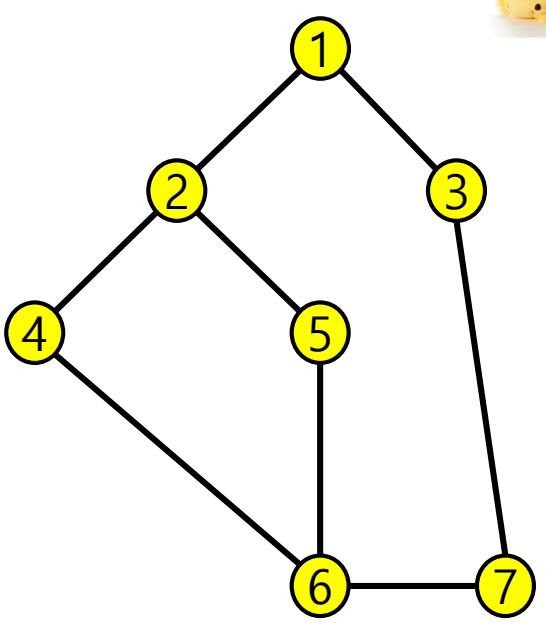
비선형자료구조의 완전탐색 : DFS, BFS

AD 보충수업 1일차

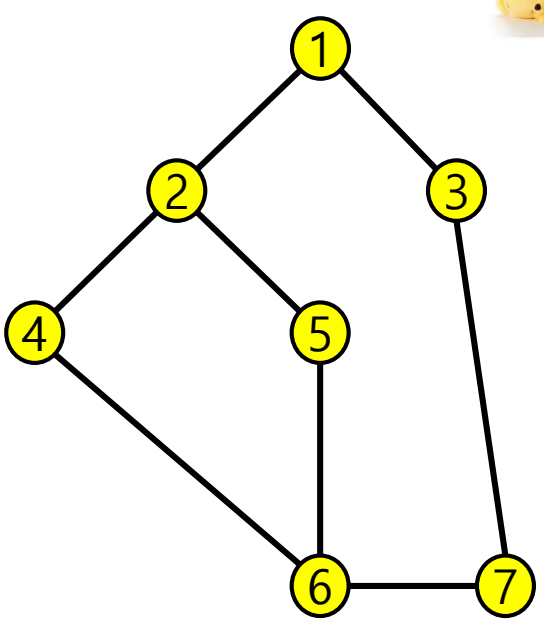
DFS




```
G = [[], [2, 3], [1, 4, 5], [1, 7], [2, 6], [2, 6], [4, 5, 7], [3, 6]]
visited = [0] * 8

dfs(1)
```

DFS

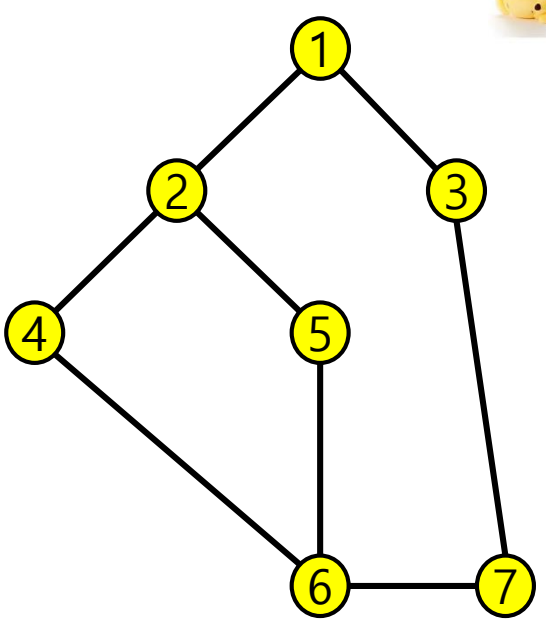
```
def dfs(v):
    s = []
    s.append(v)
    while s:
        v = s.pop(-1)
        if not visited[v]:
            visited[v] = 1
            print(v, end=' ')
            for w in G[v]:
                if not visited[w]:
                    s.append(w)
```




```
G = [[], [2, 3], [1, 4, 5], [1, 7], [2, 6], [2, 6], [4, 5, 7], [3, 6]]
visited = [0] * 8

dfs(1)
```

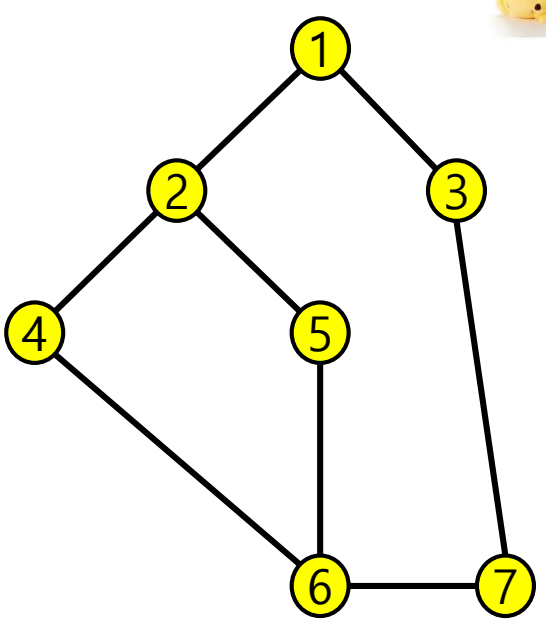
DFS

```
def dfsr(v):
    visited[v] = 1
    print(v, end =' ')
    for w in G[v]:
        if not visited[w]:
            dfsr(w)
```




```
G = [[], [2, 3], [1, 4, 5], [1, 7], [2, 6], [2, 6], [4, 5, 7], [3, 6]]
visited = [0] * 8

dfs(1)
```

BFS




```
def bfs(v):
    q = []
    q.append(v)
    while q:
        v = q.pop(0)
        if not visited[v]:
            visited[v] = 1
            print(v, end=' ')
            for w in G[v]:
                if not visited[w]:
                    q.append(w)
```

```
G = [[], [2, 3], [1, 4, 5], [1, 7], [2, 6], [2, 6], [4, 5, 7], [3, 6]]
visited = [0] * 8

dfs(1)
```

BFS 와 DFS

```python
def bfs(v):
    q = []
    q.append(v)
    while q:
        v = q.pop(0)
        if not visited[v]:
            visited[v] = 1
            print(v, end=' ')
            for w in G[v]:
                if not visited[w]:
                    q.append(w)
```

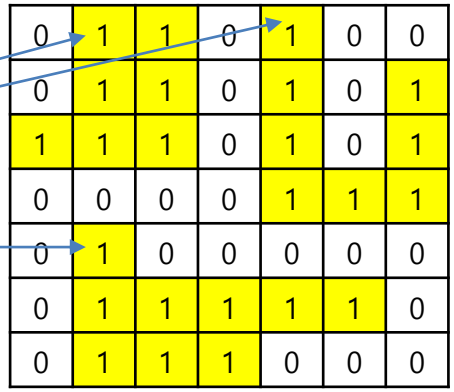
```python
def dfs(v):
    s = []
    s.append(v)
    while s:
        v = s.pop(-1)
        if not visited[v]:
            visited[v] = 1
            print(v, end=' ')
            for w in G[v]:
                if not visited[w]:
                    s.append(w)
```

단지번호붙이기

https://www.acmicpc.net/problem/2667

단지번호붙이기

```
N = int(input())
mat = [list(map(int, input())) for _ in range(N)]

res = []

for i in range(N):
    for j in range(N):
        if mat[i][j]:
            res.append(dfs(i, j))

res.sort()
print(len(res))
for i in res:
    print(i)
```



7	8	9

7	8	9

단지번호붙이기

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

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

0,1에서 시작

```
def dfs(x, y):
    mat[x][y] = 0

    ret = 0
    for (dx, dy) in [(1, 0), (-1, 0), (0, 1), (0, -1)]:
        xx, yy = x + dx, y + dy
        if not (0 <= xx < N and 0 <= yy < N): continue
        if mat[xx][yy]:
            ret += dfs(xx, yy)

    return ret + 1
```

0	0	0
1	1	0
1	0	0

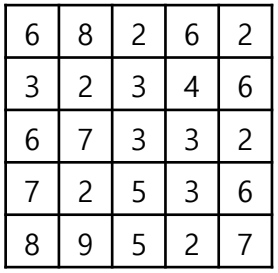
이웃한 노드의 개수와 자기 노드의 합 3을 반환

안전 영역

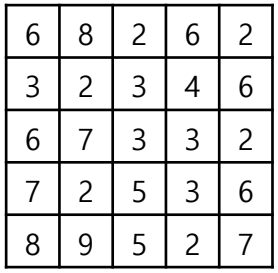
https://www.acmicpc.net/problem/2468

안전 영역

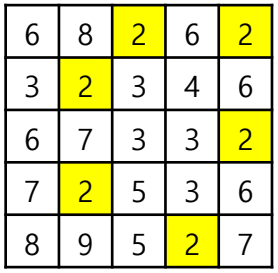
https://www.acmicpc.net/problem/2468



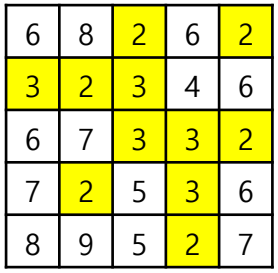
K=0



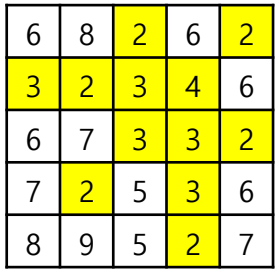
K=1



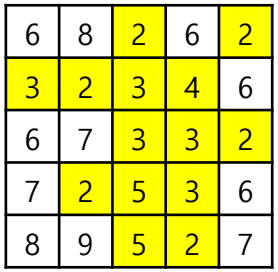
K=2



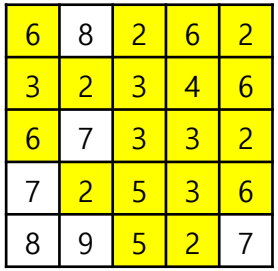
K=3



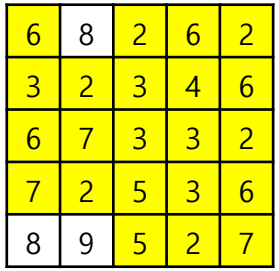
K=4



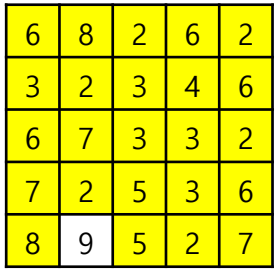
K=5



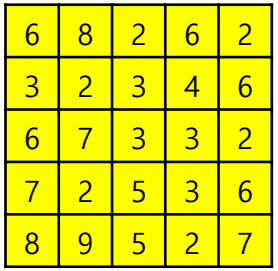
K=6



K=7



K=8



K=9

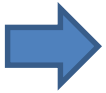
안전 영역

```
N = int(input())
mat = [list(map(int, input().split())) for _ in range(N)]

ans = 1
for k in range(1, max(sum(mat, []))):
    visited = [[0] * N for _ in range(N)]
    cnt = 0
    for i in range(N):
        for j in range(N):
            if not visited[i][j] and mat[i][j] > k:
                BFS(i, j, k)
                cnt += 1
    ans = max(ans, cnt)

print(ans)
```

6	8	2	6	2
3	2	3	4	6
6	7	3	3	2
7	2	5	3	6
8	9	5	2	7

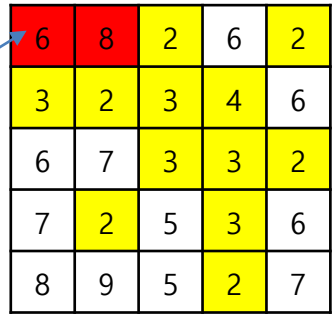
K=4

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



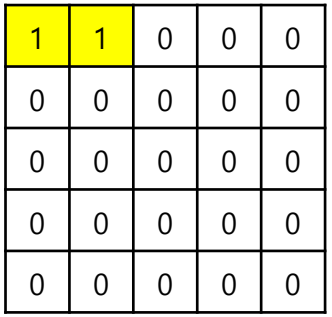
1	1	0	1	0
0	0	0	0	1
1	1	0	0	0
1	0	1	0	1
1	1	1	0	1

안전 영역



6	8	2	6	2
3	2	3	4	6
6	7	3	3	2
7	2	5	3	6
8	9	5	2	7

K=4



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

```
def BFS(x, y, k):
    q = []
    q.append((x, y))
    visited[x][y] = 1

    while q:
        x, y = q.pop(0)
        for dx, dy in (-1, 0), (1, 0), (0, -1), (0, 1):
            xx = x + dx
            yy = y + dy
            if not (0 <= xx < N and 0 <= yy < N): continue
            if not visited[xx][yy] and mat[xx][yy] > k:
                q.append((xx, yy))
                visited[xx][yy] = 1
```

치즈

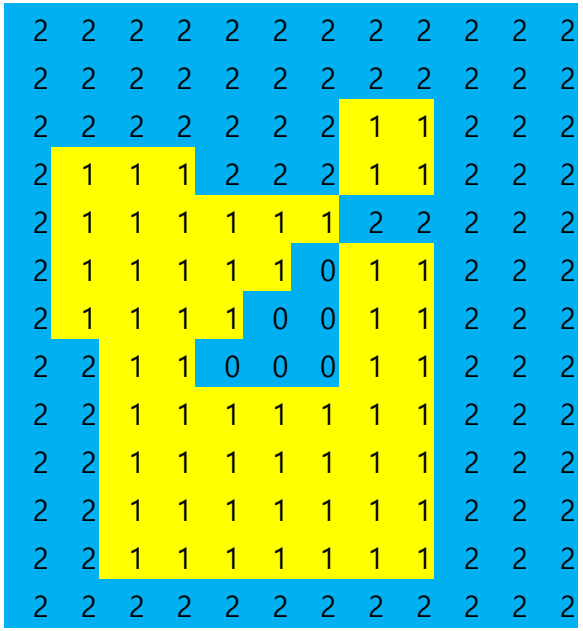
https://www.acmicpc.net/problem/2636

치즈

```
# step1 : 치즈 외부 공기 2로 표시하기
# step2 : 치즈의 공기와 접촉된 면을 표시하기 3, 치즈 내부 4
# step3 : 치즈 녹이기, 자료 재 정리, 치즈 개수 반환

N, M = map(int, input().split())
mat = [list(map(int, input().split())) for _ in range(N)]

cnt = 0
while True:
    cnt += 1
    step1(0, 0)
    for i in range(N):
        for j in range(M):
            if mat[i][j] == 1:
                step2(i, j)
    last_cheeze = step3()
    if not sum(sum(mat, [])) : break


print(cnt)
print(last_cheeze)
```

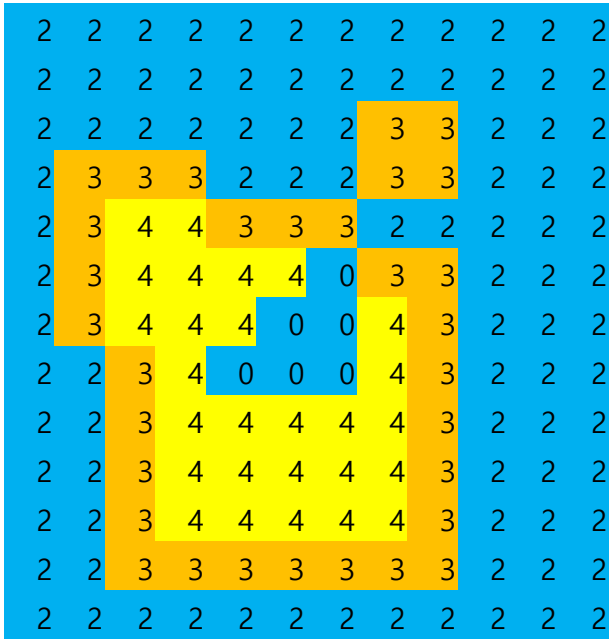
0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	1	1	0	0	0
0	1	1	1	0	0	0	1	1	0	0	0
0	1	1	1	1	1	1	0	0	0	0	0
0	1	1	1	1	1	0	1	1	0	0	0
0	1	1	1	1	0	0	1	1	0	0	0
0	0	1	1	0	0	0	1	1	0	0	0
0	0	1	1	1	1	1	1	1	0	0	0
0	0	1	1	1	1	1	1	1	0	0	0
0	0	1	1	1	1	1	1	1	0	0	0
0	0	1	1	1	1	1	1	1	0	0	0
0	0	0	0	0	0	0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

치즈



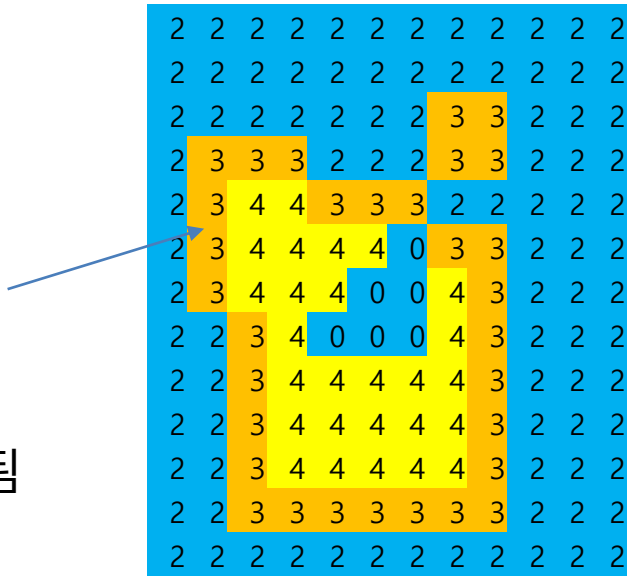

```
# 치즈 외부 공기 2로 표시하기
def step1(x, y):
    q = []
    q.append((x, y))
    mat[x][y] = 2

    while q:
        x, y = q.pop(0)

        for (dx, dy) in [(1, 0), (-1, 0), (0, 1), (0, -1)]:
            xx, yy = x + dx, y + dy

            if not (0 <= xx < N and 0 <= yy < M): continue
            if mat[xx][yy] == 0 :
                mat[xx][yy] = 2
                q.append((xx, yy))
```

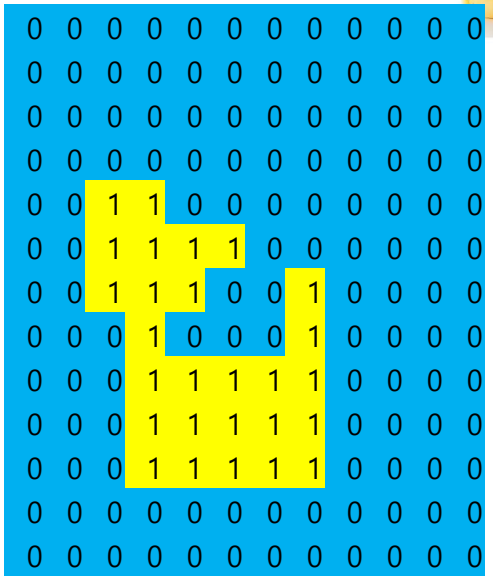
치즈




```
# 치즈의 공기와 접촉된 면을 표시하기 3, 치즈 내부 4
def step2(x, y):
    q = []

    mat[x][y] = 3

    q.append((x, y))
    while q:
        x, y = q.pop(0)

        for (dx, dy) in [(1, 0), (-1, 0), (0, 1), (0, -1)]:
            xx, yy = x + dx, y + dy

            if mat[xx][yy] == 1 :
                mat[xx][yy] = 4
                if mat[xx + 1][yy] == 2 or mat[xx - 1][yy] == 2 \
                        or mat[xx][yy + 1] == 2 or mat[xx][yy - 1] == 2:
                    mat[xx][yy] = 3
                q.append((xx, yy))
```

치즈





녹인 치즈의 개수
cnt가 반환
last_cheeze값이 됨

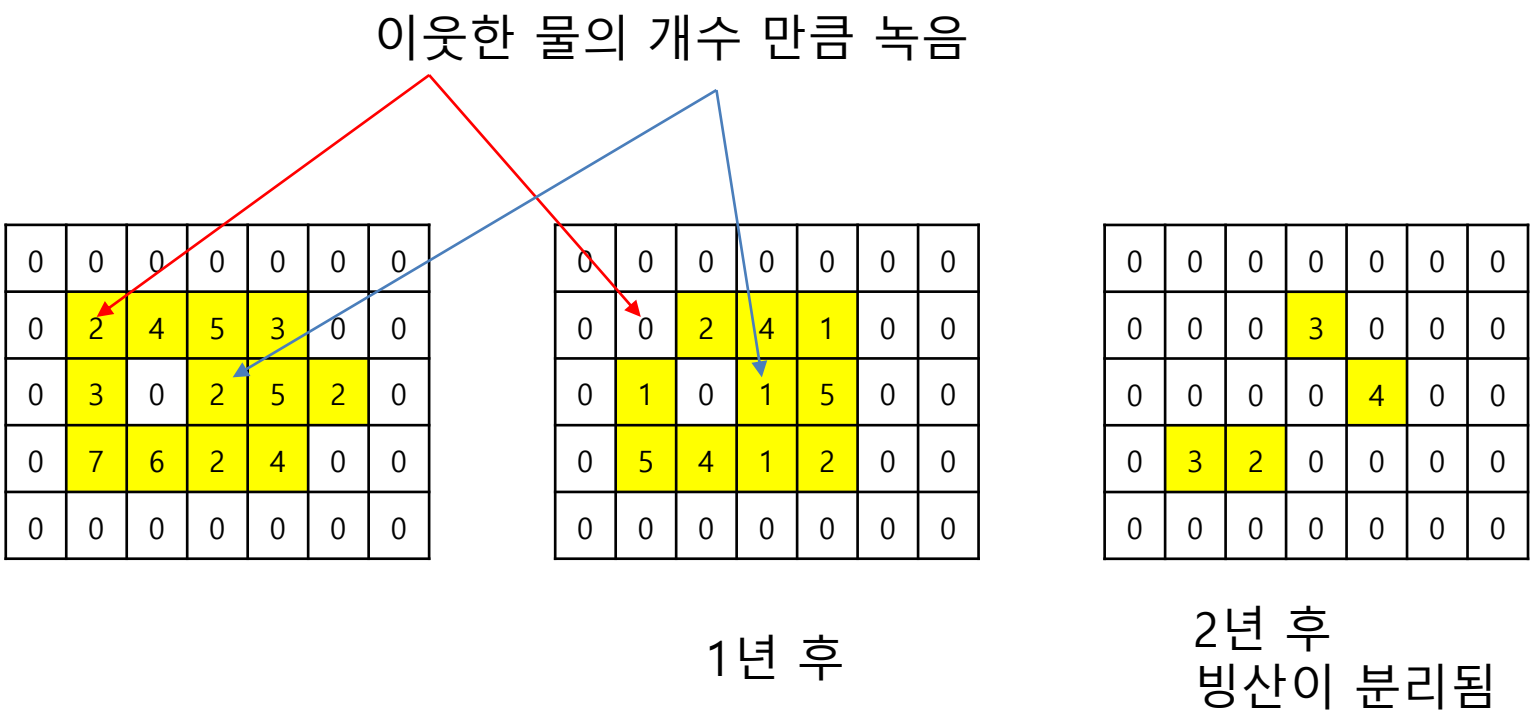
```
# 치즈 녹이기, 자료 재 정리, 녹인 치즈 개수 반환
def step3():
    cnt = 0
    for i in range(N):
        for j in range(M):
            if mat[i][j] == 2: mat[i][j] = 0
            elif mat[i][j] == 3: mat[i][j] = 0; cnt += 1
            elif mat[i][j] == 4: mat[i][j] = 1

    return cnt
```

빙산

https://www.acmicpc.net/problem/2573

빙산

https://www.acmicpc.net/problem/2573



빙산

```
N, M = map(int, input().split())
mat = [list(map(int, input().split())) for _ in range(N)]

ans = 1

while True:
    x, y, cnt = step1()    #녹인다.
    if cnt == 0:
        ans = 0
        break
    cnt1 = BFS(x, y)
    if cnt != cnt1:
        break
    ans += 1

print(ans)
```

0	0	0	0	0	0	0
0	0	2	4	1	0	0
0	1	0	1	5	0	0
0	5	4	1	2	0	0
0	0	0	0	0	0	0

1년 후

0	0	0	0	0	0	0
0	0	0	3	0	0	0
0	0	0	0	4	0	0
0	3	2	0	0	0	0
0	0	0	0	0	0	0

2년 후
빙산이 분리됨

cnt1 ➔ 빙산 내의 얼음 개수 2 반환

x, y, cnt ➔ 마지막 얼음의 좌표와 남아있는 얼음개수 3, 2, 4 반환

빙산

```
def step1():
    # 이웃한 물의 개수
    mat2 = [[0] * M for i in range(N)]

    for x in range(N):
        for y in range(M):
            if mat[x][y] == 0: continue
            for dx, dy in (-1, 0), (1, 0), (0, -1), (0, 1):
                xx, yy = x + dx, y + dy
                if mat[xx][yy] == 0:
                    mat2[x][y] += 1

    cnt = x = y = 0
    for i in range(N):
        for j in range(M):
            mat[i][j] -= mat2[i][j]
            if mat[i][j] < 0 : mat[i][j] = 0
            if mat[i][j] :
                cnt += 1
                x = i
                y = j
    return x, y, cnt
```

마지막 얼음 위치, 얼음의 개수 반환

mat

0	0	0	0	0	0	0
0	0	2	4	1	0	0
0	1	0	1	5	0	0
0	5	4	1	2	0	0
0	0	0	0	0	0	0

mat2

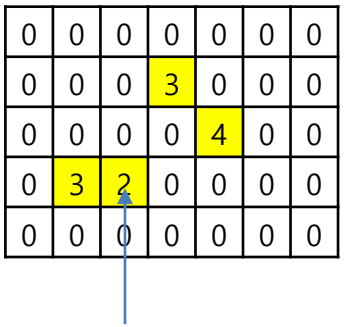
0	0	0	0	0	0	0
0	0	3	1	2	0	0
0	3	0	1	1	0	0
0	2	2	1	2	0	0
0	0	0	0	0	0	0

mat – mat2

mat'

0	0	0	0	0	0	0
0	0	0	3	0	0	0
0	0	0	0	4	0	0
0	3	2	0	0	0	0
0	0	0	0	0	0	0

빙산



빙산 내의 얼음 개수 2개

```
def BFS(x, y):
    visited = [[0] * M for _ in range(N)]
    q = []
    q.append((x, y))
    visited[x][y] = 1
    cnt = 1

    while q:
        x, y = q.pop(0)
        for dx, dy in (-1, 0), (1, 0), (0, -1), (0, 1):
            xx, yy = x + dx, y + dy
            if not visited[xx][yy] and mat[xx][yy] > 0:
                q.append((xx, yy))
                visited[xx][yy] = 1
                cnt += 1
    return cnt
```

보물섬

https://www.acmicpc.net/problem/2589

보물섬

https://www.acmicpc.net/problem/2589

W	L	L	W	W	W	L
L	L	L	W	L	L	L
L	W	L	W	L	W	W
L	W	L	W	L	L	L
W	L	L	W	L	W	W

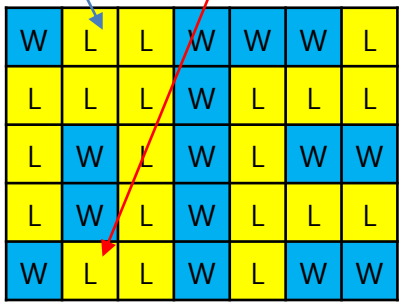
W	L	L	W	W	W	L
L	L	L	W	L	L	L
L	W	L	W	L	W	W
L	W	L	W	L	L	L
W	L	L	W	L	W	W

※ 최단 거리를 구할 때는 DFS 보다는 BFS

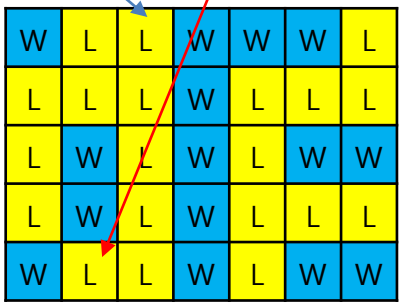
보물섬

```
N, M = map(int, input().split())
G = [input() for i in range(N)]

ans = 0
for i in range(N):
    for j in range(M):
        if G[i][j] == 'L':
            ans = max(ans, BFS(i, j))
print(ans)
```

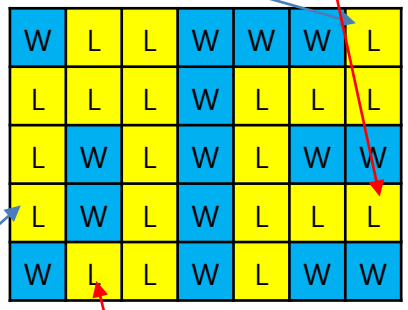
처음 시작 ➔ 6 반환



다음 시작 ➔ 5 반환

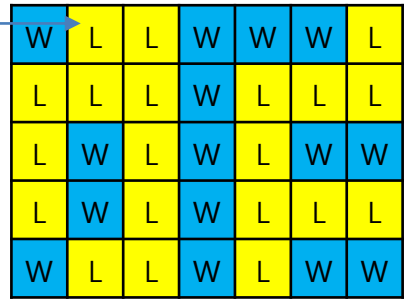


그 다음 시작 ➔ 7 반환



답은 이곳 시작 ➔ 8 반환

보물섬



dist

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

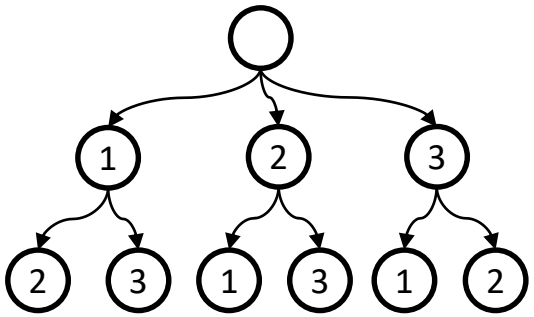
```
def BFS(x, y):
    q = []
    dist = [[0] * M for i in range(N)]

    q.append((x, y))
    dist[x][y] = 1

    while q:
        x, y = q.pop(0)

        for dx, dy in ((0, 1), (0, -1), (1, 0), (-1, 0)):
            xx, yy = x + dx, y + dy

            if not (0 <= xx < N and 0 <= yy < M): continue
            if G[xx][yy] == 'L' and dist[xx][yy] == 0:
                q.append((xx, yy))
                dist[xx][yy] = dist[x][y] + 1

    return max(sum(dist, [])) - 1
```

조합론의 완전검색(순열, 조합, 부분집합)

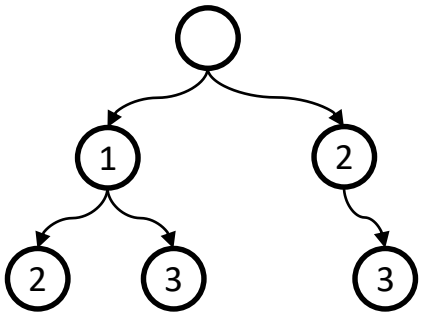
AD 보충수업 2일차

N = 3, R = 2, a = [1, 2, 3]

1 2
1 3
~~2 1~~
2 3
~~3 1~~
~~3 2~~



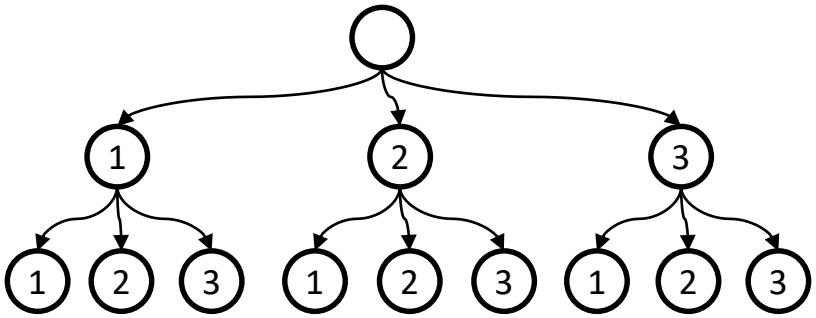
1 2
1 3
2 3



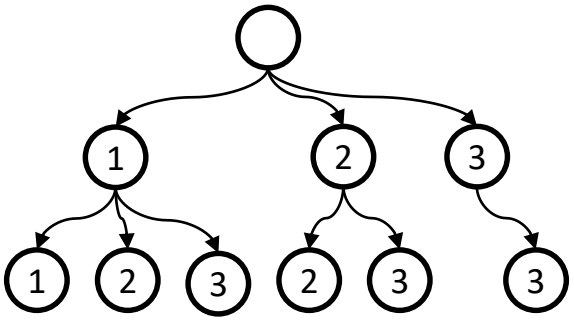
순열

조합

1 1
1 2
1 3
~~2 1~~
2 2
2 3
~~3 1~~
~~3 2~~
3 3



1 1
1 2
1 3
2 2
2 3
3 3



중복 순열

중복 조합

- **순열 생성 재귀적 알고리즘1**

```
perm(n, r)
        if (r == 0) print_arr()
        else
                for (i : n – 1 ~ 0)
                        swap(a[i], a[n - 1])
                        t[r - 1] = a[n - 1]
                        perm(n - 1, r - 1)
                        swap(a[i], a[n - 1])
```

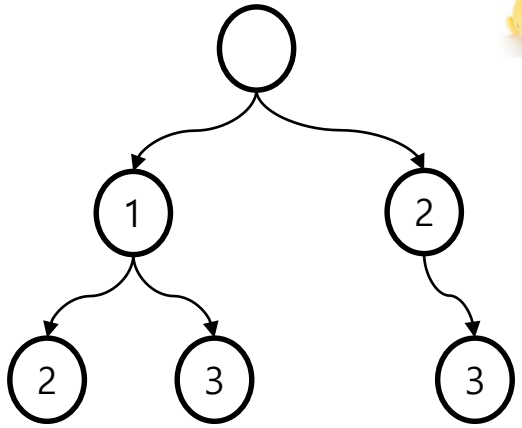
- **순열 생성 재귀적 알고리즘2**

```
perm(k)
        if (k == R) print_arr()
        else
                for (i : k ~ N - 1)
                        swap(k, i)
                        perm(k + 1)
                        swap(k, i)
```

- **순열 생성 재귀적 알고리즘2**

```
Visited[N-1]
perm(k)
        if (k == N) print_arr()
        else
                for (i : 0 ~ N - 1)
                        if (visited[i]) continue
                        t[k] = a[i]
                        visited[i] = true
                        perm(k + 1)
                        visited[i] = false
```

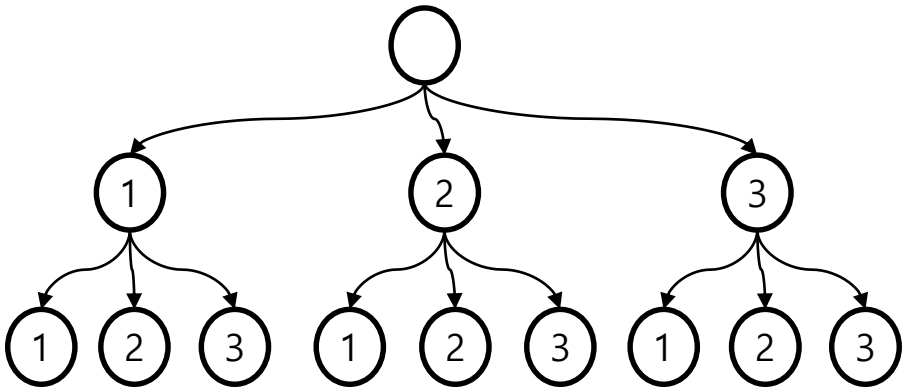
- **조합 생성 재귀적 알고리즘2**
  - 초기값 : k = 0, s = 0, N, R



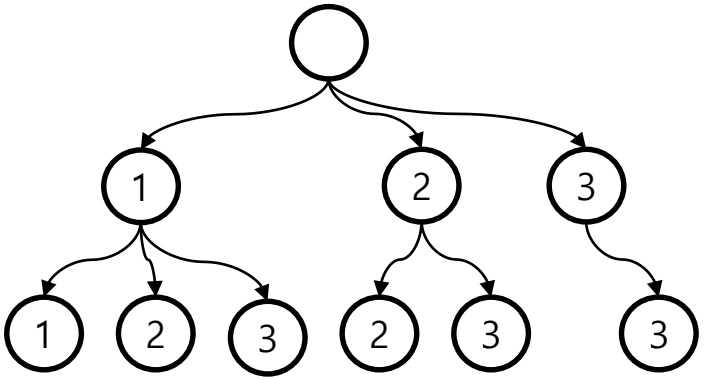

```
comb(k, s) // 깊이, 시작숫자
        if (k == R) print_arr()
        else
                for (int i : s ~ N - R + k )
                        t[k] = a[i]
                        comb(k + 1, i + 1)
```

- **중복 순열 생성 재귀적 알고리즘2** n^r

```
PI(k) // 깊이
        if (k == R) print_arr()
        else
                for (i : 0 ~ N - 1)
                        t[k] = a[i]
                        pi_r(k + 1)
```



- **중복조합 생성 재귀적 알고리즘2**
  - 초기값 : K = 0, s = 1, N, R




```
H(k, s) // 깊이, 시작숫자
        if (k == R) print_arr()
        else
                for (int i : s ~ N )
                        t[k] = a[i]
                        H(k + 1, i)
```

- **부분집합 생성 재귀적 알고리즘2**

```
def power_set_r(k):
    if k == N: print(a)
    else:
        a[k] = 1; power_set_r(k + 1)
        a[k] = 0; power_set_r(k + 1)


N = 3
a = [0] * N
power_set_r(0)
```

스타트와 링크

https://www.acmicpc.net/problem/14889

스타트와 링크

N = 6 일 경우

스타트 팀 0,1,2 그러면 링크 팀은 3,4,5

N//2개 뽑는 조합을 구하고

스타트 팀 = $S_{01} + S_{02} + S_{10} + S_{12} + S_{20} + S_{21}$

링크 팀 = $S_{34} + S_{35} + S_{43} + S_{45} + S_{53} + S_{54}$

구해진 조합에서 2개 나열하는 순열을 구한다.

스타트와 링크

```
N = int(input())
R = N // 2
t = [0] * R
mat = [list(map(int, input().split())) for _ in range(N)]

ans = 1e9
solve(0, 0)

print(ans)
```

0	1	2	3	4	5
1	0	2	3	4	5
1	2	0	3	4	5
1	2	3	0	4	5
1	2	3	4	0	5
1	2	3	4	5	0

스타트와 링크

```
def solve(k, s):
    global ans
    if k == R:
        …

    else:
        for i in range(s, N + (k - R) + 1):
            t[k] = i
            solve(k + 1, i + 1)
```

N이 6일 때 N//2 개의 조합을 구한다.

스타트와 링크

```
def solve(k, s):
    global ans
    if k == R:
        start = link = 0

        x = list(set([x for x in range(N)]) - set(t))

        for i in range(R - 1):
            for j in range(i + 1, R):
                start += (mat[t[i]][t[j]] + mat[t[j]][t[i]])
        for i in range(R - 1):
            for j in range(i + 1, R):
                link += (mat[x[i]][x[j]] + mat[x[j]][x[i]])

        ans = min(ans, abs(start - link))

    else:
        …
```

x = [3, 4, 5]

t = [0, 1, 2]

3개에서 2개 나열하는 순열

$S_{01} + S_{02} + S_{10} + S_{12} + S_{20} + S_{21}$

$S_{34} + S_{35} + S_{43} + S_{45} + S_{53} + S_{54}$

퇴사

https://www.acmicpc.net/problem/14501

퇴사

	1일	2일	3일	4일	5일	6일	7일
Ti	3	5	1	1	2	4	2
Pi	10	20	10	20	15	40	200

	1일	2일	3일	4일	5일	6일	7일
Ti	7	6	5	4	3	2	1
Pi	10	20	10	20	15	40	200

Ti 와 Pi 두 개의 인자가 있으며 항상 비례적이지는 않다.

	1일	2일	3일	4일	5일	6일	7일
Ti	1	1	1	1	1	1	1
Pi	10	20	10	20	15	40	200

완전검색 : 모든 부분 집합 조사

모든 부분 집합 ➔ Ti로 제외시키고 ➔ Pi로 최적해 구하기

퇴사

```
N = int(input())

Ti = [0] * N
Pi = [0] * N
Si = [0] * N

for i in range(N):
    Ti[i], Pi[i] = map(int, input().split())

ans = 0
solve(0)

print(ans)
```

	1일	2일	3일	4일	5일	6일	7일
Ti	3	5	1	1	2	4	2
Pi	10	20	10	20	15	40	200

퇴사

	1일	2일	3일	4일	5일	6일	7일
Ti	3	5	1	1	2	4	2
Pi	10	20	10	20	15	40	200

```
def solve(k):
    global ans
    if k == N:
        …
    else:
        Si[k] = 1; solve(k + 1)
        Si[k] = 0; solve(k + 1)
```

Si	1	1	1	1	1	1	1

N개에 대한 부분 집합

퇴사

	1일	2일	3일	4일	5일	6일	7일
Ti	3	5	1	1	2	4	2
Pi	10	20	10	20	15	40	200

선택유무조사

Si	1	1	1	1	1	1	1

```
def solve(k):
    global ans
    if k == N:
        for i in range(N):
            if Si[i]:
                for j in range(i + 1, i + Ti[i]):
                    if j >= N or Si[j] : return

        tsum = 0
        for i in range(N):
            if Si[i]:
                tsum += Pi[i]
        if tsum > ans : ans = tsum
    else:
        …
```

	1일	2일	3일	4일	5일	6일	7일
Ti	3	5	1	1	2	4	2
Pi	10	20	10	20	15	40	200

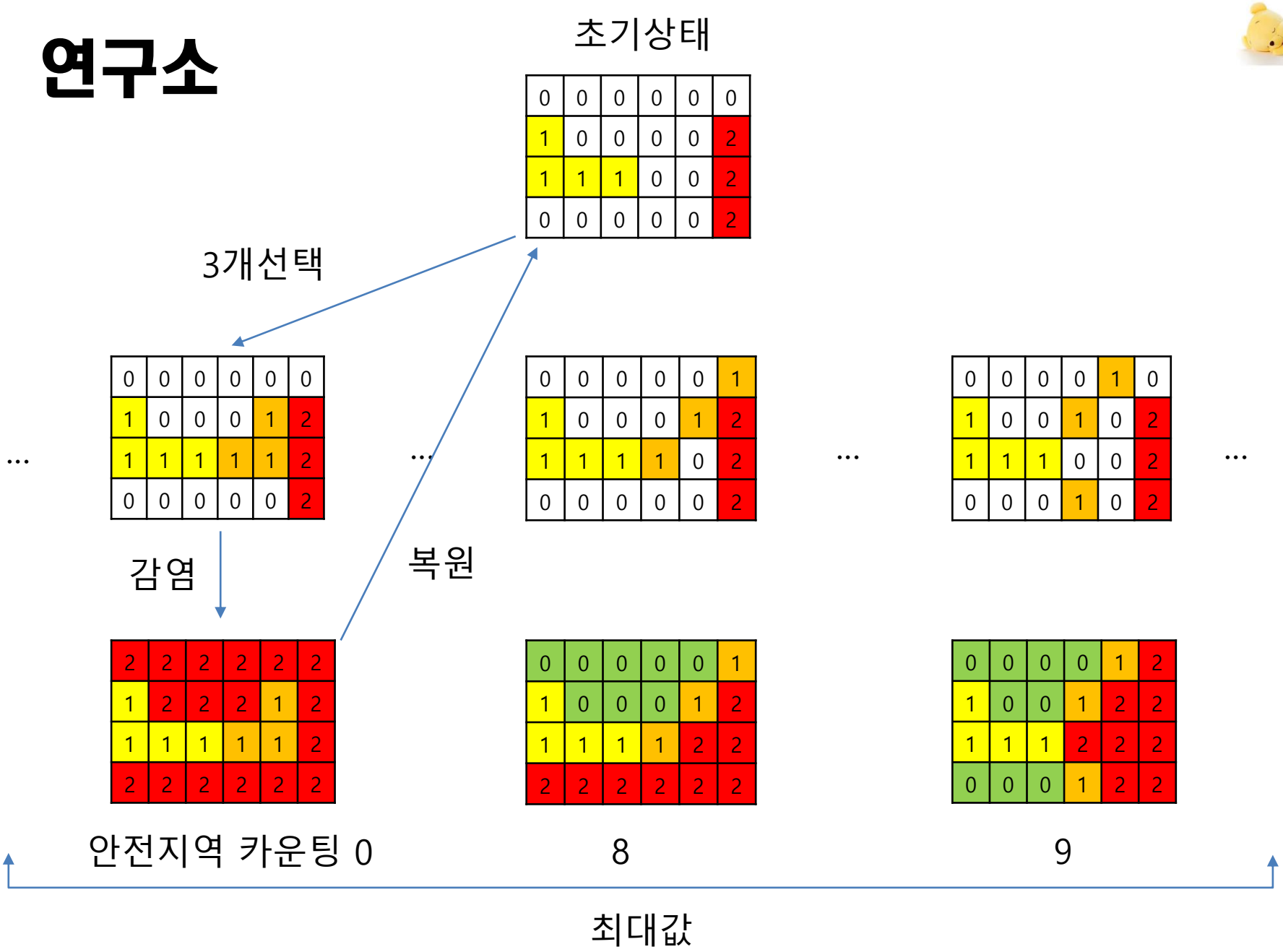
Si	1	0	0	0	1	0	0

10 + 15 = 25 가 최선인가?

연구소

https://www.acmicpc.net/problem/14502

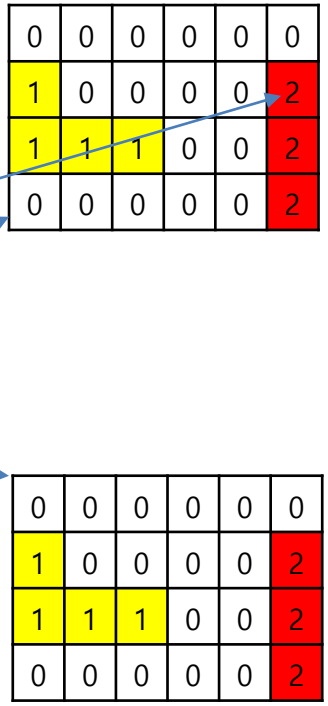
연구소



연구소

```
N, M = map(int, input().split())
mat = [list(map(int, input().split())) for _ in range(N)]

backup_mat = [[0] * M for i in range(N)]
virus_pos = []
safe_pos = []

for i in range(N):
    for j in range(M):
        if mat[i][j] == 2:
            virus_pos.append((i, j))
        elif mat[i][j] == 0:
            safe_pos.append((i, j))
        backup_mat[i][j] = mat[i][j]

ans = 0
combi = [0] * 3
solve(0, 0)
print(ans)
```

안전영역 개수에서 3개 선택 조합 저장할 배열



연구소

```
def solve(k, s):
    global ans
    if k == 3:
        ...
    else:
        for i in range(s, len(safe_pos) + (k - 3) + 1):
            combi[k] = i
            solve(k + 1, i + 1)
```

0	1	2

0, 1	0, 2	0, 3	0, 4	...	3, 4

안전영역 개수에서 임의
3개 선택 ➔ 조합 생성

0	0	0	0	0	0
1	0	0	0	0	2
1	1	1	0	0	2
0	0	0	0	0	2

연구소

```
def solve(k, s):
    global ans
    if k == 3:
        for i in range(3):
            x, y = safe_pos[combi[i]]
            mat[x][y] = 1

        for x, y in virus_pos:
            virus_infact(x, y)

        ans = max(ans, sum(mat, []).count(0))

        for i in range(N):
            for j in range(M):
                mat[i][j] = backup_mat[i][j]
    else:
        ...
```

벽 세우기

0	0	0	0	0	0
1	0	0	0	1	2
1	1	1	1	1	2
0	0	0	0	0	2

감염시키기

2	2	2	2	2	2
1	2	2	2	1	2
1	1	1	1	1	2
2	2	2	2	2	2

안전영역세기

0	0	0	0	0	0
1	0	0	0	0	2
1	1	1	0	0	2
0	0	0	0	0	2

초기 상태로 복원

연구소

```
def virus_infact(x, y):
    mat[x][y] = 2
    for dx, dy in ((0, 1), (0, -1), (1, 0), (-1, 0)):
        xx, yy = x + dx, y + dy
        if not (0 <= xx < N and 0 <= yy < M): continue
        if not mat[xx][yy]:
            virus_infact(xx, yy)
```

0	0	0	0	0	0
1	0	0	0	1	2
1	1	1	1	1	2
0	0	0	0	0	2

(1, 5)에서 감염

2	2	2	2	2	2
1	2	2	2	1	2
1	1	1	1	1	2
0	0	0	0	0	2

치킨배달

https://www.acmicpc.net/problem/15686

치킨배달

https://www.acmicpc.net/problem/15686

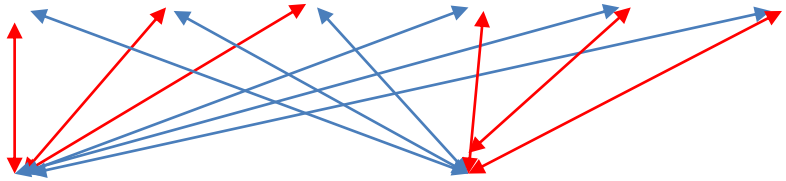
M = 2 (2개 치킨 집만 선택)

	0	1	2	3	4
0	0	2	0	1	0
1	1	0	1	0	0
2	0	0	0	0	0
3	2	0	0	1	1
4	2	2	0	1	2

집위치

0,3	1,0	1,2	3,3	3,4	4,3

모든 집과



0,1	3,0	4,0	4,1	4,4

치킨집위치

M개 선택한 치킨 집 사이의 거리 중 최소 거리 선택

치킨 집에서 임의 2개 선택

$_5C_2$

최소거리의 합 구하기

합 중 최소 구하기

치킨배달

	0	1	2	3	4
0	0	2	0	1	0
1	1	0	1	0	0
2	0	0	0	0	0
3	2	0	0	1	1
4	2	2	0	1	2

```
N, M = map(int, input().split())
mat = [list(map(int, input().split())) for _ in range(N)]

home, chicken = [], []
for i in range(N):
    for j in range(N):
        if mat[i][j] == 1:
            home.append((i, j))
        elif mat[i][j] == 2:
            chicken.append((i, j))

...

ans = 1e9
combi = [0] * M
solve(0, 0)
print(ans)
```

0,3	1,0	1,2	3,3	3,4	4,3

0,1	3,0	4,0	4,1	4,4

치킨 집에서 M개 선택하는
조합 생성

치킨배달

```
N, M = map(int, input().split())
mat = [list(map(int, input().split())) for _ in range(N)]

...

dist = [[0] * len(home) for i in range(len(chicken))]
for i in range(len(chicken)):
    for j in range(len(home)):
        dist[i][j] = abs(chicken[i][0] - home[j][0]) + abs(chicken[i][1] -
home[j][1])

ans = 1e9
combi = [0] * M
solve(0, 0)
print(ans)
```

모든 집과 모든 치킨 집 사이의 거리를 구해 놓기

치킨집 \ 집						
	2	2	2	5	6	6
	6	2	4	3	4	4
	7	3	5	4	5	3
	6	4	4	3	4	2
	5	7	5	2	1	1

치킨배달

집에서 최소거리 치킨집 선택

예

0	2

2	2	2	5	6	6
6	2	4	3	4	4
7	3	5	4	5	3
6	4	4	3	4	2
5	7	5	2	1	1

2+2+2+4+5+3=18

```
def solve(k, s):
    global ans
    if k == M:
        tsum = 0
        for h in range(len(home)):
            tmin = 1e9
            for c in combi:
                tmin = min(tmin, dist[c][h])
            tsum += tmin
        ans = min(ans, tsum)
    else:
        for i in range(s, len(chicken) + (k - M) + 1):
            combi[k] = i
            solve(k + 1, i + 1)
```

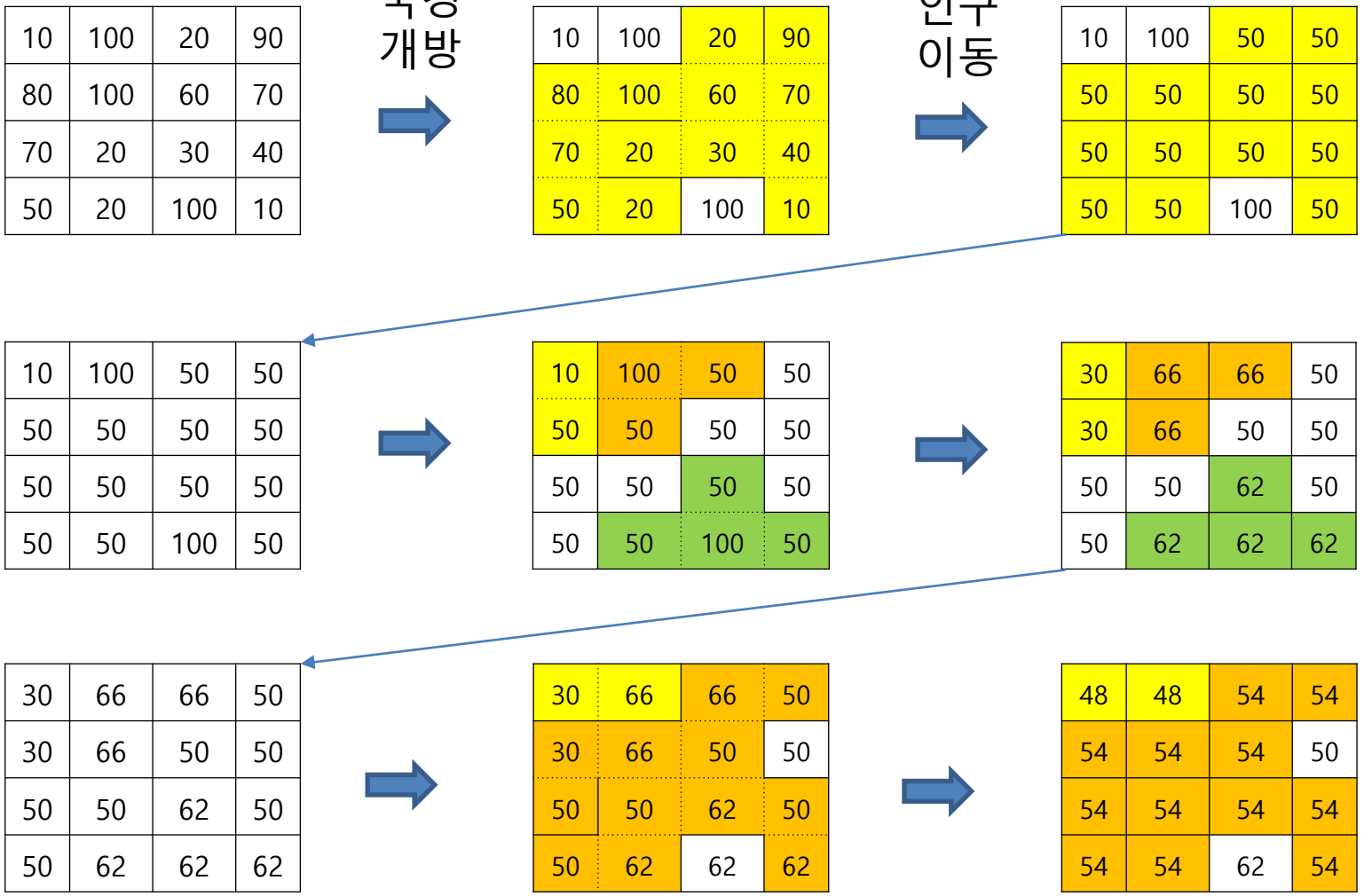
M길이 조합 구하기

인구이동

https://www.acmicpc.net/problem/16234

인구이동

L = 10, R = 50


국경
개방
인구
이동
10 100 20 90
80 100 60 70
70 20 30 40
50 20 100 10
10 100 20 90
80 100 60 70
70 20 30 40
50 20 100 10
10 100 50 50
50 50 50 50
50 50 50 50
50 50 100 50
10 100 50 50
50 50 50 50
50 50 50 50
50 50 100 50
30 66 66 50
30 66 50 50
50 50 62 50
50 62 62 62
30 66 66 50
30 66 50 50
50 50 62 50
50 62 62 62
48 48 54 54
54 54 54 50
54 54 54 54
54 54 62 54

인구이동

```
N, L, R = map(int, input().split())
mat = [list(map(int, input().split())) for _ in range(N)]

cnt = 0
while True:
    visited = [[0] * N for _ in range(N)]
    moved = False
    for i in range(N):
        for j in range(N):
            if not visited[i][j]:
                bfs(i, j)
    if moved: cnt += 1
    else: break

print(cnt)
```

매번 visited 새로 만들기

인구 이동이 없으면 중단

10	100	20	90
80	100	60	70
70	20	30	40
50	20	100	10

인구이동

10	100	20	90
80	100	60	70
70	20	30	40
50	20	100	10

```
def bfs(x, y):
    global moved
    q = []
    tList = []
    visited[x][y] = True
    q.append((x, y))

    while q:
        x, y = q.pop()
        tList.append((x, y))
        for dx, dy in ((0, 1), (0, -1), (1, 0), (-1, 0)):
            xx, yy = x + dx, y + dy
            if not (0 <= xx < N and 0 <= yy < N): continue
            if not visited[xx][yy] and L <= abs(mat[x][y] - mat[xx][yy]) <= R:
                visited[xx][yy] = 1
                q.append((xx, yy))

     ...
```

이동한 정점 저장

0,2	1,2	2,2	2,1	...	0,3

조건에 맞으면 이동

인구이동

```
def bfs(x, y):
    global moved
    q = []
    tList = []
    visited[x][y] = True
    q.append((x, y))

        ...

    tlen = len(tList)
    if tlen > 1:
        tsum = 0
        for x, y in tList:
            tsum += mat[x][y]
        for x, y in tList:
            mat[x][y] = tsum // tlen
        moved = True
```

인구 이동 있었음

10	100	20	90
80	100	60	70
70	20	30	40
50	20	100	10

10	100	50	50
50	50	50	50
50	50	50	50
50	50	100	50

0,2	1,2	2,2	2,1	...	0,3

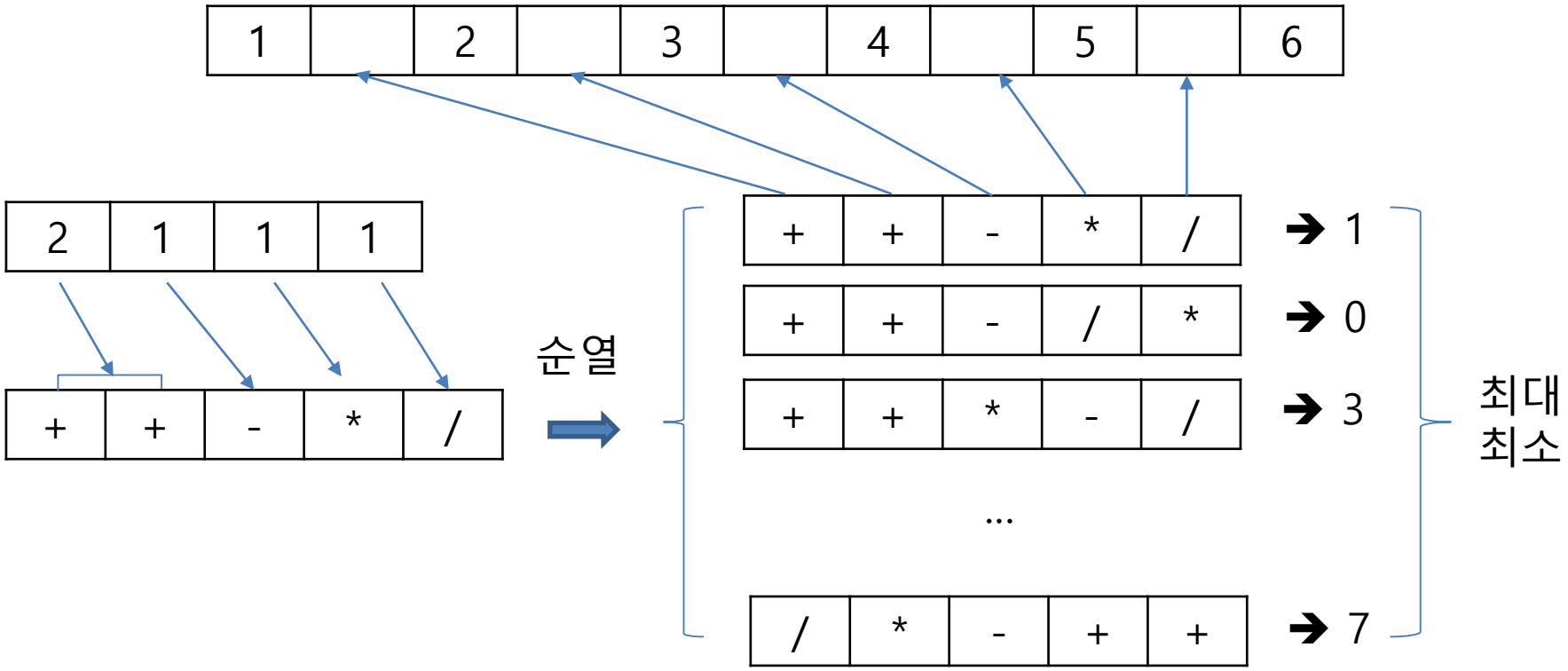
완전검색 : 백트래킹

AD 보충수업 3일차

연산자끼워넣기

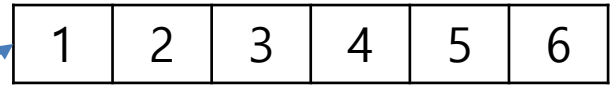
https://www.acmicpc.net/problem/14888

연산자끼워넣기

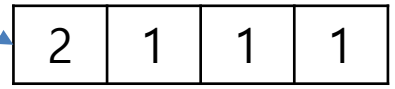


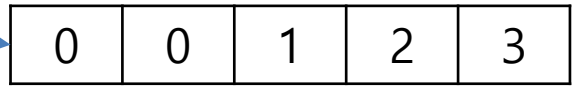
연산자끼워넣기

```
N = int(input())
nums = list(map(int, input().split()))
opc = list(map(int, input().split()))

ops = []
for i in range(4):
    ops += [i] * opc[i]

maxans, minans = -1e10, 1e10
solve(0)
print("%d₩n%d" % (maxans, minans))
```



(+, -, *, / 의 개수)

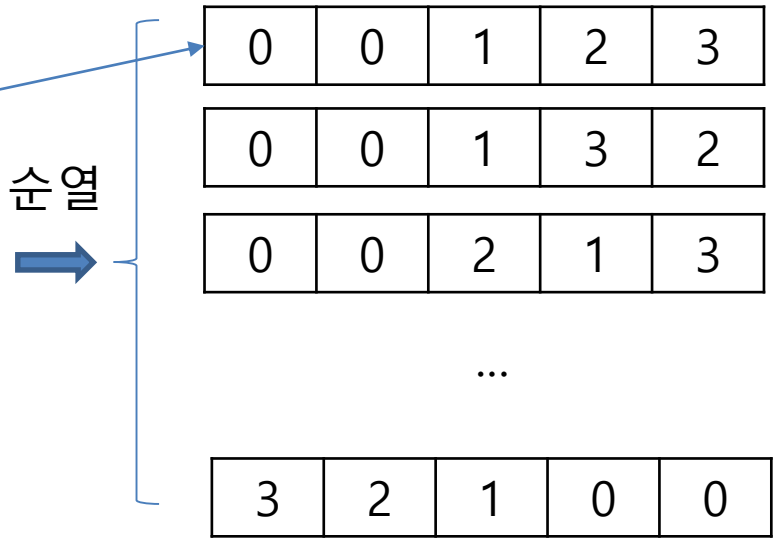




(+,+,-,*,/)

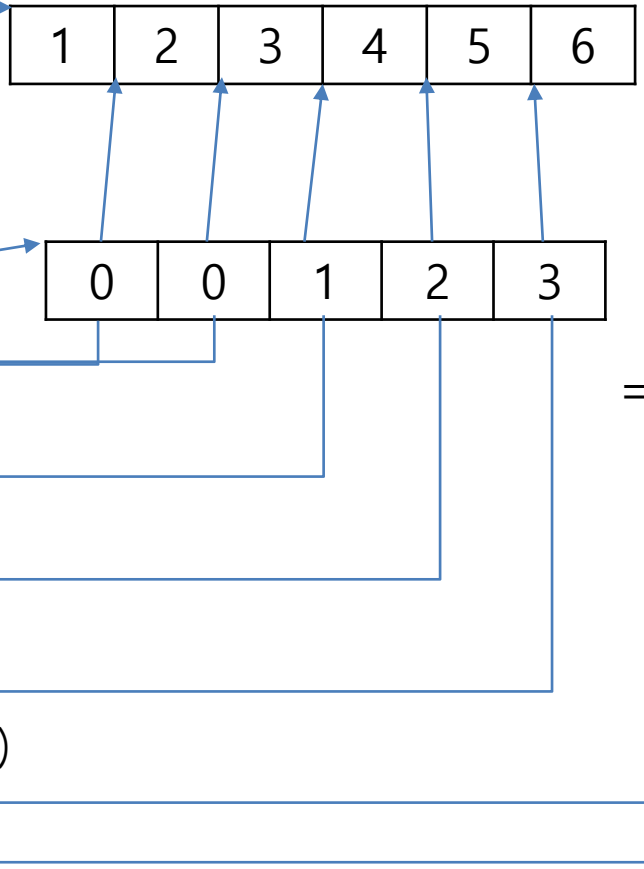
연산자끼워넣기

```
def solve(k):
    global maxans, minans
    if k == N - 1:
        ...
    else:
        for i in range(k, N - 1):
            ops[k], ops[i] = ops[i], ops[k]
            solve(k + 1)
            ops[k], ops[i] = ops[i], ops[k]
```



연산자끼워넣기

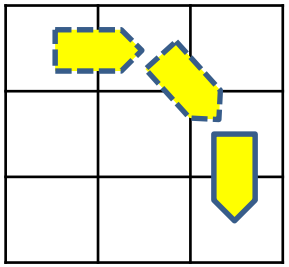
```
def solve(k):
    global maxans, minans
    if k == N - 1:
        val = nums[0]
        for i in range(N - 1):
            if ops[i] == 0:
                val += nums[i + 1]
            elif ops[i] == 1:
                val -= nums[i + 1]
            elif ops[i] == 2:
                val *= nums[i + 1]
            else:
                val = int(val / nums[i + 1])
        maxans = max(maxans, val)
        minans = min(minans, val)
    else:
        ...
```

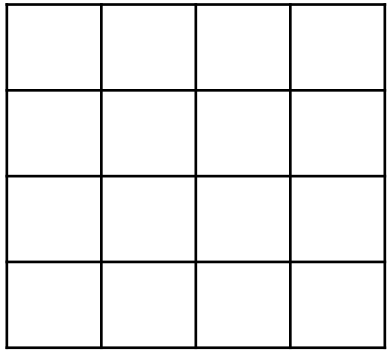


파이프 옮기기1

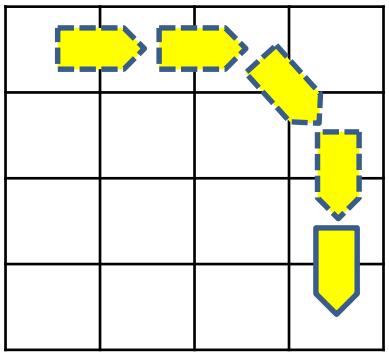
https://www.acmicpc.net/problem/17070

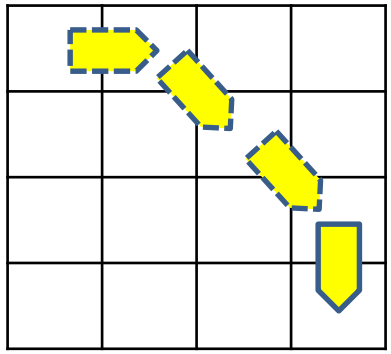
파이프 옮기기1

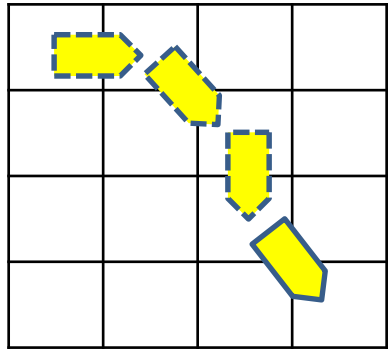




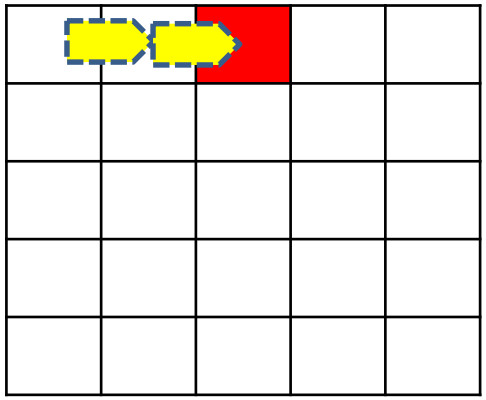
파이프 옮기기1

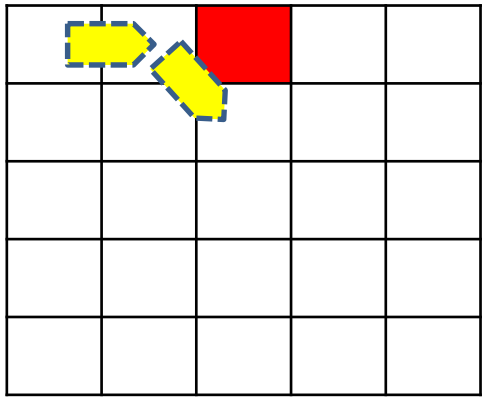




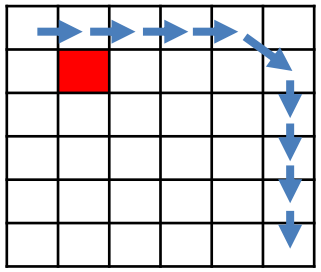
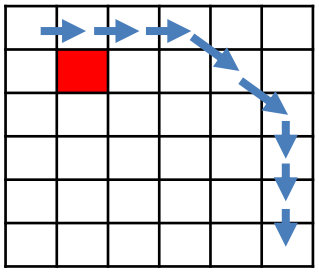
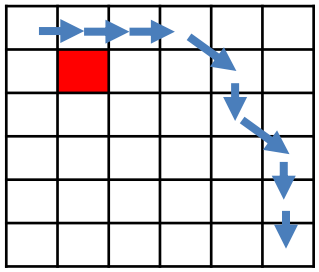
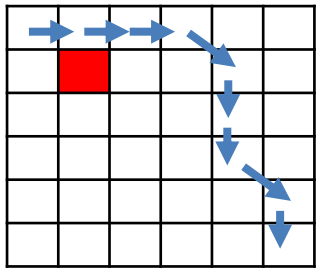
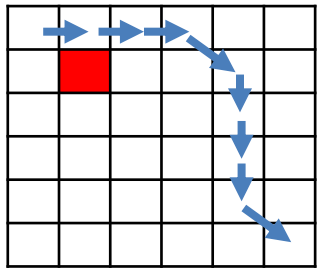
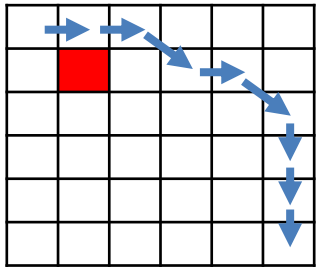
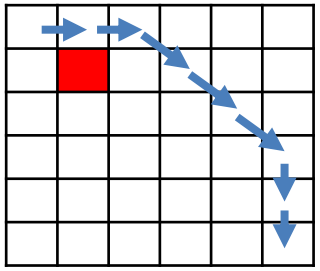
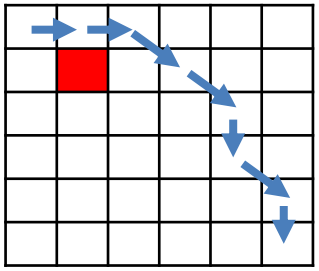
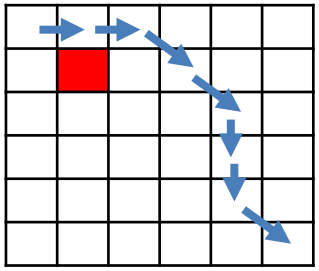
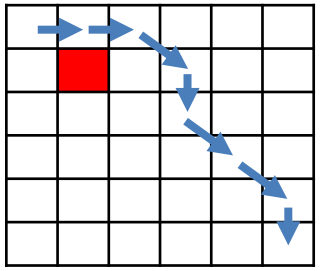
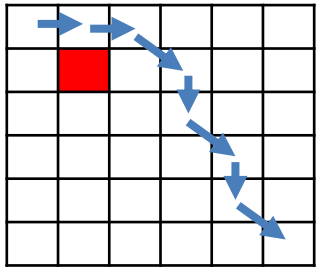
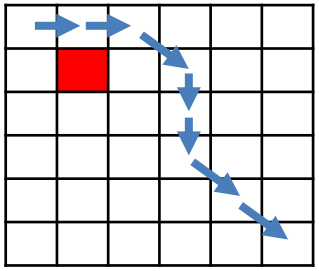
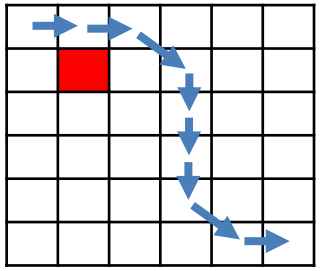


파이프 옮기기1



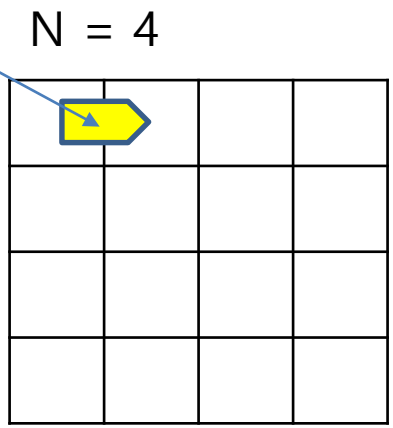


파이프 옮기기1

파이프 옮기기1

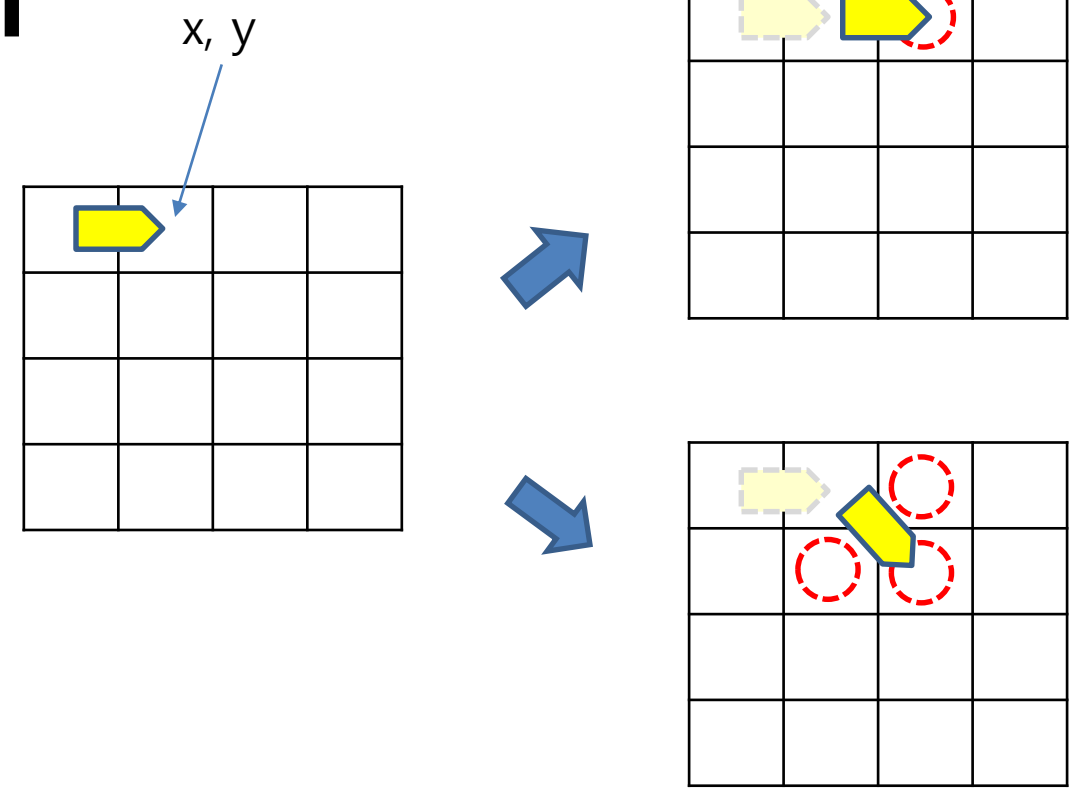
```
N = int(input())
mat = [[*map(int, input().split())] for _ in range(N)]
ans = 0
solve(0, 1, 0)  #(x,y,d) d: →0, ↓1, ↘2
print(ans)
```



파이프 옮기기1

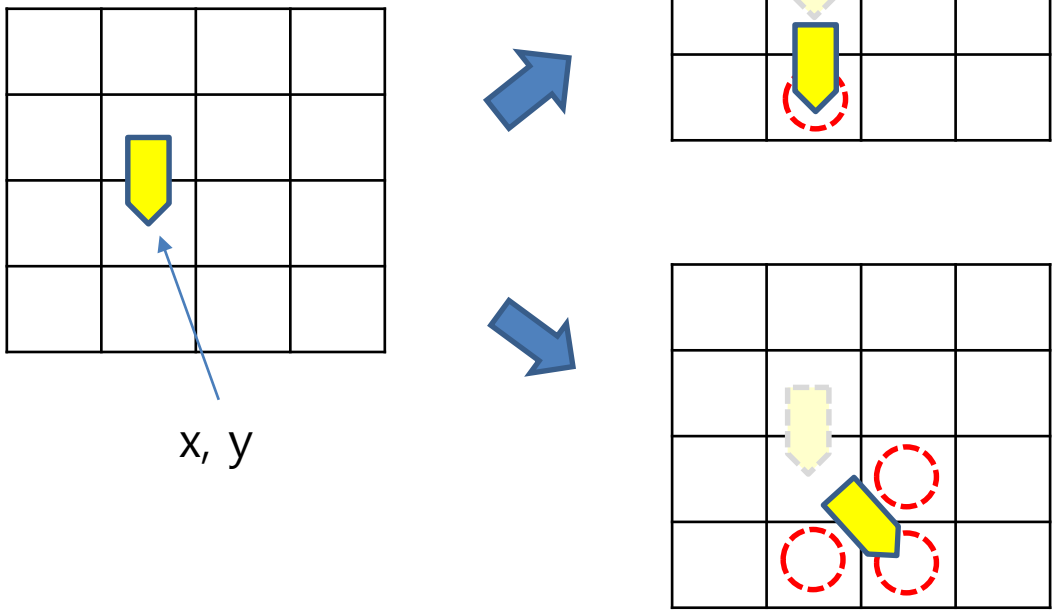
```
def solve(x, y, d):  # d: → 0, ↓ 1, ↘ 2
    global ans
    if x == N - 1 and y == N - 1:
        ans += 1
```

파이프 옮기기1



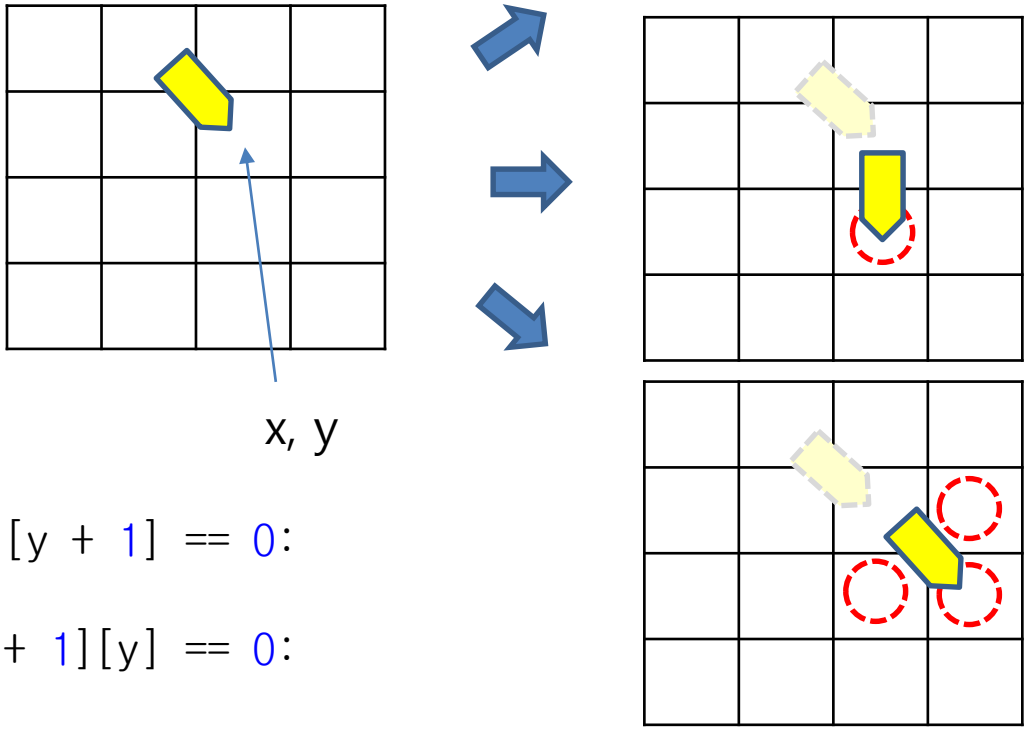

```
# d: → 0, ↓ 1, ↘ 2
def solve(x, y, d):
    global ans
    ...

    if d == 0:
        if y + 1 < N and mat[x][y + 1] == 0:
            solve(x, y + 1, 0)
        if x + 1 < N and y + 1 < N and ₩
                mat[x + 1][y] == mat[x][y + 1] == mat[x + 1][y + 1] == 0:
            solve(x + 1, y + 1, 2)

    ...
```

파이프 옮기기1




```
# d: → 0, ↓ 1, ↘ 2
def solve(x, y, d):
    global ans

    ...

    if d == 1:
        if x + 1 < N and mat[x + 1][y] == 0:
            solve(x + 1, y, 1)
        if x + 1 < N and y + 1 < N and ₩
                mat[x + 1][y] == mat[x][y + 1] == mat[x + 1][y + 1] == 0:
            solve(x + 1, y + 1, 2)

    ...
```

파이프 옮기기1




```
# d: → 0, ↓ 1, ↘ 2
def solve(x, y, d):
    global ans
    ...

    if d == 2:
        if y + 1 < N and mat[x][y + 1] == 0:
            solve(x, y + 1, 0)
        if x + 1 < N and mat[x + 1][y] == 0:
            solve(x + 1, y, 1)
        if x + 1 < N and y + 1 < N and ₩
                mat[x + 1][y] == mat[x][y + 1] == mat[x + 1][y + 1] == 0:
            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 ${}_{5}C_{3}$ = 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 ${}_5C_3 = 10$

0	0	0	0	0
0	0	0	0	0
0	0	0	0	0
1	1	1	1	1
0	0	0	0	0
^	^	^		

길이가 1 아래로 내려올때 까지 기다려야 한다.

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
^	^	^		

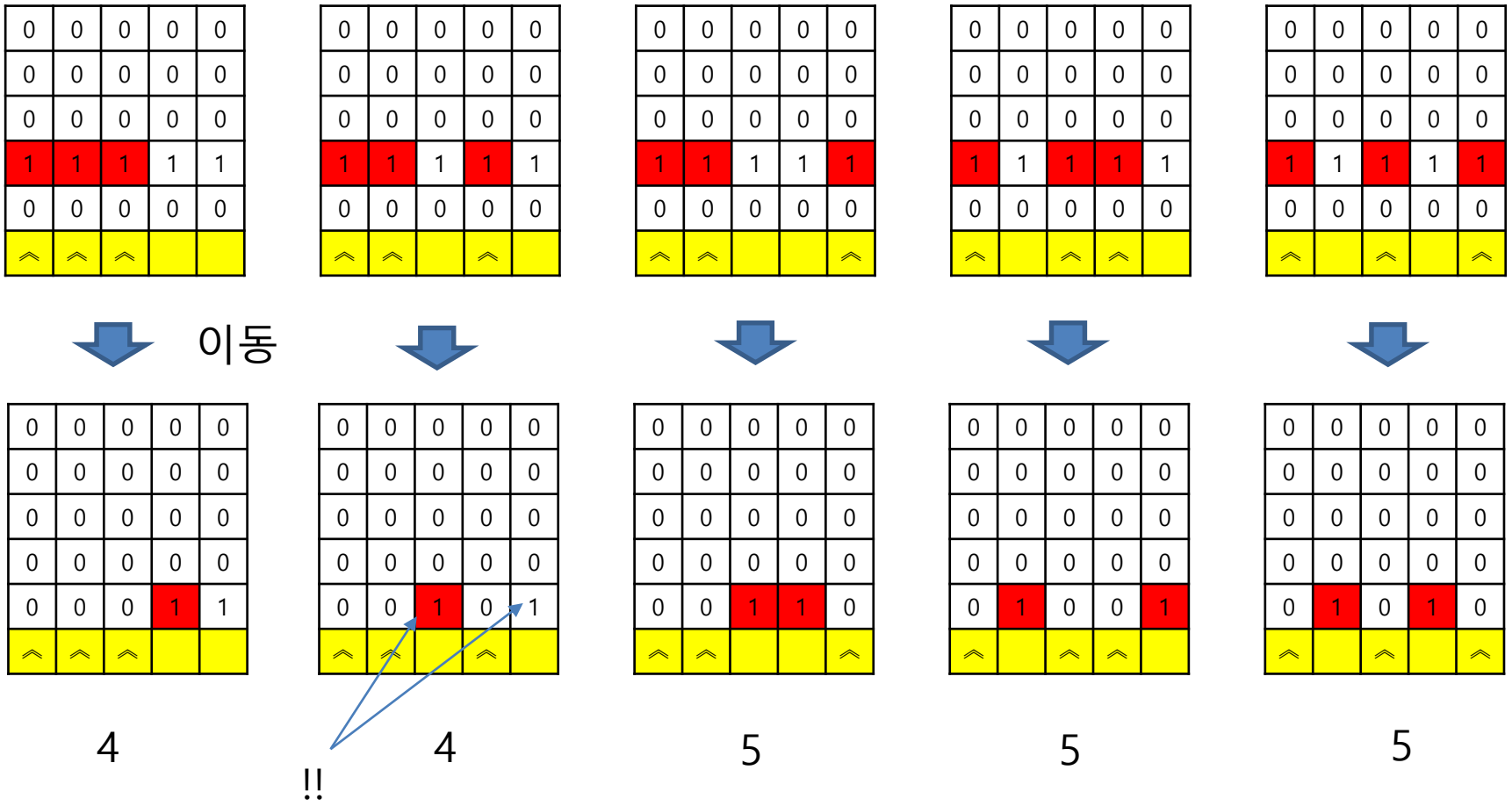
첫 번째 문제와 같은 상황

캐슬디펜스

3번예제

5, 5
길이 d = 2

5곳에서 3군데 고르는 경우의 수 $_5C_3 = 10$

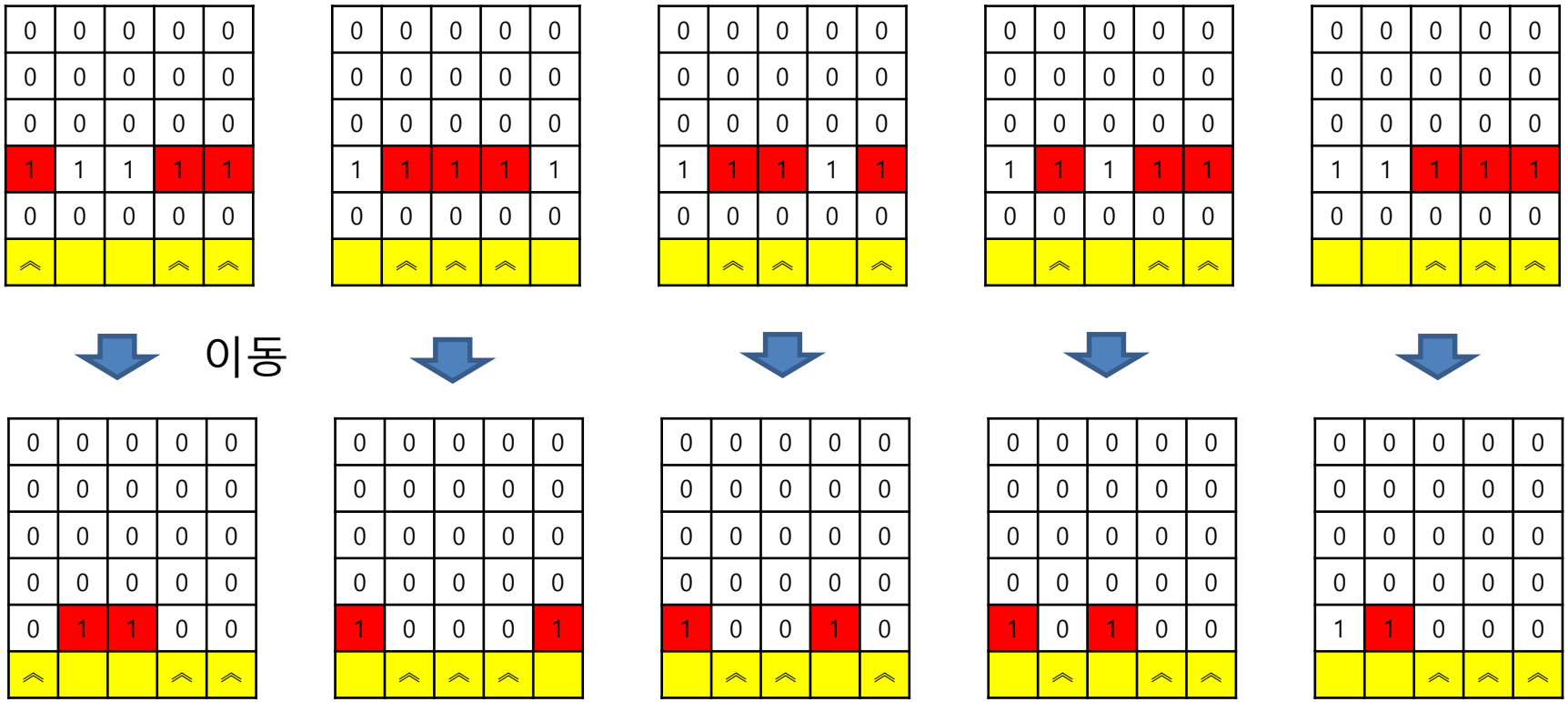


캐슬디펜스

3번예제

5, 5
길이 d = 2

5곳에서 3군데 고르는 경우의 수 $_5C_3 = 10$



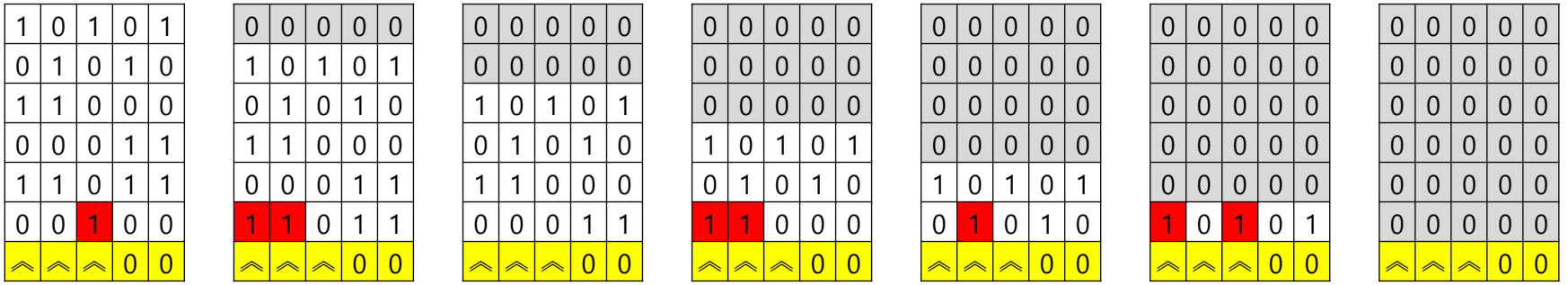
최대값 5

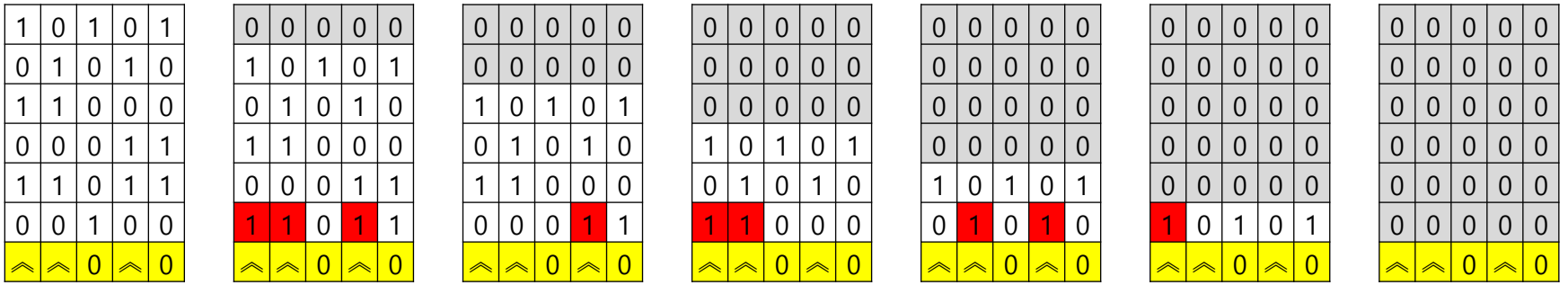
캐슬디펜스

5번예제

6, 5
길이 d = 1

5곳에서 3군데 고르는 경우의 수 $_5C_3 = 10$





...

최개 9

캐슬디펜스

6번예제

6, 5
길이 d = 2

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
0	0	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
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	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	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
0	1	0	1	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
︽	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 14

캐슬디펜스

```
N, M, D = map(int, input().split())
mat = [list(map(int, input().split())) for _ in range(N)]

archer = [0] * 3
ans = 0
for i in range(M - 2):
    for j in range(i + 1, M - 1):
        for k in range(j + 1, M):
            killed = [[0] * M for _ in range(N)]
            archer[0], archer[1], archer[2] = i, j, k
            solve(N)
            ans = max(ans, sum(sum(killed, [])))
print(ans)
```

거리

$_MC_3$ 조합생성

죽은 적군 계산하고 매번 새로 생성

N 행 이동

죽은 적군 계산, 최대값 갱신

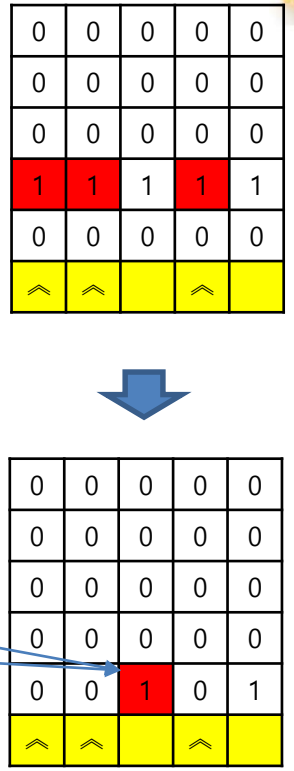
캐슬디펜스

모든 행을 처리

```
def solve(k):
    if k == 0:
        return
    else:
        t = []
        t.append(kill(k, archer[0]))
        t.append(kill(k, archer[1]))
        t.append(kill(k, archer[2]))
        for found, x, y in t:
            if found:
                killed[x][y] = 1
        solve(k - 1)
```

겹칠수 있으므로 각 궁수가 저격 가능한 적의 위치를 모은다.

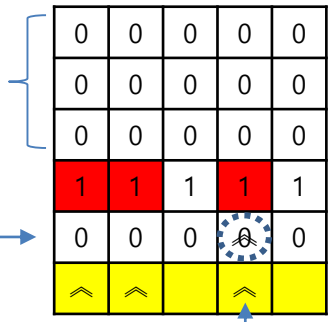
겹친다

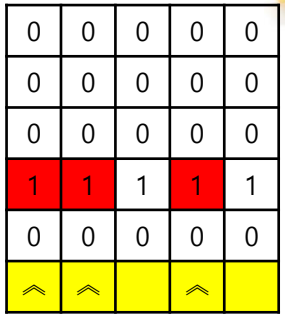


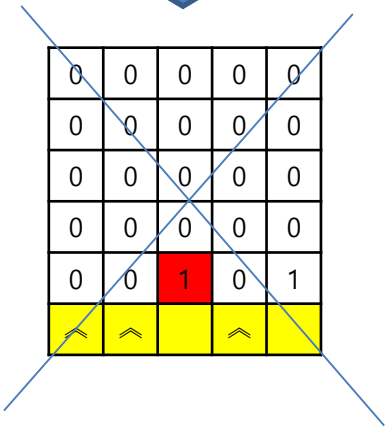
한 행씩 처리

캐슬디펜스

실제 적을 움직이는 것이 아니라 궁수가 이동한다고 가정








```
def kill(k, y):
    xx, yy, min_d = - 1, - 1, 100
    for i in range(k - 1, -1, -1):
        for j in range(M):
            if mat[i][j] and not killed[i][j]:
                td = abs(i - k) + abs(j - y)
                if td < min_d:
                    xx, yy, min_d = i, j, td
                elif td == min_d and j < yy:
                    xx, yy = i, j

    return (min_d <= D, xx, yy)
```

거리가 가장 가까운 적의 위치를 검색

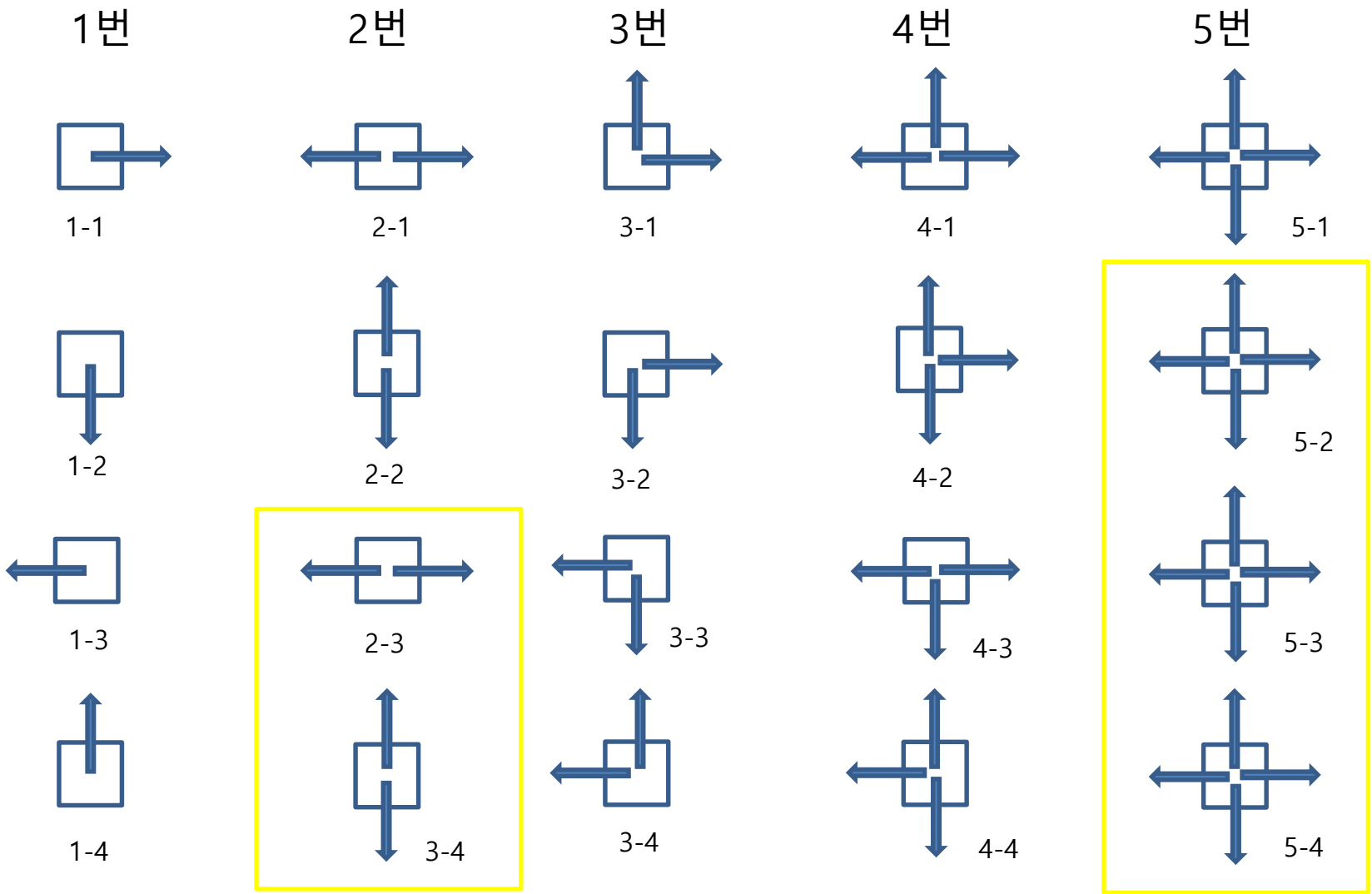
같은 거리면 왼쪽을 선택

가장 가까운 거리라도 사정 거리 이내여야 의미가 있음

감시

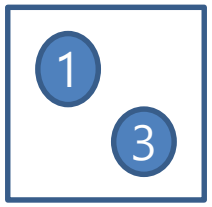
https://www.acmicpc.net/problem/15683

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

```
N, M = map(int, input().split())
mat = [list(map(int, input().split())) for _ in range(N)]
observed = [[0] * M for i in range(N)]
cctvXYC = []

...

ans = 0
direction = [0] * len(cctvXYC)
solve(0)
print(N*M - ans)
```

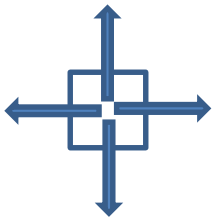
CCTV로 감시되는 칸의 정보를 저장

CCTV의 종류와 위치를 저장할 배열

중복순열을 저장할 배열

사각지대 = 전체 영역 - 최대감시영역

감시




```
...
for x in range(N):
    for y in range(M):
        if mat[x][y] == 0 : continue
        elif mat[x][y] == 6:
            observed[x][y] = 1
        elif mat[x][y] == 5:
            observed[x][y] = 1
            fill_right(x, y, observed)
            fill_left(x, y, observed)
            fill_up(x, y, observed)
            fill_down(x, y, observed)
        else:
            cctvXYC.append((mat[x][y], x, y))
...
```

벽 감시영역으로 처리

5번 CCTV는 회전의 의미가 없음. 미리 감시영역처리

CCTV 종류, 위치

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

감시

```
def fill_right(x, y, arr):
    yy = y + 1
    while yy < M and mat[x][yy] != 6:
        arr[x][yy] = 1
        yy += 1
```

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

```
def fill_left(x, y, arr):
    yy = y - 1
    while yy > -1 and mat[x][yy] != 6:
        arr[x][yy] = 1
        yy -= 1
```

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

```
def fill_up(x, y, arr):
    xx = x + 1
    while xx < N and mat[xx][y] != 6:
        arr[xx][y] = 1
        xx += 1
```

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

```
def fill_down(x, y, arr):
    xx = x - 1
    while xx > -1 and mat[xx][y] != 6:
        arr[xx][y] = 1
        xx -= 1
```

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

```
def solve(k):
    global direction
    if k == len(cctvXYC):
        observe()
    else:
        if cctvXYC[k][0] == 2:
            for i in range(2):
                direction[k] = i
                solve(k + 1)
        else:
            for i in range(4):
                direction[k] = i
                solve(k + 1)
```

4방향의 중복순열을 저장할 배열, 0,1,2,3
➔ 우,하,좌,상

5번을 제외하고 CCTV개수 만큼 중복순열을 생성했으면

2번 CCTV는 2번만 회전

2번

4방향에 대한 중복 순열 생성

감시

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

```
def observe():
    global ans
    tobserved = [[0] * M for i in range(N)]
    for i in range(N):
        for j in range(M):
            tobserved[i][j] = observed[i][j]

    for i in range(len(cctvXYC)):
        cctvC, x, y = cctvXYC[i]
        dir = direction[i]
        tobserved[x][y] = 1
        if cctvC == 1:
            ...
        elif cctvC == 2:
            ...
        elif cctvC == 3:
            ...
        elif cctvC == 4:
            ...

    ans = max(ans, sum(sum(tobserved, [])))
```

5번 CCTV를 미리 처리
한 정보를 받아옴

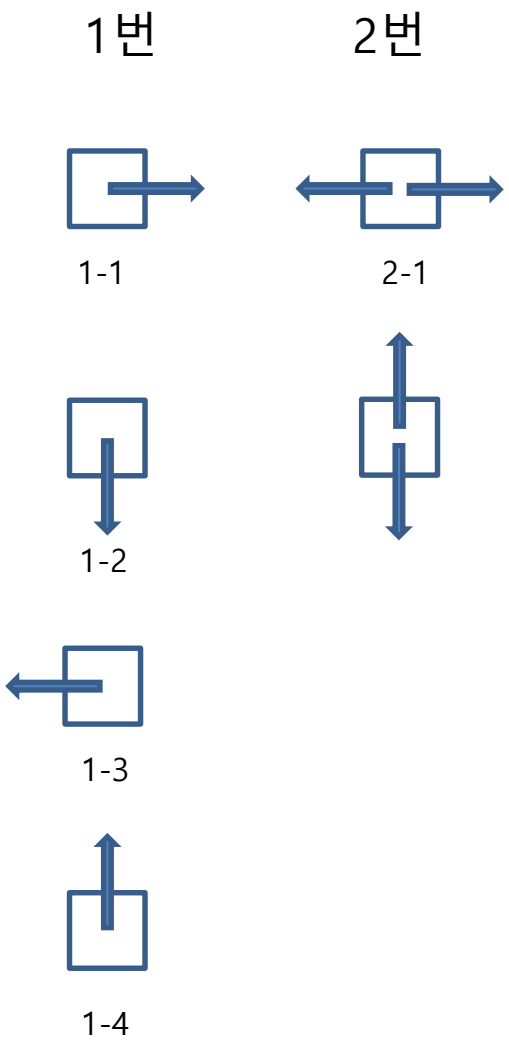
(1,0,0)	(1,1,1)	(1,2,2)	(1,3,3)	(1,4,4)	(1,5,5)

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

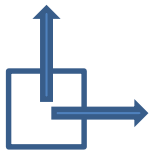
```
if cctvC == 1:
    if dir == 0:   fill_right(x, y, tobserved)
    elif dir == 1: fill_down(x, y, tobserved)
    elif dir == 2: fill_left(x, y, tobserved)
    elif dir == 3: fill_up(x, y, tobserved)
elif cctvC == 2:
    if dir == 0:
        fill_right(x, y, tobserved)
        fill_left(x, y, tobserved)
    elif dir == 1:
        fill_up(x, y, tobserved)
        fill_down(x, y, tobserved)
```



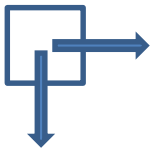
감시

```
elif cctvC == 3:
    if dir == 0:
        fill_up(x, y, tobserved)
        fill_right(x, y, tobserved)
    elif dir == 1:
        fill_right(x, y, tobserved)
        fill_down(x, y, tobserved)
    elif dir == 2:
        fill_down(x, y, tobserved)
        fill_left(x, y, tobserved)
    elif dir == 3:
        fill_left(x, y, tobserved)
        fill_up(x, y, tobserved)
```

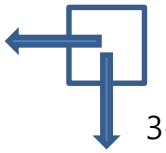
3번



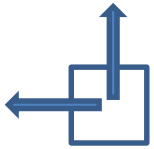
3-1



3-2



3-3

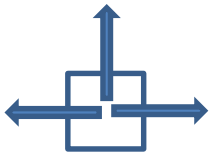


3-4

감시

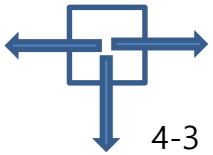
```
elif cctvC == 4:
    if dir == 0:
        fill_right(x, y, tobserved)
        fill_left(x, y, tobserved)
        fill_up(x, y, tobserved)
    elif dir == 1:
        fill_right(x, y, tobserved)
        fill_down(x, y, tobserved)
        fill_up(x, y, tobserved)
    elif dir == 2:
        fill_right(x, y, tobserved)
        fill_down(x, y, tobserved)
        fill_left(x, y, tobserved)
    elif dir == 3:
        fill_down(x, y, tobserved)
        fill_left(x, y, tobserved)
        fill_up(x, y, tobserved)
```

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

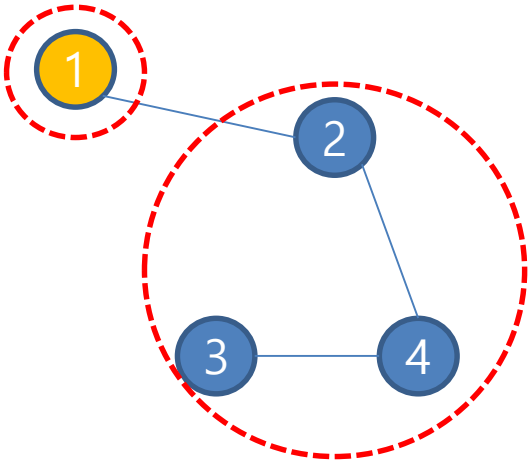
즉 부분 집합과 연관되어 있다. 1선거구를 부분 집합으로 구하고 전체 선거구에서 1선거구를 제거 하여 2선거구를 만들 수 있다.

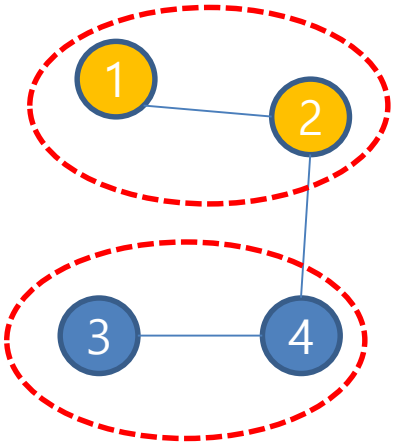
[1, 2, 3, 4]	-	[1] [2] [3] [4] [1, 2] [1, 3] [1, 4] [2, 3] [2, 4] [3, 4] [1, 2, 3] [1, 2, 4] [1, 3, 4] [2, 3, 4]	=	[2, 3, 4] [1, 3, 4] [1, 2, 4] [1, 2, 3] [3, 4] [2, 4] [2, 3] [1, 4] [1, 3] [1, 2] [4] [3] [2] [1]
		1선거구		2선거구

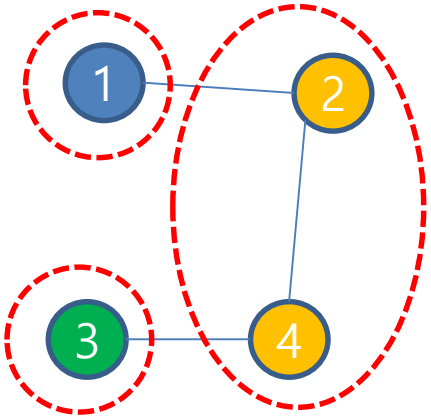
게리맨더링

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]





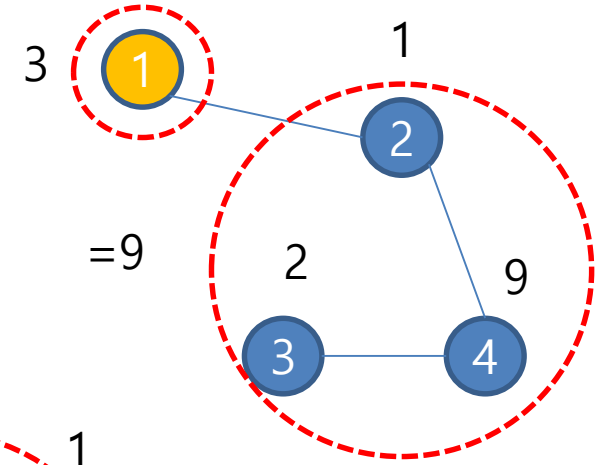


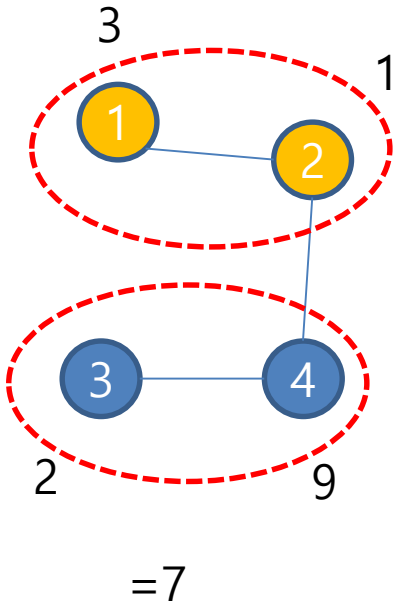
2개의 선거구로 분리되지 않는 경우도 있다.

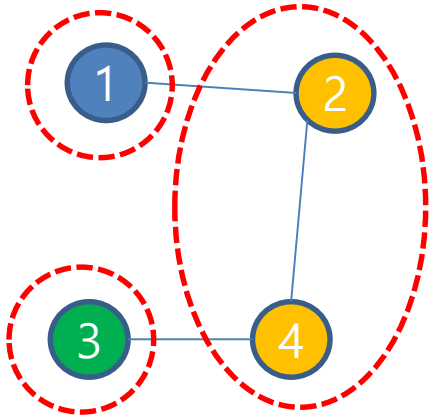
게리맨더링

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

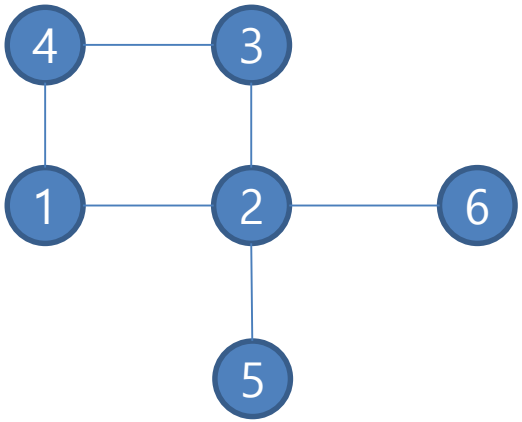
```
6
5 2 3 4 1 2
2 2 4
4 1 3 6 5
2 4 2
2 1 3
1 2
1 2
```

구역인구수

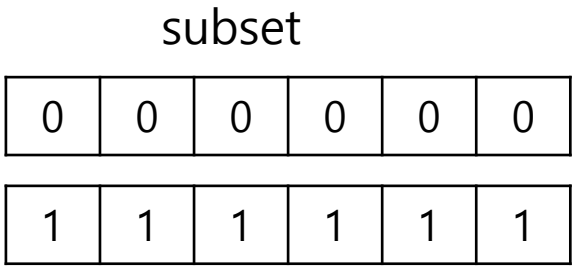
```
N = int(input())
people = list(map(int, input().split()))
G = []

for i in range(N):
    tlist = list(map(int, input().split()))
    G.append(tlist[1:])

ans = 1e9
subset = [0] * N
solve(0)

if ans == 1e9:
    print(-1)
else:
    print(ans)
```

[[2,4], [1,3,6,5], [4,2], [1,3], [2], [2]]

두 구역으로 나눌 수 있는 방법이 없으면



게리맨더링




```
def solve(k):
    global ans
    if k == N:
        if sum(subset) == 0 or sum(subset) == N: return

        ...
    else:
        subset[k] = 1; solve(k + 1)
        subset[k] = 0; solve(k + 1)
```

구역 개수의
모든 부분집합
을 생성

게리맨더링

```
def solve(k):
    global ans
    if k == N:
        ...
        area1, area2 = [], []
        for i in range(N):
            if subset[i]:
                area1.append(i)
            else:
                area2.append(i)
        visited = [0] * N
        v1 = dfs(area1[0], area1, visited)
        v2 = dfs(area2[0], area2, visited)
        if sum(visited) == N:
            ans = min(ans, abs(v1 - v2))
    else:
        ...
```

1	0	0	1	0	0

0	3

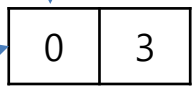
1	2	4	5

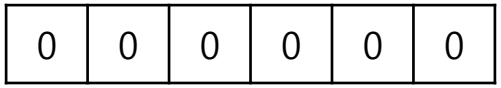
0	0	0	0	0	0

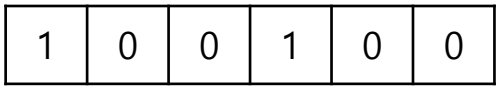
visited를 매 경우 마다 새로 만든다.

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

게리맨더링

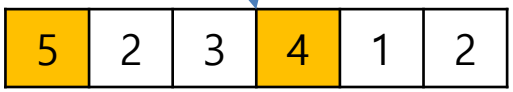


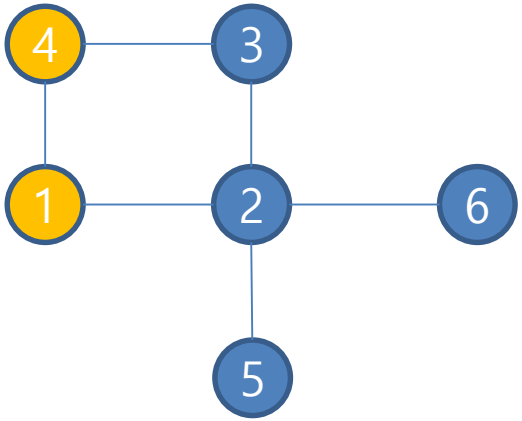




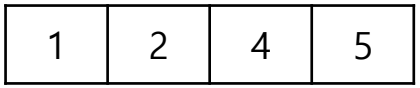

```
def dfs(v, area, visited):
    ret = people[v]
    visited[v] = 1
    for u in G[v]:
        if not visited[u - 1] and u - 1 in area:
            ret += dfs(u - 1, area, visited)
    return ret
```

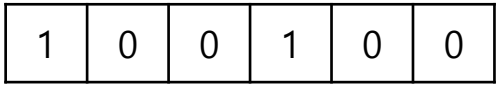
최종 9 반환

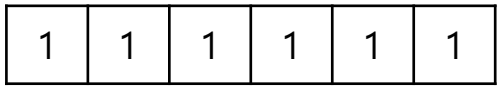




게리맨더링








```
def dfs(v, area, visited):
    ret = people[v]
    visited[v] = 1
    for u in G[v]:
        if not visited[u - 1] and u - 1 in area:
            ret += dfs(u - 1, area, visited)
    return ret
```

최종 8 반환