학습내용

백준 2591. 숫자카드

결과	메모리	시간	언어	코드 길이
맞았습니다!! ✔	12772 KB	88 ms	Java 8 / 수정	1073 B

```java
import java.util.Arrays;
import java.util.Scanner;

public class Main {
    public static void main(String[] args) {
        Scanner sc= new Scanner(System.in);

        String str = sc.next();
        int N = str.length(); // N : 입력값의 길이
        int [] arr = new int [N]; // arr[] : str을 int 형으로 형변환해 담을 배열

        for(int i = 0 ; i < N ; i++) {
            arr[i] = Character.getNumericValue(str.charAt(i));
        }
        int [] dp = new int [N]; // dp[i] i번째 숫자까지 해석할 수 있는 가짓수
        dp[0] = 1; // dp[0] 은 무조건 1

        if(arr[1] != 0)
            dp[1] += dp[0];

        if((arr[0]*10+arr[1])>=10 && (arr[0]*10+arr[1])<=34)
            dp[1] += 1;


        ////////////////////////////// 세팅 끝


        for(int i = 2 ; i < N ; i++) {
            // 1 .한자리 숫자 가능한지 체크
            if(arr[i] != 0) { // 0이 아니면 한자리 숫자 가능
                dp[i] += dp[i-1];
            }
            // 2. 두자리 숫자 가능한지 체크
            if((arr[i-1]*10+arr[i])>=10 && (arr[i-1]*10+arr[i])<=34) {
                dp[i] += dp[i-2];
            }
        }


        System.out.println(dp[N-1]);


    } // main 끝
}
```

백준 2839. 설탕배달

문제	결과	메모리	시간	언어	코드 길이
2839	맞았습니다!! ✔	12884 KB	104 ms	Java 8 / 수정	1150 B

```java
import java.util.Scanner;

public class Main {
    public static void main(String[] args) {
        Scanner sc= new Scanner(System.in);

        // 0행 => 5키로 봉지의 개수 0개
        // 1행 => 5키로 봉지의 개수 1개

        // n행 m열 => 5키로 봉지 n개와 3키로 봉지로 m키로를 만들때 필요한 봉지 개수
        // ex) 3행 18열 => 5키로 봉지 3개와 3키로 봉지 1개로 18를 만들수 있기 때문에
        //                 3행 18열에 3+1 =4 값을 입력

        // N : 3~5000
        // 필요한 행의 개수 N % 5 : 0~1000
        // 5000 * 1000 => 5,000,000

        int result = Integer.MAX_VALUE; // 결과값을 담을 변수
        int N = sc.nextInt(); // N : 3kg 과 5kg 으로 만들어야 하는 양

        for(int five = 0, three ; five <= N/5 ; five++) {

            if((N - 5*five)%3 ==0 ) { // N 을 5와 3으로 분할 가능하면
                three = (N - 5*five) / 3;

                result = Math.min(result, five + three);
            }
            // N을 5와 3으로 분할 불가능하면 아무것도 할 필요 없음.
            // else 안써도 됨.
        }

        if(result ==Integer.MAX_VALUE)
            System.out.println(-1);
        else
            System.out.println(result);

    } // main 끝
}
```

프로그래머스.양궁대회

```java
import java.util.Arrays;

class Solution {

    static int[] Lion = new int [11]; // 라이언이 각 점수마다 맞춘 화살 개수 배열
    static int scoreDiff = -1;
    static int [] answer = {-1};
    public int[] solution(int n ,int[] info ) {



        backtracking(n,info,0,0);
```

```
        return answer;
    }

    void backtracking(int n,int[] info, int index, int count) {

        // 기저조건 : n 이 0이되면 더이상 화살을 쏠수 없다고 판단 =>
        //          점수차이를 계산한 후 return
        if(index == 11) {
            if(count==n) {

                int ApeachScore = 0; // 어피치 점수와
                int LionScore = 0;  // 라이언 점수를 0으로 초기화

                for(int i = 0 ; i <= 10 ; i++) {
                    if(Lion[i]==0 && info[i]==0) // 둘다 0이면 서로 점수 X
                        continue;

                    if(Lion[i]<=info[i]) // 어피치 승
                        ApeachScore += 10-i;

                    else // 라이언 승
                        LionScore += 10-i;
                }
                // 전부 계산한 결과 라이언이 이긴경우에 answer에 Lion 배열 깊은복사
                if( LionScore > ApeachScore ){
                    if( scoreDiff < LionScore - ApeachScore ) {
                        scoreDiff = LionScore - ApeachScore;
                        answer = Lion.clone();
                    }

                    else if(scoreDiff == LionScore - ApeachScore){
                        for(int i = 10 ; i>=0 ; i--){
                            if(answer[i] < Lion[i]){
                                answer = Lion.clone();
                                return;
                            }
                            else if(answer[i] > Lion[i])
                                return;
                        }
                    }
                }
            }
            return;
        } // 기저조건 끝


        if(info[index] == 0) { // 둘다 0점으로 무승부
            backtracking(n, info, index+1, count);
        }

        if(count + 1 + info[index] <= n) { // 현재 사용한 화살수 + 1에 어피치화살수를 더해도 전체 화살수가 넘지않으면
            Lion[index] = info[index] + 1;
            backtracking(n,info,index+1,count+1+info[index]);
            Lion[index] = 0;
        }

        if(info[index] != 0) {
            for(int i = 0 ; i <= info[index]; i++) {
                Lion[index] = i;
                backtracking(n,info,index+1,count+i);
                Lion[index] = 0;
            }
        }
    } // backtracking 끝
```

```
}
```

백준 21921. 블로그

문제	결과	메모리	시간	언어	코드 길이
21921	맞았습니다!! ✔	199820 KB	852 ms	Java 8 / 수정	979 B

```java
import java.util.Arrays;
import java.util.Scanner;

public class Main {
    public static void main(String[] args) {
        Scanner sc= new Scanner(System.in);

        int N = sc.nextInt(); // N : 블로그 운영 기간 1~250,000
        int X = sc.nextInt(); // X : 최대 방문자수 기간 1~N
        int result = 0 ; // 방문자수
        int count = 0; // 2번째 줄에 출력할 기간 개수

        int [] arr = new int [N+1];

        for(int i = 1 ; i < N+1 ; i++) { // 입력 받고 누적합 구하기
            arr[i] = arr[i-1] + sc.nextInt();
        }

        for(int i = 0 ; i < N-X+1 ; i++) {
            if(result < arr[i+X]-arr[i]) { //i번째부터 i+X 까지의 합이 더 크면 갱신후 count를 1로 초기화
                result = arr[i+X]-arr[i];
                count = 1;
            }
            else if(result == arr[i+X]-arr[i]){ // 똑같은 값이 있으면 count++
                count++;
            }
        }

        if(result==0)
            System.out.println("SAD");
        else {

            System.out.println(result);
            System.out.println(count);
        }


    } // main 끝
}
```

프로그래머스.다단계 칫솔 판매

```java
import java.util.*;

class Solution {
    public int[] solution(String[] enroll, String[] referral, String[] seller, int[] amount) {
```

```
        int[] profit = new int[enroll.length]; // 각 판매원의 최종 수익을 저장할 배열

        // 이름을 인덱스와 매핑하는 HashMap 생성
        Map<String, Integer> nameToIndex = new HashMap<>();
        for (int i = 0; i < enroll.length; i++) {
            nameToIndex.put(enroll[i], i);
        }

        // 판매 데이터 처리
        for (int i = 0; i < seller.length; i++) {
            String currentSeller = seller[i];
            int totalProfit = amount[i] * 100;

            while (!currentSeller.equals("-")) { // 추천인이 없을 때는 종료
                int sellerIndex = nameToIndex.get(currentSeller);
                int toParent = totalProfit / 10; // 10%는 추천인에게 전달
                profit[sellerIndex] += totalProfit - toParent;

                if (toParent == 0) break; // 전달할 금액이 없으면 종료

                // 추천인으로 이동
                currentSeller = referral[sellerIndex];
                totalProfit = toParent;
            }
        }

        return profit;
    }
}
```

백준 1158. 요세푸스

문제	결과	메모리	시간	언어	코드 길이
1158	맞았습니다!! ✔	18252 KB	260 ms	Java 8 / 수정	802 B

```
import java.util.LinkedList;
import java.util.Scanner;

public class Main {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);

        int N = sc.nextInt(); // N : 1번부터 N번까지 N명의 사람
        int K = sc.nextInt(); // K : 순서대로 K번째 사람을 제거

        LinkedList<Integer> list = new LinkedList<>();
        for(int i = 1 ; i <= N ; i ++) {
            list.offer(i);
        }

        System.out.print("<");
        int index = K-1;
        while(true) {
            System.out.print(list.get(index));
            list.remove(index);
            // K 삭제하고
            if(list.isEmpty()) break;
```

```
            //전부 소모했으면 탈출
            //아직 소모 안했으면" ,"출력하기
            System.out.print(", ");
            index = (index +(K-1))%list.size();

        } // while 끝
        System.out.println(">");

    } // main 끝
}
```

백준 2529. 부등호

문제	결과	메모리	시간	언어	코드 길이
2529	맞았습니다!! ✔	27888 KB	156 ms	Java 8 / 수정	1255 B

```
import java.util.ArrayList;
import java.util.List;
import java.util.Scanner;

public class Main {

    static int k; // k : 부등호 개수
    static String [] arr; // arr[] : 부등호를 받을 배열
    static List<String> result = new ArrayList<>(); //부등호를 만족하는 모든 값들을 넣을 배열
    static boolean []visited = new boolean [10]; // 0~9까지 방문체크
    public static void main(String[] args) {
        Scanner sc= new Scanner(System.in);

        k = sc.nextInt();
        arr = new String [k];

        for(int i = 0 ; i < k ; i++) {
            arr[i] = sc.next();
        }

        backtracking(0,"");

        System.out.println(result.get(result.size()-1));
        System.out.println(result.get(0));

    } // main끝

    static void backtracking(int index,String str) {

        if(index==k+1) { // 기저 조건
            result.add(str);
            return;
        }

        for(int i = 0 ; i < 10 ; i++) {
            if(!visited[i]) {
                if(index == 0 || isValid(str.charAt(index-1)-'0',i,arr[index-1])) {
                    visited[i] = true;
                    backtracking(index+1,str+i);
                    visited[i] = false;

                }
```

```
            }

        }

    }

    static boolean isValid(int a , int b , String sign) {
        if(sign.equals("<"))
            return a < b;
        else if(sign.equals(">"))
            return a > b;
        return false;
    }



}
```

코드트리 메두사와 전사

```
import java.util.ArrayList;
import java.util.Arrays;
import java.util.Collection;
import java.util.Collections;
import java.util.LinkedList;
import java.util.List;
import java.util.Queue;
import java.util.Scanner;

public class Main {

    static int N; // N : 마을의 크기 N x N
    static int M; // M : 전사의 수
    static int [] medusa; // medusa[] : 메두사의 위치정보
    static int [] park; // park[] : 공원의 위치정보
    static List<int[]> warrior; // warrior : 전사의 위치정보
    static int [][] map; // map[][] : 마을의 길위치를 담은 배열 (지도)
    static boolean check = false; // check : 메두사가 공원까지 갈수 있는지 여부
    static boolean [][] medusaVision; // medusaVision : 메두사의 시야
    static int [][] medusaRoute; // medusaRoute : 메두사 이동경로
    static List<Integer> medusaMove; // medusaMove : 메두사의 이동순서를 담을 리스트
    static int warriorMoveCount;
    static int warriorStoneCount;
    static int warriorAttackCount;
    static List<Integer> StoneWarrior = new ArrayList<>(); // StoneWarrior : 석화된 전사들의 인덱스를 저장할

    static int [] R = {-1, 1, 0, 0};
    static int [] C = { 0, 0,-1, 1};
                    //  상  하  좌  우

    public static void main(String[] args) {
        Scanner sc =  new Scanner(System.in);

        N = sc.nextInt();
        M = sc.nextInt();

        // 메두사 위치 입력받기
        medusa = new int [2];
```

```
        medusa[0] = sc.nextInt();
        medusa[1] = sc.nextInt();

        // 공원 위치 입력받기
        park = new int [2];
        park[0] = sc.nextInt();
        park[1] = sc.nextInt();

        // 전사 위치 입력받기
        warrior = new ArrayList<>();
        for(int i = 0 ; i < M ; i++)
            warrior.add(new int[2]);

        for(int r = 0 ; r < M ; r++)
            for(int c = 0 ; c < 2 ; c++)
                warrior.get(r)[c] =sc.nextInt();

        // 길 위치 입력받기 (지도)
        map = new int [N][N];
        for(int r = 0 ; r < N ; r++)
            for(int c = 0 ; c < N ; c++)
                map[r][c] = sc.nextInt();

        // 메두사의 시야 초기화
        medusaVision = new boolean [N][N];
        // 메두사 경로 초기화
        medusaRoute = new int [N][N];
        // 메두사 이동순서 초기화
        medusaMove = new ArrayList<>();
        // 메두사의 처음위치 -1로 초기화
        medusaRoute[medusa[0]][medusa[1]] = -1;

        checkBFS();
        if(!check) // 메두사가 공원까지 갈 수 없는 상황이라면 -1를 출력
            System.out.println(-1);
        else { // 메두사가 공원까지 갈 수 있음
            int medusaX = medusa[0];
            int medusaY = medusa[1];

            for(int i = 0 ; i < medusaMove.size() ; i++) { // 메두사의 이동횟수만큼 턴진행
                // 메두사의 위치 파악
                medusaX = medusaMove.get(i)/N;
                medusaY = medusaMove.get(i)%N;

                warriorMoveCount = 0;
                warriorStoneCount = 0;
                warriorAttackCount = 0;

            // 1. 메두사 이동
                // 메두사가 이동한후 전사들의 위치와 겹치면 전사들 사망
                for(int j = warrior.size()-1 ; j>=0 ; j--) {
                    if(warrior.get(j)[0]==medusaX && warrior.get(j)[1]==medusaY) {
                        // 메두사가 전사의 위치로 이동하면 아무 카운트도 증가시키지 않고 전사들만 사망
                        warrior.remove(j); // j번째 전사 사망
                    }
                }

            // 2.전사들의 위치를 파악해 메두사의 시야방향을 결정할 알고리즘 필요
                findMedusaVision(medusaX,medusaY);

            // 3 && 4. 전사들의 첫번째 이동 및 공격
                for(int j = warrior.size()-1 ; j >=0 ; j--) {
                    firstStep(j,medusaX,medusaY);
                }
```

```java
            // 3 && 4. 전사들의 두번째 이동 및 공격
                for(int j = warrior.size()-1 ; j >=0 ; j--) {
                    seconedStep(j,medusaX,medusaY);
                }

                StoneWarrior.clear(); // 턴이 끝났으면 석화된 전사들을 초기화
                System.out.println(warriorMoveCount + " " + warriorStoneCount + " " + warriorAttackCoun
            } // medusaMove(메두사 이동) for문 끝
            System.out.println(0);
        } // else 끝 (메두사가 공원으로 이동 가능한경우)

    } // main 끝




    // 메두사의 시야 방향을 결정할 메서드
    private static void findMedusaVision(int medusaX, int medusaY) {
        int [][] mapWarrior = new int [N][N];
        for(int k = 0 ; k < warrior.size() ; k++) {
            mapWarrior[warrior.get(k)[0]][warrior.get(k)[1]] += 1;
        } // 맵의 각위치에 전사들의 수를 입력한 2차원 배열

//      for(int m = 0 ; m < N ;m++)
//      System.out.println(Arrays.toString(mapWarrior[m]));
//
//      System.out.println();



        boolean [][] visited = new boolean[N][N];
        int count = 0; // 석화된 전사들의 수를 계산할 변수
        int maxValue=0; // 석화된 전사들의 수를 계산해
        // 싱히좌우 순서대로 살펴본후  maxValue 보다 큰경우 값 갱신
        boolean isUp = true;
        boolean isDown = true;
        boolean isLeft = true;
        boolean isRight = true;

        int [][] medusaVisionUp = new int[N][N]; // 메두사의 위쪽시야를 체크할 2차원 배열
        int [][] medusaVisionDown = new int[N][N]; // 메두사의 아래쪽시야를 체크할 2차원 배열
        int [][] medusaVisionLeft = new int[N][N]; // 메두사의 왼쪽시야를 체크할 2차원 배열
        int [][] medusaVisionRight = new int[N][N]; // 메두사의 오른쪽시야를 체크할 2차원 배열
        int x = medusaX; // 메두사의 위치로부터 위쪽방향으로 한칸씩 이동할 예정
        int y = medusaY;
        int index = 1; // index 를 조정하여 한칸,두칸씩 대각선방향을 탐색;



        // 상상상상상상상상상상상상상상상상상상상상상상상상상상상상상상상상상상상상상상상상상상상상상상상상상상상상상상상
        // 상상상상상상상상상상상상상상상상상상상상상상상상상상상상상상상상상상상상상상상상상상상상상상상상상상상상상상
        while(x>=0) { // 한칸씩 위로 올라가다가 끝점에 도달할때까지 반복
            // 해당위치(x,y)에서 isUp이 true면 1 값을 입력
            if(isUp) {
                medusaVisionUp[x][y] = 1;
            }
            // x,y에 true 값을 입력한후 만약 해당위치에 전사가 있다면 isUp을 false 로 바꾸기
            if(mapWarrior[x][y]!=0)
                isUp = false;
```

```
            // x,y(메두사의 위치)를 기준으로 왼쪽대각선을 탐색
            // x-index , y-index
            while(x-index >= 0 && y-index >= 0) {
                // 해당위치의 medusaVisionUp이 -1 로 입력되어있으면
                // isLeft 를 false 로바꾸기
                if(medusaVisionUp[x-index][y-index] == -1)
                    isLeft = false;

                if(isLeft) // isLeft 가 true 인 상태이면 메두사의 시야로 판단, 1입력
                    medusaVisionUp[x-index][y-index] = 1;

                if(mapWarrior[x-index][y-index]!=0) {
                    isLeft = false;
                    // 전사들의 위쪽medusaVisionUp에 -1를 입력해야함
                    int i = 1;
                    while(x-index-i>=0) { // 전사들의 위쪽을 -1 로 바꾸다가 범위를 벗어나면 중지
                        medusaVisionUp[x-index-i][y-index] = -1;
                        i++;
                    }
                }
                index++;
            } // 왼쪽대각선 탐색 while문 끝
            isLeft = true;
            index = 0;

            while(x-index >=0 && y+index < N) {
                // 해당위치의 medusaVisionUp이 -1 로 입력되어있으면
                // isRight 를 false 로바꾸기
                if(medusaVisionUp[x-index][y+index] == -1)
                    isRight = false;

                if(isRight) // isRight 가 true 인 상태이면 메두사의 시야로 판단, 1입력
                    medusaVisionUp[x-index][y+index] = 1;

                if(mapWarrior[x-index][y+index]!=0) {
                    isRight = false;
                    // 전사들의 위쪽medusaVisionUp에 -1를 입력해야함
                    int i = 1;
                    while(x-index-i>=0) { // 전사들의 위쪽을 -1 로 바꾸다가 범위를 벗어나면 중지
                        medusaVisionUp[x-index-i][y+index] = -1;
                        i++;
                    }
                }
                index++;
            } // 오른쪽대각선 탐색 while문 끝
            isRight = true;
            index = 0;
            x -= 1;
        } // 위쪽방향 시야를 체크하는 while 문 끝

        medusaVisionUp[medusaX][medusaY] = 0; // 메두사의 위치는 시야에서 없애기
        isUp= true;
        isDown= true;
        isLeft= true;
        isRight= true;
        x = medusaX;
        y = medusaY;
        index = 1;
        // 위쪽시야에 전사들의 수가 몇명인지 체크후 갱신하는 알고리즘 필요
        for (int r = 0; r < N; r++)
            for (int c = 0; c < N; c++)
                if(medusaVisionUp[r][c] == 1)
                    count += mapWarrior[r][c];
```

```
        if(maxValue<count) {
            maxValue = count;
            // medusaVision 을 medusaVisionUp으로 다채우기
            for (int r = 0; r < N; r++) {
                for (int c = 0; c < N; c++) {
                    if(medusaVisionUp[r][c] == 1)
                        medusaVision[r][c] = true;
                    else
                        medusaVision[r][c] = false;
                }
            }
        } // if(maxValue<count) { 끝
        count = 0;

        //하하하하하하하하하하하하하하하하하하하하하하하하하하하하하하하하하하하하하하하하하하하하하하하하하하하하하하하하하하하
        //하하하하하하하하하하하하하하하하하하하하하하하하하하하하하하하하하하하하하하하하하하하하하하하하하하하하하하하하하하하

        while(x<N) { // 한칸씩 아래로 내려가다가 끝점에 도달할때까지 반복
            // 해당위치(x,y)에서 isDown이 true면 1 값을 입력
            if(isDown) {
                medusaVisionDown[x][y] = 1;
            }
            // x,y에 true 값을 입력한후 만약 해당위치에 전사가 있다면 isUp을 false 로 바꾸기
            if(mapWarrior[x][y]!=0)
                isDown = false;

            // x,y(메두사의 위치)를 기준으로 왼쪽대각선을 탐색
            // x+index , y-index
            while(x+index < N && y-index >= 0) {
                // 해당위치의 medusaVisionDown이 -1 로 입력되어있으면
                // isLeft 를 false 로바꾸기
                if(medusaVisionDown[x+index][y-index] == -1)
                    isLeft = false;

                if(isLeft) // isLeft 가 true 인 상태이면 메두사의 시야로 판단, 1입력
                    medusaVisionDown[x+index][y-index] = 1;

                if(mapWarrior[x+index][y-index]!=0) {
                    isLeft = false;
                    // 전사들의 아래쪽 medusaVisionDown에 -1를 입력해야함
                    int i = 1;
                    while(x+index+i < N) { // 전사들의 아래쪽을 -1 로 바꾸다가 범위를 벗어나면 중지
                        medusaVisionDown[x+index+i][y-index] = -1;
                        i++;
                    }
                }
                index++;
            } // 왼쪽대각선 탐색 while문 끝
            isLeft = true;
            index = 0;

            // 오른쪽 대각선 탐색
            while(x+index <N && y+index < N) {
                // 해당위치의 medusaVisionDown이 -1 로 입력되어있으면
                // isRight 를 false 로바꾸기
                if(medusaVisionDown[x+index][y+index] == -1)
                    isRight = false;

                if(isRight) // isLeft 가 true 인 상태이면 메두사의 시야로 판단, 1입력
                    medusaVisionDown[x+index][y+index] = 1;

                if(mapWarrior[x+index][y+index]!=0) {
                    isRight = false;
                    // 전사들의 위쪽medusaVisionUp에 -1를 입력해야함
```

```
                int i = 1;
                while(x+index+i<N) { // 전사들의 아래쪽을 -1 로 바꾸다가 범위를 벗어나면 중지
                    medusaVisionDown[x+index+i][y+index] = -1;
                    i++;
                }
            }
            index++;
        } // 오른쪽대각선 탐색 while문 끝
        isRight = true;
        index = 0;
        x += 1;
    } // 아래쪽방향 시야를 체크하는 while 문 끝

    medusaVisionDown[medusaX][medusaY] = 0; // 메두사의 위치는 시야에서 없애기
    isUp= true;
    isDown= true;
    isLeft= true;
    isRight= true;
    x = medusaX;
    y = medusaY;
    index = 1;
    // 아래쪽시야에 전사들의 수가 몇명인지 체크후 갱신하는 알고리즘 필요
    for (int r = 0; r < N; r++)
        for (int c = 0; c < N; c++)
            if(medusaVisionUp[r][c] == 1)
                count += mapWarrior[r][c];

    if(maxValue<count) {
        maxValue = count;
        // medusaVision 을 medusaVisionUp으로 다채우기
        for (int r = 0; r < N; r++) {
            for (int c = 0; c < N; c++) {
                if(medusaVisionDown[r][c] == 1)
                    medusaVision[r][c] = true;
                else
                    medusaVision[r][c] = false;
            }
        }
    } // if(maxValue<count) { 끝
    count = 0;

    // 좌좌좌좌좌좌좌좌좌좌좌좌좌좌좌좌좌좌좌좌좌좌좌좌좌좌좌좌좌좌좌좌좌좌좌좌좌좌좌좌좌좌좌좌좌좌좌
    // 좌좌좌좌좌좌좌좌좌좌좌좌좌좌좌좌좌좌좌좌좌좌좌좌좌좌좌좌좌좌좌좌좌좌좌좌좌좌좌좌좌좌좌좌좌좌좌

    while(y>=0) { // 한칸씩 왼쪽으로 가다가 끝점에 도달할때까지 반복
        // 해당위치(x,y)에서 isLeft이 true면 1 값을 입력
        if(isLeft) {
            medusaVisionLeft[x][y] = 1;
        }
        // x,y에 true 값을 입력한후 만약 해당위치에 전사가 있다면 isLeft을 false 로 바꾸기
        if(mapWarrior[x][y]!=0)
            isLeft = false;

        // x,y(메두사의 위치)를 기준으로 위쪽대각선을 탐색
        // x-index , y-index
        while(x-index >= 0 && y-index >= 0) {
            // 해당위치의 medusaVisionLeft가 -1 로 입력되어있으면
            // isUp 를 false 로바꾸기
            if(medusaVisionLeft[x-index][y-index] == -1)
                isUp = false;

            if(isUp) // isUp 가 true 인 상태이면 메두사의 시야로 판단, 1입력
                medusaVisionUp[x-index][y-index] = 1;
```

```
            if(mapWarrior[x-index][y-index]!=0) {
                isUp = false;
                // 전사들의 위쪽medusaVisionLeft에 -1를 입력해야함
                int i = 1;
                while(y-index-i>=0) { // 전사들의 왼쪽을 -1 로 바꾸다가 범위를 벗어나면 중지
                    medusaVisionLeft[x-index][y-index-i] = -1;
                    i++;
                }
            }
            index++;
        } // 왼쪽대각선 탐색 while문 끝
        isUp = true;
        index = 0;

        // 아래쪽 대각선 탐색
        while(x+index < N && y-index >= 0) {
            // 해당위치의 medusaVisionLeft이 -1 로 입력되어있으면
            // isDown 를 false 로바꾸기
            if(medusaVisionLeft[x+index][y-index] == -1)
                isDown = false;

            if(isDown) // isDown 가 true 인 상태이면 메두사의 시야로 판단, 1입력
                medusaVisionLeft[x+index][y-index] = 1;

            if(mapWarrior[x+index][y-index]!=0) {
                isDown = false;
                // 전사들의 위쪽medusaVisionLeft에 -1를 입력해야함
                int i = 1;
                while(y-index-i>=0) { // 전사들의 왼쪽을 -1 로 바꾸다가 범위를 벗어나면 중지
                    medusaVisionLeft[x+index][y-index-i] = -1;
                    i++;
                }
            }
            index++;
        } // 아래쪽대각선 탐색 while문 끝
        isDown = true;
        index = 0;
        y -= 1;
    } // 왼쪽방향 시야를 체크하는 while 문 끝

    medusaVisionLeft[medusaX][medusaY] = 0; // 메두사의 위치는 시야에서 없애기
    isUp= true;
    isDown= true;
    isLeft= true;
    isRight= true;
    x = medusaX;
    y = medusaY;
    index = 1;
    // 위쪽시야에 전사들의 수가 몇명인지 체크후 갱신하는 알고리즘 필요
    for (int r = 0; r < N; r++)
        for (int c = 0; c < N; c++)
            if(medusaVisionLeft[r][c] == 1)
                count += mapWarrior[r][c];

    if(maxValue<count) {
        maxValue = count;
        // medusaVision 을 medusaVisionUp으로 다채우기
        for (int r = 0; r < N; r++) {
            for (int c = 0; c < N; c++) {
                if(medusaVisionLeft[r][c] == 1)
                    medusaVision[r][c] = true;
                else
                    medusaVision[r][c] = false;
```

```
            }
        }
    } // if(maxValue<count) { 끝
    count = 0;

    // 우우우우우우우우우우우우우우우우우우우우우우우우우우우우우우우우우우우우우우우우우우우우우우
    // 우우우우우우우우우우우우우우우우우우우우우우우우우우우우우우우우우우우우우우우우우우우우우우

    while(y<N) { // 한칸씩 오른쪽으로 가다가 끝점에 도달할때까지 반복
        // 해당위치(x,y)에서 isRight이 true면 1 값을 입력
        if(isRight) {
            medusaVisionRight[x][y] = 1;
        }
        // x,y에 true 값을 입력한후 만약 해당위치에 전사가 있다면 isRight을 false 로 바꾸기
        if(mapWarrior[x][y]!=0)
            isRight = false;

        // x,y(메두사의 위치)를 기준으로 위쪽대각선을 탐색
        // x-index , y+index
        while(x-index >= 0 && y+index < N) {
            // 해당위치의 medusaVisionRight가 -1 로 입력되어있으면
            // isUp 를 false 로바꾸기
            if(medusaVisionRight[x-index][y+index] == -1)
                isUp = false;

            if(isUp) // isUp 이 true 인 상태이면 메두사의 시야로 판단, 1입력
                medusaVisionUp[x-index][y+index] = 1;

            if(mapWarrior[x-index][y+index]!=0) {
                isUp = false;
                // 전사들의 위쪽medusaVisionRight에 -1를 입력해야함
                int i = 1;
                while(y+index+i < N) { // 전사들의 오른쪽을 -1 로 바꾸다가 범위를 벗어나면 중지
                    medusaVisionRight[x-index][y+index+i] = -1;
                    i++;
                }
            }
            index++;
        } // 왼쪽대각선 탐색 while문 끝
        isUp = true;
        index = 0;

        // 아래쪽 대각선 탐색
        while(x+index < N && y+index < N) {
            // 해당위치의 medusaVisionRight이 -1 로 입력되어있으면
            // isDown 를 false 로바꾸기
            if(medusaVisionRight[x+index][y+index] == -1)
                isDown = false;

            if(isDown) // isDown 가 true 인 상태이면 메두사의 시야로 판단, 1입력
                medusaVisionRight[x+index][y+index] = 1;

            if(mapWarrior[x+index][y+index]!=0) {
                isDown = false;
                // 전사들의 위쪽medusaVisionRight에 -1를 입력해야함
                int i = 1;
                while(y+index+i <= 0) { // 전사들의 오른쪽을 -1 로 바꾸다가 범위를 벗어나면 중지
                    medusaVisionRight[x+index][y+index+i] = -1;
                    i++;
                }
            }
            index++;
        } // 아래쪽대각선 탐색 while문 끝
        isDown = true;
```

```
            index = 0;
            y += 1;
        } // 왼쪽방향 시야를 체크하는 while 문 끝

        medusaVisionRight[medusaX][medusaY] = 0; // 메두사의 위치는 시야에서 없애기
        isUp= true;
        isDown= true;
        isLeft= true;
        isRight= true;
        x = medusaX;
        y = medusaY;
        index = 1;
        // 위쪽시야에 전사들의 수가 몇명인지 체크후 갱신하는 알고리즘 필요
        for (int r = 0; r < N; r++)
            for (int c = 0; c < N; c++)
                if(medusaVisionRight[r][c] == 1)
                    count += mapWarrior[r][c];

        if(maxValue<count) {
            maxValue = count;
            // medusaVision 을 medusaVisionRight으로 다채우기
            for (int r = 0; r < N; r++) {
                for (int c = 0; c < N; c++) {
                    if(medusaVisionRight[r][c] == 1)
                        medusaVision[r][c] = true;
                    else
                        medusaVision[r][c] = false;

                }
            }
        } // if(maxValue<count) { 끝
        count = 0;

        warriorStoneCount = maxValue;
        // 석화된 전사들의 수에 maxValue 를 저장

        System.out.println();
            for (int r = 0; r < N; r++) {
                for (int c = 0; c < N; c++) {
                    if(medusaVision[r][c])
                        System.out.print(1 + " ");
                    else
                        System.out.print(0 + " ");
                }
                System.out.println();
            }
            System.out.println();
        } // findMedusaVision 메서드 끝









    static void firstStep(int j, int medusaX, int medusaY) {
        // bfs를 사용할 필요 없이 전사들의 위치가
        // 메두사보다 하,상,우,좌 일때 1칸 이동하는 것으로 충분

        // 석화된 전사들이라면 이동을 체크할 필요없음
        for(int k = 0 ; k <= StoneWarrior.size()-1 ; k++)
            if(StoneWarrior.get(k) == j)
```

```
                return; // 석화된 전사라면 메서드를 끝내 이동하지않기

        int x = warrior.get(j)[0]; // 전사의 x좌표
        int y = warrior.get(j)[1]; // 전사의 y좌표

        //
        if(x > medusaX && !medusaVision[x-1][y]) {
            warrior.get(j)[0] -= 1; // 전사의 위치를 1칸 위로 이동시킨다
            // 이동 시켰으면 카운트++
            warriorMoveCount++;
        }
        else if(x < medusaX && !medusaVision[x+1][y]) {
            warrior.get(j)[0] += 1; // 전사의 위치를 1칸 아래로 이동시킨다
            warriorMoveCount++;
        }
        else if(y > medusaY && !medusaVision[x][y-1]) {
            warrior.get(j)[1] -= 1; // 전사의 위치를 1칸 좌로 이동시킨다
            warriorMoveCount++;
        }
        else if(y < medusaY && !medusaVision[x][y+1]) {
            warrior.get(j)[1] += 1; // 전사의 위치를 1칸 우로 이동시킨다
            warriorMoveCount++;
        }
        // 이동시킨후 해야할 일
        // 이동한결과 메두사의 위치가 전사의 위치와 같으면
        // 전사가 사망하고 메두사를 공격한 전사의 수 카운트++
        if(warrior.get(j)[0] == medusaX && warrior.get(j)[1] == medusaY) {
            warrior.remove(j);
            warriorAttackCount++;
        }


    }

    static void seconedStep(int j, int medusaX, int medusaY) {
        // bfs를 사용할 필요 없이 전사들의 위치가
        // 메두사보다 하,상,우,좌 일때 1칸 이동하는 것으로 충분
        int x = warrior.get(j)[0]; // 전사의 x좌표
        int y = warrior.get(j)[1]; // 전사의 y좌표

        // 석화된 전사들이라면 이동을 체크할 필요없음
        for(int k = 0 ; k <= StoneWarrior.size()-1 ; k++)
            if(StoneWarrior.get(k) == j)
                return; // 석화된 전사라면 메서드를 끝내 이동하지않기

        //
        if(y > medusaY && !medusaVision[x][y-1]) {
            warrior.get(j)[1] -= 1; // 전사의 위치를 1칸 좌로 이동시킨다
            warriorMoveCount++;
        }
        else if(y < medusaY && !medusaVision[x][y+1]) {
            warrior.get(j)[1] += 1; // 전사의 위치를 1칸 우로 이동시킨다
            warriorMoveCount++;
        }
        else if(x > medusaX && !medusaVision[x-1][y]) {
            warrior.get(j)[0] -= 1; // 전사의 위치를 1칸 위로 이동시킨다
            // 이동 시켰으면 카운트++
            warriorMoveCount++;
        }
        else if(x < medusaX && !medusaVision[x+1][y]) {
            warrior.get(j)[0] += 1; // 전사의 위치를 1칸 아래로 이동시킨다
            warriorMoveCount++;
        }
        // 이동시킨후 해야할 일
```

```
        // 이동한결과 메두사의 위치가 전사의 위치와 같으면
        // 전사가 사망하고 메두사를 공격한 전사의 수 카운트++
        if(warrior.get(j)[0] == medusaX && warrior.get(j)[1] == medusaY) {
            warrior.remove(j);
            warriorAttackCount++;
        }
    }


    static void checkBFS() {
        boolean [][] visited = new boolean [N][N];
        Queue<int []> queue = new LinkedList<>();

        visited[medusa[0]][medusa[1]] = true;
        queue.offer(new int [] {medusa[0] , medusa[1]});

        while(!queue.isEmpty()) {
            int [] current = queue.poll();
            int mr = current[0];
            int mc = current[1];

            // 기저조건
            if(mr==park[0] && mc == park[1]) {
                check = true;
                // 메두사가 공원까지 이동할 수 있다고 판단된 시점에서
                // 메두사의 이동경로를 medusaMove 리스트에 담기
                int x =mr*N + mc;
//              medusaMove.add(x); // 메두사 좌표도 리스트에 담기
                while(!(x==medusa[0]*N + medusa[1])) {
                    medusaMove.add(medusaRoute[x/N][x%N]);
                    x = medusaRoute[x/N][x%N];
                }
                Collections.reverse(medusaMove);
                medusaMove.remove(0);
                break;
            }

            for(int d = 0 ; d < 4 ; d++) {
                int dr = mr + R[d];
                int dc = mc + C[d];
                if(dr>=0 && dr<N && dc>=0 && dc<N && !visited[dr][dc] && map[dr][dc]==0) {
                    queue.offer(new int [] {dr,dc});
                    visited[dr][dc] = true;
                    // 메두사의 이동경로 표시
                    medusaRoute[dr][dc] = mr*N + mc;
                }
            }
        } // while문 끝
    } // checkBFS 끝


}
```