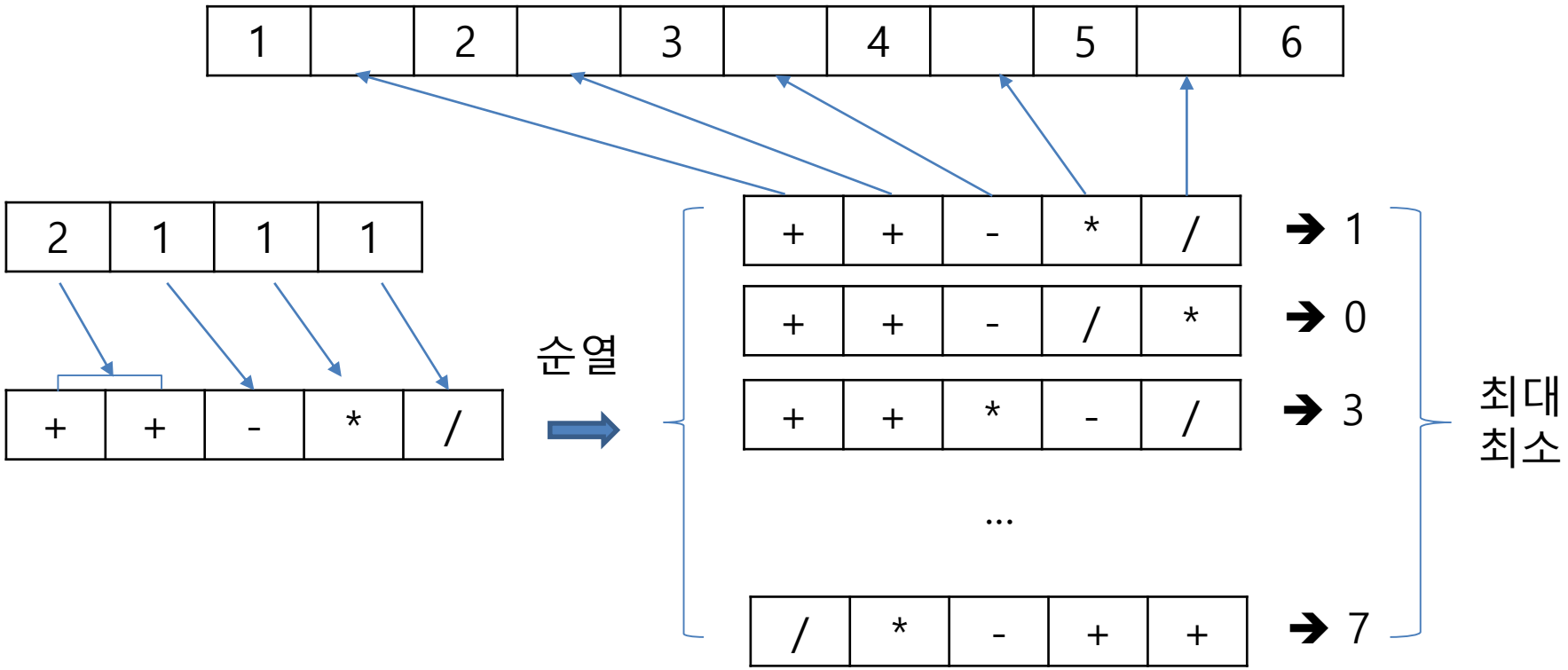
연산자끼워넣기

https://www.acmicpc.net/problem/14888

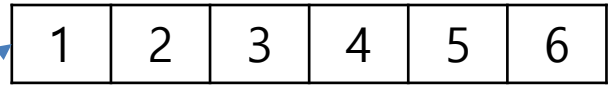
연산자끼워넣기

수의 개수 N(2 ≤ N ≤ 11)



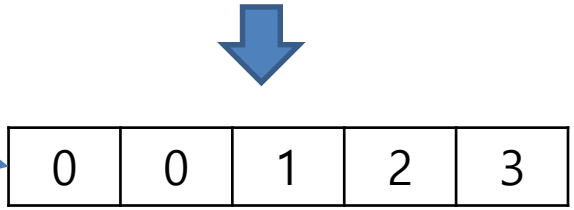
연산자끼워넣기

```
public static void main(String[] args)

        N = sc.nextInt();
        nums = new int [N];

        for (int i = 0; i < N; i++)
            nums[i] = sc.nextInt();

        for (int i = 0, cnt = 0; i < 4; i++)
        {
            int t;
            t = sc.nextInt();
            for (int j = 0; j < t; j++)
                ops[cnt++] = i;
        }

        maxans = Integer.MIN_VALUE;
        minans = Integer.MAX_VALUE;

        solve(0);

        System.out.println(maxans);
        System.out.println(minans);
```



(+, -, *, / 의 개수)

2 1 1 1



(+,+,-,*,/)

연산자끼워넣기

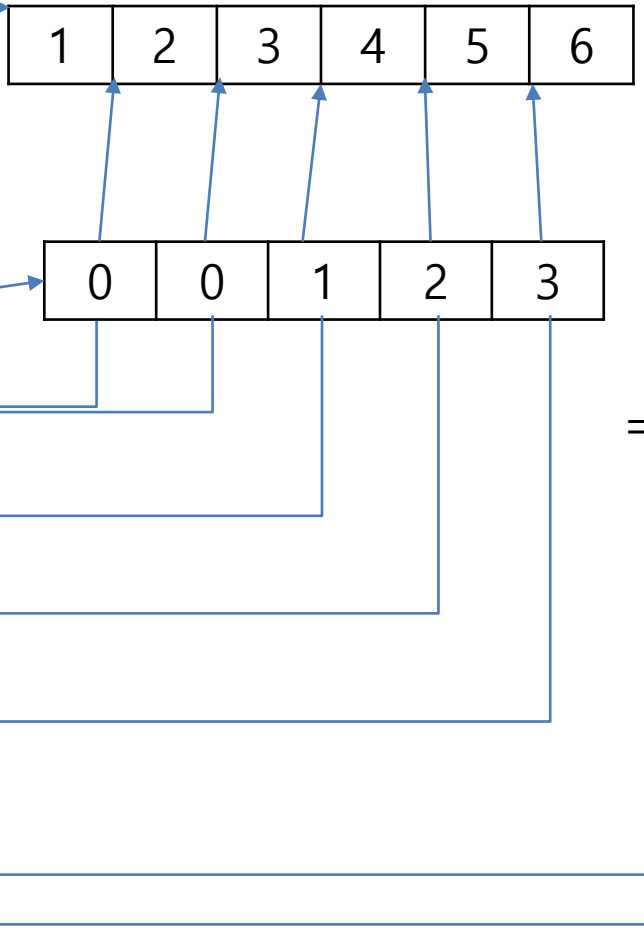
```
public static void solve(int k)

    if (k == N - 1)
    {
          …
    }
    else
        for (int i = k; i < N - 1; i++)
        {
            int t = ops[k]; ops[k] = ops[i]; ops[i] = t;
            solve(k + 1);
            t = ops[k]; ops[k] = ops[i]; ops[i] = t;
        }
```

순열

0	0	1	2	3
0	0	1	3	2
0	0	2	1	3
…				
3	2	1	0	0

연산자끼워넣기

```
public static void solve(int k)

    if (k == N - 1)
    {
        int val = nums[0];
        for (int i = 0; i < N - 1; i++)
        {
            if (ops[i] == 0)
                val += nums[i + 1];
            else if (ops[i] == 1)
                val -= nums[i + 1];
            else if (ops[i] == 2)
                val *= nums[i + 1];
            else
                val /= nums[i + 1];
        }
        maxans = maxans < val ? val : maxans;
        minans = minans > val ? val : minans;
    }
    else
        …
```



연산자끼워넣기

SWEA
게임 판에 적힌 숫자의 개수 N 은 3 이상 12 이하의 정수이다. (3 ≤ N ≤ 12)

```
public static void main(String[] args)

    N = sc.nextInt();
    nums = new int [N];

    for (int i = 0; i < N; i++)
    nums[i] = sc.nextInt();

    for (int i = 0; i < 4; i++)
        ops[i] = sc.nextInt();

    maxans = Integer.MIN_VALUE;
    minans = Integer.MAX_VALUE;

    solve(1, nums[0]);

    System.out.println(maxans);
    System.out.println(minans);
```

연산자끼워넣기

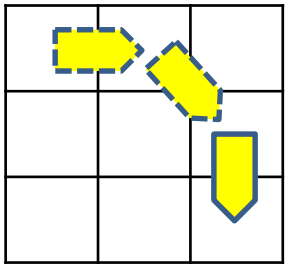
게임 판에 적힌 숫자의 개수 N 은 3 이상 12 이하의 정수이다. (3 ≤ N ≤ 12)

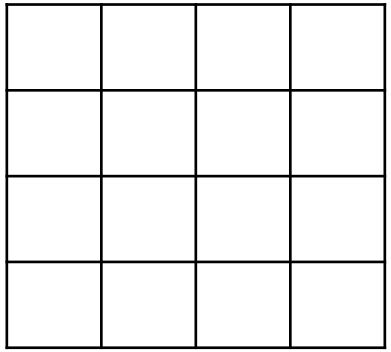
```
public static void solve(int k, int v ) {

    if (k == N)
    {
        maxans = Math.max(maxans, v);
        minans = Math.min(minans, v);
    }

    for (int i = 0; i < 4; i++) {
       if (ops[i] > 0) {
           ops[i]--;

           switch (i) {
               case 0 : solve(k + 1, v + nums[k]); break;
               case 1 : solve(k + 1, v - nums[k]); break;
               case 2 : solve(k + 1, v * nums[k]); break;
               case 3 : solve(k + 1, v / nums[k]); break;
           }
           ops[i]++;
      }
   }
}
```

파이프 옮기기1

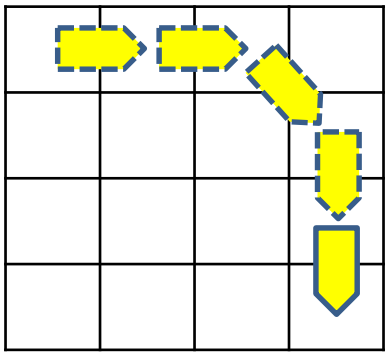
https://www.acmicpc.net/problem/17070

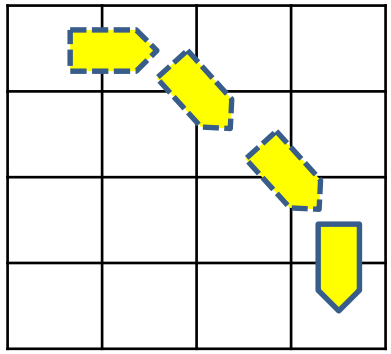
파이프 옮기기1

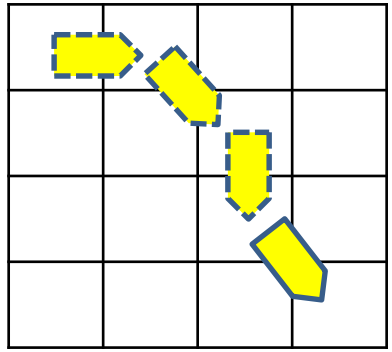




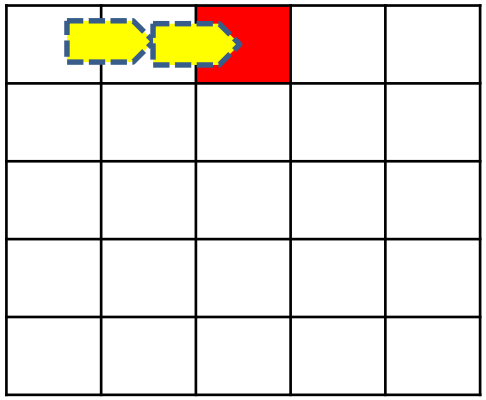
파이프 옮기기1

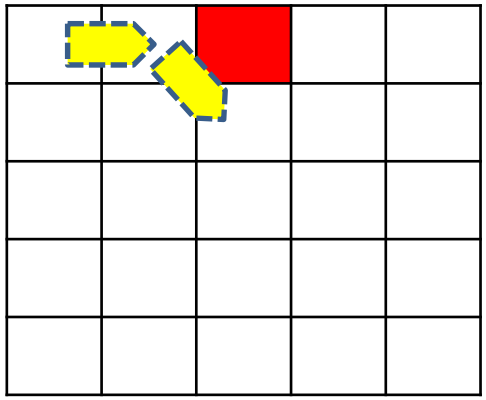




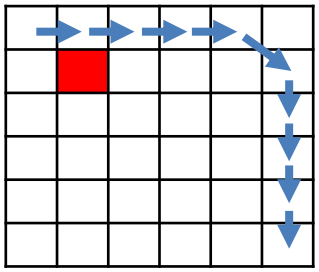
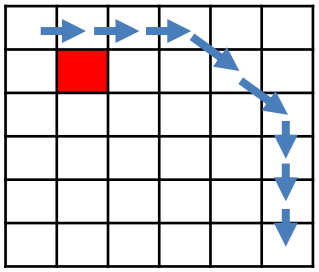
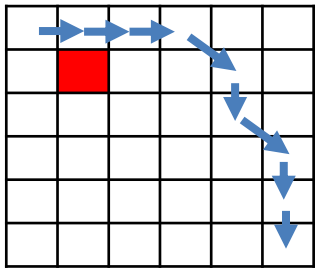
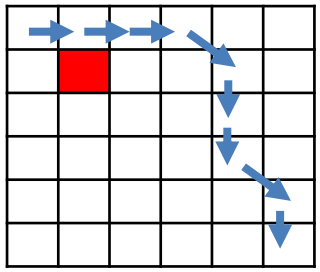
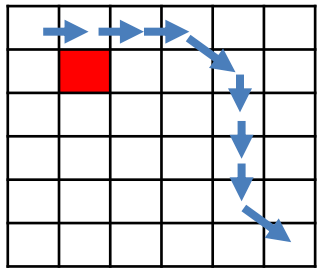
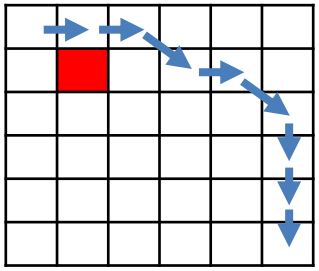
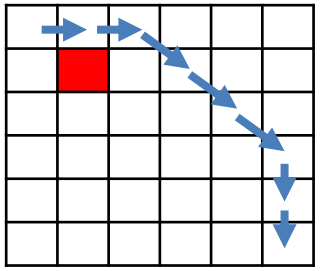
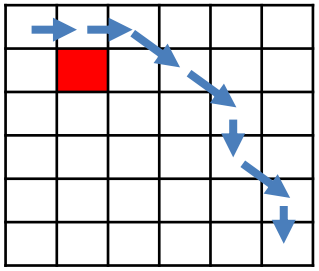
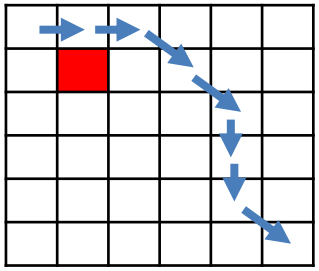
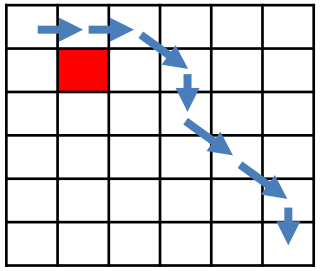
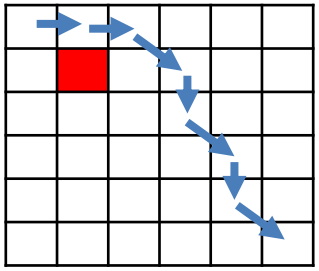
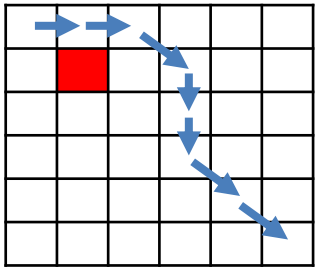
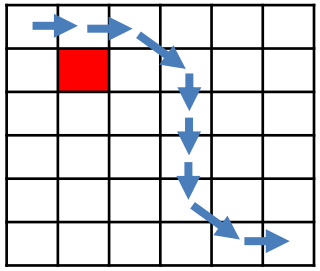


파이프 옮기기1



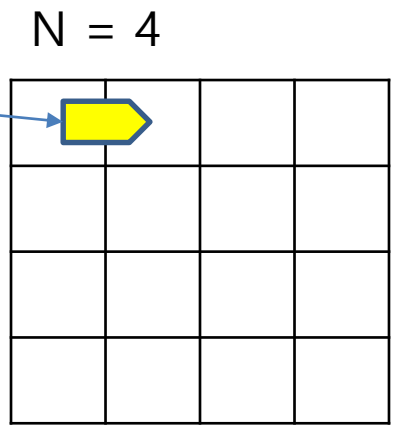


파이프 옮기기1

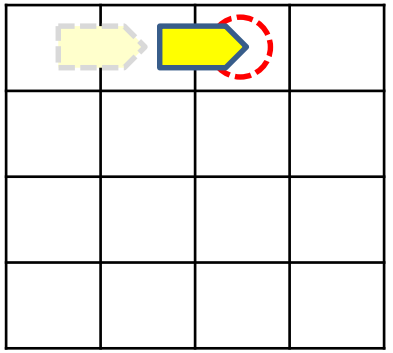
파이프 옮기기1

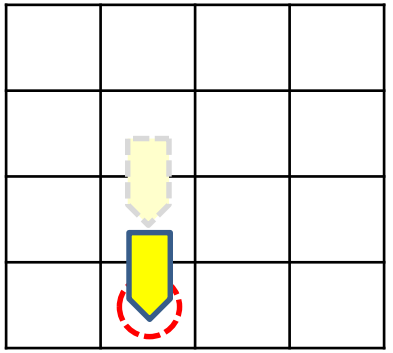
```
public static void main(String[] args)


    N = sc.nextInt();
    mat = new int [N][N];

    for(int i = 0; i < N; i++)
        for(int j = 0; j < N; j++)
            mat[i][j] = sc.nextInt();

    solve(0, 1, 0); //(x,y,d) d: →0, ↓1, ↘2
    System.out.println(ans);
```

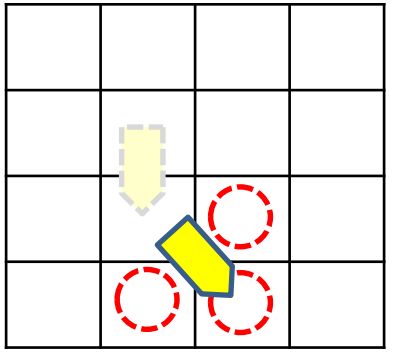


파이프 옮기기1

```
static boolean checkRight(int x, int y) {

    if (y + 1 < N && mat[x][y + 1] == 0)
        return true;
    return false;
}

static boolean checkDown(int x, int y) {

    if (x + 1 < N && mat[x + 1][y] == 0)
        return true;
    return false;
}

static boolean checkDigonal(int x, int y) {
    if (x + 1 < N && y + 1 < N &&
        mat[x + 1][y] == 0 &&
        mat[x][y + 1] == 0 &&
        mat[x + 1][y + 1] == 0)
        return true;
    return false;
}
```



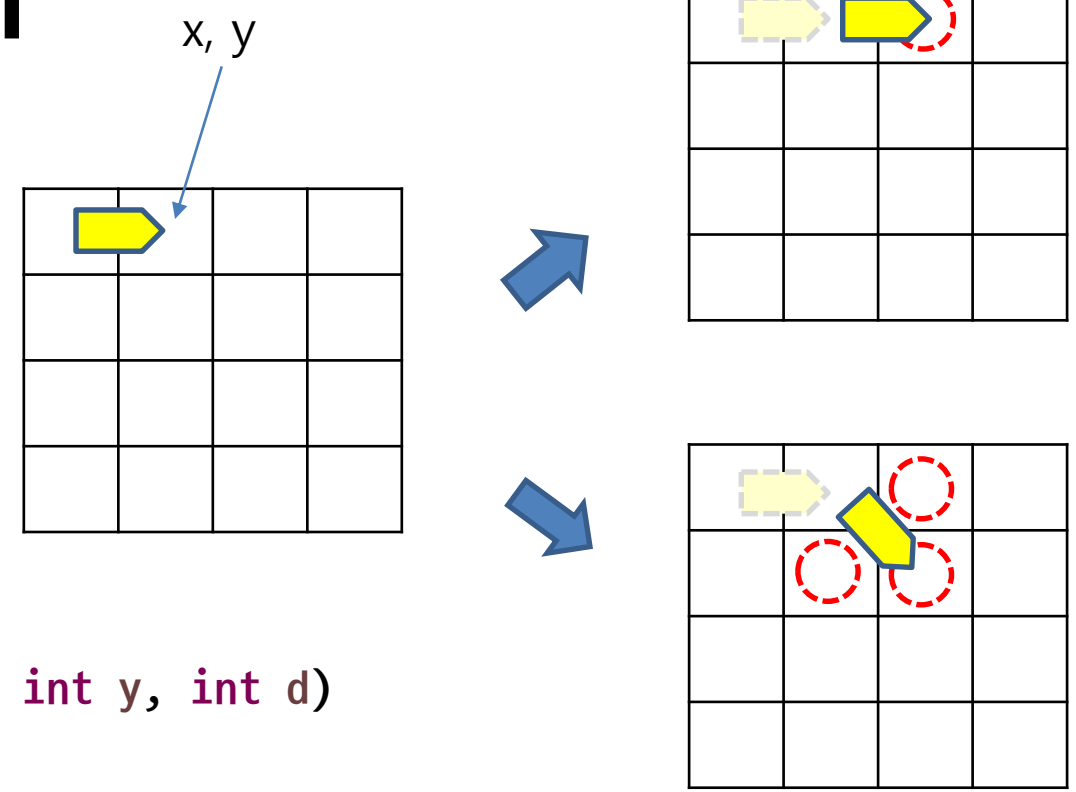




파이프 옮기기1

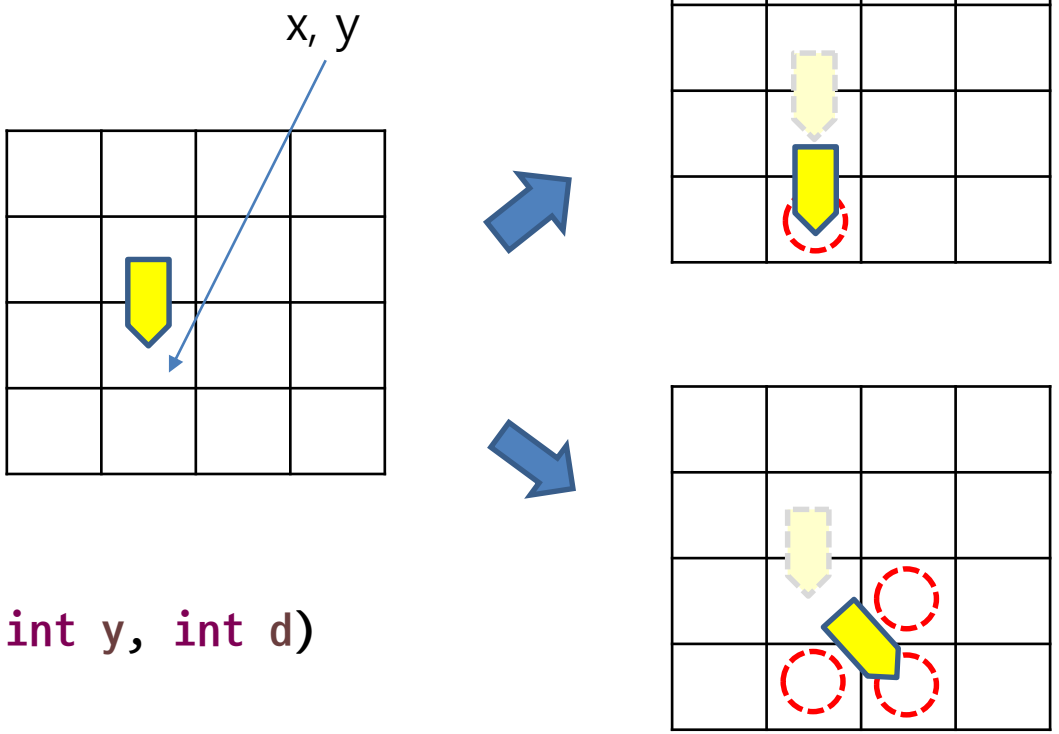
```
static void solve(int x, int y, int d) // d : → 0, ↓ 1,  ↘ 2
{
    if (x == N - 1 && y == N - 1) ans++;
    ...
```

파이프 옮기기1



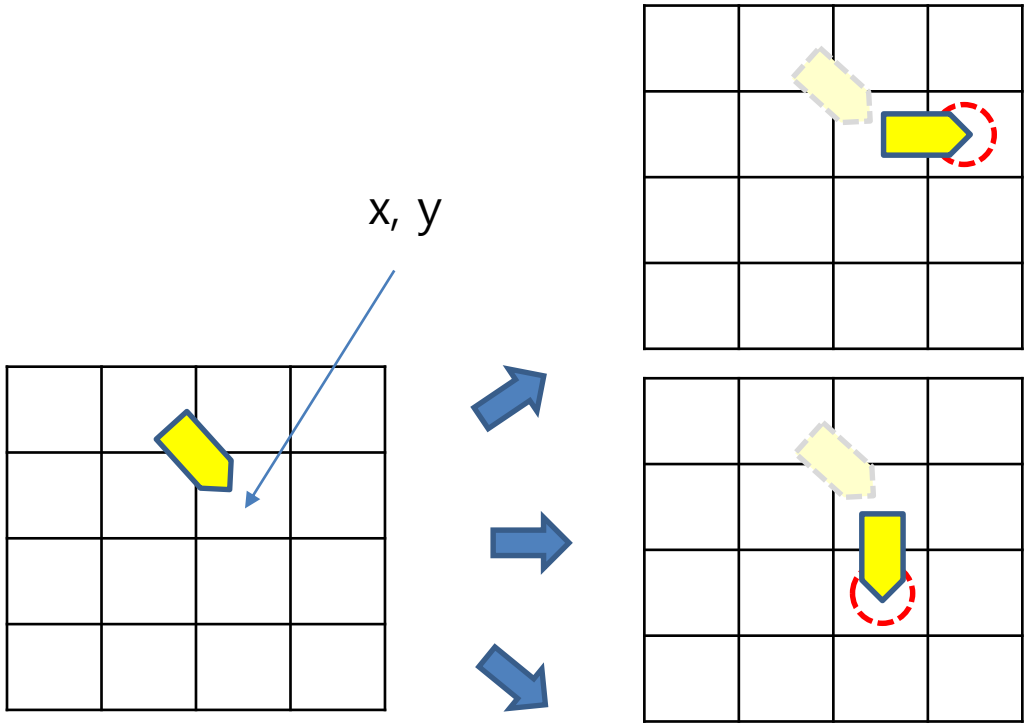

```
// d : → 0, ↓ 1,  ↘ 2
static void solve(int x, int y, int d)

    if (d == 0)
    {
        if (checkRight(x, y)) solve(x, y + 1, 0);

        if (checkDigonal(x, y)) solve(x + 1, y + 1, 2);
    }
```

파이프 옮기기1




```
// d : → 0, ↓ 1,  ↘ 2
static void solve(int x, int y, int d)

    if (d == 1)
    {
        if (checkDown(x, y)) solve(x + 1, y, 1);

        if (checkDigonal(x, y))solve(x + 1, y + 1, 2);
    }
```

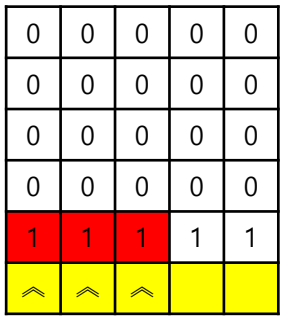
파이프 옮기기1

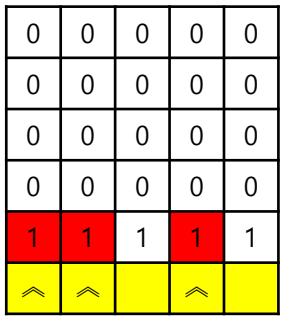


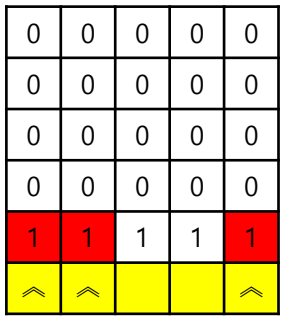

```
static void solve(int x, int y, int d)

    if (d == 2)
    {
        if (checkRight(x, y)) solve(x, y + 1, 0);

        if (checkDown(x, y)) solve(x + 1, y, 1);

        if (checkDigonal(x, y))solve(x + 1, y + 1, 2);
    }
```

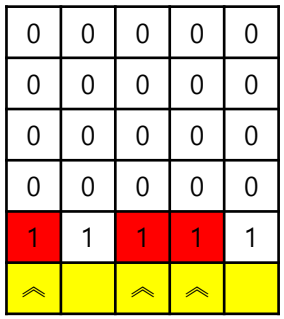
캐슬디펜스

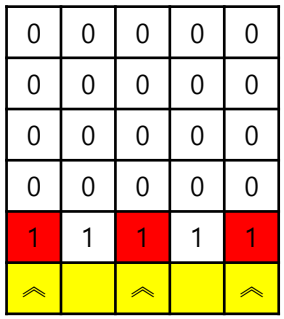
https://www.acmicpc.net/problem/17135

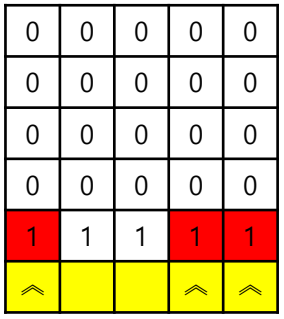


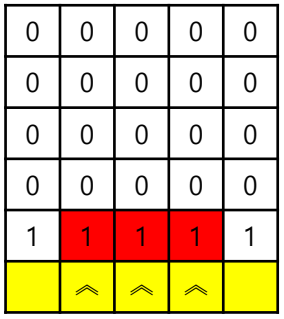


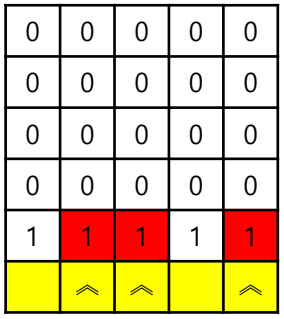


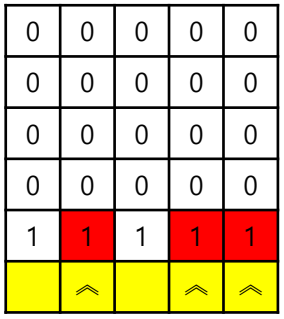


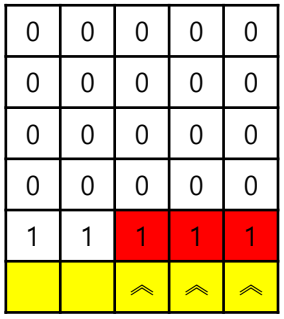












캐슬디펜스

1번예제

5, 5
길이 d = 1

5곳에서 3군데 고르는 경우의 수 ${}_5C_3$ = 10

0	0	0	0	0
0	0	0	0	0
0	0	0	0	0
0	0	0	0	0
1	1	1	1	1
︽	︽	︽		

3

0	0	0	0	0
0	0	0	0	0
0	0	0	0	0
0	0	0	0	0
1	1	1	1	1
︽	︽		︽	

3

0	0	0	0	0
0	0	0	0	0
0	0	0	0	0
0	0	0	0	0
1	1	1	1	1
︽	︽			︽

3

0	0	0	0	0
0	0	0	0	0
0	0	0	0	0
0	0	0	0	0
1	1	1	1	1
︽		︽	︽	

3

0	0	0	0	0
0	0	0	0	0
0	0	0	0	0
0	0	0	0	0
1	1	1	1	1
︽		︽		︽

3

0	0	0	0	0
0	0	0	0	0
0	0	0	0	0
0	0	0	0	0
1	1	1	1	1
︽			︽	︽

3

0	0	0	0	0
0	0	0	0	0
0	0	0	0	0
0	0	0	0	0
1	1	1	1	1
	︽	︽	︽	

3

0	0	0	0	0
0	0	0	0	0
0	0	0	0	0
0	0	0	0	0
1	1	1	1	1
	︽	︽		︽

3

0	0	0	0	0
0	0	0	0	0
0	0	0	0	0
0	0	0	0	0
1	1	1	1	1
	︽		︽	︽

3

0	0	0	0	0
0	0	0	0	0
0	0	0	0	0
0	0	0	0	0
1	1	1	1	1
		︽	︽	︽

3

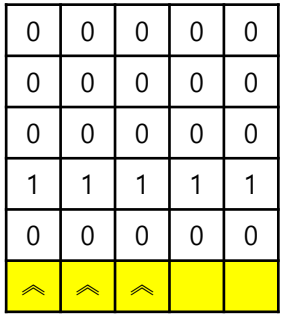
최대값 3

캐슬디펜스

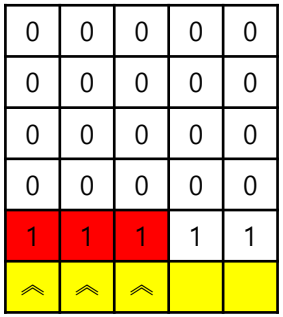
2번예제

5, 5
길이 d = 1

5곳에서 3군데 고르는 경우의 수 ${}_5C_3 = 10$



길이가 1 아래로 내려올때 까지 기다려야 한다.



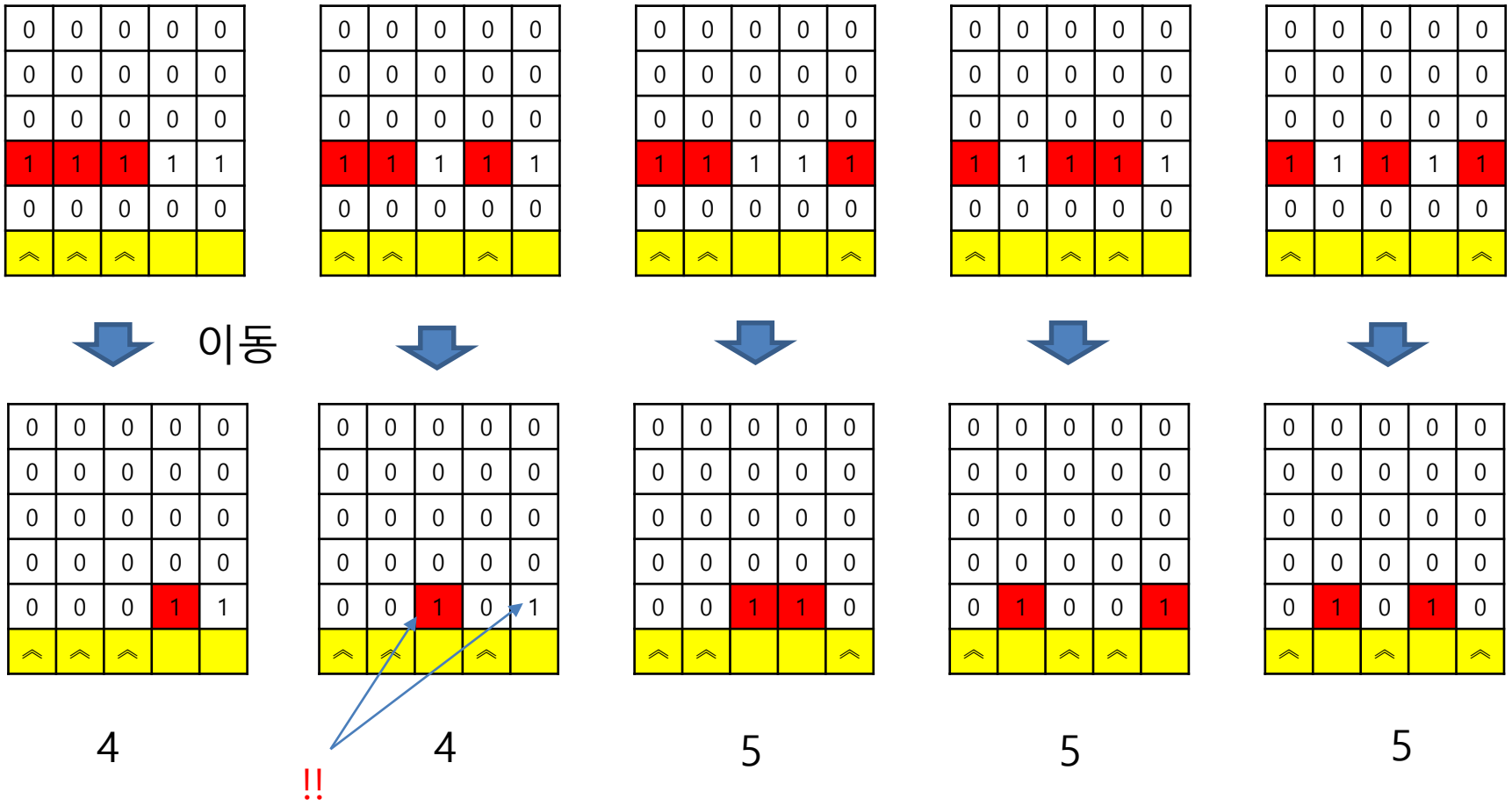
첫 번째 문제와 같은 상황

캐슬디펜스

3번예제

5, 5
길이 d = 2

5곳에서 3군데 고르는 경우의 수 $_5C_3 = 10$



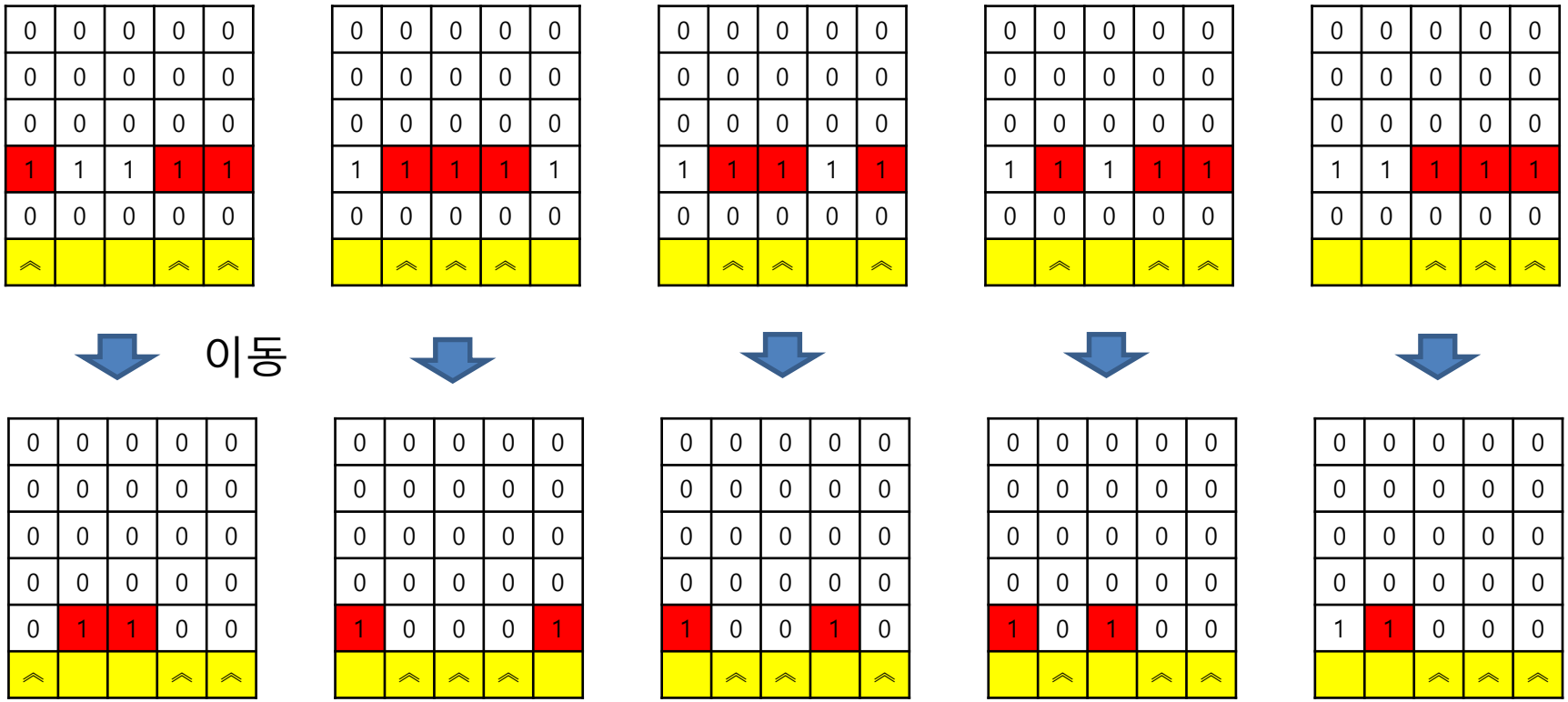
궁수가 공격하는 적은 거리가 D이하인 적 중에서 가장 가까운 적이고,
그러한 적이 여럿일 경우에는 가장 왼쪽에 있는 적을 공격한다.

캐슬디펜스

3번예제

5, 5
길이 d = 2

5곳에서 3군데 고르는 경우의 수 $_5C_3 = 10$



최대값 5

캐슬디펜스

5번예제

6, 5
길이 d = 1

5곳에서 3군데 고르는 경우의 수 ${}_5C_3$ = 10

1	0	1	0	1
0	1	0	1	0
1	1	0	0	0
0	0	0	1	1
1	1	0	1	1
0	0	1	0	0
︽	︽	︽	0	0

0	0	0	0	0
1	0	1	0	1
0	1	0	1	0
1	1	0	0	0
0	0	0	1	1
1	1	0	1	1
︽	︽	︽	0	0

0	0	0	0	0
0	0	0	0	0
1	0	1	0	1
0	1	0	1	0
1	1	0	0	0
0	0	0	1	1
︽	︽	︽	0	0

0	0	0	0	0
0	0	0	0	0
0	0	0	0	0
1	0	1	0	1
0	1	0	1	0
1	1	0	0	0
︽	︽	︽	0	0

0	0	0	0	0
0	0	0	0	0
0	0	0	0	0
0	0	0	0	0
1	0	1	0	1
0	1	0	1	0
︽	︽	︽	0	0

0	0	0	0	0
0	0	0	0	0
0	0	0	0	0
0	0	0	0	0
0	0	0	0	0
1	0	1	0	1
︽	︽	︽	0	0

0	0	0	0	0
0	0	0	0	0
0	0	0	0	0
0	0	0	0	0
0	0	0	0	0
0	0	0	0	0
︽	︽	︽	0	0

1	0	1	0	1
0	1	0	1	0
1	1	0	0	0
0	0	0	1	1
1	1	0	1	1
0	0	1	0	0
︽	︽	0	︽	0

0	0	0	0	0
1	0	1	0	1
0	1	0	1	0
1	1	0	0	0
0	0	0	1	1
1	1	0	1	1
︽	︽	0	︽	0

0	0	0	0	0
0	0	0	0	0
1	0	1	0	1
0	1	0	1	0
1	1	0	0	0
0	0	0	1	1
︽	︽	0	︽	0

0	0	0	0	0
0	0	0	0	0
0	0	0	0	0
1	0	1	0	1
0	1	0	1	0
1	1	0	0	0
︽	︽	0	︽	0

0	0	0	0	0
0	0	0	0	0
0	0	0	0	0
0	0	0	0	0
1	0	1	0	1
0	1	0	1	0
︽	︽	0	︽	0

0	0	0	0	0
0	0	0	0	0
0	0	0	0	0
0	0	0	0	0
0	0	0	0	0
1	0	1	0	1
︽	︽	0	︽	0

0	0	0	0	0
0	0	0	0	0
0	0	0	0	0
0	0	0	0	0
0	0	0	0	0
0	0	0	0	0
︽	︽	0	︽	0

...

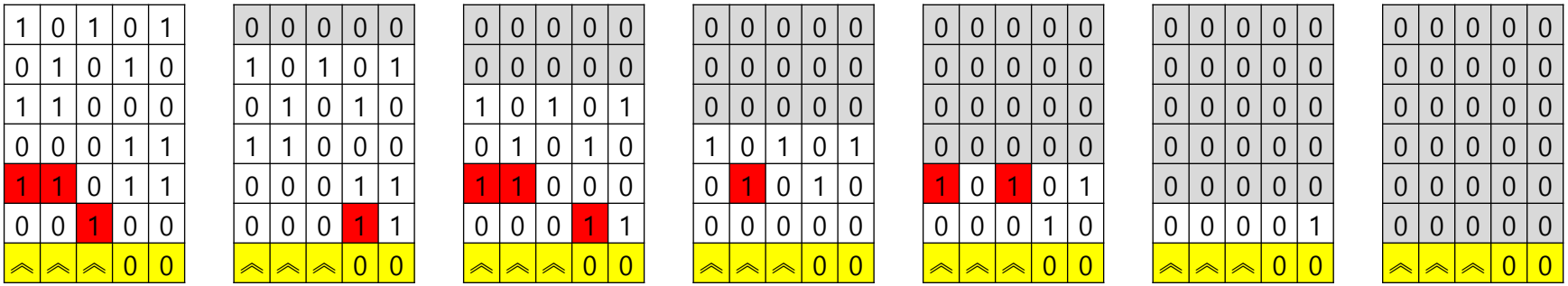
최개 9

캐슬디펜스

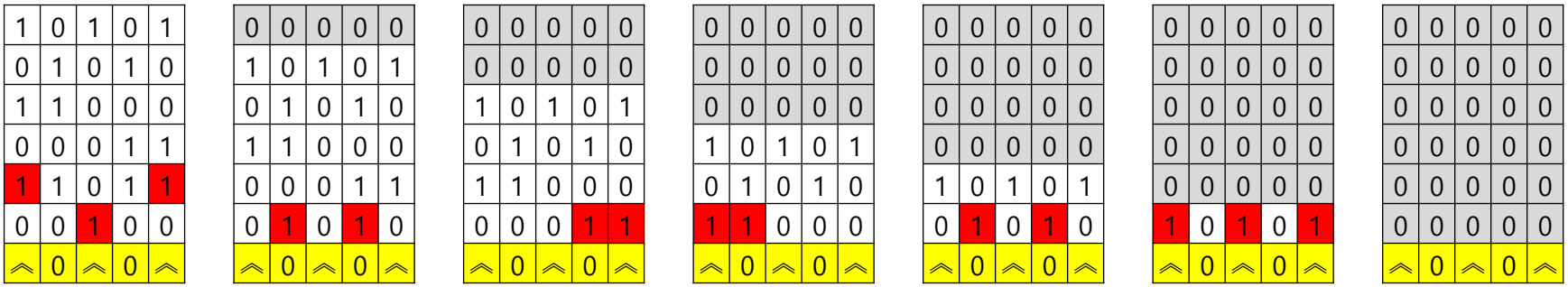
6번예제

6, 5
길이 d = 2

5곳에서 3군데 고르는 경우의 수 $_5C_3 = 10$



...



...

최개 14

캐슬디펜스

```
public static void main(String[] args)

        ans = 0;

        for(int i = 0; i < M - 2; i++) {
        for(int j = i + 1; j < M - 1; j ++) {
        for(int k = j + 1; k < M; k++) {
            killed = new int [N][M];
            tans = 0;
            archer[0] = i;
            archer[1] = j;
            archer[2] = k;
            solve(N);
            ans = Math.max(ans, getKilled());
        } } }

        System.out.println(ans);
```

$_{M}C_{3}$ 조합생성

죽은 적군 계산하고 매번 새로 생성

N 행 이동

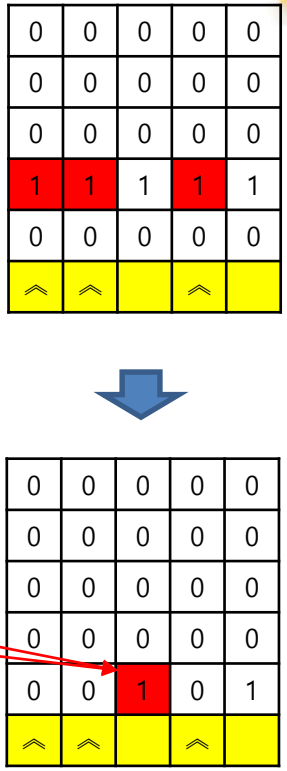
죽은 적군 계산, 최대값 갱신

```
static int getKilled()
   for(int i = 0, tsum = 0; i < N; i++)
       for(int j = 0; j < M; j++)
          if(killed[i][j] == 1)
             tsum++;
   return tsum;
```

캐슬디펜스

```
static void solve(int k) {

    if(k == 0) return;
    else {
        kill(k, archer[0]);
        kill(k, archer[1]);
        kill(k, archer[2]);

        while(f != r)
        {
            int x = Q[++f];
            int y = Q[++f];
            killed[x][y] = 1;
        }
        solve(k - 1);
    }
}
```

모든 행을 처리



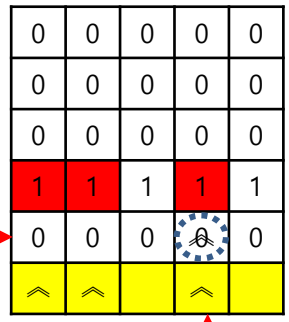
겹친다

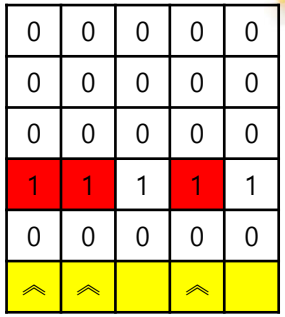
두 번째 궁수가 타겟을 제거하면 세번째 궁수는 다른 타겟을 제거하게 된다. 오답이 된다. 가능한 타켓 위치를 모두 모은 후 한꺼번에 처리한다.!!

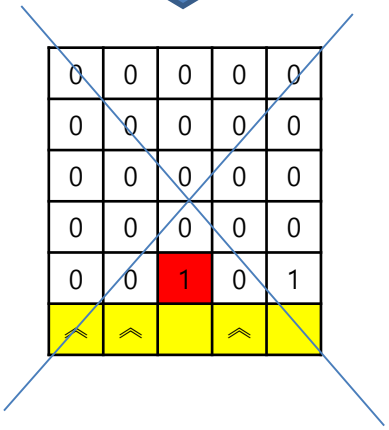
한 행씩 처리

캐슬디펜스

실제 적을 움직이는 것이 아니라 궁수가 이동한다고 가정








```
static void kill(int k, int y)
    int xx = -1, yy = -1, min_d = 100;

    for(int i = k - 1; i > -1; i--) {
    for(int j = 0; j < M; j++) {
        if(mat[i][j] == 1 && killed[i][j] == 0) {
        int td = Math.abs(i - k) + Math.abs(j - y);
        if(td < min_d) {
            xx = i; yy = j; min_d = td;
        }
        else if(td == min_d && j < yy) {
            xx = i; yy = j;
        }
    }}}

    if(min_d <= D) {
        Q[++r] = xx;    Q[++r] = yy;
    }
}
```

거리가 가장 가까운 적의 위치 찾기

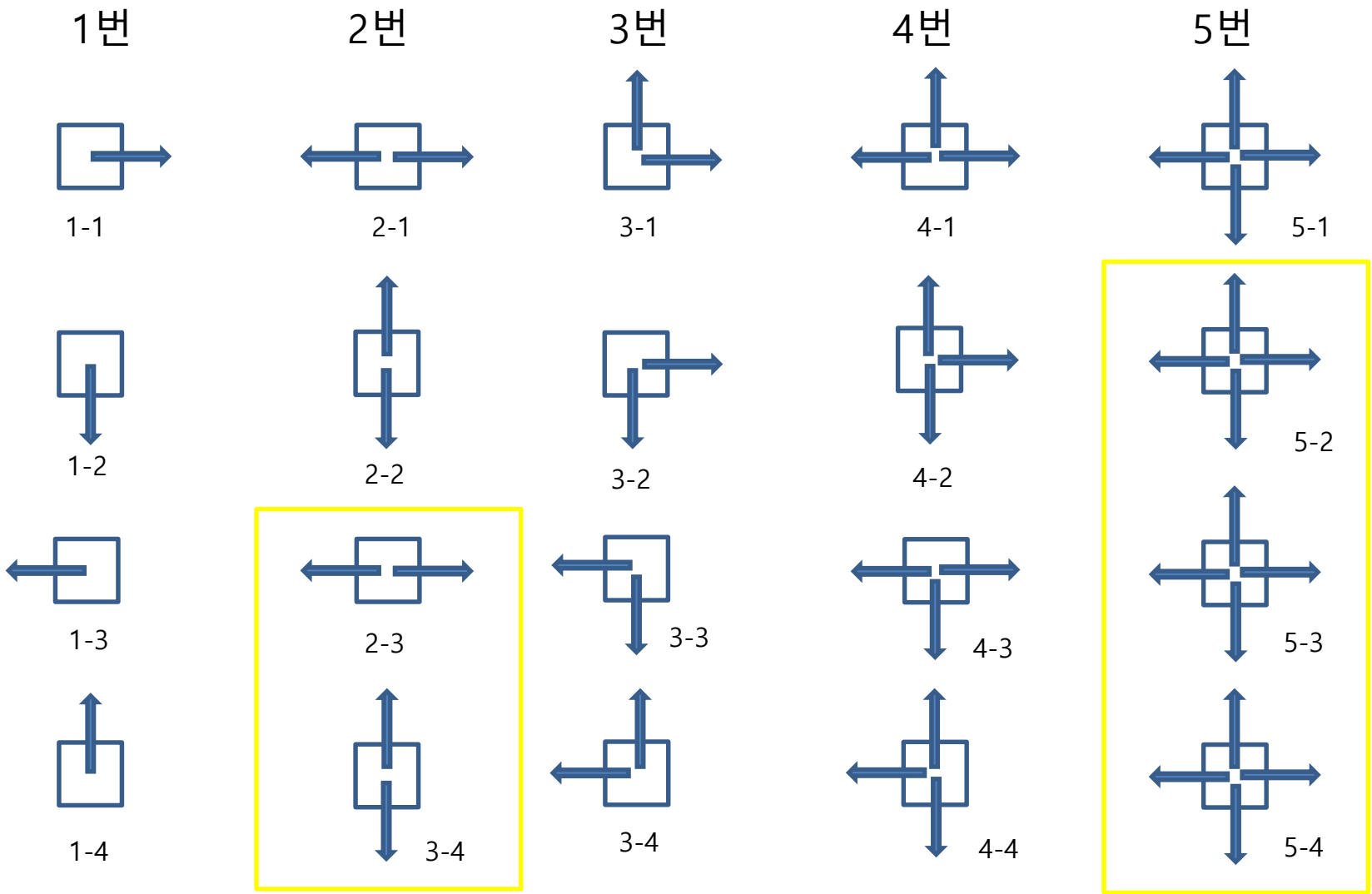
같은 거리면 왼쪽을 선택

가능한 적의 위치를 저장한다.

감시

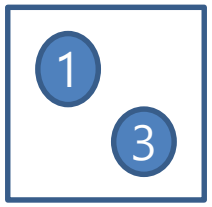
https://www.acmicpc.net/problem/15683

감시



각 종류 CCTV 90도 씩 회전한 모습

감시



사무실 안에 1,3 두 종류의
CCTV가 있다면

1-1 3-1	1-2 3-1	1-3 3-1	1-4 3-1
1-1 3-2	1-2 3-2	1-3 3-2	1-4 3-2
1-1 3-3	1-2 3-3	1-3 3-3	1-4 3-3
1-1 3-4	1-2 3-4	1-3 3-4	1-4 3-4

각 CCTV를 회전 했을 때

서로 다른 16가지 경우가
생긴다.

중복순열이다.

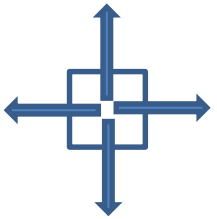
감시

```
public static void main(String[] args)

    mat = new int [N][M];
    visited = new boolean [N][M];

      …

    for(int x = 0; x < N; x++) {
        for(int y = 0; y < M; y++) {
            if ( mat[x][y] == 0 )
                continue;
            else if ( mat[x][y] == 6 )
                visited[x][y] = true;
            else if ( mat[x][y] == 5 ) {
                visited[x][y] = true;
                fill_right(x, y, visited);
                fill_left(x, y, visited);
                fill_up(x, y, visited);
                fill_down(x, y, visited);
            }
            else {
                cctvXYC[cctvCnt][0] = mat[x][y];
                cctvXYC[cctvCnt][1] = x;
                cctvXYC[cctvCnt++][2] = y;
    }}}
```

CCTV로 감시되는 칸의 정보를 저장

벽 감시영역으로 처리

5번



5번 CCTV는 회전의 의미가 없음. 미리 감시영역처리

1	0	0	0	0	0
0	1	0	0	0	0
0	0	1	5	0	0
0	0	5	1	0	0
0	0	0	0	1	0
0	0	0	0	0	1

CCTV의 종류와 위치를 저장할 배열

감시

```
public static void main(String[] args){

        …
        ans = 0;
        direction = new int [cctvCnt];
        solve(0);

        System.out.println(N*M - ans);
```

중복순열을
저장할 배열

사각지대 = 전체 영역 - 최대감시영역

감시

```
public static void fill_right(int x, int y,
boolean tvisited[][]) {
    int yy = y + 1;
    while (yy < M && mat[x][yy] != 6)
        tvisited[x][yy++] = true;
}
```

1	0	0	0	0	0
0	1	0	0	0	0
0	0	1	5	0	0
0	0	5	1	0	0
0	0	0	0	1	0
0	0	0	0	0	1

mat

0	0	0	0	0	0
0	0	0	0	0	0
0	0	0	1	1	1
0	0	0	0	0	0
0	0	0	0	0	0
0	0	0	0	0	0

arr

```
public static void fill_left(int x, int y,
boolean tvisited[][]) {
    int yy = y - 1;
    while (yy > -1 && mat[x][yy] != 6)
        tvisited[x][yy--] = true;
}
```

1	0	0	0	0	0
0	1	0	0	0	0
0	0	1	5	0	0
0	0	5	1	0	0
0	0	0	0	1	0
0	0	0	0	0	1

mat

0	0	0	0	0	0
0	0	0	0	0	0
1	1	1	1	1	1
0	0	0	0	0	0
0	0	0	0	0	0
0	0	0	0	0	0

arr

감시

```
public static void fill_up(int x, int y,
boolean tvisited[][]) {
      int xx = x + 1;
      while (xx < N && mat[xx][y] != 6)
            tvisited[xx++][y] = true;
}
```

1	0	0	0	0	0
0	1	0	0	0	0
0	0	1	5	0	0
0	0	5	1	0	0
0	0	0	0	1	0
0	0	0	0	0	1

mat

0	0	0	1	0	0
0	0	0	1	0	0
1	1	1	1	1	1
0	0	0	0	0	0
0	0	0	0	0	0
0	0	0	0	0	0

arr

```
public static void fill_down(int x, int y,
boolean tvisited[][]) {
      int xx = x - 1;
      while ( xx > -1 && mat[xx][y] != 6)
            tvisited[xx--][y] = true;
}
```

1	0	0	0	0	0
0	1	0	0	0	0
0	0	1	5	0	0
0	0	5	1	0	0
0	0	0	0	1	0
0	0	0	0	0	1

mat

0	0	0	1	0	0
0	0	0	1	0	0
1	1	1	1	1	1
0	0	0	1	0	0
0	0	0	1	0	0
0	0	0	1	0	0

arr

감시

```
public static void solve(int k) {
    if( k == cctvCnt)
        observe();
    else{
        if (cctvXYC[k][0] == 2) {
            for (int i = 0; i < 2; i++) {
                direction[k] = i;
                solve(k + 1);
            }
        }
        else{
            for (int i = 0; i < 4; i++) {
                direction[k] = i;
                solve(k + 1);
            }
}}}
```

5번을 제외하고 CCTV개수 만큼 중복순열을 생성했으면

2번 CCTV는 2번만 회전

2번

4방향에 대한 중복 순열 생성

4방향의 중복순열을 저장할 배열, 0,1,2,3
➔ 우,하,좌,상

```
public static void observe() {
        boolean [][] tvisited = new boolean [N][M];

        for (int i = 0; i < N; i++)
            for (int j = 0; j < M; j++)
                tvisited[i][j] = visited[i][j];

        for (int i = 0; i < cctvCnt; i++) {
            int cctvC = cctvXYC[i][0];
            int x = cctvXYC[i][1];
            int y = cctvXYC[i][2];

            int dir = direction[i];

            tvisited[x][y] = true;

            if (cctvC == 1) {  …  }
            else if(cctvC == 2) { … }
            else if(cctvC == 3) { … }
            else if(cctvC == 4) { … }
        }

        int tsum = 0;
        for (int i = 0; i < N; i++)
            for (int j = 0; j < M; j++)
                if(tvisited[i][j]) tsum++;

        ans = ans < tsum ? tsum : ans;
}
```

5번 CCTV를 미리 처리한 정보를 받아옴

mat

1	0	0	0	0	0
0	1	0	0	0	0
0	0	1	5	0	0
0	0	5	1	0	0
0	0	0	0	1	0
0	0	0	0	0	1

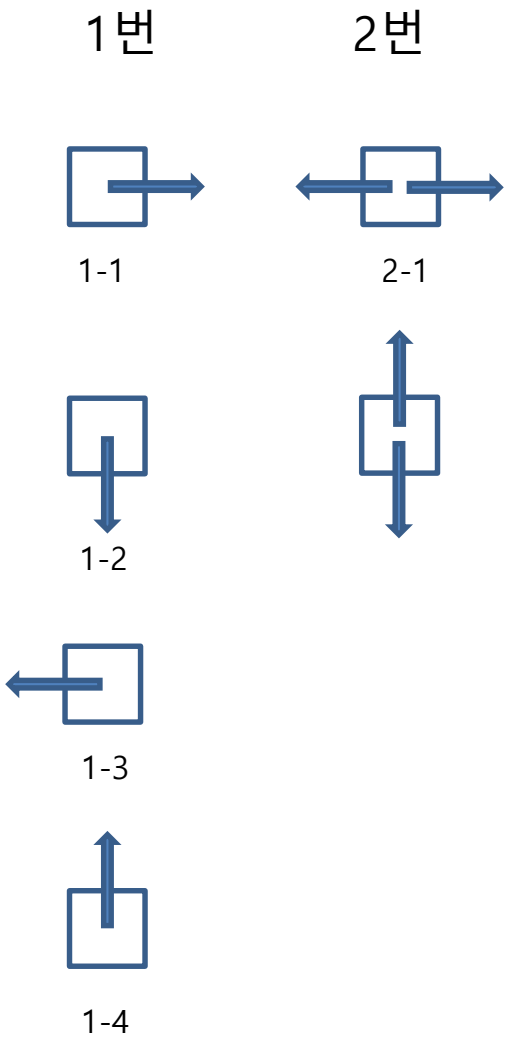
(1,0,0)	(1,1,1)	(1,2,2)	(1,3,3)	(1,4,4)	(1,5,5)

1	1	1	1	1	1

1	1	1	1	1	1
0	1	1	1	1	1
0	0	1	1	1	1
1	1	1	1	1	1
0	0	1	0	1	1
0	0	1	0	0	1

감시

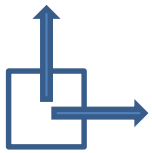
```
if (cctvC == 1) {
    if (dir == 0)   fill_right(x, y, tvisited);
    else if( dir == 1) fill_down(x, y, tvisited);
    else if( dir == 2) fill_left(x, y, tvisited);
    else if( dir == 3) fill_up(x, y, tvisited);
}
else if( cctvC == 2) {
    if (dir == 0) {
        fill_right(x, y, tvisited);
        fill_left(x, y, tvisited);
    }
    else if(dir == 1) {
        fill_up(x, y, tvisited);
        fill_down(x, y, tvisited);
    }
}
```



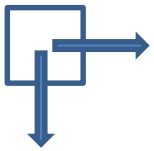
감시

```
else if(cctvC == 3) {
    if (dir == 0) {
        fill_up(x, y, tvisited);
        fill_right(x, y, tvisited);
    }
    else if(dir == 1) {
        fill_right(x, y, tvisited);
        fill_down(x, y, tvisited);
    }
    else if (dir == 2) {
        fill_down(x, y, tvisited);
        fill_left(x, y, tvisited);
    }
    else if (dir == 3) {
        fill_left(x, y, tvisited);
        fill_up(x, y, tvisited);
    }
}
```

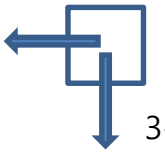
3번



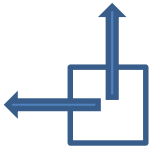
3-1



3-2



3-3

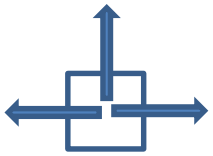


3-4

감시

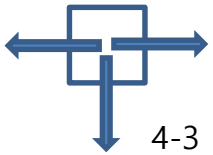
```
else if (cctvC == 4) {
    if (dir == 0) {
        fill_right(x, y, tvisited);
        fill_left(x, y, tvisited);
        fill_up(x, y, tvisited);
    }
    else if (dir == 1) {
        fill_right(x, y, tvisited);
        fill_down(x, y, tvisited);
        fill_up(x, y, tvisited);
    }
    else if (dir == 2) {
        fill_right(x, y, tvisited);
        fill_down(x, y, tvisited);
        fill_left(x, y, tvisited);
    }
    else if (dir == 3) {
        fill_down(x, y, tvisited);
        fill_left(x, y, tvisited);
        fill_up(x, y, tvisited);
    }
}
```

4번



4-1

4-2



4-3

게리맨더링

https://www.acmicpc.net/problem/17471

게리맨더링

구역이 [1,2,3,4], 4개 있다면 하나의 선거구가 구성할 수 있는 방법은?

[]
[1]
[2]
[3]
[4]
[1, 2]
[1, 3]
[1, 4]
[2, 3]
[2, 4]
[3, 4]
[1, 2, 3]
[1, 2, 4]
[1, 3, 4]
[2, 3, 4]
[1, 2, 3, 4]

게리맨더링

다른 선거구가 가지는 방법이 있어야 하므로 [], [1, 2, 3, 4]을 제외하면 14가지 경우가 생긴다.

[1]
[2]
[3]
[4]
[1, 2]
[1, 3]
[1, 4]
[2, 3]
[2, 4]
[3, 4]
[1, 2, 3]
[1, 2, 4]
[1, 3, 4]
[2, 3, 4]

1선거구

[2, 3, 4]
[1, 3, 4]
[1, 2, 4]
[1, 2, 3]
[3, 4]
[2, 4]
[2, 3]
[1, 4]
[1, 3]
[1, 2]
[4]
[3]
[2]
[1]

2선거구

게리맨더링

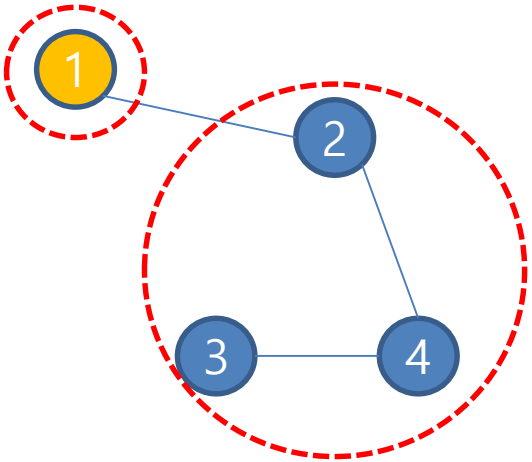
즉 부분 집합과 연관되어 있다. 1선거구를 부분 집합으로 구하고 전체 선거구에서 1선거구를 제거 하여 2선거구를 만들 수 있다.

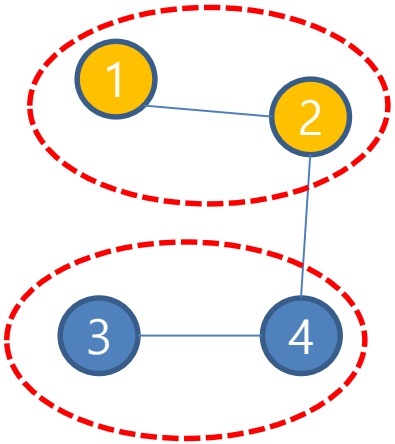
[1, 2, 3, 4]	-	[1]	=	[2, 3, 4]
		[2]		[1, 3, 4]
		[3]		[1, 2, 4]
		[4]		[1, 2, 3]
		[1, 2]		[3, 4]
		[1, 3]		[2, 4]
		[1, 4]		[2, 3]
		[2, 3]		[1, 4]
		[2, 4]		[1, 3]
		[3, 4]		[1, 2]
		[1, 2, 3]		[4]
		[1, 2, 4]		[3]
		[1, 3, 4]		[2]
		[2, 3, 4]		[1]
		1선거구		2선거구

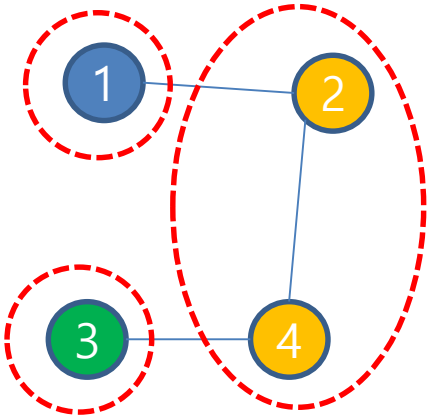
게리맨더링

나누어진 2개 선거구를 이용하여 선거구의 연결 상태를 확인한다.

1선거구		2선거구
[1]		[2, 3, 4]
[2]		[1, 3, 4]
[3]	…	[1, 2, 4]
[4]		[1, 2, 3]
[1, 2]		[3, 4]
[1, 3]		[2, 4]
[1, 4]	…	[2, 3]
[2, 3]		[1, 4]
[2, 4]		[1, 3]
[3, 4]		[1, 2]
[1, 2, 3]		[4]
[1, 2, 4]	…	[3]
[1, 3, 4]		[2]
[2, 3, 4]		[1]





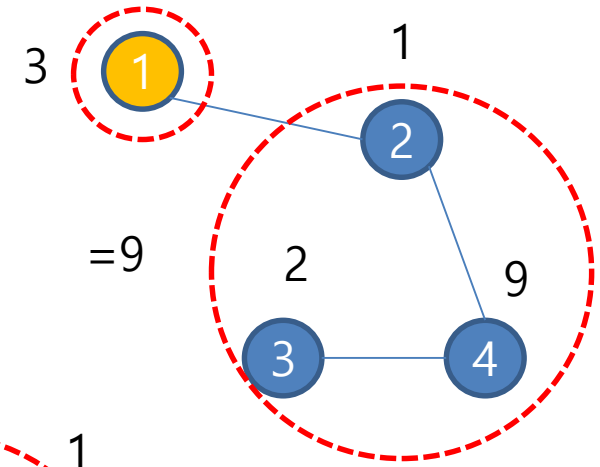


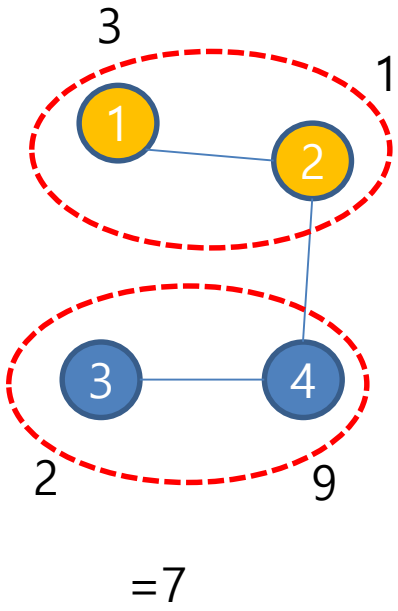
2개의 선거구로 분리되지 않는 경우도 있다.

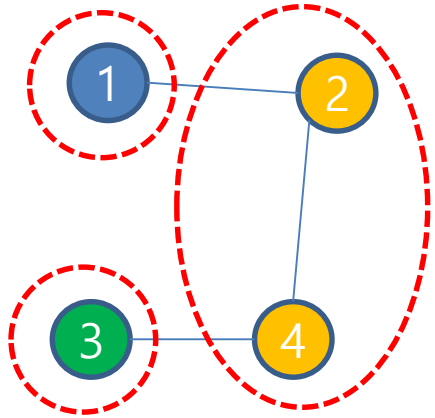
게리맨더링

두 개 선거구의 인구수를 구하고 그 차가 최소인 것을 찾는다.

1선거구		2선거구
[1]		[2, 3, 4]
[2]		[1, 3, 4]
[3]	…	[1, 2, 4]
[4]		[1, 2, 3]
[1, 2]		[3, 4]
[1, 3]		[2, 4]
[1, 4]	…	[2, 3]
[2, 3]		[1, 4]
[2, 4]		[1, 3]
[3, 4]		[1, 2]
[1, 2, 3]		[4]
[1, 2, 4]	…	[3]
[1, 3, 4]		[2]
[2, 3, 4]		[1]







2개의 선거구로 분리되지 않는 경우도 있다.

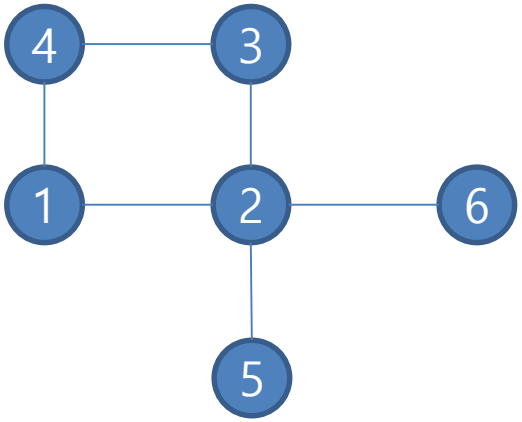
게리맨더링

5	2	3	4	1	2

```
6
5 2 3 4 1 2
2 2 4
4 1 3 6 5
2 4 2
2 1 3
1 2
1 2
```

구역인구수

```
public static void main(String[] args){

    N = sc.nextInt();
    people = new int [N];
    subset = new int [N];

    for(int i = 0; i < N; i++)
        people[i] = sc.nextInt();

    G = new int [N][10];
    for (int i = 0; i < N; i++){
        G[i][0] = sc.nextInt();
        for (int j = 1; j <= G[i][0]; j++)
            G[i][j] = sc.nextInt();
    }

    solve(0);

    if (ans == Integer.MAX_VALUE)System.out.println(-1);
    elseSystem.out.println(ans);
```



두 구역으로 나눌 수 있는 방법이 없으면

게리맨더링

subset

0	0	0	0	0	0

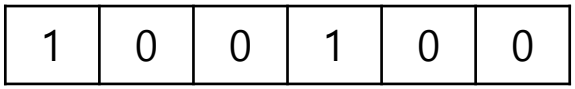
1	1	1	1	1	1

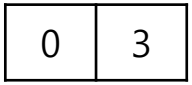
```
static void solve(int k)
{
    if (k == N)
    {
        int tsum = 0;
        for (int i = 0; i < N; i++)
            tsum += subset[i];
        if (tsum == 0 || tsum == N) return;

        …
    else
    {
        subset[k] = 1; solve(k + 1);
        subset[k] = 0; solve(k + 1);
    }
}
```

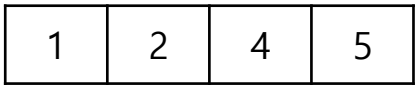
구역 개수의
모든 부분집합
을 생성

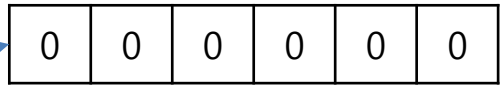
게리맨더링

```
static void solve(int k) {
    if (k == N) {
        …

        int [] area1 = new int [10]; int t1 = 0;
        int [] area2 = new int [10]; int t2 = 0;

        for (int i = 0; i < N; i++){
            if (subset[i] == 1)area1[t1++] = i;
            else area2[t2++] = i;
        }

        int [] visited = new int [10];
        int v1 = dfs(area1[0], area1, t1, visited);
        int v2 = dfs(area2[0], area2, t2, visited);

        tsum = 0;
        for (int i = 0; i < N; i++)
            tsum += visited[i];
        if (tsum == N)
ans = ans > Math.abs(v1 - v2) ? Math.abs(v1 - v2) : ans;
    }
    else
        …
```









visited를 매 경우 마다 새로 만든다.

두 지역구를 조사 한 후 모든 구역이 선택 됐으면…
두 구역의 인구수의 최소 차를 구한다.

게리맨더링

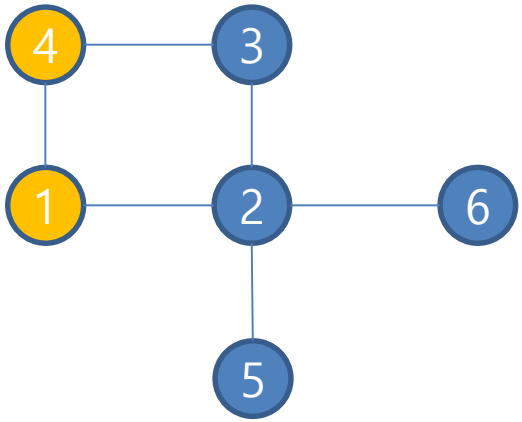
```
static int dfs(int v, int area[], int areacnt, int visited[]){
    int ret = people[v];
    visited[v] = 1;

    for (int u = 1; u <= G[v][0]; u++)  {
        if (visited[G[v][u] - 1] == 1) continue;

        boolean found = false;
        for (int j = 0; j < areacnt; j++) {
            if (area[j] == (G[v][u] - 1)) {
                found = true;
                break;
            }
        }

         if (found)
              ret += dfs(G[v][u] - 1, area, areacnt, visited);
    }

    return ret;
}
```

0	3

0	0	0	0	0	0

1	0	0	1	0	0

5	2	3	4	1	2



최종 9 반환