最短経路を求めるには（例１　深さ優先探索）(3/3)

疑似コード

```
歩数マップの作成関数( ) {
   ゴール座標のマスの歩数 = 0;
   再帰的探索関数( ゴールのxy座標 );
}
```

```
再帰的探索関数( 対象のマスのxy座標 ) {
   if ( 対象のマスの上に壁がない && 上のマスはまだ通っていない ) {
      上のマスの歩数 = 対象のマスの歩数 + 1;
      再帰的探索関数( 上のマスのxy座標 );
   }
   if ( 対象のマスの右に壁がない && 右のマスはまだ通っていない ) {
      右のマスの歩数 = 対象のマスの歩数 + 1;
      再帰的探索関数( 右のマスのxy座標 );
   }
   if ( 対象のマスの下に壁がない && 下のマスはまだ通っていない ) {
      下のマスの歩数 = 対象のマスの歩数 + 1;
      再帰的探索関数( 下のマスのxy座標 );
   }
   if ( 対象のマスの左に壁がない && 左のマスはまだ通っていない ) {
      左のマスの歩数 = 対象のマスの歩数 + 1;
      再帰的探索関数( 左のマスのxy座標 );
   }
}
```

最短経路を求めるには（例２　幅優先探索）(3/3)

疑似コード

```
歩数マップの作成関数( ) {
  ゴール座標のマスの歩数 = 0;
  for ( 調査歩数 = 0; 調査歩数 < 36; 調査歩数 ++ ) {
    for ( 座標x = 0; 座標x <= 5; 座標x ++ ) {
      for ( 座標y = 0; 座標y <= 5; 座標y ++ ) {
        if ( 座標xyのマスの歩数 == 調査歩数 ) {
          if ( 対象のマスの上に壁がない && 上のマスはまだ通っていない ) {
              上のマスの歩数 = 対象のマスの歩数 + 1;
          }
          if ( 対象のマスの右に壁がない && 右のマスはまだ通っていない ) {
              右のマスの歩数 = 対象のマスの歩数 + 1;
          }
          if ( 対象のマスの下に壁がない && 下のマスはまだ通っていない ) {
              下のマスの歩数 = 対象のマスの歩数 + 1;
          }
          if ( 対象のマスの左に壁がない && 左のマスはまだ通っていない ) {
              左のマスの歩数 = 対象のマスの歩数 + 1;
          }
        }
      }
    }
  }
}
```