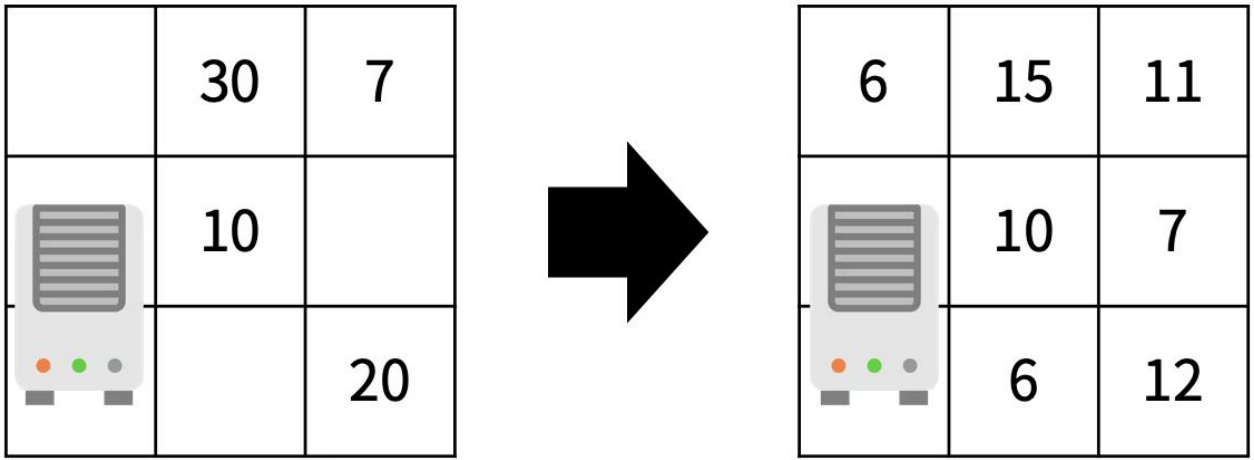
17144: 미세먼지 안녕!

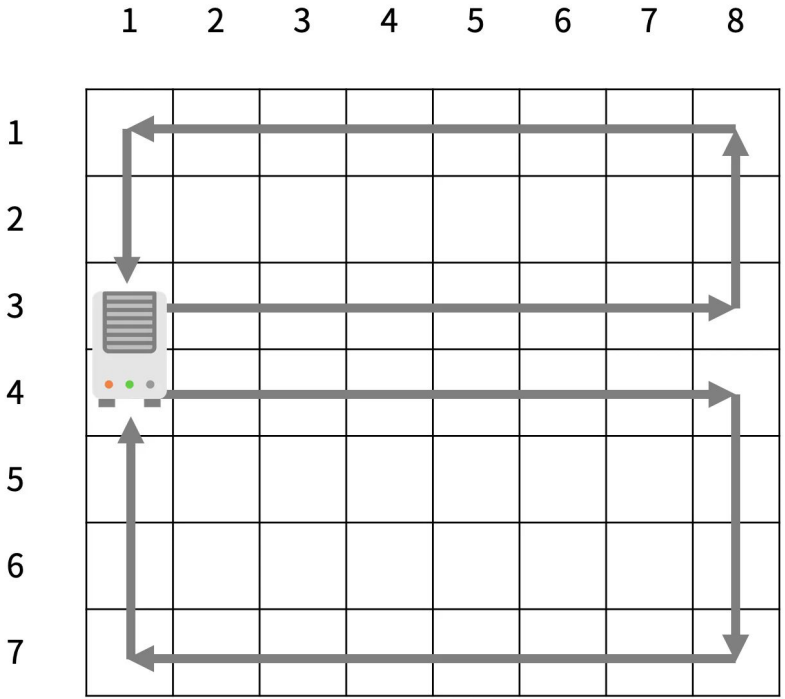
진짜 단순 구현: T초마다 1. 확산 2. shift 해주면 됨

미세먼지가 확산(확산은 미세먼지가 있는 **모든 칸에서 동시에** 일어난다)

- (r, c)에 있는 미세먼지는 인접한 네 방향으로 확산된다.
- 인접한 방향에 공기청정기가 있거나, 칸이 없으면 그 방향으로는 확산이 일어나지 않는다.
- 확산되는 양은 Ar,c/5이고 소수점은 버린다. 즉, ⌊Ar,c/5⌋이다.
- (r, c)에 남은 미세먼지의 양은 Ar,c - ⌊Ar,c/5⌋×(확산된 방향의 개수) 이다.



17144: 미세먼지 안녕!

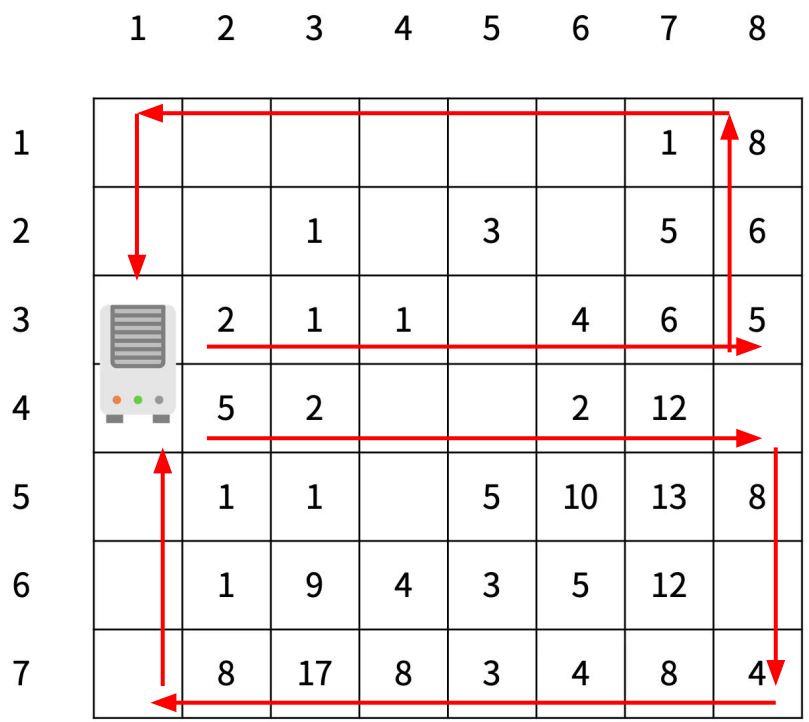


공기청정기가 작동한다.

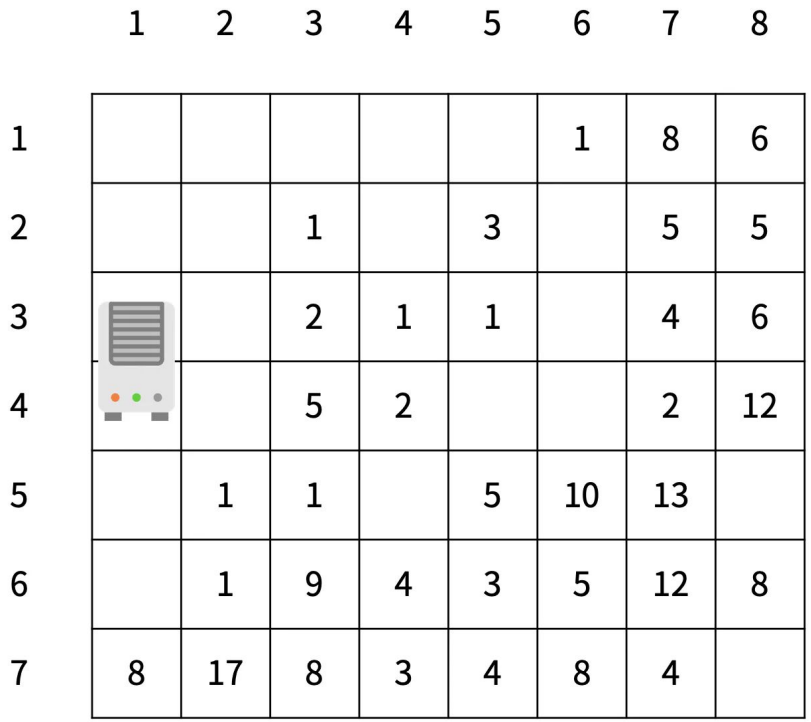
- **위쪽 공기청정기의 바람은 반시계방향**으로 순환하고, **아래쪽 공기청정기의 바람은 시계방향으로 순환**한다.
- 바람이 불면 미세먼지가 **바람의 방향대로 모두 한 칸씩 이동**한다.
- 공기청정기에서 부는 바람은 미세먼지가 없는 바람이고, **공기청정기로 들어간 미세먼지는 모두 정화**된다.

17144: 미세먼지 안녕!

예제1로 보는 공청기 작동 예시



	1	2	3	4	5	6	7	8
1							1	8
2			1		3		5	6
3		2	1	1		4	6	5
4		5	2			2	12	
5		1	1		5	10	13	8
6		1	9	4	3	5	12	
7		8	17	8	3	4	8	4



	1	2	3	4	5	6	7	8
1						1	8	6
2			1		3		5	5
3			2	1	1		4	6
4			5	2			2	12
5		1	1		5	10	13	
6		1	9	4	3	5	12	8
7	8	17	8	3	4	8	4	

17144: 미세먼지 안녕!


```java
int[][] afterStatus = new int[r][c];

for (int x = 0; x < r; x++) {
    for (int y = 0; y < c; y++) {
        // 확산되는 양(=4방향에 더해줄 값)
        int amount = map[x][y] / 5;
        if (amount == 0) {
            afterStatus[x][y] += map[x][y];
            continue;
        }

        int cnt = 0; // 확산된 방향의 개수
        for (int dir = 0; dir < 4; dir++) {
            int nx = x + dx[dir];
            int ny = y + dy[dir];
            if (nx < 0 || nx >= r || ny < 0 || ny >= c || map[nx][ny] == -1) continue
            afterStatus[nx][ny] += amount;
            cnt++;
        }

        // 남은 미세먼지 양
        int remain = map[x][y] - amount * cnt;
        afterStatus[x][y] += remain;
    }
}
```

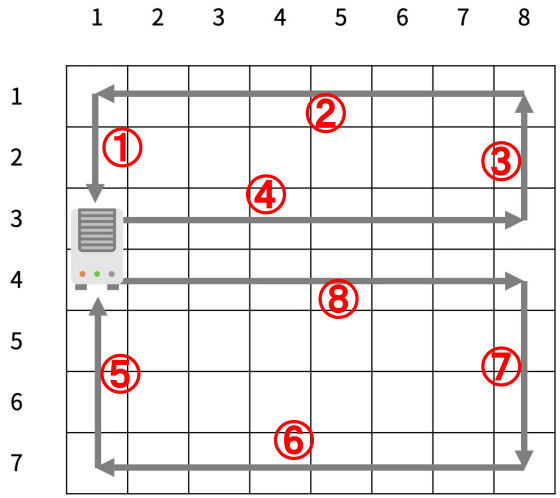
미세먼지가 확산(확산은 미세먼지가 있는 **모든 칸에서 동시에** 일어난다)

- (r, c)에 있는 미세먼지는 인접한 네 방향으로 확산된다.
- 인접한 방향에 공기청정기가 있거나, 칸이 없으면 그 방향으로는 확산이 일어나지 않는다.
- 확산되는 양은 Ar,c/5이고 소수점은 버린다. 즉, ⌊Ar,c/5⌋이다.
- (r, c)에 남은 미세먼지의 양은 Ar,c - ⌊Ar,c/5⌋×(확산된 방향의 개수) 이다.

17144: 미세먼지 안녕!

진짜 단순 구현: T초마다 1. 확산 2. shift 해주면 됨

```java
int ax = airCleaner[0].x;
int ay = airCleaner[0].y;
// 위쪽 공기청정기의 바람은 반시계방향으로 순환
// [top cleaner] ↓ down shift
① for (int x = ax - 1; x > 0; x--) {
    map[x][ay] = map[x - 1][ay];
}
// [top cleaner] ← left shift
② for (int y = 0; y < c - 1; y++) {
    map[0][y] = map[0][y + 1];
}
// [top cleaner] ↑ up shift
③ for (int x = 0; x < ax; x++) {
    map[x][c - 1] = map[x + 1][c - 1];
}
// [top cleaner] → right shift
④ for (int y = c - 1; y > 1; y--) {
    map[ax][y] = map[ax][y - 1];
}
map[ax][ay + 1] = 0;
```



바람이 시작하는 곳부터 shift하면 tmp를 저장하는 게 조금 까다로울 거 같아서 공청기에 들어오는 바람부터 shift해줬음
dx, dy써서 돌리는 게 더 깔끔한 방법인 듯.
그냥 처음 생각난 방법으로 풀었다

```java
// 아래쪽 공기청정기의 바람은 시계방향으로 순환
ax = airCleaner[1].x;
ay = airCleaner[1].y;
// [bottom cleaner] ↑ up shift
⑤ for (int x = ax + 1; x < r - 1; x++) {
    map[x][0] = map[x + 1][0];
}
// [bottom cleaner] ← left shift
⑥ for (int y = 0; y < c - 1; y++) {
    map[r - 1][y] = map[r - 1][y + 1];
}
// [bottom cleaner] ↓ down shift
⑦ for (int x = r - 1; x > ax; x--) {
    map[x][c - 1] = map[x - 1][c - 1];
}
// [bottom cleaner] → right shift
⑧ for (int y = c - 1; y > 1; y--) {
    map[ax][y] = map[ax][y - 1];
}
map[ax][ay + 1] = 0;
```

32643: 정민이의 수열 제조법

알고리즘 분류

사실 분류를 보고 풀 수 있었지, 코테에서 나왔으면 문제 이해 못하고 버렸을 것 같다.

- 수학
- 정수론
- 누적 합
- 소수 판정
- 에라토스테네스의 체

1부터 N까지 한 개 씩 들어있는 수열을 만들기 위해 최소 수열을 준비했고, 익준이는 이 수열에 a이상 b이하 수가 몇 개인지 물어본다.

수열에 있는 정수를 제곱하여 추가, 수열에 있는 두 정수를 곱하여 수열해 추가하는 작업을 반복해서, 1~N까지 숫자가 하나씩 들어있는 수열을 만들 수 있어야 함

1~10을 하나씩 만들기 위해 최소 수열에 필요한 수들

- 1: 제곱, 곱셈으로 만들 수 없음 == 필수
- 2: 제곱, 곱셈으로 만들 수 없음 == 필수
- 3: 제곱, 곱셈으로 만들 수 없음 == 필수
- 4: 2^2로 만들 수 있음 필수x
- 5: 제곱, 곱셈으로 만들 수 없음 == 필수
- 6: 2*3으로 만들 수 있음 필수x
- 7: 제곱, 곱셈으로 만들 수 없음 == 필수
- 8: 2*4로 만들 수 있음 필수 x
- 9: 3^2으로 만들 수 있음 필수 x
- 10: 2*5로 만들 수 있음 필수 x

⇒ 1~10까지 최소 수열은 {1, 2, 3, 5, 7} 1을 제외하면 10까지 소수 집합이 된다.

1 + N까지 소수를 구하면 됨

32643: 정민이의 수열 제조법

이 문제가 소수 구하기 문제라는 걸 깨닫는 게 어렵지, 나머지는 그냥 나와 있는 알고리즘을 사용하면 됨

```
// 소수 배열 만들기(에라토스테네스의 체)
primes = new boolean[n + 1];
Arrays.fill(primes, val: true);
for (int i = 2; i * i <= n; i++) {
    if (!primes[i]) continue;

    for (int j = i * i; j <= n; j += i) {
        primes[j] = false;
    }
}
```

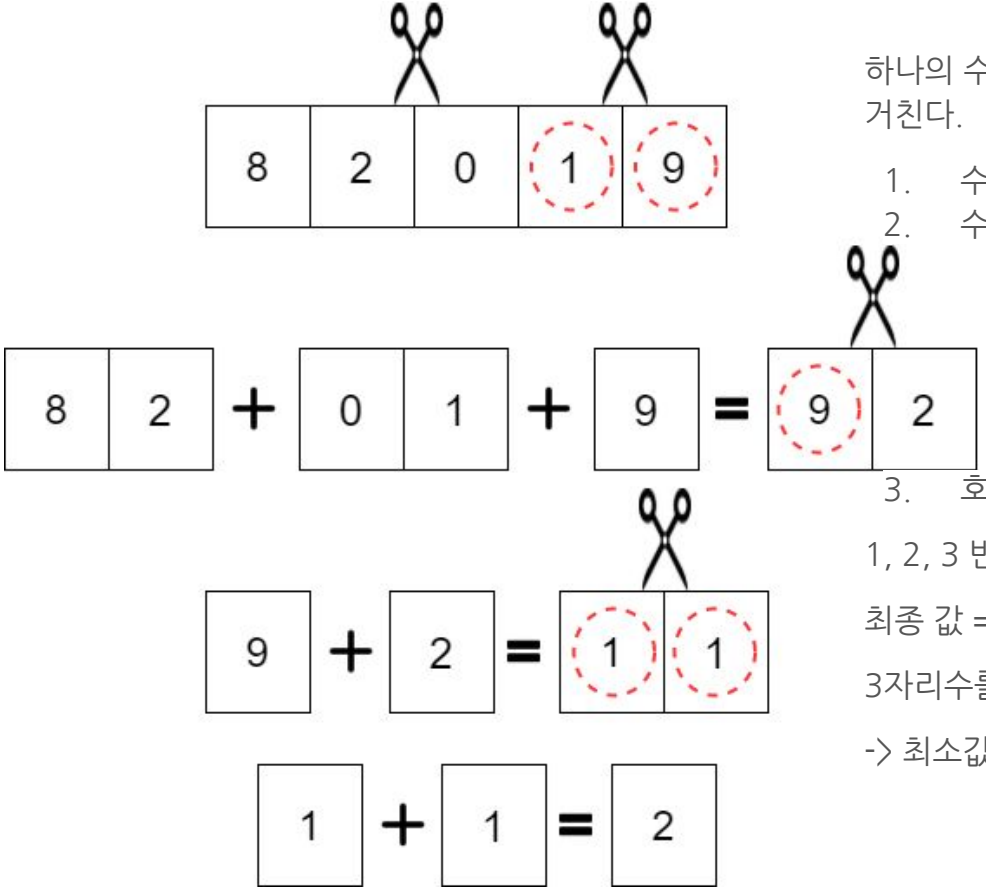
에라토스테네스의 체

2는 소수임.
2*3, 2*4, …은 전부 소수가 아님
근데 보통 약수는 짝지어져 있음
예를 들어 8은 {1, 2, 4, 8} 1*8과 2*4가 짝임
그니까 sqrt(n)까지만 보면 그 뒤는 다 본 거나 진배 없음
왜냐하면 곱해지는 두 수가 sqrt(n)보다 크면 n을 넘으니까
9는 1, 3, 9잖아. 3=루트9 3보다 큰 수 2개가 곱해지면 9보다 클 수 밖에 없음 => 이걸 이용해서 sqrt(n)까지만 보면 된다는 수학 머시기

```
// 질의가 100만개라서 i까지의 소수의 개수를 미리 계산해둬야 빠름
int[] prefixSum = new int[n + 1];
prefixSum[1] = 1; // 1은 소수가 아니지만 수열을 만들 때 필요하므로 소수 취급 함
for (int i = 2; i <= n; i++) {
    prefixSum[i] = (primes[i]) ? prefixSum[i - 1] + 1 : prefixSum[i - 1];
}
```

익준이가 질문을 100만 개나 하기 때문에 i까지 소수 개수를 미리 구해둬야 빠르게 대답해줄 수 있음

20164: 홀수 홀릭 호석



하나의 수가 주어졌을 때 호석이는 한 번의 연산에서 다음과 같은 순서를 거친다.

1. 수의 각 자리 숫자 중에서 홀수의 개수를 종이에 적는다.
2. 수의 자릿수에 따라 아래 작업 수행
   - 수가 **한 자리**이면 더 이상 아무것도 하지 못하고 종료한다.
   - 수가 **두 자리**이면 2개로 나눠서 합을 구하여 새로운 수로 생각한다.
   - 수가 **세 자리 이상**이면 임의의 위치에서 끊어서 3개의 수로 분할하고, 3개를 더한 값을 새로운 수로 생각한다.
3. 호석이는 연산이 종료된 순간에 종이에 적힌 수들을 모두 더한다.

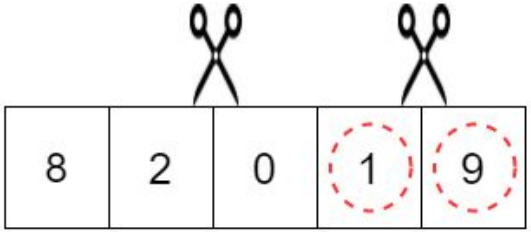
1, 2, 3 반복

최종 값 = 한자릿 수 만들 때까지 나온 홀수 개수의 합

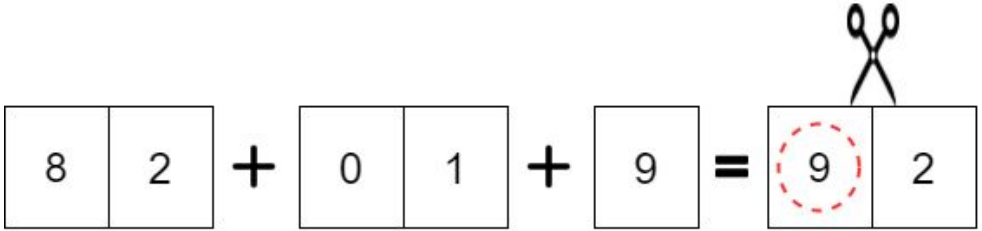
3자리수를 어떻게 나누냐에 따라 홀수 개수가 달라짐

-> 최소값 최대값 출력하기

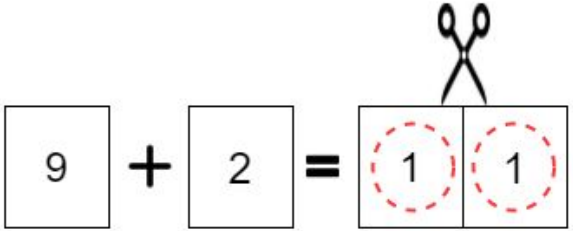
20164: 홀수 홀릭 호석



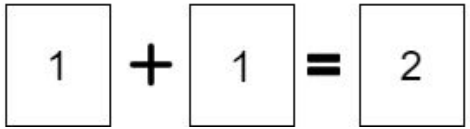
82019에서 각 자리 숫자 중 홀수 개수 세기
세 자리 이상 이므로 임의의 위치에서 끊어서 3개의 수로 분할
3개의 수를 더한 값을 새로운 수로 생각
82+01+9 = 92



92에서 각 자리 숫자 중 홀수 개수 세기
두 자릿 수이므로 2개로 나눠서 합을 구한 후 새로운 수로 생각
9+2 = 11

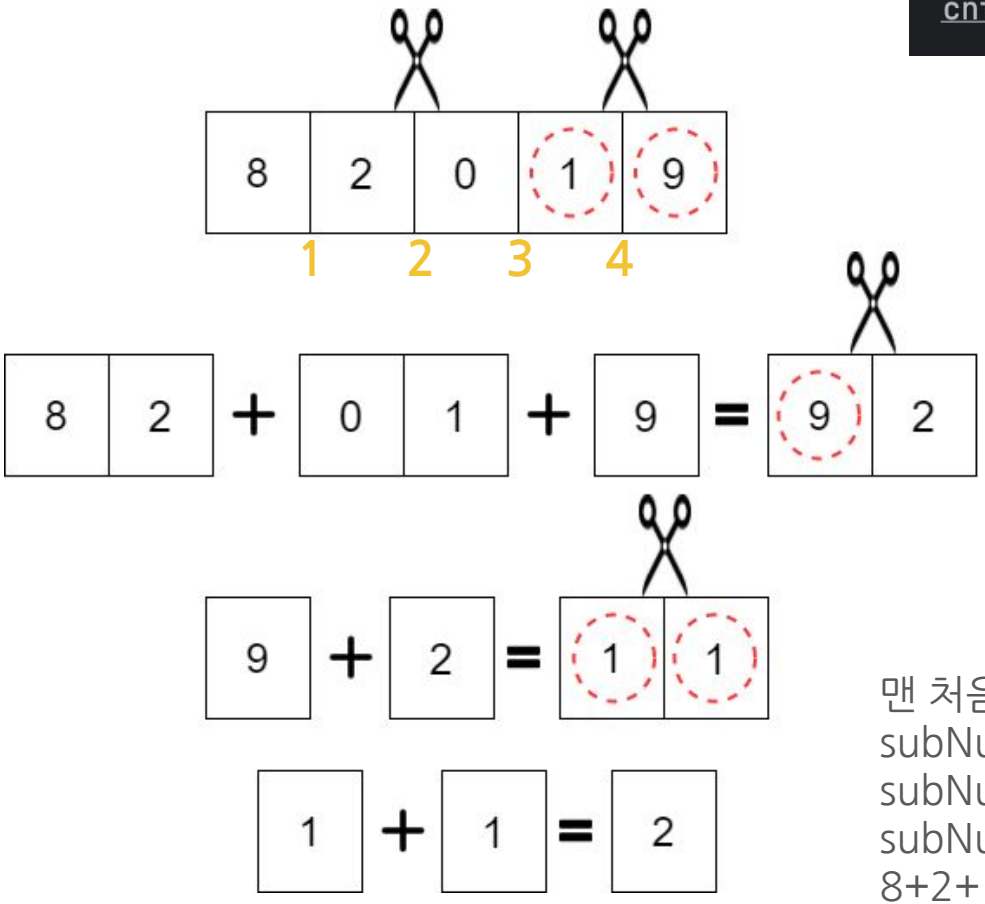


11에서 각 자리 숫자 중 홀수 개수 세기
두 자릿 수이므로 2개로 나눠서 합을 구한 후 새로운 수로 생각
1+1 = 2



2에서 각 자리 숫자 중 홀수 개수 세기(없음)
2는 한 자리 수이므로 아무 것도 하지 못하고 종료

20164: 홀수 홀릭 호석




```
// 1. 각 자리 수 홀수 세기
cnt += CountOdd(num);
```

3자리 이상일 때는 가위 2개의 위치를 구하면 됨
이중 for문으로 가위 위치 정하기

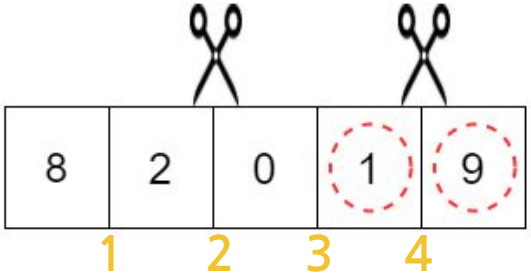
```
// 2-3. 3자리 수 = 나누는 곳이 2곳
for (int i = 1; i < len - 1; i++) {
    for (int j = i + 1; j < len; j++) {
        // substring은 startIdx부터 endIdx-1까지 자름
        int subNum1 = Integer.parseInt(s.substring(0, i));
        int subNum2 = Integer.parseInt(s.substring(i, j));
        int subNum3 = Integer.parseInt(s.substring(j));

        recur( num: subNum1 + subNum2 + subNum3, cnt);
    }
}
```

맨 처음 i = 1, j=2라면
subNum1 = substring(0, 1) → 0 ~ 1-1=0 → s[0]
subNum2 = substring(1, 2) → 1~2-1 = 1 → s[1]
subNum3 = substring(2) → 2~ → s[2:]
8+2+19 = 29를 재귀호출하게 됨

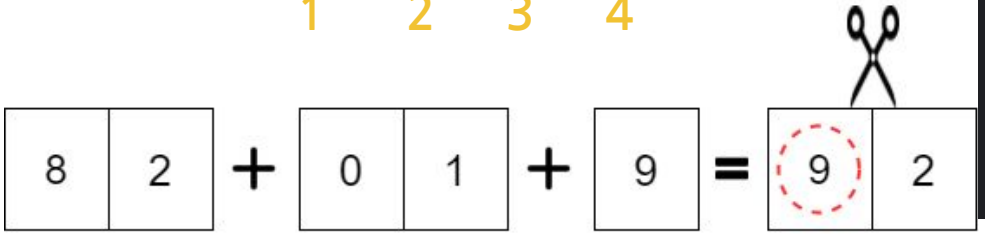
20164: 홀수 홀릭 호석

```
// 1. 각 자리 수 홀수 세기
cnt += CountOdd(num);
```

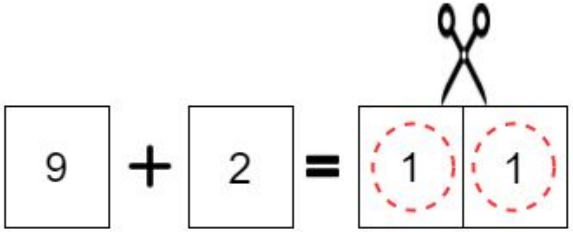
2자리는 그냥 십의자리 + 일의자리 해주면 됨

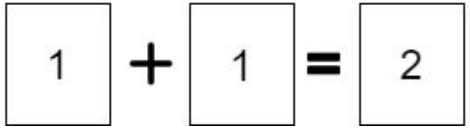



```
// 2-2. 2자리 수
if (len == 2) {
    int next = (num / 10) + (num % 10);
    recur(next, cnt);
    return;
}
```

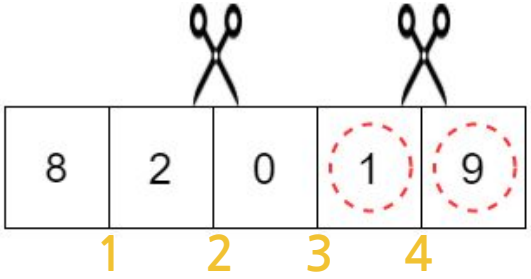


82 → 8+2 = 10
10을 재귀 호출

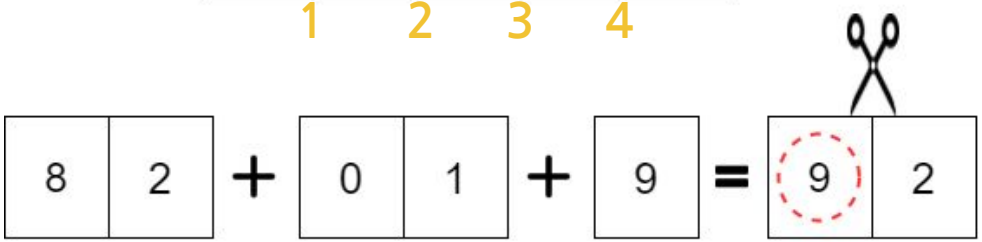


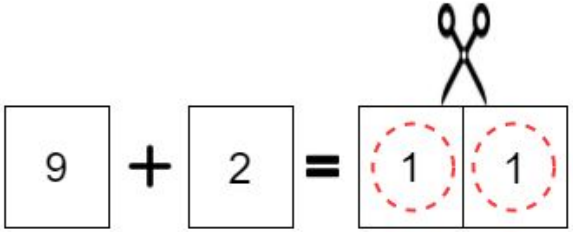


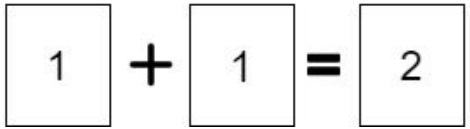
20164: 홀수 홀릭 호석




```
// 1. 각 자리 수 홀수 세기
cnt += CountOdd(num);
```








```
// 2-1. 한자리 수인 경우 갱신
if (num < 10) {
    maxCnt = Math.max(maxCnt, cnt);
    minCnt = Math.min(minCnt, cnt);
    return;
}
```

1자리만 남았으면 할 일이 이제 끝남
이때동안 셌던 홀수의 개수로 maxCnt, minCnt 갱신

20164: 홀수 홀릭 호석


```java
// num은 현재 수, cnt는
public static void recur(int num, int cnt) {  3 usages
    // 1. 각 자리 수 홀수 세기
    cnt += CountOdd(num);

    // 2-1. 한자리 수인 경우 갱신
    if (num < 10) {
        maxCnt = Math.max(maxCnt, cnt);
        minCnt = Math.min(minCnt, cnt);
        return;
    }

    // 두자리 이상인 경우 나눠야 함
    String s = String.valueOf(num);
    int len = s.length();

    // 2-2. 2자리 수
    if (len == 2) {
        int next = (num / 10) + (num % 10);
        recur(next, cnt);
        return;
    }

    // 2-3. 3자리 수 = 나누는 곳이 2곳
    for (int i = 1; i < len - 1; i++) {
        for (int j = i + 1; j < len; j++) {
            // substring은 startIdx부터 endIdx-1까지 자름
            int subNum1 = Integer.parseInt(s.substring(0, i));
            int subNum2 = Integer.parseInt(s.substring(i, j));
            int subNum3 = Integer.parseInt(s.substring(j));

            recur( num: subNum1 + subNum2 + subNum3, cnt);
        }
    }
}

private static int CountOdd(int num) {  1 usage
    int cnt = 0;
    while (num > 0) {
        int digit = num % 10;
        if (digit % 2 != 0) cnt++;
        num /= 10;
    }
    return cnt;
}
```