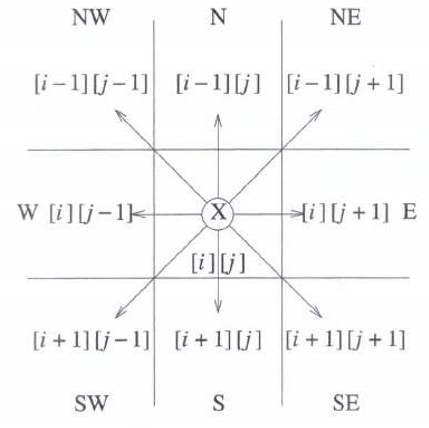
# 3.7 미로 찾기

미로에서 mouse가 입구에서 출발하여 출구로 나올 수 길을 찾는 문제[1]이다. 다음 행렬에서 mouse는 0이면 지나갈 수가 있고 1이면 지나가지 못한다. 그리고 행렬 밖으로는 나가지 못한다. 미로 행렬을 2차원 배열 maze[i][j]로 표현할 때 1<= i < m, 1<= j <= p 이고 blocked path은 1이고 지나갈 수 있는 경로는 0이다. i= 0 또는 i = m+1, j = 0 또는 j = p+1은 미로 밖으로 나갈 수 없다. maze[1][1]에서 시작하며, exit는 maze[m][p]에서 종료된다.



mouse의 현재 위치가 [i][j]일 때 이동할 수 있는 경로는 다음과 같이 8가지가 있다. 8가지는 north, northeast, east, southeast, south, southwest, west, northwest이고 각각을 N, NE, E, SE, S, SW, W, NW로 표기한다. 0 ~ 7 사이의 이동 방향을 숫자 0 ~ 7 대신에 “enum directions {N, NE, E, SE, S, SW, W, NW};”을 사용한다.

mouse의 현재 위치 maze[i][j]가 미로의 가장자리에 있을 때, 즉 i = 1 or m, j = 1 or p 일 때는 가장자리 밖으로 이동할 수가 없다. 이러한 border를 검사하는 if 문 사용을 줄이기 위해(소스 코드의 line의 숫자를 줄이는 것이 가장 잘하는 코딩이다) 미로를 maze[m+2][p+2]으로 선언한다. border를 모두 1로 설정하기 위하여 i = 0 or m+1, j = 0 or p+1에 대하여 “m[i][j] = 1;”를 초기화한다.

현재 위치에서 이동 가능한 8가지 방향은 “enum directions q;”로 선언되고 0 ~ 7의 값을 갖게 된다. mouse의 현재 위치가 maze[i][j]라면 이동후의 위치 maze[g][h]는 다음과 같은 문장으로 표현된다.

struct offsets {

int a, b;

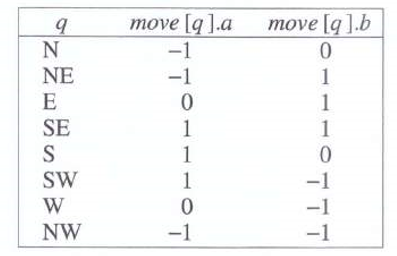
}

offsets move[8];

//q는 0 ~ 7 값을 가짐

g = i + move[q].a; h = j + move[q].b;

q의 값에 따라 다음 이동 위치 maze[g][h]는 “offsets move[8];”로 설정된 값을 사용하여 변경된다. 배열 move[q]의 값을 사용하여 q의 값으로 다음 위치를 이동한다. mouse가 막다른 길에 도달하면 stack을 이용하여 왔던 길로 되돌아가 가보지 않은 다른 경로를 찾아보아야 한다. 이때 한번 지나간 경로를 다시 선택하는 것을 없애기 위하여 또 다른 배열 mark[m+2][p+2]을 만들고 초기 값은 0으로 하고 지나가면 mark[i][j]을 1로 변경한다.



maze[][]를 사용한 미로 찾기 알고리즘은 현재 위치 maze[i][j]를 stack에 push한 후에 생성하여 다음 위치로 이동한다. 더 이상 경로가 없으면 stack에서 pop하여 되돌아가 가지 않은 다른 경로를 선택하여 미로 찾기를 계속한다. stack에 현재 위치를 push할 때 다음 이동 방향인 dir를 같이 포함한다. 미로 찾기 시작시에 (1,1,N)을 stack에 push한 후에 while() {}에서 pop한 결과를 temp 변수가 갖게 되고 d = temp.dir이 이동할 방향이 된다.

struct items {

int x, y, dir;

}

operator<<() 함수는 stack과 items에 대하여 overload되었다. operator<<이 stack의 private data member를 접근하기 위해서 class Stack의 friend로 선언되었다.

struct items {

int x, y, dir;

};

ostream& operator<<(ostream& os, items& item)

{

return os << item.x << "," << item.y << "," << item.dir;

}

struct offsets {

int a, b;

};

enum directions { N, NE, E, SE, S, SW, W, NW };

offsets moves[8];

int maze[100][100];

int mark[100][100];

void path(int m, int p)

// Output a path (if any) in the maze; rows is m, cols is p;

# initialize stack to the maze entrance coordinates and direction east;

# while (stack is not empty)

# {

# (i,j,dir) = coordinates and direction deleted from top of stack;

# while (there are more moves)

# {

# (g,h) = coordinates of next move;

# if ((g == m) && (h == p)) success;

# if ((!maze[g][h]) //legal move

# && (!mark[g][h]) //haven't been here before

# {

# mark[g][h] = 1;

# dir = next direction to try;

# add (i,j,dir) to top of stack;

# i = g; j = h; dir = N;

# }

# }

# }

# cout << "No path found" << endl;

}

void main() {

int input[12][15] = {

{ 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1 },

{ 1, 0, 0, 0, 1, 1, 0, 1, 1, 1, 0, 0, 1, 1, 1 },

{ 0, 1, 1, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 1, 1 },

{ 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 0, 0 },

{ 1, 1, 0, 1, 0, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1 },

{ 0, 0, 1, 1, 0, 1, 1, 1, 0, 1, 0, 0, 1, 0, 1 },

{ 0, 0, 1, 1, 0, 1, 1, 1, 0, 1, 0, 0, 1, 0, 1 },

{ 0, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1 },

{ 0, 0, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1 },

{ 1, 1, 0, 0, 0, 1, 1, 0, 1, 1, 0, 0, 0, 0, 0 },

{ 0, 0, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 0 },

{ 0, 1, 0, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0 },

};

moves[N].a = -1; moves[N].b = 0;

moves[NE].a = -1; moves[NE].b = 1;

moves[E].a = 0; moves[E].b = 1;

moves[SE].a = 1; moves[SE].b = 1;

moves[S].a = 1; moves[S].b = 0;

moves[SW].a = 1; moves[SW].b = -1;

moves[W].a = 0; moves[W].b = -1;

moves[NW].a = -1; moves[NW].b = -1;

path(12, 15);

int end;

cin >> end;

}