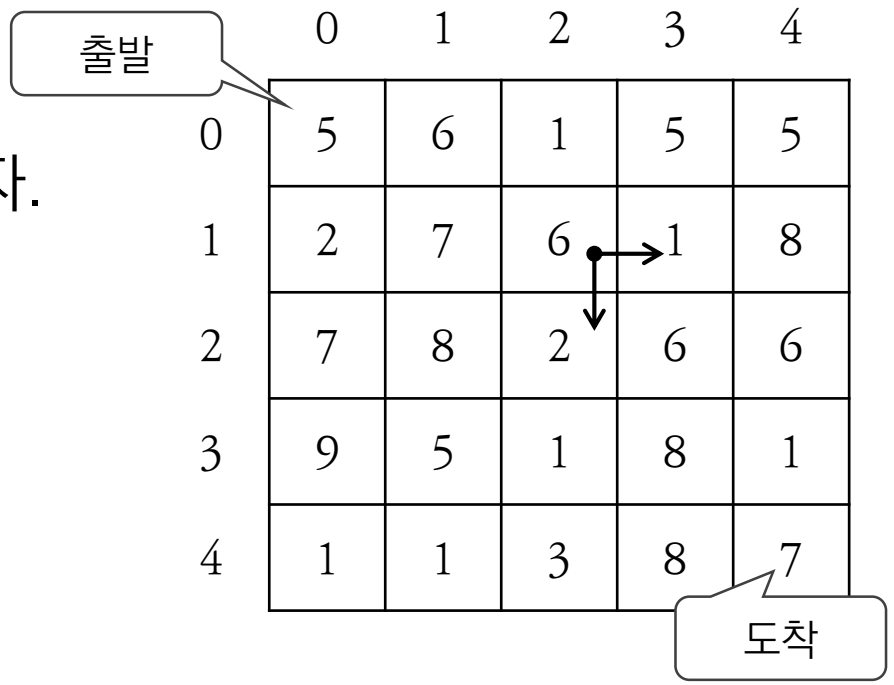
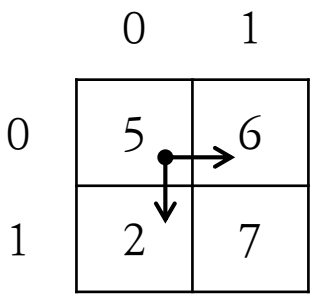
미로

오른쪽과 아래로만 이동하는 경우

- 각 칸에서는 오른쪽이나 아래로만 이동할 수 있다.
- 출발은 맨 왼쪽 위, 도착은 맨 오른쪽 아래이다.
- 출발부터 도착까지 지나는 각 칸의 합계가 최소가 되도록 움직였을 때, 합계를 계산하라.
- 각 칸은 1에서 9사이의 숫자.



- 다음 칸의 좌표 계산.
  - 현재 위치가 (0, 0)이면, 갈 수 있는 칸은 (0, 1)이나 (1, 0)이다.
  - 재귀 호출의 각 단계에서는 현재 위치를 저장한다.
  - 오른 쪽 칸으로 가는 경우와 아래로 가는 경우를 나누어 호출한다.
  - 현재 칸 까지의 숫자의 합을 항상 구한다.



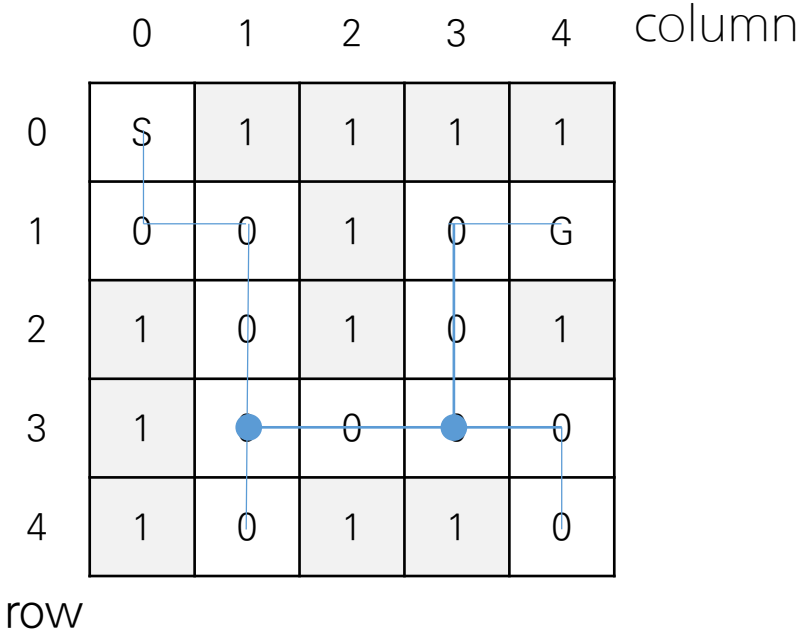

```
f( row, col, sum)
{
    if( 도착점이면 )
        if( sum+m[row][col] < min )
            min = sum + m[row][col];
    else
        if(col+1<N)
            f( row, col + 1, sum + m[row][col]);
        if(row+1<N)
            f( row + 1, col, sum + m[row][col]);
}
```

- 모든 칸을 지나지 않고 답을 찾는 방법.
  - 도착점이 아닌 경우, 지나온 숫자의 합이 min보다 크면 return.

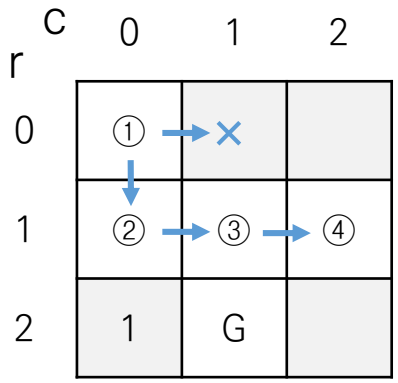
```
f( row, col, sum)
{
    if( 도착점이면 )
        if( sum+m[row][col] < min )
            min = sum + m[row][col];
    else if( sum + m[row][col] > min )
        return;
    else
        f( row, col + 1, sum + m[row][col]);
        f( row + 1, col, sum + m[row][col]);
}
```

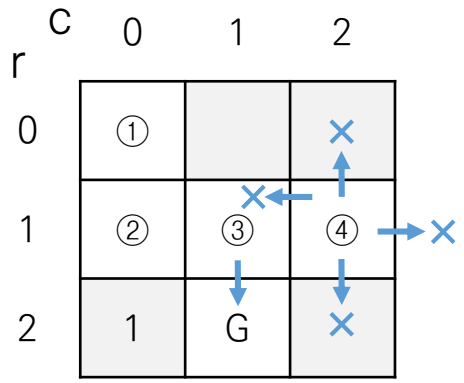
미로의 응용

- 배열에 저장한 미로
  - 1-벽, 0-통로, S-출발, G-도착
  - 도착 가능 여부 판단, 최단 거리 구하기, 경로의 수 구하기 등.



▪ 미로에서의 이동

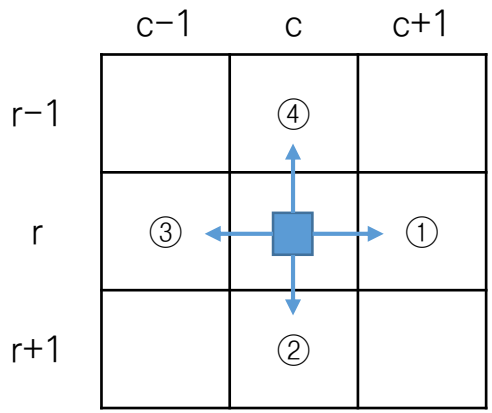




	r	c	
①	0	0	오른쪽 불가. 아래 가능.
②	1	0	오른쪽 가능.
③	1	1	오른쪽 가능.

	r	c	
④	1	2	오른쪽/아래 불가. 왼쪽 **이미 방문**. 위쪽 불가.
③	1	1	**이전 위치**로 되돌아감. 아래 가능.
G	2	1	도착

■ 이동할 칸의 좌표 계산

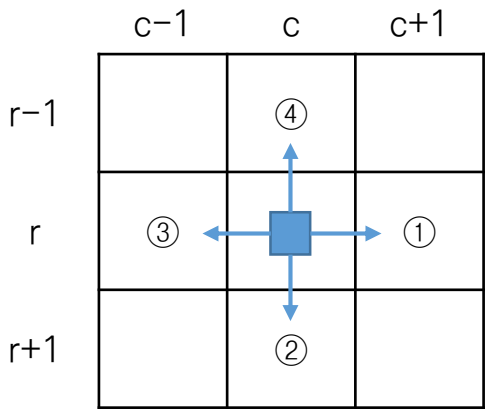



```
//크기가 NxN인 2차원 배열 일 때,
//현재 위치 ( r, c )에서 새 좌표로 이동하기.

if ( c+1 < N )
    next( r, c+1 ); // ①
if ( r+1 < N )
    next( r+1, c ); // ②
if ( c > 0 )
    next( r, c-1 ); // ③
if ( r > 0 )
    next( r-1, c ); // ④
```

벽으로 둘러싸인 미로의 경우 유효 좌표 검사가 필요없다.

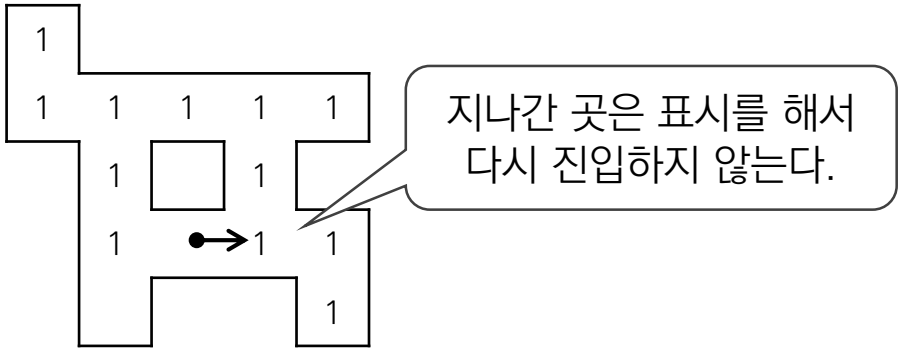
▪ 반복문을 이용한 이동할 칸의 좌표 계산

```
int dr[] = { 0, 1, 0, -1 };
int dc[] = { 1, 0, -1, 0};

...
for( i = 0 ; i < 4 ; i++ )
{
      nr = r + dr[i];
      nc = c + dc[i];
      if(( nr >= 0 )&&( nr < N )&&( nc >=0 )&&( nc < N ))
      {
            next( nr, nc );
      }
}
```



■ 경로의 존재만 확인하는 경우

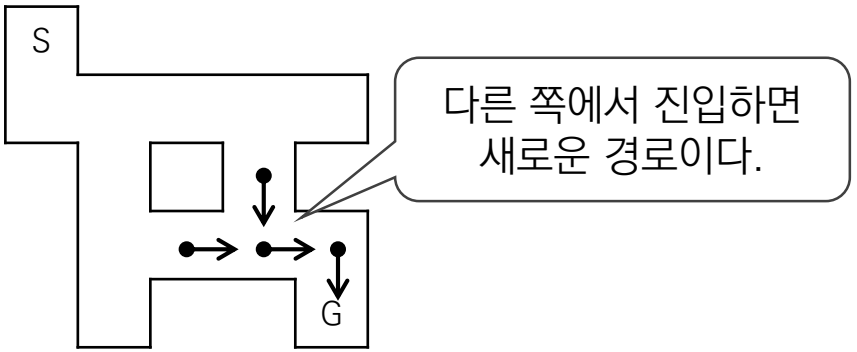
- 한번 지나간 칸은 표시를 해서 다시 들어가지 않는다.
- 일단 목적지에 도착하면 나머지 경로는 확인하지 않는다.

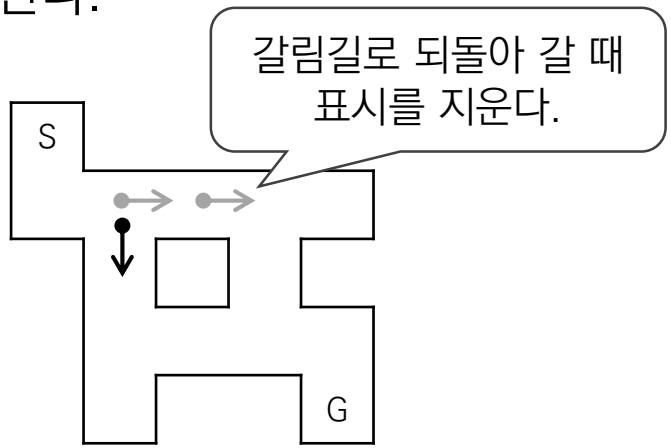



```
map[r][c] = 1; // 현재 위치를 벽으로 변경
for i : 1 -> 4
    //새 좌표 계산
    …
    next( nr, nc );
```

▪ 경로의 수를 찾는 경우.

- 목적지에 도착 가능한 모든 경로를 지나야 한다.

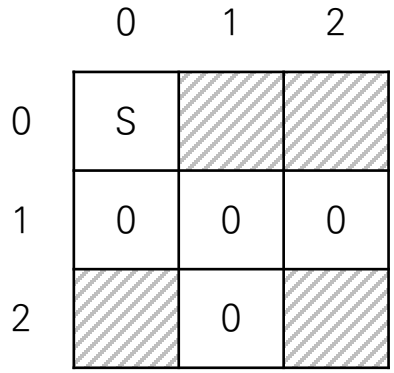


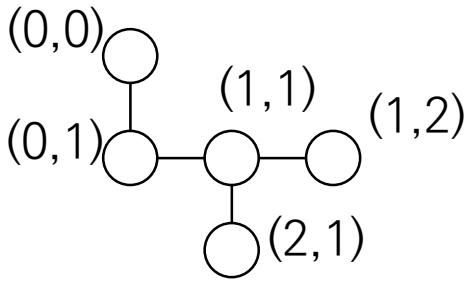



```
map[r][c] = 1; // 현재 위치를 벽으로 표시
for i : 1 -> 4
    //새 좌표 계산
    …
    next( nr, nc );
map[r][c] = 0; // 되돌아 가기 전에 표시 지움
```

방문 표시가 없으면 무한 반복의 위험이 있다.

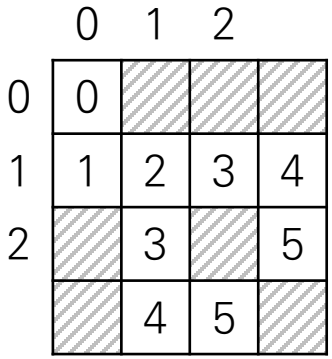
- 미로와 그래프.





상하좌우만 인접할 수 있기때문에 인접행렬이 필요 없음.

- BFS를 적용한 최단 거리 계산



시작 위치부터 거리가 같은 곳을 찾거나 최단 거리를 찾을 수 있다. 미로에 직접 표시하는 대신 별도의 배열에 거리를 기록한다.

- NxN 미로에 대한 BFS 적용
  - map[][] : 미로, 1 : 벽

```
BFS(r, c)
    dr[] = { 0 , 1, 0, -1 }
    dc[] = { 1, 0, -1, 0 }
    enQ(r, c)                    // 시작 좌표 enqueue
    v[r][c] = 1                  // 방문 표시
    while( Q_is_not_empty())
        t = deQ()
        for i : 0 -> 3
            r = t.r + dr[i]
            c = t.c + dc[i]
            if( r>=0 && r<N && c>=0 && c<N )
                if( map[r][c] != 1 && v[r][c] == 0 )
                    enQ(r, c)
                    v[r][c] = v[t.r][t.c] + 1
```

✓ 좌표를 큐에 저장하기.

배열

```
// enqueue
rear++;
qr[rear] = row;
qc[rear] = col;

// dequeue
front++;
row = qr[front];
col = qc[front];
```

Java Point Class

```
import java.awt.Point;

Queue<Point> q = new LinkedList<>();

q.add(new Point(row, col));  // enqueue

Point p = q.poll();                // dequeue

row = p.x;
col = p.y;
```

연습

- 숫자가 적혀 있는 2차원 배열이 있다. 이웃한 칸으로 움직여 1 2 3 4 5 6 3 이란 수열을 찾을 수 있으면 1, 없으면 0을 출력하라. 배열안의 숫자는 1번씩만 사용할 수 있고, 대각선으로는 이동할 수 없다.

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

수열을 찾을 수 있는 경우

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

수열을 찾을 수 없는 경우

▪ 방문 표시 처리

- 다음 경우, 다른 경로를 찾을 때는 방문 표시를 지워야 한다.

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

수열을 찾을 수 있는 경우