1. N 으로 표현

문제 설명
아래와 같이 5 와 사칙연산만으로 12 를 표현할 수 있습니다.
12 = 5 + 5 + (5 / 5) + (5 / 5)
12 = 55 / 5 + 5 / 5
12 = (55 + 5) / 5
5 를 사용한 횟수는 각각 6,5,4 입니다. 그리고 이중 가장 작은 경우는 4 입니다.
이처럼 숫자 N 과 number 가 주어질 때, N 과 사칙연산만 사용해서 표현 할 수 있는 방법 중 N 사용횟수의 최솟값을 return 하도록 solution 함수를 작성하세요.
제한사항
N 은 1 이상 9 이하입니다.
number 는 1 이상 32,000 이하입니다.
수식에는 괄호와 사칙연산만 가능하며 나누기 연산에서 나머지는 무시합니다.
최솟값이 8 보다 크면 -1 을 return 합니다.
입출력 예

| N | number | return |
|---|---|---|
| 5 | 12 | 4 |
| 2 | 11 | 3 |

입출력 예 설명
예제 #1
문제에 나온 예와 같습니다.
예제 #2
11 = 22 / 2 와 같이 2 를 3 번만 사용하여 표현할 수 있습니다.
출처
※ 공지 - 2020 년 9 월 3 일 테스트케이스가 추가되었습니다.

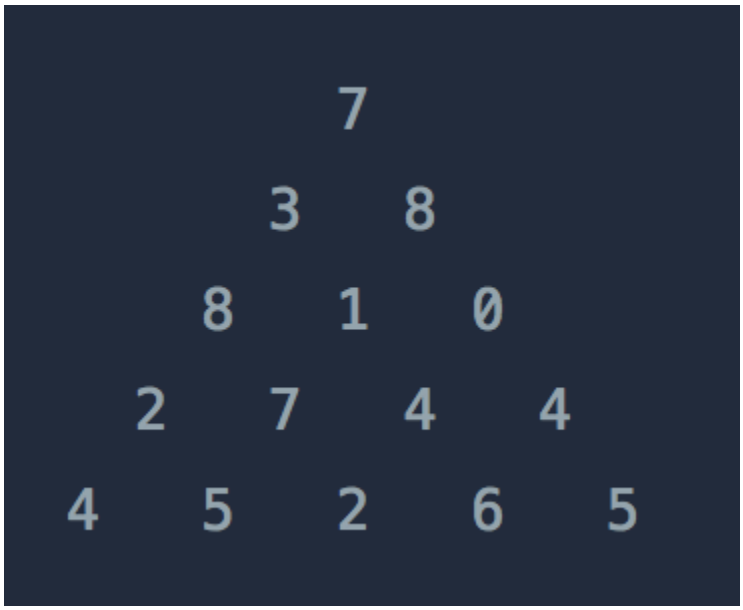
```
class Solution {
   int answer = -1;

   public int solution(int N, int number) {
      dfs(N, 0, 0, number, "");
      return answer;
   }

   public void dfs(int n, int pos, int num, int number, String s)
{
      if (pos > 8)
         return;
      if (num == number) {
         if (pos < answer || answer == -1) {
            System.out.println(s);
            answer = pos;
         }
         return;
      }
      int nn=0;
```

```
        for (int i = 0; i < 8; i++) {
            nn=nn*10+n;
            dfs(n, pos + 1+i, num + nn, number, s + "+");
            dfs(n, pos + 1+i, num - nn, number, s + "-");
            dfs(n, pos + 1+i, num * nn, number, s + "*");
            dfs(n, pos + 1+i, num / nn, number, s + "/");
        }
        // dfs(n,pos+1,num*10+n,number,s+"5");
    }
}
```

https://heunnnn.github.io/DP(1)-N%EC%9C%BC%EB%A1%9C-%ED%91%9C%ED%98%84/

2. 정수 삼각형

문제 설명




위와 같은 삼각형의 꼭대기에서 바닥까지 이어지는 경로 중, 거쳐간 숫자의 합이 가장 큰 경우를 찾아보려고 합니다. 아래 칸으로 이동할 때는 대각선 방향으로 한 칸 오른쪽 또는 왼쪽으로만 이동 가능합니다. 예를 들어 3 에서는 그 아래칸의 8 또는 1 로만 이동이 가능합니다.

삼각형의 정보가 담긴 배열 triangle 이 매개변수로 주어질 때, 거쳐간 숫자의 최댓값을 return 하도록 solution 함수를 완성하세요.

제한사항

삼각형의 높이는 1 이상 500 이하입니다.

삼각형을 이루고 있는 숫자는 0 이상 9,999 이하의 정수입니다.

입출력 예

| triangle | result |
|---|---|
| [[7], [3, 8], [8, 1, 0], [2, 7, 4, 4], [4, 5, 2, 6, 5]] | 30 |

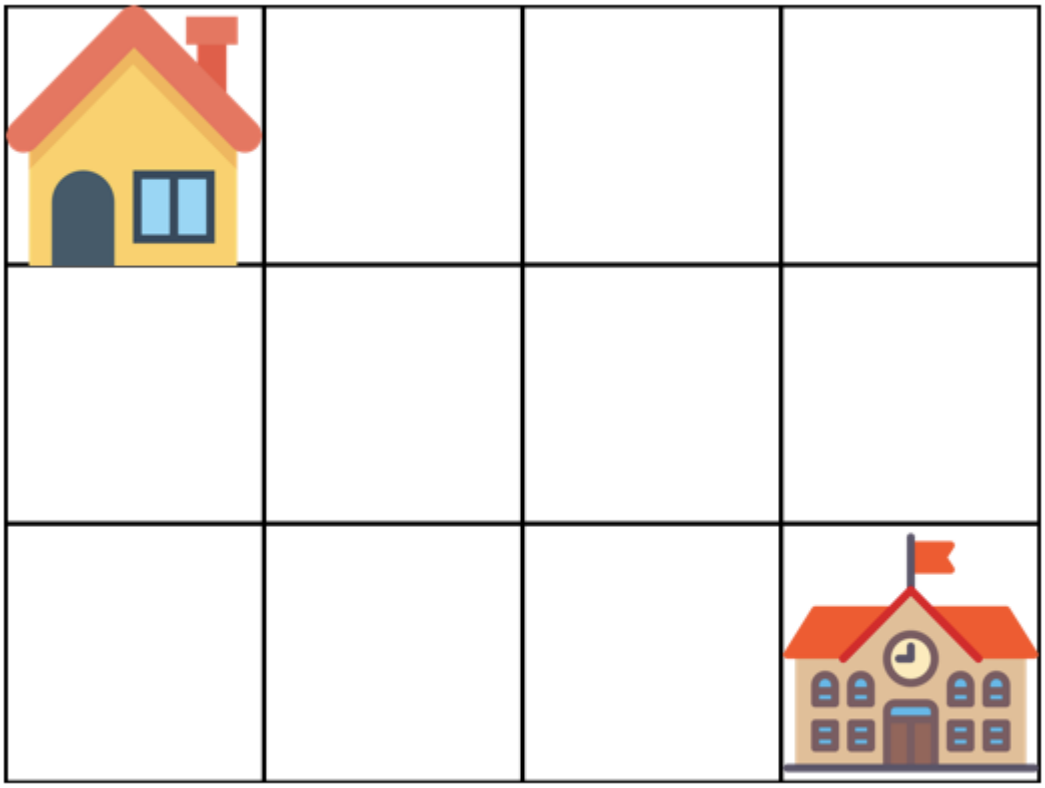
```
public int solution(int[][] triangle) {
     // 1. 기본값 초기화  //
   int[][] dp = new int[triangle.length][triangle.length];
     dp[0][0] = triangle[0][0];
     for(int i = 1; i < triangle.length; i++) {
        dp[i][0] = dp[i - 1][0] + triangle[i][0];
        dp[i][i] = dp[i - 1][i - 1] + triangle[i][i];
     }

     // 2. 동적계획법 //
     for(int i = 2; i < triangle.length; i++)
        for(int j = 1; j < i; j++)
           dp[i][j] = Math.max(dp[i - 1][j - 1], dp[i - 1][j]) +
triangle[i][j];
```

```
    // 3. 최대값 반환 //
    int max = 0;
    for(int i = 0; i < dp.length; i++)
       max = Math.max(max, dp[dp.length - 1][i]);

  return max;
}
```

https://lkhlkh23.tistory.com/117

3. 등굣길

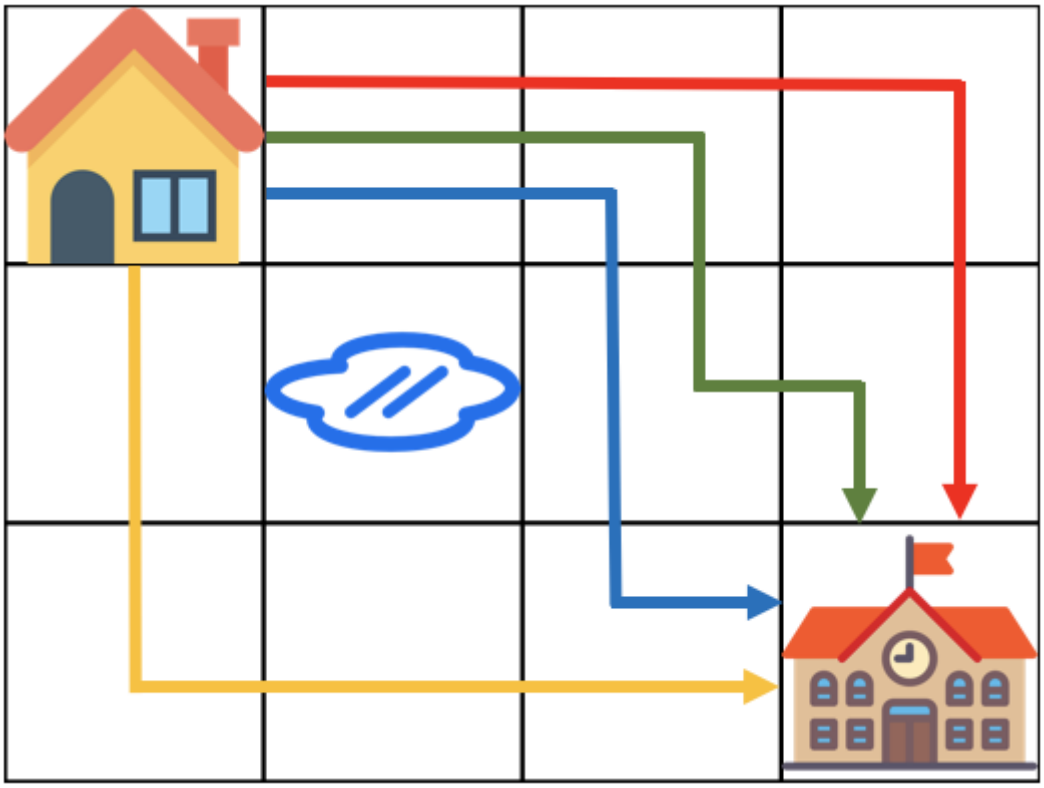
문제 설명
계속되는 폭우로 일부 지역이 물에 잠겼습니다. 물에 잠기지 않은 지역을 통해 학교를 가려고 합니다. 집에서 학교까지 가는 길은 m x n 크기의 격자모양으로 나타낼 수 있습니다.
아래 그림은 m = 4, n = 3 인 경우입니다.



가장 왼쪽 위, 즉 집이 있는 곳의 좌표는 (1, 1)로 나타내고 가장 오른쪽 아래, 즉 학교가 있는 곳의 좌표는 (m, n)으로 나타냅니다.
격자의 크기 m, n 과 물이 잠긴 지역의 좌표를 담은 2 차원 배열 puddles 이 매개변수로 주어집니다. 오른쪽과 아래쪽으로만 움직여 집에서 학교까지 갈 수 있는 최단경로의 개수를 1,000,000,007 로 나눈 나머지를 return 하도록 solution 함수를 작성해주세요.
제한사항
격자의 크기 m, n 은 1 이상 100 이하인 자연수입니다.
m 과 n 이 모두 1 인 경우는 입력으로 주어지지 않습니다.
물에 잠긴 지역은 0 개 이상 10 개 이하입니다.
집과 학교가 물에 잠긴 경우는 입력으로 주어지지 않습니다.
입출력 예

| m | n | puddles | return |
|---|---|---|---|
| 4 | 3 | [[2, 2]] | 4 |

입출력 예 설명



```
public class Solution {
  public int solution(int m, int n, int[][] puddles) {
    int[][] street = new int[n][m];

    // 웅덩이는 -1
    for (int[] puddle : puddles)
      street[puddle[1] - 1][puddle[0] - 1] = -1;

    street[0][0] = 1;

    for (int i = 0; i < n; i++) { // 시작점은 1 로 저장
      for (int j = 0; j < m; j++) {

        if(street[i][j] == -1) { // 웅덩이면 0 으로 바꾸고 continue
          street[i][j] = 0;
          continue;
        }

        if(i != 0)
          street[i][j] += street[i - 1][j] % 1000000007; // 숫자가
이 값을 초과할 수 있기 때문에 계산 과정에서 나머지 구하기

        if(j != 0)
          street[i][j] += street[i][j - 1] % 1000000007; // 왼쪽
      }
```

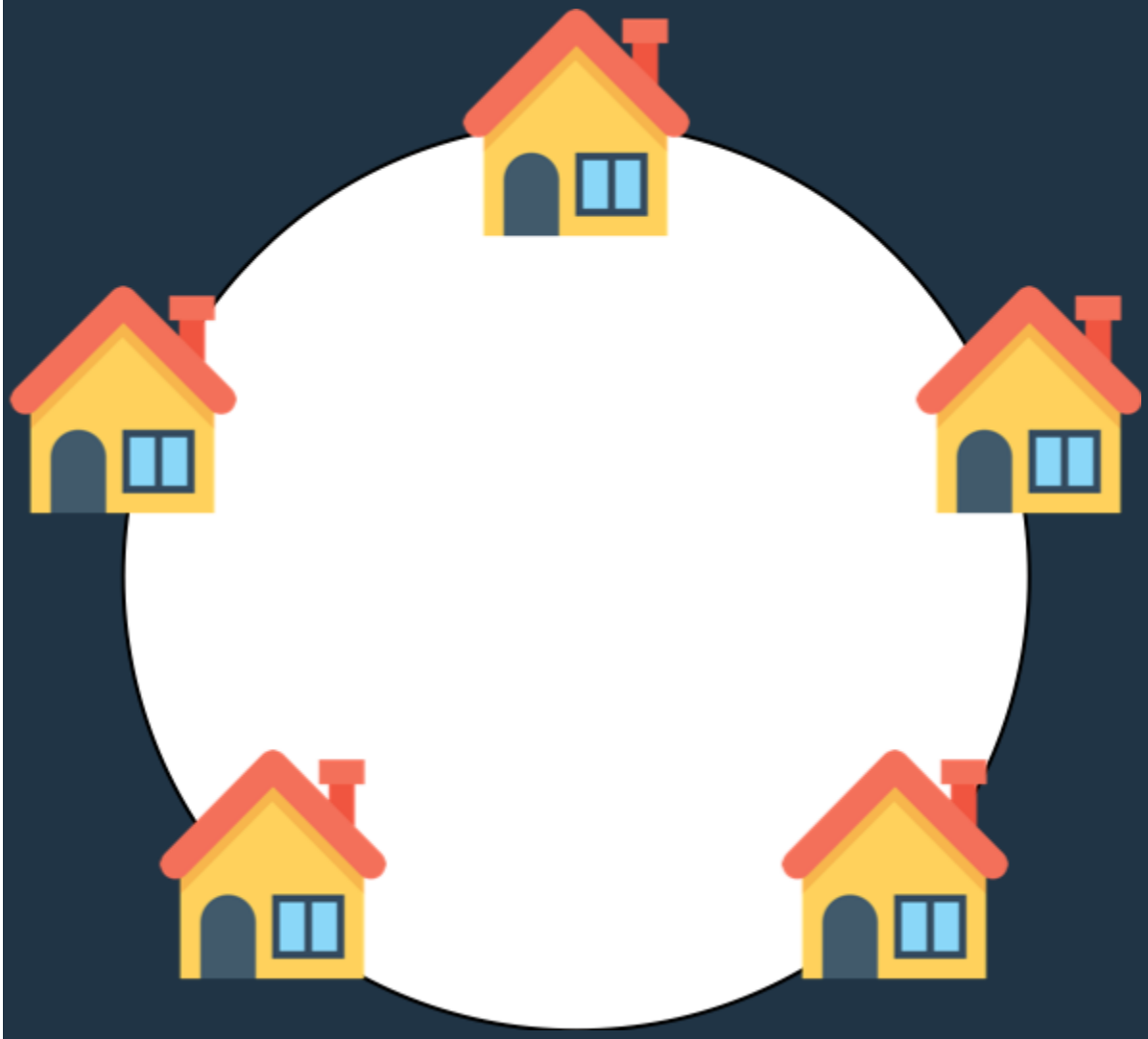
```
    }

    return street[n - 1][m - 1] % 1000000007;
  }

  @Test
  public void 정답(){
    Assert.assertEquals(4, solution(4, 3, new int[][]{{2,2}}));
    Assert.assertEquals(7, solution(4, 4, new int[][]{{3,2},
{2,4}}));
    Assert.assertEquals(7, solution(5, 3, new int[][]{{4,2}}));
    Assert.assertEquals(0, solution(2, 2, new int[][]{{2,1}, {1,
2}}));
    Assert.assertEquals(0, solution(3, 1, new int[][]{{2,1}}));

  }
}
```

https://velog.io/@ajufresh/%EB%93%B1%EA%B5%A3%EA%B8%B8

## 4. 도둑질

### 문제 설명

도둑이 어느 마을을 털 계획을 하고 있습니다. 이 마을의 모든 집들은 아래 그림과 같이 동그랗게 배치되어 있습니다.



각 집들은 서로 인접한 집들과 방범장치가 연결되어 있기 때문에 인접한 두 집을 털면 경보가 울립니다.

각 집에 있는 돈이 담긴 배열 money 가 주어질 때, 도둑이 훔칠 수 있는 돈의 최댓값을 return 하도록 solution 함수를 작성하세요.

제한사항

- 이 마을에 있는 집은 3 개 이상 1,000,000 개 이하입니다.
- money 배열의 각 원소는 0 이상 1,000 이하인 정수입니다.

입출력 예

| money | return |
| --- | --- |
| [1, 2, 3, 1] | 4 |

```
class Solution {
    public int solution(int[] money) {
        int answer = 0;
        int length = money.length;
        int[] dp =new int[length-1];
        int[] dp2= new int[length];

        dp[0]=money[0];
        dp[1]=money[0];
        dp2[0]=0;
        dp2[1]=money[1];
        for(int i=2;i<length-1;i++){
            dp[i]=Math.max(dp[i-2]+money[i],dp[i-1]);
        }
        for(int i=2;i<length;i++){
            dp2[i]=Math.max(dp2[i-2]+money[i],dp2[i-1]);
        }

        return Math.max(dp[length-2],dp2[length-1]);
    }
}
```

https://doohong.github.io/2019/03/14/Algorithm-%20thievery/