セル [0,0] からセル [*n*,*m*] への道を求めるには, セル [*n*,*m*] から隣接するセルを順に調べ, 各セルに付与された番号 *Maze*[*i*,*j*] が 1 づつ小さくなるように辿ることができればよい. そのため, まず, **1-2-1°** の操作を下記のように変更し, *Maze*[*n*,*m*] に番号を付与すると共に, セルを逆に辿るための手続き *Backtrack* を呼ぶことにする.

```
    {
0-1°        セル [0,0] をスタック S に入れる ;
0-2°        Maze[0,0] := 1 ;
1°  while ( スタック S が空でない ) {
1-1°              スタック S からセルを取り出す. そのセルが [i,j] であったとする ;
1-2°              for ( セル [i,j] に隣接する各セル [ii,jj] ) {
1-2-1°            if ( ii = n  かつ  jj = m ) {
                        Maze[n,m] := Maze[i,j] + 1 ;
                        Backtrack ;
                        return true ;
                  }
1-2-2°            else if ( Maze[ii,jj] = 0 ) {
1-2-2-1°                  セル [ii,jj] をスタック S に入れる ;
1-2-2-2°                  Maze[ii,jj] := Maze[i,j] + 1 ;
      }   }       }
2°  return false ;
    }
```

Backtrack の操作は, 下記のようにすればよい.

なお, 下記の操作では, セルをリスト *Path* の先頭の position に入れているので, リストを本に書かれたような配列で実現する場合, リストの要素を毎回後ろへ移動させねばならないため, 効率的でなくなる. 従って, 連結リストが適切なデータ構造となる.

しかし, セル [0,0] からセル [*n*,*m*] への道をリストに覚えるのではなく, セル [*n*,*m*] からセル [0,0] への道を覚える (すなわち, セル [0,0] からセル [*n*,*m*] への道上のセルを逆順に覚える) ことにすれば, リストを配列で実現しても効率的になる.

```
void   Backtrack( )
        入力:   Maze[1..n,1..m]  :        整数の 2 次元配列(グローバル変数)
        出力:   Path   :   LIST  ([0,0] 〜 [n,m] への道上のセルのリスト. グローバル変数)
    {
    p  :=  Makenull( Path ) ;                    /*  Path を空リストにする  */
    Insert( [n,m], First( Path ), Path ) ;       /*  セル [n,m] を Path に入れる  */
    セル [n,m] をセル [i,j] とする ;             /*  i  :=  n,    j  :=  m  */
    do                        /*  セル [0,0] に到達するまで下記を繰返す  */
        found  :=  false  ;          /*  番号の小さなセルが見つかっていない  */
        k  :=  0  ;          /*  隣接セルを順番に調べるためのカウンタ  */
        while (  found  =  false  ) {
            if (  k = 0  ) {
                [ii,jj] を [i,j] の上のセルとする  ;        /*  ii  :=  i−1,   jj  :=  j  ;  */
                k  :=  1  ;
            }
            else  if (  k = 1  ) {
                [ii,jj] を [i,j] の右のセルとする  ;        /*  ii  :=  i  , jj  :=  j + 1  ;  */
                k  :=  2  ;
            }
            else  if (  k = 2  ) {
                [ii,jj] を [i,j] の下のセルとする  ;        /*  ii  :=  i + 1  , jj  :=  j  ;  */
                k  :=  3  ;
            else {          /*  k = 3  */
                [ii,jj] を [i,j] の左のセルとする  ;        /*  ii  :=  i  , jj  :=  j−1  ;  */
            }
            if (  セル [ii,jj] が存在し, かつ  Maze[ii,jj] = Maze[i,j]−1  ) {
                Insert( [ii,jj], First( Path ), Path )  ;
                found  :=  true  ;
                [i,j]  :=  [ii,jj] ;                          /*  i  :=  ii,    j  :=  jj  */
          }     }
    while (  セル [i,j] はセル [0,0] でない  )
}  /*  Backtrack  */
```