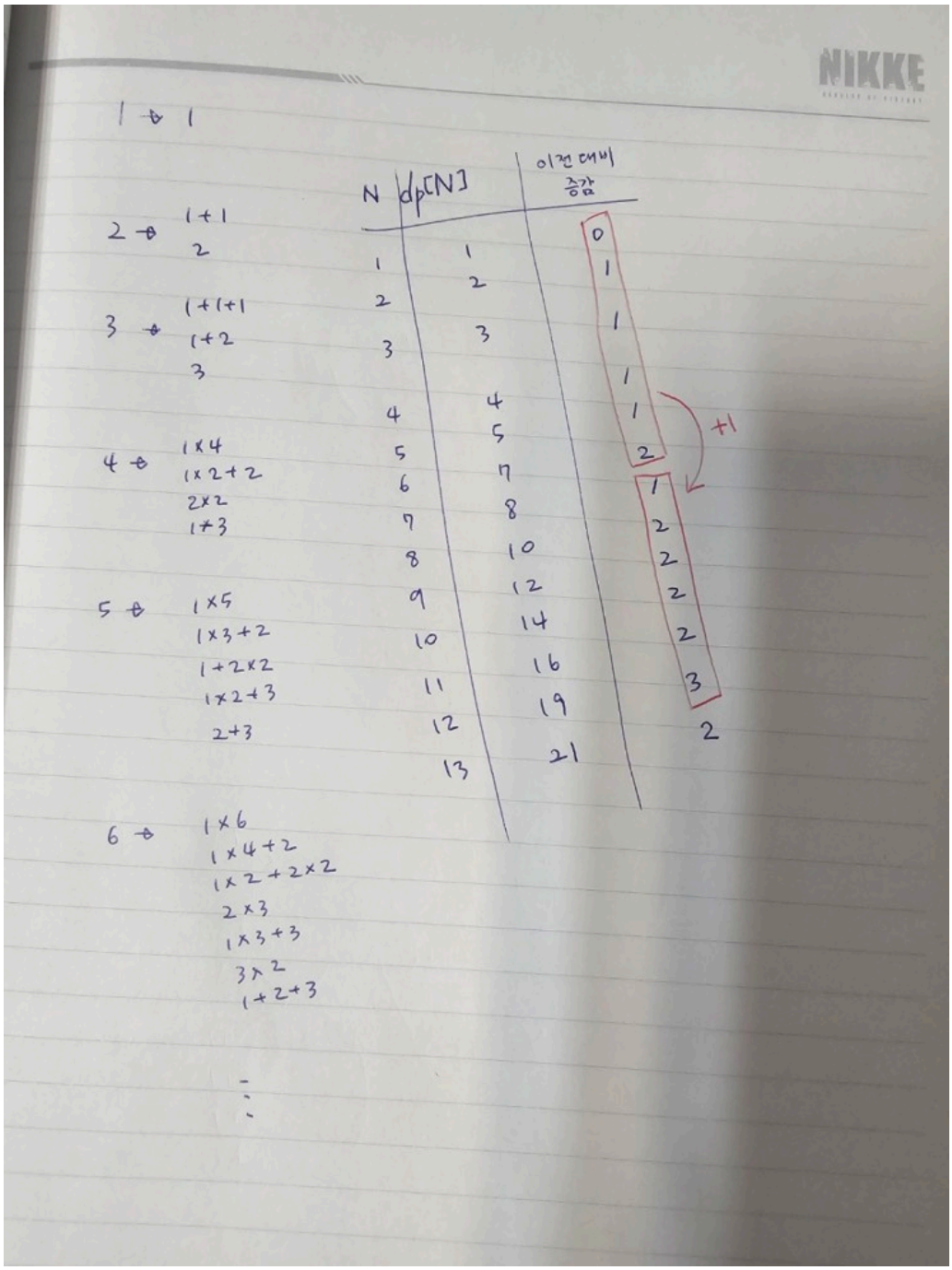
첫째 줄에 테스트 케이스의 개수 T가 주어진다. 각 테스트 케이스는 한 줄로 이루어져 있고, 정수 n이 주어진다. n은 양수이며 10,000보다 작거나 같다.

출력

각 테스트 케이스마다, n을 1, 2, 3의 합으로 나타내는 방법의 수를 출력한다.

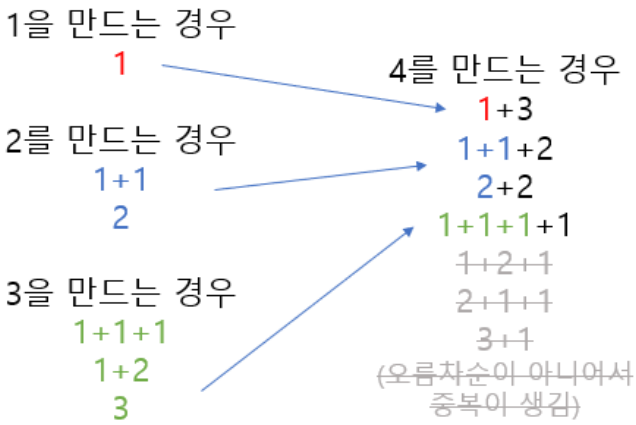
n의 범위가 10^4 이하의 양수로 바뀌었고, 합을 이루고 있는 수의 순서만 다른 것은 같은 것으로 치도록 바뀌었습니다.

이 문제는 운이 좋아서 풀었습니다. 규칙성이 잘 안 보이길래 무작정 수를 적어나갔는데, 6개 주기로 이전 대비 증감 패턴이 0, 1, 1, 1, 1, 2로 반복된다는 사실을 알아냈습니다. 한 주기가 끝날 때마다 패턴 전체에 1만 더해주면 되고요.



합의 형태에서 발견한 규칙이 아니기 때문에, 대다수 분들 풀이와 제 풀이는 조금 다르고 이해하기 쉽지 않습니다. 보통은 이런 규칙성을 발견하고 푸시더라고요.

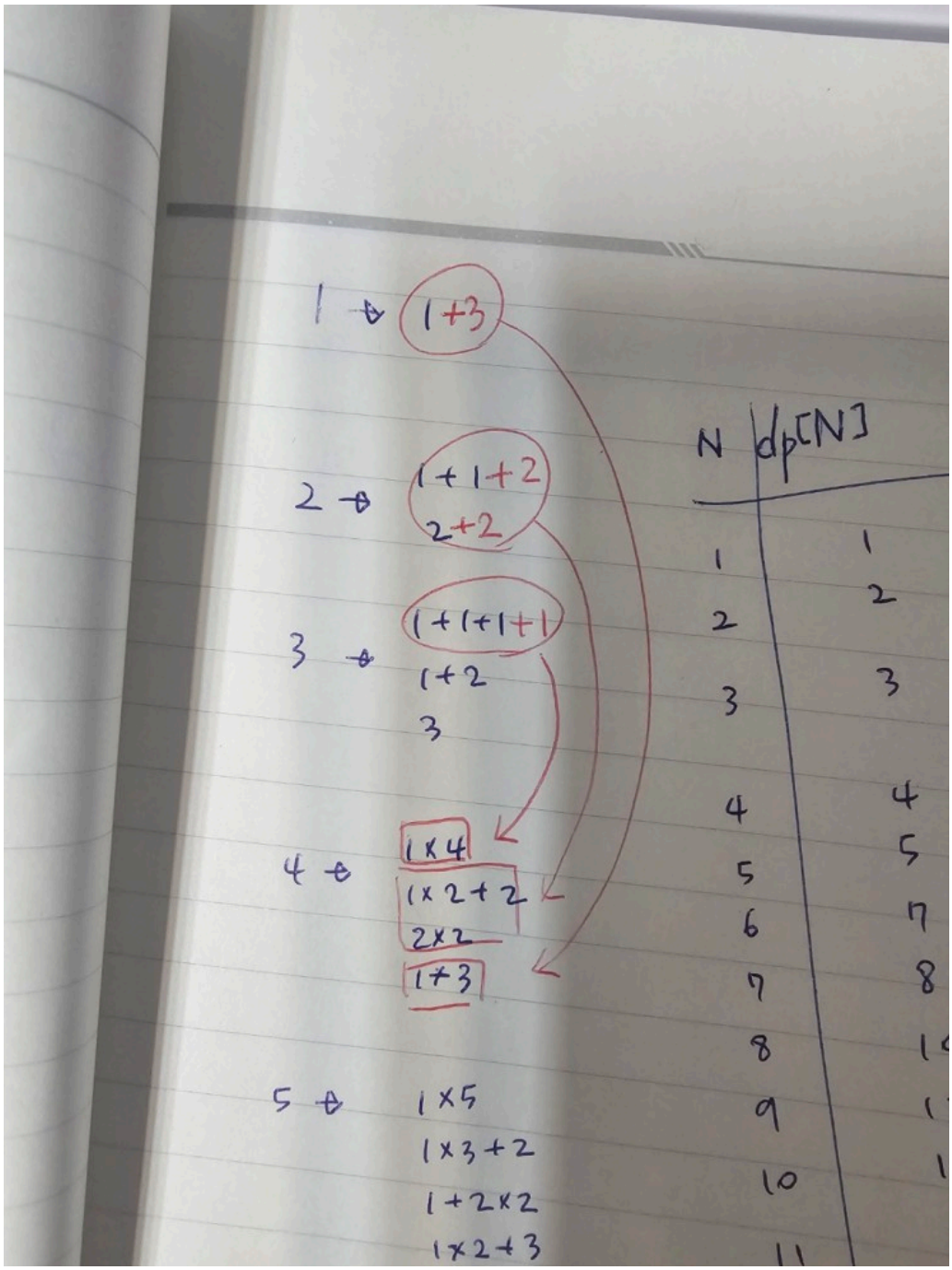
합을 이루고 있는 순서를 오름차순으로 정리한 뒤 1, 2, 3으로 끝나는 조합을 구분하여 보겠습니다. 4 이상의 n을 1, 2, 3의 합으로 나타내는 방법의 수는 **n-1에서 1, 2, 3으로 끝나는 방법의 수와 n-2에서 1, 2로 끝나는 방법의 수와 n-1에서 1로 끝나는 방법의 수의 합**과 같습니다. 이해를 돕고자 아래 사진과 관련 풀이 글 링크를 첨부합니다.

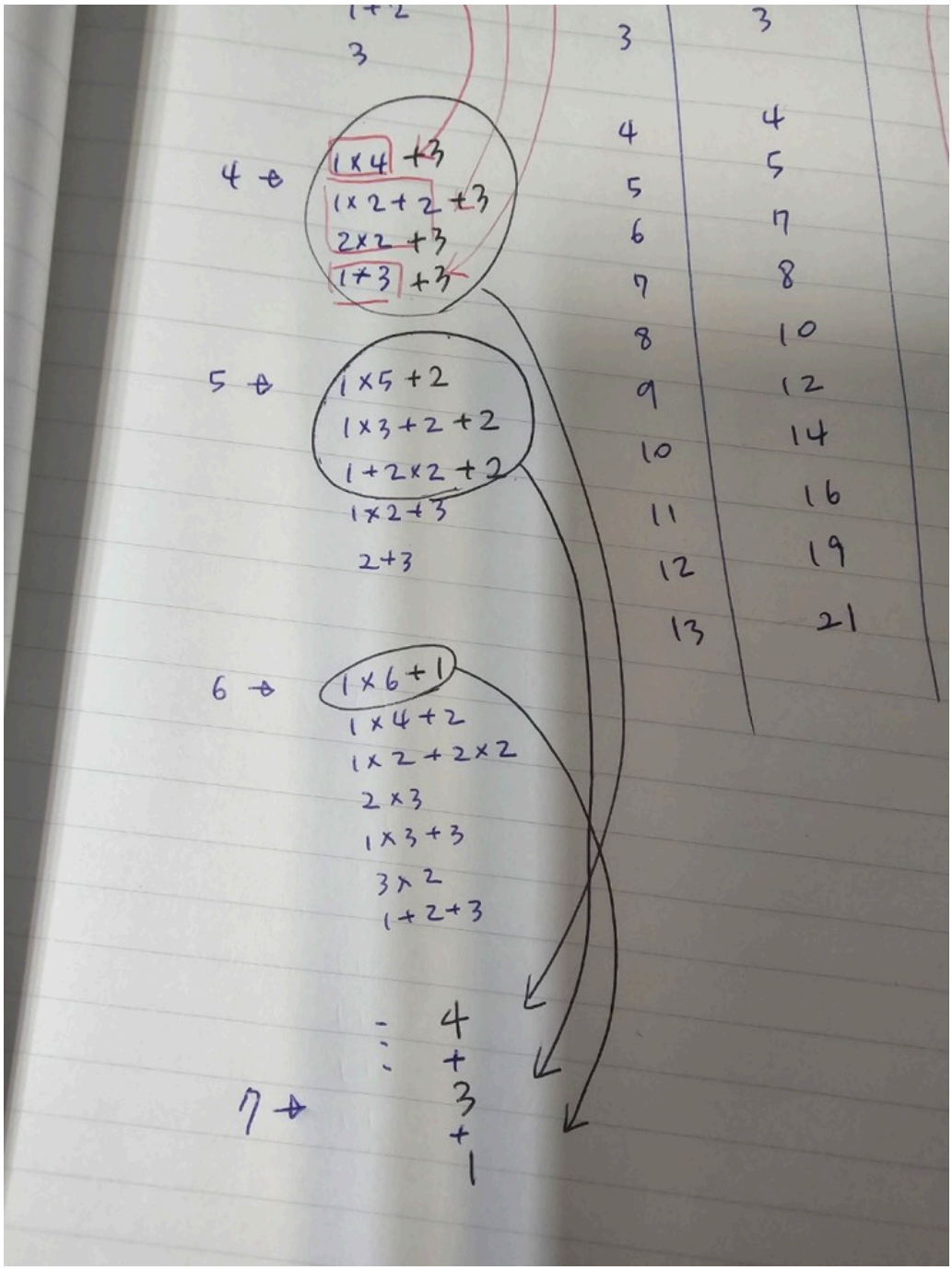


[앝고리즘] 백준 15989 1, 2, 3 더하기 (4) Java

문제 정보플랫폼 : 백준분듀 : Dynamic Programming (동적 프로그래밍…

velog.io


1 → 1+3
2 → 1+1+2
2+2
3 → 1+1+1+1
1+2
3
4 → 1×4
1×2+2
2×2
1+3
5 → 1×5
1×3+2
1+2×2
1×2+3
N dp[N]
1 1
2 2
3 3
4 4
5 5
6 7
7 8
8
9
10
11



이렇게 풀이하는 경우 dp[i][j]는 순서만 다른 것은 같은 것으로 할 때, i+1을 1, 2, 3의 합으로 나타냈을 때 j+1이 끝에 오는 경우의 수로 정의하고 풀면 되겠네요. 제 풀이에서 dp[i][0]은 순서만 다른 것은 같은 것으로 할 때, i+1을 1, 2, 3의 합으로 나타내는 경우의 수이고 dp[i][1]은 이전 대비 증감, 즉 dp[i][0] - dp[i-1][0]입니다. dp[0][1]은 0으로 합니다.

```
N_MAX = 10**4
dp = [[1, 0], [2, 1], [3, 1], [4, 1], [5, 1], [7, 2]] + [[0, 0] for _ in range(N_MAX - 6

for i in range(6, N_MAX):
    dp[i][1] = dp[i-6][1] + 1
    dp[i][0] = dp[i-1][0] + dp[i][1]

t = int(input())
for _ in range(t):
    n = int(input())
    print(dp[n-1][0])
```

▶ 15990번: 1, 2, 3 더하기 5

1, 2, 3 더하기 5

실버 II

시간 제한	메모리 제한	제출	정답	맞힌 사람	정답 비율
1 초 (추가 시간 없음)	512 MB	28719	9686	6817	30.864%

문제

정수 4를 1, 2, 3의 합으로 나타내는 방법은 총 3가지가 있다. 합을 나타낼 때는 수를 1개 이상 사용해야 한다. 단, 같은 수를 두 번 이상 연속해서 사용하면 안 된다.

- 1+2+1
- 1+3
- 3+1

정수 n이 주어졌을 때, n을 1, 2, 3의 합으로 나타내는 방법의 수를 구하는 프로그램을 작성하시오.

입력

첫째 줄에 테스트 케이스의 개수 T가 주어진다. 각 테스트 케이스는 한 줄로 이루어져 있고, 정수 n이 주어진다. n은 양수이며 100,000보다 작거나 같다.

출력

각 테스트 케이스마다, n을 1, 2, 3의 합으로 나타내는 방법의 수를 1,000,000,009로 나눈 나머지를 출력한다.

같은 수를 두 번 이상 연속해서 사용할 수 없고, n은 10^5 이하의 양수입니다. 방법의 수가 많은지 10^9+9로 나눈 나머지를 출력해야 합니다.

개인적으로는 가장 어려웠던 문제였습니다. 이전 문제를 요행으로 풀어서, 끝나는 수를 기준으로 조합을 분듀할 생각을 하지 못했거든요. 만약 이전 문제를 끝에 오는 수를 기준으로 분듀하여 푸셨다면 어렵지 않을 문제입니다.

n이 4 이상일 때, n을 같은 수를 두 번 이상 연속해서 사용하지 않고 1, 2, 3의 합으로 나타낸 방법의 수는 **(n-1을 같은 수를 두 번 이상 연속해서 사용하지 않고 1, 2, 3의 합으로 나타낸 방법의 수 중 2 또는 3으로 끝나는 수) + (n-2을 같은 수를 두 번 이상 연속해서 사용하지 않고 1, 2, 3의 합으로 나타낸 방법의 수 중 1 또는 3으로 끝나는 수) + (n-3을 같은 수를 두 번 이상 연속해서 사용하지 않고 1, 2, 3의 합으로 나타낸 방법의 수 중 1 또는 2로 끝나는 수)** 와 같습니다.

4를 만드는 경우로 예를 들어보겠습니다. 1을 만드는 경우에 3을 붙이고, 2를 만드는 경우에 2를 붙이고, 3을 만드는 경우에 1을 붙이면 4를 만들 수 있습니다. 그런데 숫자를 붙일 때 끝에 같은 숫자가 붙어 있으면, 같은 수가 두 번 연속되기 때문에 문제의 조건에 부합하지 않습니다. 따라서 붙이려는 숫자를 제외한 나머지 숫자로 끝나는 조합에만 붙이는 것이죠.

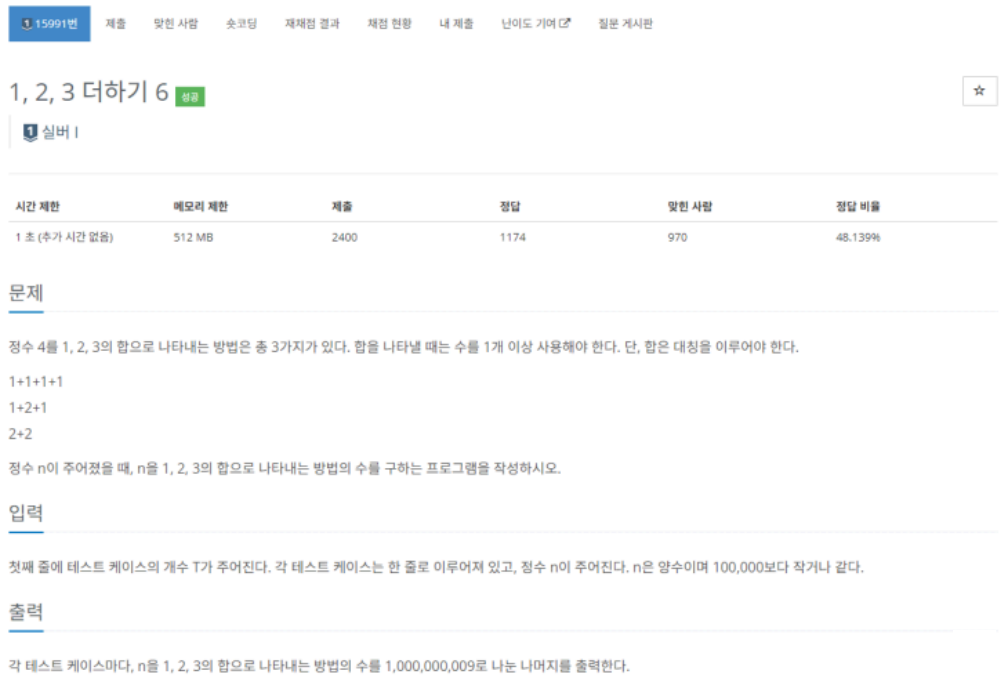
dp[i][j]를 i+1을 같은 수를 두 번 이상 연속해서 사용하지 않고 1, 2, 3의 합으로 나타낸 방법의 수 중 끝에 오는 수가 j+1인 경우의 수로 정의했습니다. 이전 풀이와 비슷하게 초기 3열만 채워두고 그다음 열부터 이전 3열을 참조하여 채워나가면 됩니다.

```
import sys
input = lambda: sys.stdin.readline().rstrip()

N_MAX = 10**5
dp = [[0, 0, 0] for _ in range(N_MAX)]
dp[0] = [1, 0, 0]
dp[1] = [0, 1, 0]
dp[2] = [1, 1, 1]
for i in range(3, N_MAX):
    dp[i][0] = (dp[i-1][1] + dp[i-1][2]) % (10**9 + 9)
    dp[i][1] = (dp[i-2][0] + dp[i-2][2]) % (10**9 + 9)
    dp[i][2] = (dp[i-3][0] + dp[i-3][1]) % (10**9 + 9)

t = int(input())
for _ in range(t):
    n = int(input())
    print(sum(dp[n-1]) % (10**9 + 9))
```

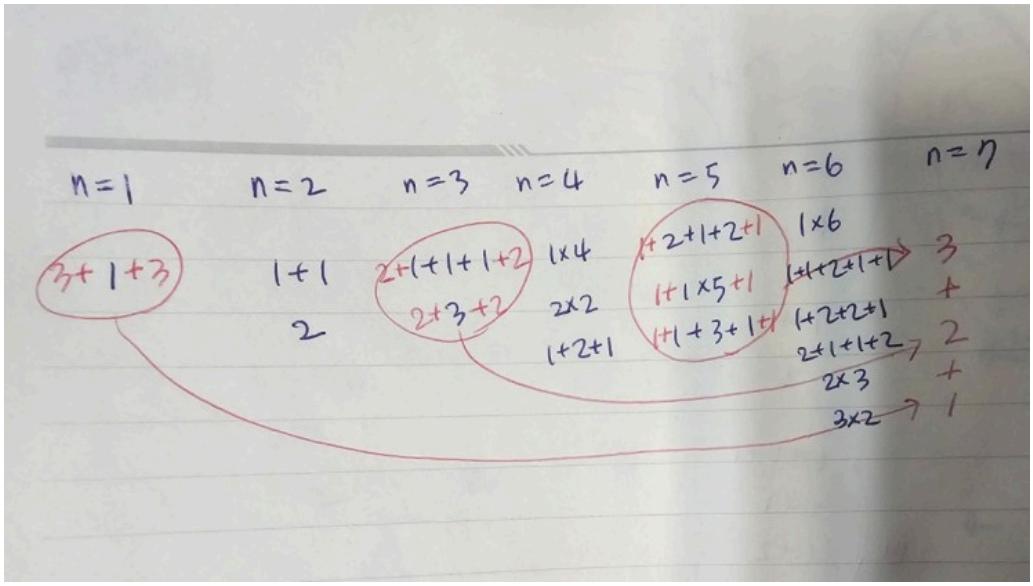
▶ 15991번: 1, 2, 3 더하기 6



이번에는 합이 대칭을 이루어야 합니다. 이번 문제는 n에 따른 수열이 한 번 쉬어가며 숫자가 바뀌어서 규칙성을 찾기 쉬웠습니다. (1, 2, 2, 3, 3, 6, 6, ...)

이전 조합의 형태를 유지하면서 다음 대칭 조합의 형태를 예측하려면, 이전 조합의 양 끝에 숫자를 붙이면 됩니다. n이 7 이상일 때, n을 대칭이 되도록 1, 2, 3의 합으로 나타낸 방법의 수는 **(n-2, n-4, n-6은 대칭이 되도록 1, 2, 3의 합으로 나타낸 방법의 수의 총합)**과 같습니다.

n-2일 때는 양 끝에 1을, n-4일 때는 양 끝에 2를, n-6일 때는 양 끝에 3을 붙여준다고 생각하면 됩니다. 이렇게 하면 겹치는 경우 없이 모든 조합을 구할 수 있습니다.



풀고 나서 정리하니 말은 쉬운데, 푸는 당시에는 확신이 어렵습니다. 검증을 위해 추가로 조합을 그려보아야 하고, 그 과정에서 실수라도 하게 되면 혼란은 더 커집니다. 결국 많이 풀어보며 공부하는 수밖에 없는 것 같아요.

```python
import sys
input = lambda: sys.stdin.readline().rstrip()

N_MAX = 10**5
MOD = 10**9 + 9
dp = [1, 2, 2, 3, 3, 6] + [0] * (N_MAX - 6)
for i in range(6, N_MAX):
    dp[i] = (dp[i-6] + dp[i-4] + dp[i-2]) % MOD

t = int(input())
for _ in range(t):
    n = int(input())
    print(dp[n-1])
```

▶ 15992번: 1, 2, 3 더하기 7

1, 2, 3 더하기 7 성공

실버 I

시간 제한	메모리 제한	제출	정답	맞힌 사람	정답 비율
0.25 초	512 MB	1895	967	769	51.995%

문제

정수 4를 1, 2, 3의 합으로 나타내는 방법은 총 7가지가 있다. 합을 나타낼 때는 수를 1개 이상 사용해야 한다.

- 1+1+1+1
- 1+1+2
- 1+2+1
- 2+1+1
- 2+2
- 1+3
- 3+1

정수 n과 m이 주어졌을 때, n을 1, 2, 3의 합으로 나타내는 방법의 수를 구하는 프로그램을 작성하시오. 단, 사용한 수의 개수는 m개 이어야 한다.

입력

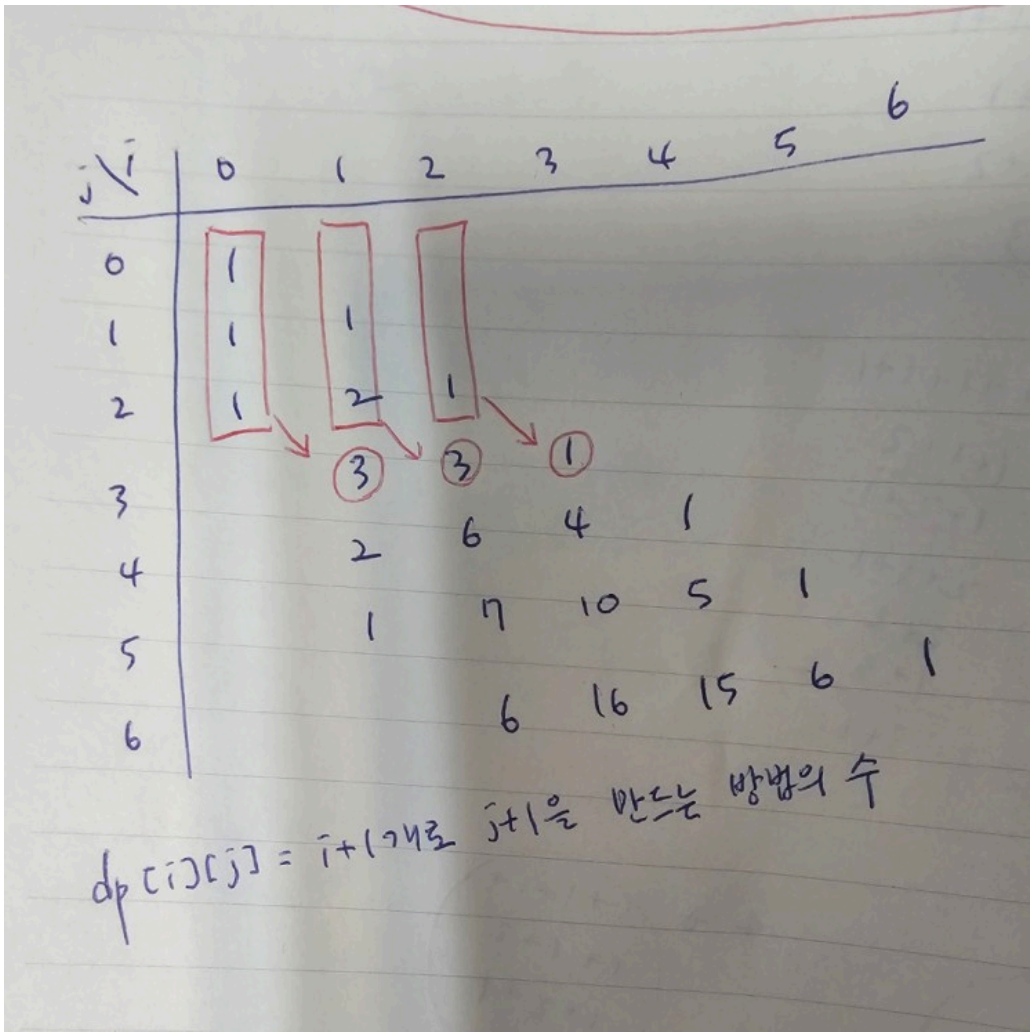
첫째 줄에 테스트 케이스의 개수 T가 주어진다. 각 테스트 케이스는 한 줄로 이루어져 있고, 정수 n과 m이 주어진다. n은 양수이며 1,000보다 작거나 같다. m도 양수이며, n보다 작거나 같다.

출력

각 테스트 케이스마다, n을 1, 2, 3의 합으로 나타내는 방법의 수를 1,000,000,009로 나눈 나머지를 출력한다. 단, 사용한 수의 개수는 m개 이어야 한다.

사용한 수의 개수가 m개인 방법의 수를 구해야 합니다. 앞서 끝자리 수에 따라 방법의 수를 나누었듯, 사용한 수의 개수에 따라 방법의 수를 나누어 저장해 주면 됩니다.

n이 4 이상일 때, n을 m개의 수를 사용하여 1, 2, 3의 합으로 나타낸 방법의 수는 **(n - 1, n - 2, n - 3은 m - 1개의 수를 사용하여 1, 2, 3의 합으로 나타낸 방법의 수의 총합)**과 같습니다. 아래 사진이 있는데, i, j가 거꾸로 되어 있으니 그 점 참고해서 봐주시면 되겠습니다.



4를 예로 다시 설명하자면 4를 2개의 수를 사용하여 만드는 경우의 수는 1을 1개의 수를 사용하여 만든 경우에 3을 더하거나, 2를 1개의 수를 사용하여 만든 경우에 2를 더하거나, 3을 1개의 수를 사용하여 만든 경우에 1을 더하는 경우의 수와 같습니다. 워낙 직관적이라 쉽게 납득할 수 있었습니다.

```
import sys
input = lambda: sys.stdin.readline().rstrip()

N_MAX = 10**3
dp = [[0]*N_MAX for _ in range(N_MAX)]
dp[0] = [1] + [0]*(N_MAX-1)
dp[1] = [1, 1] + [0]*(N_MAX-2)
dp[2] = [1, 2, 1] + [0]*(N_MAX-3)
for i in range(3, N_MAX):
    for j in range(1, N_MAX):
        dp[i][j] = (dp[i-1][j-1] + dp[i-2][j-1] + dp[i-3][j-1]) % (10**9 + 9)

t = int(input())
for _ in range(t):
    n, m = map(int, input().split())
    print(dp[n-1][m-1])
```

▶ 15993번: 1, 2, 3 더하기 8

1, 2, 3 더하기 8 성공

실버 I

시간 제한	메모리 제한	제출	정답	맞힌 사람	정답 비율
1 초 (추가 시간 없음)	512 MB	1215	703	593	56.964%

문제

정수 4를 1, 2, 3의 합으로 나타내는 방법은 총 7가지가 있다. 합을 나타낼 때는 수를 1개 이상 사용해야 한다.

- 1+1+1+1
- 1+1+2
- 1+2+1
- 2+1+1
- 2+2
- 1+3
- 3+1

정수 n이 주어졌을 때, n을 1, 2, 3의 합으로 나타내는 방법의 수를 구하는 프로그램을 작성하시오.

입력

첫째 줄에 테스트 케이스의 개수 T가 주어진다. 각 테스트 케이스는 한 줄로 이루어져 있고, 정수 n이 주어진다. n은 양수이며 100,000보다 작거나 같다.

출력

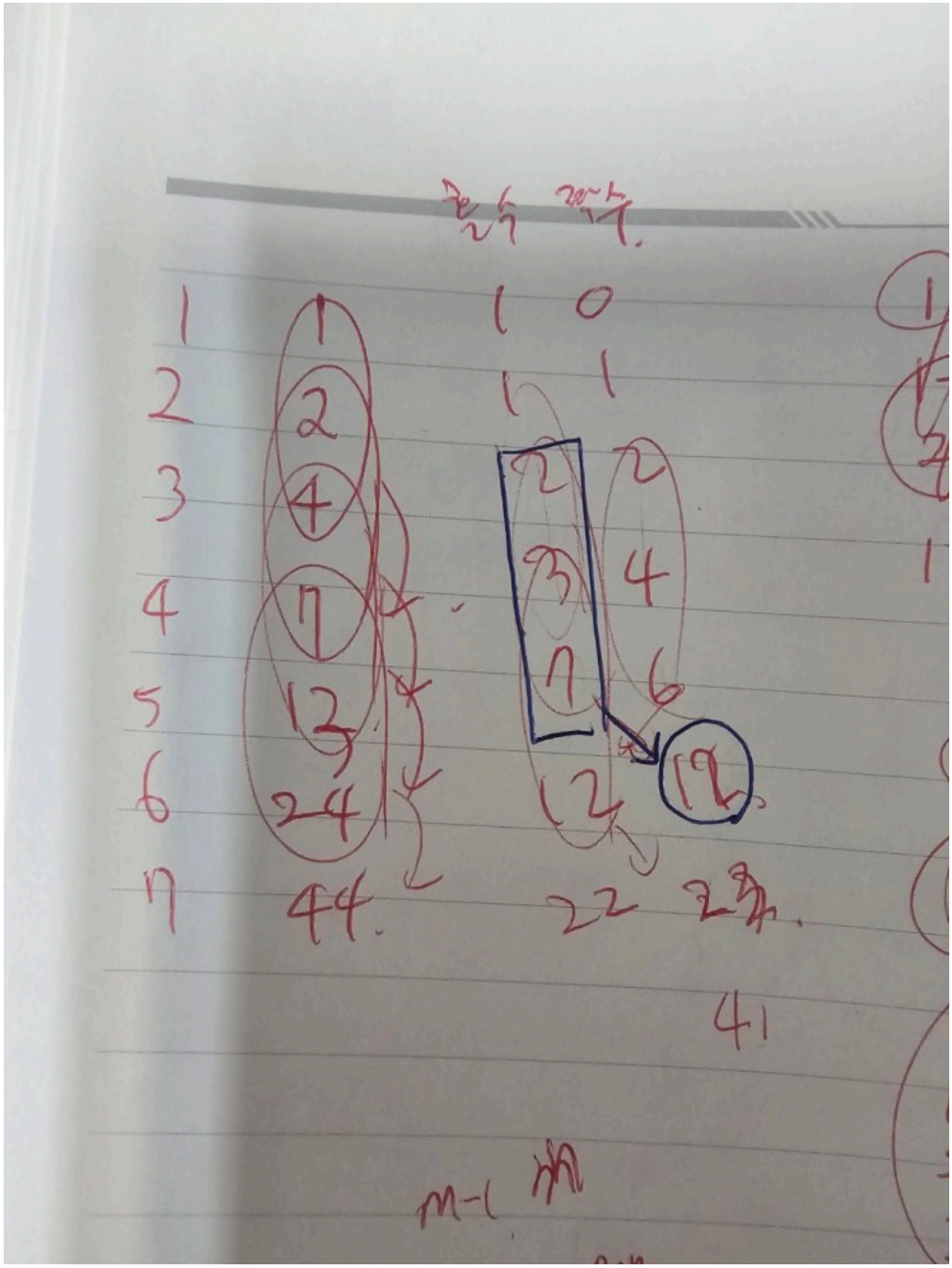
각 테스트 케이스마다, n을 나타낼 때 사용한 수의 개수가 홀수인 방법의 수, 짝수인 방법의 수를 공백으로 구분해 출력한다.

방법의 수는 1,000,000,009로 나눈 나머지를 출력해야 한다.

이번엔 n을 나타낼 때 사용한 수의 개수가 홀수인 방법의 수, 짝수인 방법의 수를 공백으로 구분해 출력하랍니다.

이번에는 홀수, 짝수에 따라 방법의 수를 나누어주면 되겠네요. n이 4 이상일 때, n을 1, 2, 3의 합으로 나타낸 방법의 수 중 사용한 수의 개수가 홀수인 방법의 수는 **(n - 1, n -2, n - 3은 1, 2, 3의 합으로 나타낸 방법의 수 중 사용한 수의 개수가 짝수인 방법의 수의 총합)**과 같습니다. 반대로 짝수인 방법의 수는 홀수인 방법의 수의 총합과 같고요.

이유는 간단합니다. n - 1인 경우에는 1, n - 2인 경우에는 2, n - 3인 경우에는 3을 더해주는데 숫자가 하나 추가되었으니 사용한 수의 개수는 하나 늘어 홀수였으면 짝수, 짝수였으면 홀수가 됩니다. 그래서 그렇습니다.



dp[i][j]는 i+1을 1, 2, 3의 합으로 나타낸 방법의 수(j가 0이면 홀수, 1이면 짝수)입니다. 코드 재활용하다 실수로 N_MAX = 10**6으로 지수를 범위보다 1 높게 설정했더니 시간 초과가 발생하더라고요. 참고가 되었으면 좋겠습니다.

```python
import sys
input = lambda: sys.stdin.readline().rstrip()

N_MAX = 10**5
dp = [[0, 0] for _ in range(N_MAX)]
dp[0] = [1, 0]
dp[1] = [1, 1]
dp[2] = [2, 2]
for i in range(3, N_MAX):
    dp[i][0] = (dp[i-3][1] + dp[i-2][1] + dp[i-1][1]) % (10**9 + 9)
    dp[i][1] = (dp[i-3][0] + dp[i-2][0] + dp[i-1][0]) % (10**9 + 9)
t = int(input())
for _ in range(t):
    n = int(input())
    print(*dp[n-1])
```

▶ 16915번: 1, 2, 3 더하기 9

1, 2, 3 더하기 9 성공

실버 I

시간 제한	메모리 제한	제출	정답	맞힌 사람	정답 비율
1 초	512 MB	1804	888	713	50.460%

문제

정수 4를 1, 2, 3의 합으로 나타내는 방법은 총 7가지가 있다. 합을 나타낼 때는 수를 1개 이상 사용해야 한다.

- 1+1+1+1
- 1+1+2
- 1+2+1
- 2+1+1
- 2+2
- 1+3
- 3+1

정수 n과 m이 주어졌을 때, n을 1, 2, 3의 합으로 나타내는 방법의 수를 구하는 프로그램을 작성하시오. 단, 사용한 수의 개수는 m개 이하 이어야 한다.

입력

첫째 줄에 테스트 케이스의 개수 T가 주어진다. 각 테스트 케이스는 한 줄로 이루어져 있고, 정수 n과 m이 주어진다. n은 양수이며 1,000보다 작거나 같다. m도 양수이며, n보다 작거나 같다.

출력

각 테스트 케이스마다, n을 1, 2, 3의 합으로 나타내는 방법의 수를 1,000,000,009로 나눈 나머지를 출력한다. 단, 사용한 수의 개수는 m개 이하 이어야 한다.

사용한 수의 개수가 m개 이하인 방법의 수를 구하랍니다. 15992번: 1, 2, 3 더하기 7의 코드를 재활용하면 됩니다.

dp[i][j]에는 i+1을 1, 2, 3 중 j+1개를 사용한 합으로 나타내는 방법의 수가 담겨있습니다. 따라서, i+1을 1, 2, 3 중 j+1개 이하를 사용한 합으로 나타내는 방법의 수는 sum(dp[i][:j+2])가 되겠습니다.

sum()을 사용할 경우 나머지 연산 붙이는 것을 잊지 맙시다. 제가 그래서 한 번 틀렸거든요...

```python
import sys
input = lambda: sys.stdin.readline().rstrip()

N_MAX = 10**3
dp = [[0]*N_MAX for _ in range(N_MAX)]
dp[0] = [1] + [0]*(N_MAX-1)
dp[1] = [1, 1] + [0]*(N_MAX-2)
dp[2] = [1, 2, 1] + [0]*(N_MAX-3)
for i in range(3, N_MAX):
    for j in range(1, N_MAX):
        dp[i][j] = (dp[i-1][j-1] + dp[i-2][j-1] + dp[i-3][j-1]) % (10**9 + 9)

t = int(input())
for _ in range(t):
    n, m = map(int, input().split())
    ans = 0
    for i in range(m):
        ans += dp[n-1][i]
        ans %= 10**9 + 9
    print(ans)
```

사실 조금 더 난이도 있는 DP 문제 풀다가 머리 좀 식히려고 풀었습니다. 파이썬이 주는 어드밴티지가 상당한 것을 감안하더라도, 실버 상위 문제가 나름 쉽게 풀리니까 뿌듯합니다. 본격적으로 프로그래밍 공부 시작한 지도 꽤 됐으니 사실 부끄러운 수준의 발전이지만, 부지런히 공부해서 빠르게 성장해 보겠습니다.

알고리즘 문제 풀이 덕분에 골방에서 썩어가던 홍련 노트를 쓰게 되었습니다. 융합 프로그래밍 강의 시간, 문제 풀기 전에 종이에 먼저 적고 시작하라는 교수님 말씀을 뒤늦게서야 깨닫는 중입니다.

도형... 내 갓으로
내삼 구워먹지말게..