Table of Contents

The **All shortest Paths** algorithm was introduced by [Steffan Westcott](#) [1]:

The Algorithm

The following routine determines if a path of set bits in 'path' 8-way connect (ie. king move) any set bit in sq1 with any set bit in sq2 (this is the same as Squares Are Connected). Also, it will return an [array](#) of bitboards showing all possible shortest paths - This is probably best explained by showing an example:

Setting sq1 to contain e4, sq2 to have both a1 and h8, and mask to be ~0 yields this all shortest paths array of bitboards:

```
. . . . . .  . . . . . .    . . . . . .  . . . . . .
.  . . . . . 1
. . . . . .  . . . . . .    . . . . . .  . . . . . 1
1  . . . . . .
. . . . . .  . . . . . .    . . . . 1 1 .  . . . . . .
.  . . . . . .
. . . . . .  . . . 1 1 . .  . . . . . .  . . . . . .
.  . . . . . .
. . . . 1 . .  . . . 1 . . .  . . . . . .  . . . . . .
.  . . . . . .
. . . . . .  . . . 1 . . . .  . . 1 . . . .  . . . . . .
.  . . . . . .
. . . . . .  . . . . . .    . . 1 . . . .  . 1 . . . .
.  . . . . . .
. . . . . .  . . . . . .    . . . . . .  . 1 . . . .
```

```
.   1 . . . . . . .
```

The king needs 4 moves to reach either a1 or h8, so both possibilities are shown in the last bitboard. Also, the king has a choice of squares along the way, meaning there are many shortest paths.

```
////////////////////////////////////////////////////////////////////
///
//
// Returns length of shortest path of set bits present in 'path' that
// 8-way connect any set bit in sq1 to any set bit of sq2. 0 is return
ed
// if no such path exists. Also fills a sequence of bitboards asp[leng
th]
// that describes all such shortest paths.

int allShortestPaths(U64 sq1, U64 sq2, U64 path, U64* asp)
{

// Do an 8-way flood fill with sq1, masking off bits not in path and
    // storing the fill frontier at every step. Stop when fill reaches
    // any set bit in sq2 or quit if fill cannot progress any further.
    // Then do 8-way flood fill from reached bits in sq2, ANDing the
    // frontiers with those from the first fill in reverse order.

    if (!(sq1 &= path) || !(sq2 &= path)) return 0;
                        // Drop bits not in path
                        // Early exit if sq1 or sq2 not on any path
    int i = 1;
    asp[0] = sq1;

    while(1)  // Fill from all set bits in sq1, to any set bit in sq2
    {
        if (sq1 & sq2) break;                       // Found good path
        U64 temp = sq1;
        sq1 |= eastOne(sq1) | westOne(sq1);
// Set all 8 neighbours
        sq1 |= soutOne(sq1) | nortOne(sq1);
        sq1 &= path;
                               // Drop bits not in path
        if (sq1 == temp) return 0;                  // Fill has stopped
        asp[i++] = sq1 & ~temp;
// Store fill frontier
    }

    int length = i;
```

```
                                // Remember path length
    asp[--i] = (sq2 &= sq1);
                      // Drop unreached bits

    while(i)  // Fill from reached bits in sq2
    {
        U64 temp = sq2;
        sq2 |= eastOne(sq2) | westOne(sq2);
// Set all 8 neighbours
        sq2 |= soutOne(sq2) | nortOne(sq2);
        sq2 &= path;
                               // Drop bits not in path
        asp[--i] &= sq2 & ~temp;
// Intersect frontiers
    }
    return length;
}
```

As a reminder - one step only.

So far we've just mentioned 8-way flood filling for king/queen moves, and diagonal or rank/file 4-way filling for bishop or rook moves. A simple variant exists for knight moves too, which is handy for calculating knight mobility, and helping solve "Knight's Tour" type puzzles. Its even useful for pawns too, for quick determination of outposts outside all enemy pawn "attack cones", for example.

A fill iteration need not just go one square in each direction. With appropriate bitboard manipulation, it may extend the full ray of a sliding piece, for example. Examining the filled bitboard for 1 or 2 iterations will enable design of piece mobility terms with some simulated ply.

Optimization

Edmund Moshammer proposed following optimization with an inlined king fill routine [2]:

```
// Set all up to eight neighbours
// inline
U64 fillKing(U64 b) {
    b |= eastOne(b) | westOne(b);
    b |= soutOne(b) | nortOne(b);
    return b;
}

int allShortestPaths(U64 sq1, U64 sq2, U64 path, U64* asp) {
```

```
    int i, length;
    asp[0] = sq1;
    for (i=0; !(asp[i] & sq2); i++)
        if (!(asp[i+1] = fillKing(asp[i]) & path &~asp[i])) return 0;
    length = i;
    asp[i] = sq2;
    for (; i; i--)
       asp[i-1] &= fillKing(asp[i]) & path;
    return length;
}
```

External Links

- Pathfinding from Wikipedia
- Remember Shakti [3], Finding the way, part 1, YouTube Video
  Zakir Hussain, U. Srinivas, John McLaughlin, V. Selvaganesh

References

1. ^ All shortest paths & other flood-fill based algorithms by Steffan Westcott, CCC, September 13, 2002
2. ^ Discussion about this page
3. ^ Shakti from Wikipedia

What links here?

Page	Date Edited
Algorithms	May 5, 2017
All Shortest Paths	Jan 21, 2014
Edmund Moshammer	Sep 23, 2017
Fill Algorithms	Nov 6, 2013
Helmut Horacek	Oct 31, 2016
King Pattern	Nov 15, 2013
Mobility	Jan 17, 2018
PawnKing	Sep 5, 2014
Steffan Westcott	Jan 21, 2014
Trajectory	Jan 3, 2015

Up one Level