# 2020 Goricon 풀이

2020. 11. 08

# 출제 및 검수

skeep194

exqt

dogdriip

annaria

jame0313

dlstj0923

urd05

# 사토르 마방진

| | | |
|---|---|---|
| A | A | B |
| A | C | D |
| B | D | E |

| | | | |
|---|---|---|---|
| A | P | P | L |
| P | P | A | P |
| P | A | D | D |
| L | P | O | V |

# 사토르 마방진

| | | |
|---|---|---|
| A | A | B |
| A | C | D |
| B | D | E |

| | | | |
|---|---|---|---|
| A | P | P | L |
| P | P | A | P |
| P | A | D | D |
| L | P | O | V |

2중 for문으로 대각선 원소끼리 같은지 체크
or
가로방향, 세로방향으로 순회해서 얻은 문자열이 같은지 비교

# 긴급 회의

빈도의 최댓값을 구하고, 해당 최댓값이 몇 개인지 구하면 됩니다

- 가장 많은 표를 받은 사람이 1명뿐이라면 해당 idx 출력
- 최댓값이 2명 이상이라면 skipped
- 투표를 건너뛴 경우(0 입력)는 계산하지 않고, 1~N만 봐도 됨

참고로 실제 어몽어스 게임 룰과는 조금 다르다고 합니다. 제가 막상 게임은 안 해봐서... :(

# 미아 노트

| | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| ? | ? | ? | r | r | r | u | u | u | ? | ? | ? | t | t | t | ? | ? | ? |
| f | ? | f | ? | r | r | u | u | u | ? | ? | ? | ? | ? | t | ? | ? | ? |

# 미아 노트

| | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| ? | ? | ? | r | r | r | u | u | u | ? | ? | ? | t | t | t | ? | ? | ? |
| f | ? | f | ? | r | r | u | u | u | ? | ? | ? | ? | ? | t | ? | ? | ? |

H*W칸씩 묶어서 왼쪽부터 나누면 각 묶음이 원래 복원할 문자가 된다

# 미아 노트

| | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| ? | ? | ? | r | r | r | u | u | u | ? | ? | ? | t | t | t | ? | ? | ? |
| f | ? | f | ? | r | r | u | u | u | ? | ? | ? | ? | ? | t | ? | ? | ? |

f r u ? t ?

묶음 내에 문자가 하나라도 있는 경우에는 해당 문자로 복원하면 되고,
묶음 내의 모든 칸이 물음표인 경우는 복원할 수 없다

# 에너지 드링크

- 큰 수를 절반 나누는 것이 작은 수를 절반 나누는 것보다 손실이 더 크다
- 따라서 더 적은 양의 에너지 드링크를 붓는 게 이득
  - 덜 흘리므로
- 또한, 이미 한번 부어진 애를 다시 다른 곳에 부으면 흘리는 양이 늘어남을 알 수 있음
  - 양이 x인 애를 한번 부으면 x/2가 되고, 그 드링크가 들어있는 드링크를 또 부으면 x/4가 되고, …
- 따라서 한 곳에다 모으는 것이 최적

# 에너지 드링크

- 가장 양이 많은 에너지 드링크에 나머지 에너지 드링크들을 모두 부으면 됨
- 가장 큰 수에 나머지 수들의 절반을 모두 더하면 됨

- 절반씩 더하는 과정에서, 단순히 (최댓값을 제외한 수들을) 각각 절반씩 나눠서 누적해가면 실수 오차가 크게 발생할 수 있음
- 수들을 일단 모두 더하고 마지막에 절반을 나누는 식으로 해결 가능
- (다행히) 이 문제는 절대/상대 오차가 $10^{-5}$이므로 해당 풀이로 풀어도 무방함

# 호반우 상인의 이상한 품질 계산법

묶음이 홀수인 경우와 짝수인 경우로 나눠보자.

[1, 2, 3] -> 중앙값인 2로 계산
[1, 2, 3, 4] -> 중앙값인 3으로 계산

명백하게 짝수 개수로 묶는 것이 이득이다.

# 호반우 상인의 이상한 품질 계산법

그렇다면, 2개로 묶는 경우는 손해를 보지 않고 항상 이득을 취할 수 있다.

[1, 4], [2, 3] …

따라서 이렇게 [제일 큰 가격, 제일 작은 가격], … ,[n/2번째 큰 가격, n/2번째 작은 가격]으로 묶을 경우가 최적이 된다.

n이 홀수인 경우 전체 배열의 중앙값은 1묶음으로 묶인다.

# 호반우 상인의 이상한 품질 계산법

정렬 후 배열을 한 번만 돌면 되므로 총 시간복잡도는 O(NlogN)이 된다.

# 호반우가 길을 건너간 이유

xor 연산은 자기 자신과 연산하면 0이 된다는 성질이 있다.

a xor a = 0

같은 좌표를 중복해서 방문할 수 있으므로, (0, 0)에서 (n-1, m-1)까지 가는 동안 방문한 길을 한 번 더 방문하면 된다.

# 호반우가 길을 건너간 이유

예를 들어 (0, 0), (1, 0), (0, 0), (1, 0) 으로 이동하면 현재 답은 0이면서 1, 0에서 다음에 시작할 좌표로 이동할 수 있다.

홀수 x 짝수인 경우 (0, m-2)까지 0으로 만들면서 이동이 가능하고 (0, m-1)에서 (n-1, m-1)까지 0으로 만들면서 이동이 가능하다.

1, 2, 1, 2, 3, 6, 3, 6

| | | |
|---|---|---|
| 1 | 2 | 3 |
| 4 | 5 | 6 |

# 호반우가 길을 건너간 이유

짝수 x 짝수, 홀수 x 홀수인 경우 처음에 대각선으로 이동하면 부분문제를 짝수 x 홀수 혹은 홀수 x 짝수로 만들 수 있다.

1, 5, 1, 5, (6, 9, 6, 9)

괄호가 있는 부분부터는 부분문제가 홀수 x 짝수

| | | |
|---|---|---|
| 1 | 2 | 3 |
| 4 | 5 | 6 |
| 7 | 8 | 9 |

# 호반우가 길을 건너간 이유

따라서 모든 경우에 2*(N+M)번 이하로 답을 구할 수 있다.

# 호반우와 리듬게임

dp 상태공간을 아래와 같이 정의하자.

dp[i][j][k] : i번째 노트를 처리할 때 j콤보를 완성했고 연속으로 k번 실패한 부분문제

k는 3이상이 될 수 없으므로 k > 2인 모든 경우는 답이 0이다.

# 호반우와 리듬게임

현재 상태에서 할 수 있는 행동은 두 가지다.

1. 이 노트를 치는 경우
2. 이 노트를 버리는 경우

1번 경우는 note[i]*j+dp(i+1,j+1,0)이 되고 2번 경우는 k가 2미만인 경우만 dp(i+1,1,k+1)이 된다.

두 가지 경우 중 최대값을 구해서 현재 답으로 하면 된다.

# 호반우와 리듬게임

dp(i,j,k)는 실행할 때 마다 값이 같으므로 메모이제이션할 수 있다.

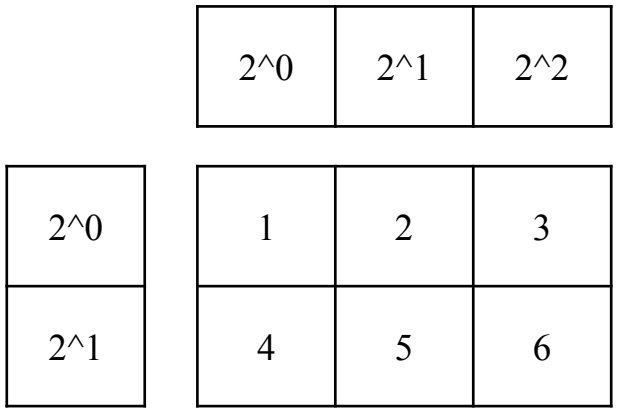
k가 3미만인 경우에 대해서만 구했으므로 마지막 결과값은 0과 비교해서 0보다 작을 경우 0을 출력해야 한다.

따라서 총 시간복잡도는 O(N^2)이 된다.

# 중2병 호반우

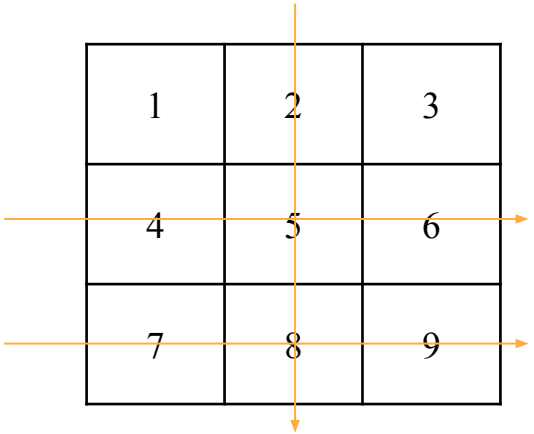
N * M 격자에서 빔을 발사하는 경우는 총 N + M 가지 있고 같은 자리에는 빔을 한 번만 발사하면 되므로 발사 여부를 비트마스크로 표현할 수 있다.

N <= 10이고 M <= 10이므로 총 상태공간은 1024*1024개 있다.



# 중2병 호반우

예를 들어, 오른쪽 격자의 상태는 [010_2][110_2] 가 된다.



# 중2병 호반우

그렇다면 dp 부분문제를 아래와 같이 정의할 수 있다.

dp[i][j] = i: 수직으로 발사한 비트마스크, j: 수평으로 발사한 비트마스크

부분문제의 값은 방문여부에 따라서만 달라지므로 메모이제이션이 가능하다.

# 중2병 호반우

현재 상태에서 할 수 있는 행동은 아래와 같다.

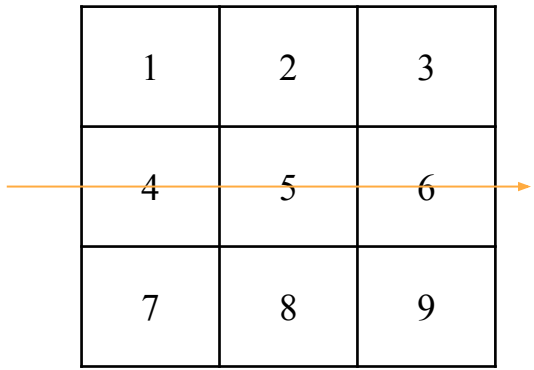
1. 빔을 수직 방향으로 발사
2. 빔을 수평 방향으로 발사

두 경우에 따라 처리하는 방법은 같으므로 1번 경우를 예시로 생각해보자.

# 중2병 호반우

수직으로 발사하는 경우 현재 상태의 답은 수평으로 이미 발사된 경우를 제외하고 계산하면 된다.

예를 들어, 여기서 수직으로 1번째 위치에서 발사하는 경우 수평으로 2번째는 이미 발사되었으므로 1, 7번째 칸만 계산된다.



# 중2병 호반우

현재 상태를 계산한 후 그 다음 부분문제는 지금 발사한 곳만 비트마스크를 갱신하면 된다.

따라서 k번째 위치에서 발사했을 때 다음 부분문제는

dp(i | (1<<k), j) 가 된다.

이 때 지나온 c값의 곱을 다음 부분문제의 답에 곱해줘야 한다.

# 중2병 호반우

답은 최댓값과 최솟값이 있고 c값의 곱은 음수가 될 수 있으므로 최종적으로

최댓값

(현재 부분문제 + max((c값의 곱)*(다음 부분문제의 최댓값), (c값의 곱) * (다음 부분문제의 최솟값)))

최솟값

(현재 부분문제 + min((c값의 곱)*(다음 부분문제의 최댓값), (c값의 곱) * (다음 부분문제의 최솟값)))

# 중2병 호반우

단, 호반우가 모든 적을 죽이지 않아도 되므로 현재 상태의 초기값은 0으로 초기화해야 한다.

부분 문제의 개수가 2^N * 2^M 개 있고 각 부분문제를 계산하는 데 2*N*M의 시간이 드므로 총 시간복잡도는 O(2^N*2^M*N*M)이다.

문제의 제한으로 계산해보면 2억인데 제한 시간이 3초이므로 충분히 가능하다.

# 박스의 균형

- $a_n$
- $(a_n+a_{n-1})/2$
- $(a_n+a_{n-1}+a_{n-2})/3$
- …
- $(a_n+a_{n-1}+a_{n-2}\ \ldots a_1)/n$
- 을 효율적으로 계산하는 문제

- 나머진 지문에 있는 그대로 구현

# 박스의 균형

- $(a_n + a_{n-1}) + \mathrm{a}_{\mathrm{n}-2} = (a_n + a_{n-1} + a_{n-2})$

- 분자 부분만 따로 계산하고
  - 매번 a_i 를 누적
- 범위 검사할 때 분모로 나누어 계산

# 클레어와 물약

- 무식하게 하는 방법
- 각 레시피 마다 새로운 물약을 만들 수 있는지 검사
- 만들 수 있다면 만들 수 있는 물약 리스트에 넣는다

- 모든 레시피를 한번씩 검사했는데 새로 만들 수 있는게 없다면 종료

클레어와 물약

- 최악의 경우 시간이 N*sum(레시피 크기) 에 비례
  - 모든 한번씩 검사했는데 하나만 업데이트 되는 경우
- 개선할 수 있는 부분
  - 레시피의 물약을 하나하나 검사하지 말고 남은 개수를 카운트

- 레시피 (…) -> y 를 통하여 새롭게 y를 만들 수 있다면
- y를 사용하는 모든 레시피의 남은 개수 카운트를 -1
- 0이 된다면 (=모든 재료가 모였다면) 해당 레시피를 업데이트

# 클레어와 물약

7 5
1 x x 7 2
1 x 3 x 7
1 3 x 5
1 x x 3
0 x x 4
3
1 5 6 4

# 클레어와 물약

- 업데이트 하는 함수를 upd(u: 물약 번호)라고 하자
- upd(1), upd(2) .. upd(n) 의 총 수행 시간은 (sum(레시피 크기) <= 400,000) 에 비례

- 구현은 BFS 와 유사하게 Queue 를 사용하여 구현

# 클레어와 물약

```
// i번 물약을 포함하는 레시피 번호들
vector<vector<int>> recipes(n+1);
// 레시피에 필요한 재료 개수
vector<int> cnt(m+1, 0);
// i번째 레시피의 산출물
vector<int> target(m+1);
```

```
vector<int> available(n+1, false); // i번 물약을 만들 수 있는가?
 queue<int> que; // 만들 수 있어서 리스트에 추가시킬 것들
 auto upd = [&](int u) {
   if(available[u]) return;
    available[u] = true;

   for(int r : recipes[u]) {
   cnt[r]--;
     if(cnt[r] == 0) {
       que.push(target[r]);
     }
   }
};
```

클레어와 물약

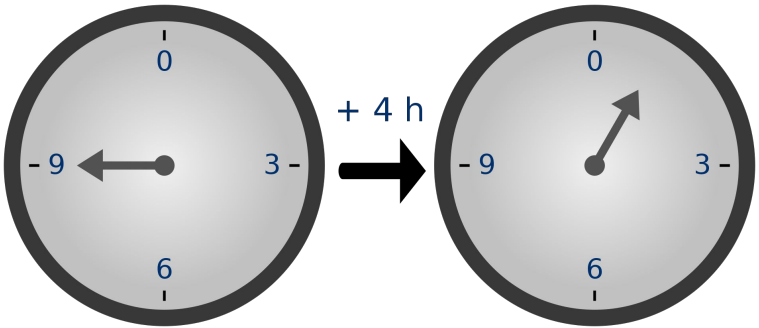
```
int l; cin >> l; // 처음에 가지고 있는 물약 업데이트
  while(l--) {
    int x; cin >> x;
    upd(x);
  }

  while(que.size()) { // 추가적으로 만들 수 있는 물약이 있을 때 까지
    int t = que.front(); que.pop();
    upd(t);
  }
```

# 카드 셔플

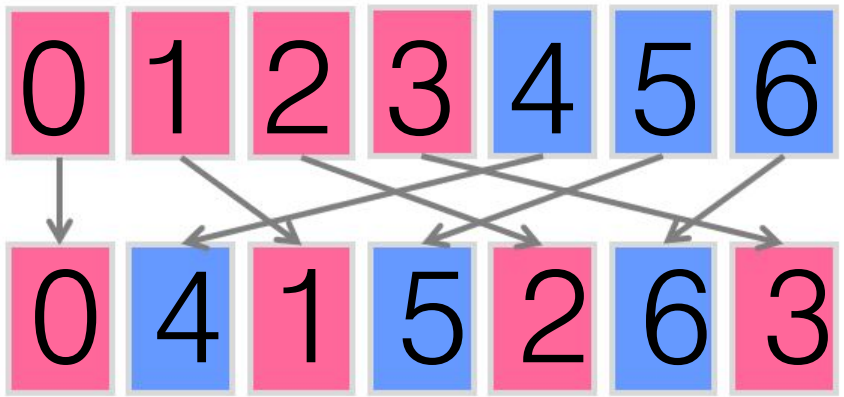
- 1-base 대신 0-base로 변환
- 1 2 3 4 5 6 7 > 0 1 2 3 4 5 6

- modular 산술
- 9 + 4 = 1 (mod 12), (1 − 3) = 10 (mod 12)



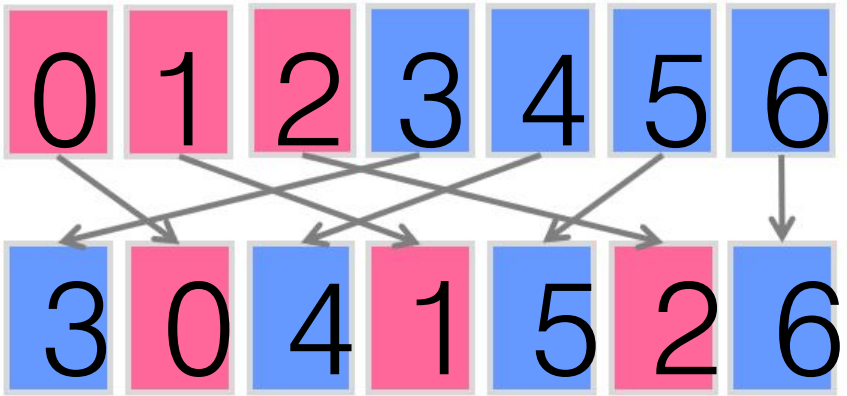
# 카드 셔플

- X셔플을 했을 때 a번째 위치에 있는 카드는
- (2a) 번째로 이동
- 2 => (2*2) = 4
- 4 => (2*4) = 1
- 5 => (2*5) = 3



# 카드 셔플

- Y셔플을 했을 때 a번째 위치에 있는 카드는
- (2a+1) 번째로 이동
- 2 => (2*2+1) = 5
- 4 => (2*4+1) = 2
- 5 => (2*5+1) = 4



# 카드 셔플

- a가 있을 때 f(a) = 2a 또는 g(a) = 2a+1 을 k번 수행하여 만들 수 있는 수들은?

- a = 1, k = 2 인 경우
- f(f(a)) = 4, g(f(a)) = 5, f(g(a)) = 6, g(g(a)) = 7

- [2^k * a, 2^k * a + 2^k)

# 카드 셔플

- 최대 ceil(log(n))번 (n 제한이 10^9 이니 30 정도)
- k=1부터 30까지 다 해본다

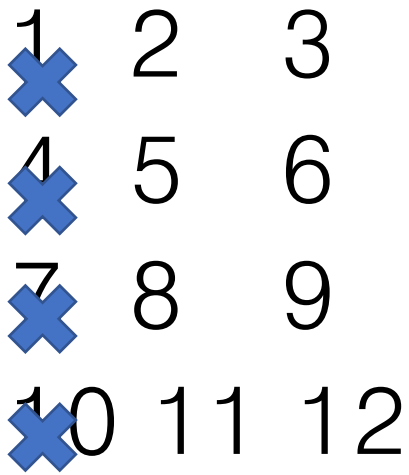
- 방법은 어떻게?

- 최종 위치 B와 2^k * a 의 차이 (B – 2^k * a)를 d라고 하자
- d를 이진수로 표현 했을 때 01을 XY로 바꿔주면 된다

# L-트로미노 계단

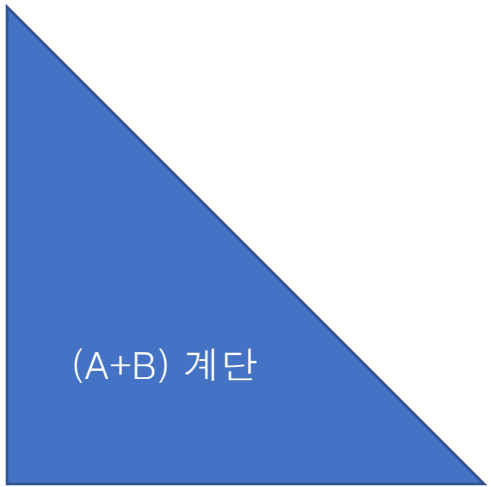
- 1부터 하나씩 가능한지 손으로 판별해보자
- 필요한 L-트로미노 개수는 n*(n+1)/2/3
- n을 3으로 나눈 나머지가 1일 때는 정수가 아니므로 불가능

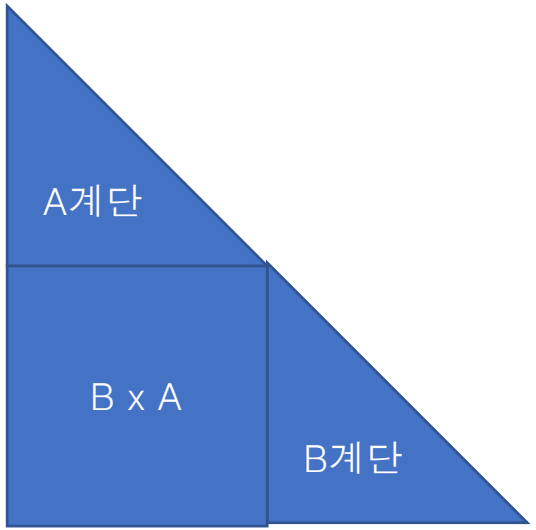
L-트로미노 계단



# L-트로미노 계단

- 아이디어: 큰 계단을 작은 계단 2개와 직사각형으로 나눠서 볼 수 있다.
- 분할정복?



# L-트로미노 계단

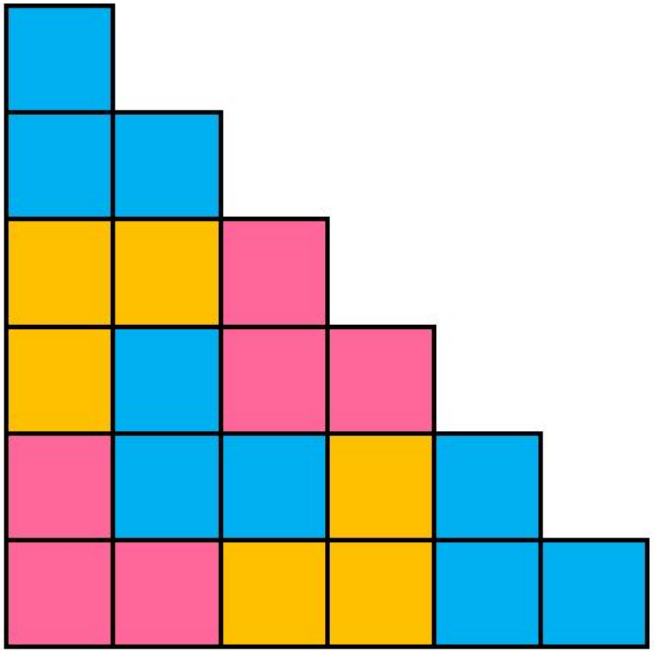
1 2 3

4 5 6

7 8 9

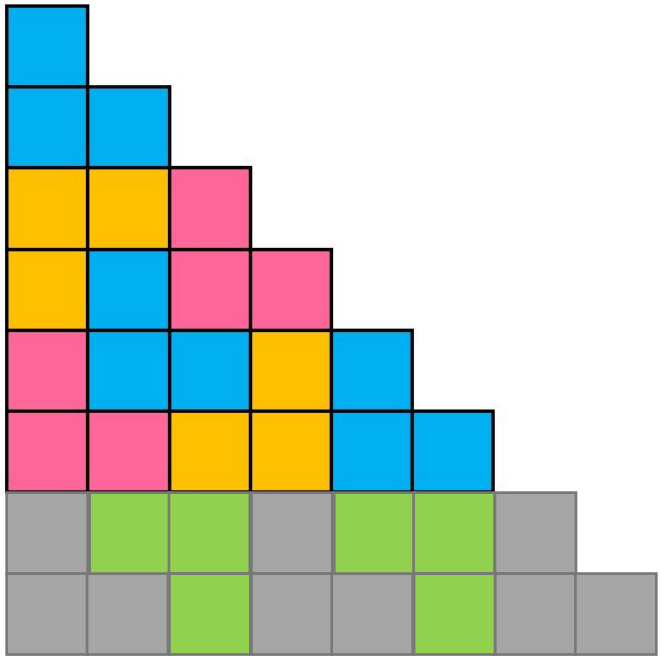
10 11 12

...



# L-트로미노 계단

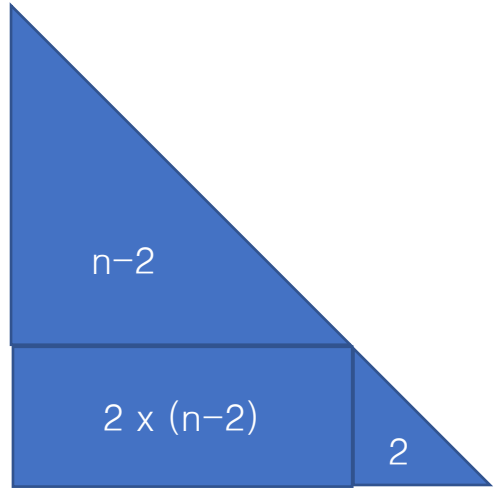
- n = 8 인 경우 n = 6인 경우에서 쉽게 만들 수 있다.

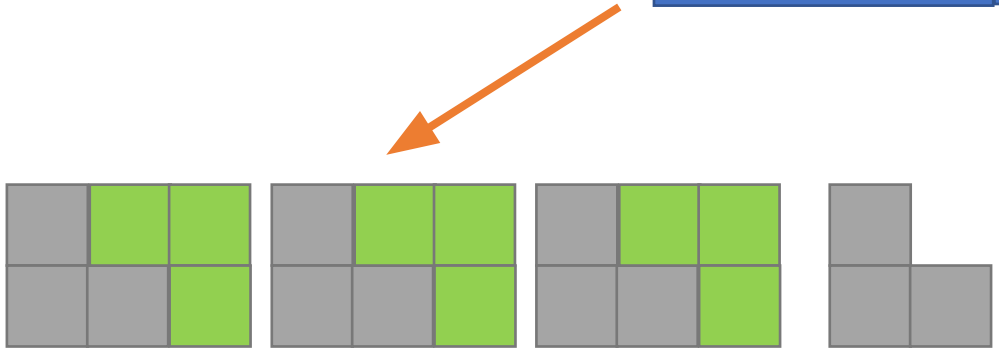


L-트로미노 계단

일반화 해보면

- n % 3 = 2 이고
- n-2 일 때 가능하면 가능



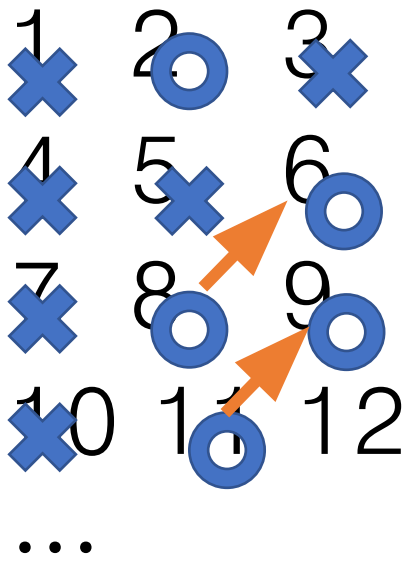



L-트로미노 계단

- n = 9 인 경우
- 딱히 방법이 없으므로 손으로 찾는다

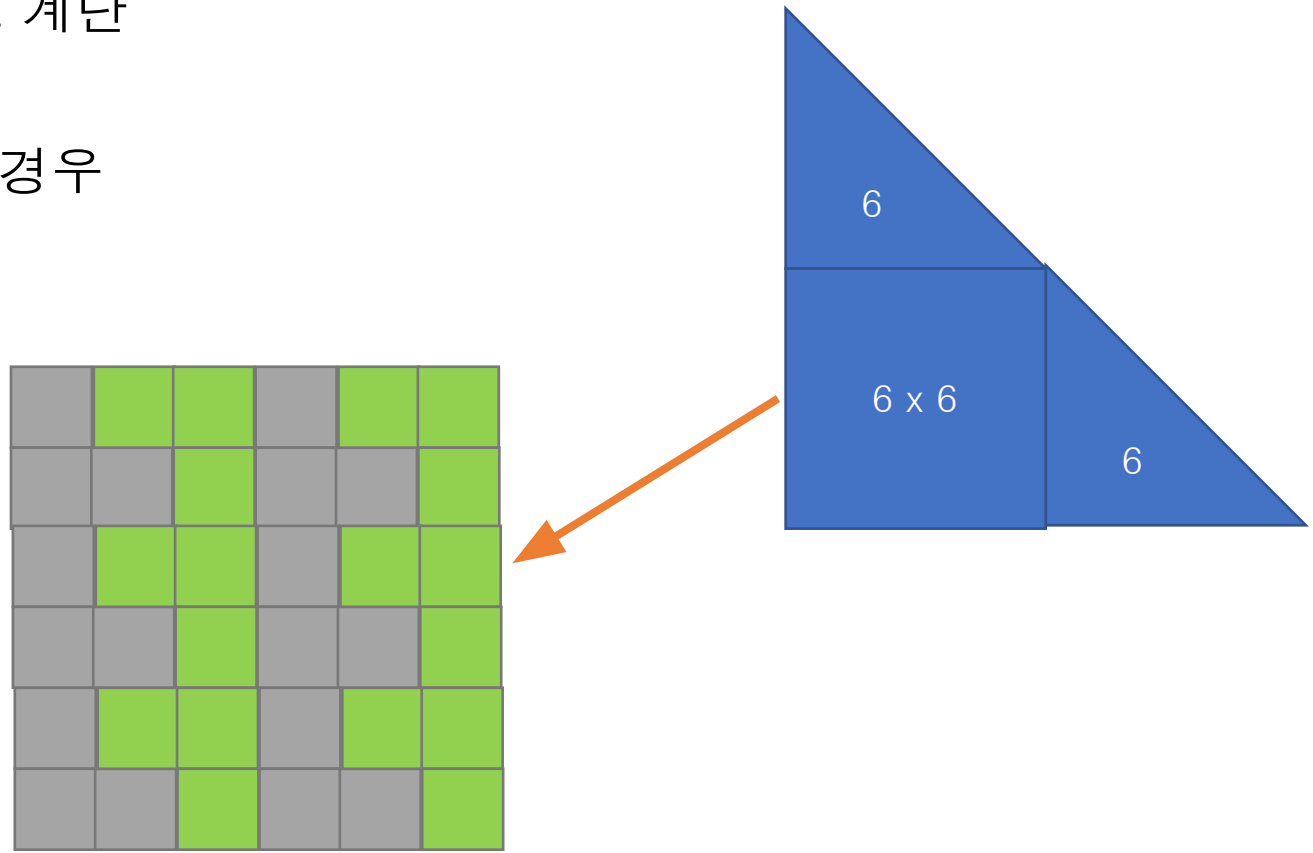
- ~~암튼 가능함~~

# L-트로미노 계단



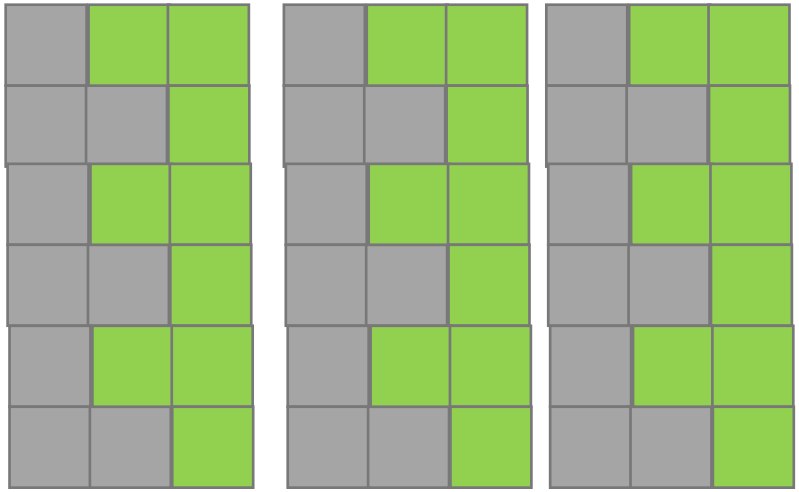
L-트로미노 계단

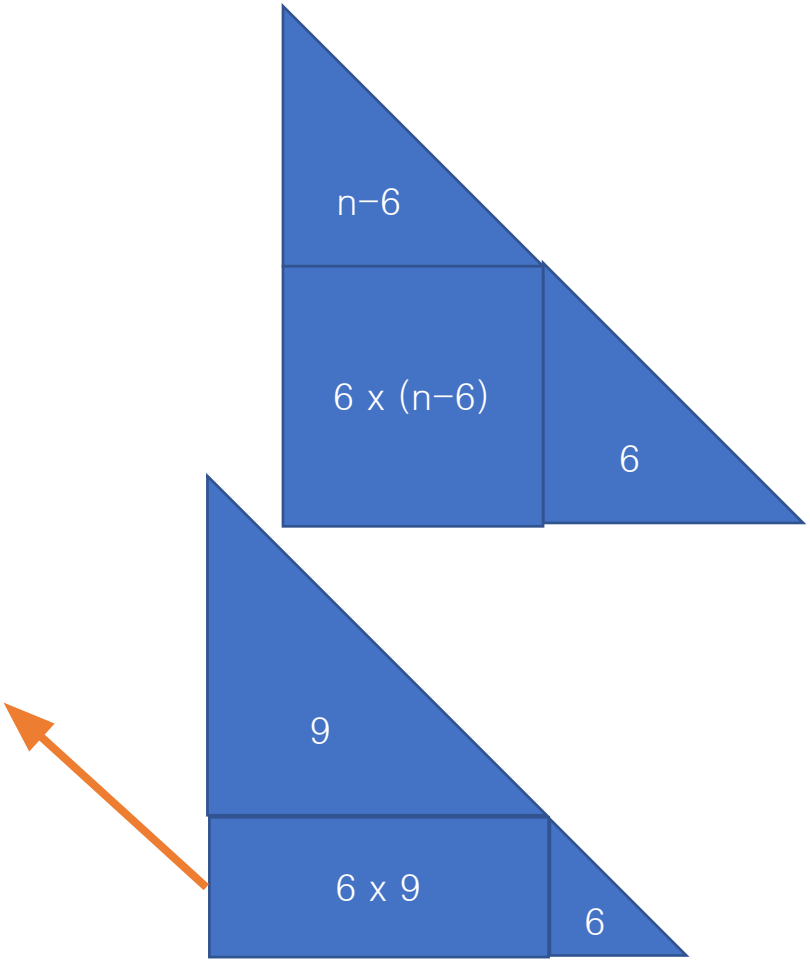
- n = 12의 경우



L-트로미노 계단

- 마찬가지로 n % 3 = 0 이고
- n – 6 이 가능하면 가능





L-트로미노 계단

