Dumb7Fill (/Dumb7Fill)

Table of Contents

Dumb7Fill - the obvious, straight forward flood-fill approach works **set-wise** - **seven** times one fill-cycle by one step only in one of eight directions. We rely on the compass rose to identify ray-directions.

```
  noWe         nort         noEa
          +7    +8    +9
              \  |  /
  west    -1 <-  0 -> +1    east
              /  |  \
          -9    -8    -7
  soWe         sout         soEa
```

Code samples and bitboard diagrams rely on Little endian file and rank mapping.

Occluded Fill

An occluded fill includes the flood generating sliding pieces, but excludes the blocker. It is base of attack fills. One additional direction shift excludes the sliders but includes the blocker.

Seven Fill Cycles

The sliding pieces generate the flood. They were shifted one step in the desired direction and intersected with the set of empty squares, the propagator. The flood aggregates the intersection and the cycle repeats six further times to cover the maximum distance on the chessboard. A blocker, not member of propagator, stops the flood in that particular ray-direction for one sliding piece. Therefor occluded fill contains the initial generator but excludes any blocker.

```
U64 soutOccl(U64 gen, U64 pro) {
   for (int cycle = 0; cycle < 7; cycle++)
      gen |= pro & (gen >> 8);
   return gen;
}
```

Fill Loop

Alternatively one may save move-instructions by introducing an explicit flood accumulator, and to probably terminate the loop early if the flood stops:

```
U64 soutOccl(U64 gen, U64 pro) {
   U64 flood = 0;
   while (gen) {
      flood |= gen;
      gen = (gen >> 8) & pro;
   }
   return flood;
}
```

Unrolled Loop

To avoid conditional branches and to schedule several directions in parallel, the "real" dumb7fill unrolls the loop:

```
U64 soutOccl(U64 gen, U64 pro) {
   U64 flood = gen;
   flood |= gen = (gen >> 8) & pro;
   flood |= gen = (gen >> 8) & pro;
   flood |= gen = (gen >> 8) & pro;
   flood |= gen = (gen >> 8) & pro;
   flood |= gen = (gen >> 8) & pro;
   flood |= gen = (gen >> 8) & pro;
   flood |=       (gen >> 8) & pro;
   return flood;
}
```

Some south fill cycles in slow motion:

```
flood = gen =
brooks|bqueen      empty
1 . . . . . . .    . 1 1 1 1 1 . 1
. . . 1 . . . .    1 1 . . 1 . 1 .
. . . . . . . .    1 . 1 1 1 1 . 1
. . . . . . . .    1 1 . 1 1 1 1 1
. . . . . 1 . .    1 1 1 . 1 . 1 1
. . . . . . . .    1 1 . 1 1 1 1 .
. . . . . . . .    . . . 1 1 . . 1
. . . . . . . .    . 1 1 . 1 1 . 1
1.fill
 gen = gen >> 8   &  empty           =>  flood
. . . . . . . .     . . . . . . . .     1 . . . . . . .
1 . . . . . . .     1 . . . . . . .     1 . . 1 . . . .
. . . 1 . . . .     . . . 1 . . . .     . . . 1 . . . .
. . . . . . . .     . . . . . . . .     . . . . . . . .
. . . . . . . .     . . . . . . . .     . . . . . 1 . .
. . . . . 1 . .     . . . . . 1 . .     . . . . . 1 . .
. . . . . . . .     . . . . . . . .     . . . . . . . .
. . . . . . . .     . . . . . . . .     . . . . . . . .
2.fill
 gen = gen >> 8   &  empty               flood  |= ...
. . . . . . . .     . . . . . . . .     1 . . . . . . .
. . . . . . . .     . . . . . . . .     1 . . 1 . . . .
1 . . . . . . .     1 . . . . . . .     1 . . 1 . . . .
. . . 1 . . . .     . . . 1 . . . .     . . . 1 . . . .
. . . . . . . .     . . . . . . . .     . . . . . 1 . .
. . . . . . . .     . . . . . . . .     . . . . . 1 . .
. . . . . 1 . .     . . . . . 0 . .     . . . . . . . .
. . . . . . . .     . . . . . . . .     . . . . . . . .
3.fill
 gen = gen >> 8   &  empty               flood |= ...
. . . . . . . .     . . . . . . . .     1 . . . . . . .
. . . . . . . .     . . . . . . . .     1 . . 1 . . . .
. . . . . . . .     . . . . . . . .     1 . . 1 . . . .
1 . . . . . . .     1 . . . . . . .     1 . . 1 . . . .
. . . 1 . . . .     . . . 0 . . . .     . . . . . 1 . .
. . . . . . . .     . . . . . . . .     . . . . . 1 . .
. . . . . . . .     . . . . . . . .     . . . . . . . .
. . . . . . . .     . . . . . . . .     . . . . . . . .

...

6.fill
 gen = gen >> 8  &   empty               flood  |= ...
. . . . . . . .     . . . . . . . .     1 . . . . . . .
. . . . . . . .     . . . . . . . .     1 . . 1 . . . .
. . . . . . . .     . . . . . . . .     1 . . 1 . . . .
. . . . . . . .     . . . . . . . .     1 . . 1 . . . .
. . . . . . . .     . . . . . . . .     1 . . . . 1 . .
. . . . . . . .     . . . . . . . .     1 . . . . 1 . .
1 . . . . . . .     0 . . . . . . .     . . . . . . . .
. . . . . . . .     . . . . . . . .     . . . . . . . .
```

The flood already stopped, the final 7th fill cycles don't change anything.

Attack Fill

To get attacks, one additional direction shift of the occluded fill is necessary to exclude the rooks/queen and include the blocker:

```
U64 soutAttacks (U64 rooks, U64 empty) {return soutOne(soutOccl(rooks, empty));}
```

Comparison with Kogge-Stone

To combine the dumb7fill as attack getter, we also can take advantage of outer square occupancy doesn't affect the attack set. We need one fill cycle less, before the final shift. That takes 19 operations. In anticipation to parallel prefix, a Kogge-Stone approach takes 14 instructions, 5 less. Kogge-Stone needs more move-instructions and a temporary register to compute generator as well as propagator, while dumb7fill uses a const propagator:

```
dumb7fill                                  | Kogge-Stone Algorithm
                                           |
U64 soutAttacks(U64 rooks, U64 empty) {    | U64 soutAttacks(U64 rooks, U64 empty) {
   U64 flood = rooks;                      |    rooks |= empty & (rooks >>  8);
   flood |= rooks = (rooks >> 8) & empty;  |    empty  = empty & (empty >>  8);
   flood |= rooks = (rooks >> 8) & empty;  |    rooks |= empty & (rooks >> 16);
   flood |= rooks = (rooks >> 8) & empty;  |    empty  = empty & (empty >> 16);
   flood |= rooks = (rooks >> 8) & empty;  |    rooks |= empty & (rooks >> 32);
   flood |= rooks = (rooks >> 8) & empty;  |    return            rooks >>  8;
   flood |=         (rooks >> 8) & empty;  | }
   return            flood >> 8;           |
}                                          |
                                           |
in x86/64 assembly                         | in x86/64 assembly
; dumb7fill                                |  ; koggeStone
; rooks rdx                                |  ; rooks rdx
; empty rcx                                |  ; empty rcx
                                           |
  mov   rax, rdx                           |    mov   rax, rdx
                                           |    shr   rax, 8
  shr   rdx, 8                             |    and   rax, rcx
  and   rdx, rcx                           |    or    rdx, rax
  or    rax, rdx                           |
                                           |    mov   rax, rcx
  shr   rdx, 8                             |    shr   rax, 8
  and   rdx, rcx                           |    and   rcx, rax
  or    rax, rdx                           |
                                           |    mov   rax, rdx
  shr   rdx, 8                             |    shr   rax, 16
  and   rdx, rcx                           |    and   rax, rcx
  or    rax, rdx                           |    or    rdx, rax
                                           |
  shr   rdx, 8                             |    mov   rax, rcx
  and   rdx, rcx                           |    shr   rax, 16
  or    rax, rdx                           |    and   rcx, rax
                                           |
  shr   rdx, 8                             |    mov   rax, rdx
  and   rdx, rcx                           |    shr   rax, 32
  or    rax, rdx                           |    and   rax, rcx
                                           |    or    rax, rdx
  shr   rdx, 8                             |
  and   rdx, rcx                           |    shr   rax, 8
  or    rax, rdx                           |
                                           |
  shr   rax, 8                             |
                                           |
   1 move                                  |     5 moves
  19 operations                            |    14 operations
  20 instructions                          |    19 instructions
```

Thus, dumb7fill is not that bad, specially if processing several directions in parallel, like south and north, all east and all west attacks.

All Directions

```
U64 soutAttacks(U64 rooks, U64 empty) {
   U64 flood = rooks;
   flood |= rooks = (rooks >> 8) & empty;
   flood |= rooks = (rooks >> 8) & empty;
   flood |= rooks = (rooks >> 8) & empty;
   flood |= rooks = (rooks >> 8) & empty;
   flood |= rooks = (rooks >> 8) & empty;
   flood |=         (rooks >> 8) & empty;
   return            flood >> 8;
}

U64 nortAttacks(U64 rooks, U64 empty) {
   U64 flood = rooks;
   flood |= rooks = (rooks << 8) & empty;
   flood |= rooks = (rooks << 8) & empty;
   flood |= rooks = (rooks << 8) & empty;
   flood |= rooks = (rooks << 8) & empty;
   flood |= rooks = (rooks << 8) & empty;
   flood |=         (rooks << 8) & empty;
   return            flood << 8;
}
```

Horizontal fills need to consider wraps from H-file to A-file and vice versa. Fortunately this can be combined by intersection of ~A-file or ~H-file with the propagator:

```
U64 eastAttacks(U64 rooks, U64 empty) {
   const U64 notA = C64(0xfefefefefefefefe);
   U64 flood = rooks;
   empty &= notA;
   flood |= rooks = (rooks << 1) & empty;
   flood |= rooks = (rooks << 1) & empty;
   flood |= rooks = (rooks << 1) & empty;
   flood |= rooks = (rooks << 1) & empty;
   flood |= rooks = (rooks << 1) & empty;
   flood |=         (rooks << 1) & empty;
   return           (flood << 1) & notA ;
}

U64 noEaAttacks(U64 bishops, U64 empty) {
   const U64 notA = C64(0xfefefefefefefefe);
   U64 flood = bishops;
   empty &= notA;
   flood |= bishops = (bishops << 9) & empty;
   flood |= bishops = (bishops << 9) & empty;
   flood |= bishops = (bishops << 9) & empty;
   flood |= bishops = (bishops << 9) & empty;
   flood |= bishops = (bishops << 9) & empty;
   flood |=           (bishops << 9) & empty;
   return               (flood << 9) & notA ;
}

U64 soEaAttacks(U64 bishops, U64 empty) {
   const U64 notA = C64(0xfefefefefefefefe);
   U64 flood = bishops;
   empty &= notA;
   flood |= bishops = (bishops >> 7) & empty;
   flood |= bishops = (bishops >> 7) & empty;
   flood |= bishops = (bishops >> 7) & empty;
   flood |= bishops = (bishops >> 7) & empty;
   flood |= bishops = (bishops >> 7) & empty;
   flood |=           (bishops >> 7) & empty;
   return               (flood >> 7) & notA ;
}

U64 westAttacks(U64 rooks, U64 empty) {
   const U64 notH = C64(0x7f7f7f7f7f7f7f7f);
   U64 flood = rooks;
   empty &= notH;
   flood |= rooks = (rooks >> 1) & empty;
   flood |= rooks = (rooks >> 1) & empty;
   flood |= rooks = (rooks >> 1) & empty;
   flood |= rooks = (rooks >> 1) & empty;
   flood |= rooks = (rooks >> 1) & empty;
   flood |=         (rooks >> 1) & empty;
   return           (flood >> 1) & notH ;
}

U64 soWeAttacks(U64 bishops, U64 empty) {
   const U64 notH = C64(0x7f7f7f7f7f7f7f7f);
   U64 flood = bishops;
   empty &= notH;
   flood |= bishops = (bishops >> 9) & empty;
   flood |= bishops = (bishops >> 9) & empty;
   flood |= bishops = (bishops >> 9) & empty;
   flood |= bishops = (bishops >> 9) & empty;
   flood |= bishops = (bishops >> 9) & empty;
   flood |=           (bishops >> 9) & empty;
   return               (flood >> 9) & notH ;
}

U64 noWeAttacks(U64 bishops, U64 empty) {
   const U64 notH = C64(0x7f7f7f7f7f7f7f7f);
   U64 flood = bishops;
   empty &= notH;
   flood |= bishops = (bishops << 7) & empty;
   flood |= bishops = (bishops << 7) & empty;
   flood |= bishops = (bishops << 7) & empty;
   flood |= bishops = (bishops << 7) & empty;
   flood |= bishops = (bishops << 7) & empty;
   flood |=           (bishops << 7) & empty;
   return               (flood << 7) & notH ;
}
```

Generalized Rays

Since rotate works like a generalized shift with positive or negative shift amount - since it internally applies a modulo 64 and makes -i = 64-i. We need to clear either the lower or upper bits by intersection with a mask, which might be combined with the wrap-ands for one step. It might be applied to get attacks for both sides with a direction parameter and small lookups for shift amount and wrap-ands - instead of multiple code for eight directions. Of course generalized shift will be a bit slower due to lookups and using cl as the shift amount register.

Loop Version

This is the loop-version:

```
U64 occludedFill (U64 gen, U64 pro, int dir8) {
   U64 flood = 0;
   if (gen) {
      int r = shift[dir8]; // {+-1,7,8,9}
      pro  &= avoidWrap[dir8];
      do {
         flood |= gen;
         gen = rotateLeft(gen, r) & pro;
      } while (gen);
   }
   return flood;
}

U64 shiftOne (U64 b, int dir8) {
   int r = shift[dir8]; // {+-1,7,8,9}
   return rotateLeft(b, r) & avoidWrap[dir8];
}

U64 slidingAttacks (U64 sliders, U64 empty, int dir8) {
   U64 fill = occludedFill(slider, empty, dir8)
   return shiftOne(fill, dir8);
}

// positve left, negative right shifts
int shift[8] = {9, 1,-7,-8,-9,-1, 7, 8};

U64 avoidWrap[8] =
{
   0xfefefefefefefe00,
   0xfefefefefefefefe,
   0x00fefefefefefefe,
   0x00ffffffffffffff,
   0x007f7f7f7f7f7f7f,
   0x7f7f7f7f7f7f7f7f,
   0x7f7f7f7f7f7f7f00,
   0xffffffffffffff00,
};
```

The avoidWrap masks by some arbitrary dir8 enumeration and shift amount:

```
6 == noWe -> +7     7 == nort -> +8     0 == noEa -> +9
0x7F7F7F7F7F7F7F00  0xFFFFFFFFFFFFFF00  0xFEFEFEFEFEFEFE00
1 1 1 1 1 1 1 .     1 1 1 1 1 1 1 1     . 1 1 1 1 1 1 1
1 1 1 1 1 1 1 .     1 1 1 1 1 1 1 1     . 1 1 1 1 1 1 1
1 1 1 1 1 1 1 .     1 1 1 1 1 1 1 1     . 1 1 1 1 1 1 1
1 1 1 1 1 1 1 .     1 1 1 1 1 1 1 1     . 1 1 1 1 1 1 1
1 1 1 1 1 1 1 .     1 1 1 1 1 1 1 1     . 1 1 1 1 1 1 1
1 1 1 1 1 1 1 .     1 1 1 1 1 1 1 1     . 1 1 1 1 1 1 1
1 1 1 1 1 1 1 .     1 1 1 1 1 1 1 1     . 1 1 1 1 1 1 1
. . . . . . . .     . . . . . . . .     . . . . . . . .

5 == west -> -1                         1 == east -> +1
0x7F7F7F7F7F7F7F7F                      0xFEFEFEFEFEFEFEFE
1 1 1 1 1 1 1 .                         . 1 1 1 1 1 1 1
1 1 1 1 1 1 1 .                         . 1 1 1 1 1 1 1
1 1 1 1 1 1 1 .                         . 1 1 1 1 1 1 1
1 1 1 1 1 1 1 .                         . 1 1 1 1 1 1 1
1 1 1 1 1 1 1 .                         . 1 1 1 1 1 1 1
1 1 1 1 1 1 1 .                         . 1 1 1 1 1 1 1
1 1 1 1 1 1 1 .                         . 1 1 1 1 1 1 1
1 1 1 1 1 1 1 .                         . 1 1 1 1 1 1 1

4 == soWe -> -9     3 == sout -> -8     2 == soEa -> -7
0x007F7F7F7F7F7F7F  0x00FFFFFFFFFFFFFF  0x00FEFEFEFEFEFEFE
. . . . . . . .     . . . . . . . .     . . . . . . . .
1 1 1 1 1 1 1 .     1 1 1 1 1 1 1 1     . 1 1 1 1 1 1 1
1 1 1 1 1 1 1 .     1 1 1 1 1 1 1 1     . 1 1 1 1 1 1 1
1 1 1 1 1 1 1 .     1 1 1 1 1 1 1 1     . 1 1 1 1 1 1 1
1 1 1 1 1 1 1 .     1 1 1 1 1 1 1 1     . 1 1 1 1 1 1 1
1 1 1 1 1 1 1 .     1 1 1 1 1 1 1 1     . 1 1 1 1 1 1 1
1 1 1 1 1 1 1 .     1 1 1 1 1 1 1 1     . 1 1 1 1 1 1 1
1 1 1 1 1 1 1 .     1 1 1 1 1 1 1 1     . 1 1 1 1 1 1 1
```

Unrolled Attacks

The generalized unrolled sliding attack getter:

```
U64 slidingAttacks (U64 sliders, U64 empty, int dir8) {
   U64 flood = sliders;
   int r = shift[dir8]; // {+-1,7,8,9}
   empty &= avoidWrap[dir8];
   flood |= sliders = rotateLeft(sliders , r) & empty;
   flood |= sliders = rotateLeft(sliders , r) & empty;
   flood |= sliders = rotateLeft(sliders , r) & empty;
   flood |= sliders = rotateLeft(sliders , r) & empty;
   flood |= sliders = rotateLeft(sliders , r) & empty;
   flood |=         = rotateLeft(sliders , r) & empty;
   return   rotateLeft(flood, r)  &   avoidWrap[dir8];
}
```

See also

- AVX2 Dumb7Fill
- Fill Algorithms
- Kogge-Stone Algorithm
- Pieces versus Directions

External Links

- bitboard mobility   Copyright (c) 2003, Gunnar Andersson » Othello, Mobility

- [Weather Report](#) - [Seventh Arrow / Umbrellas, 1971](#) , [YouTube](#) Video
  [Joe Zawinul](#), [Wayne Shorter](#), [Miroslav Vitouš](#), [Alphonse Mouzon](#), [Dom Um Romão](#)

What links here?

Page	Date Edited
AVX2	Aug 8, 2017
DirGolem	Jun 5, 2016
Dumb7Fill	May 27, 2016
Efficient Generation of Sliding Piece Attacks	Nov 5, 2016
Fill Algorithms	Nov 6, 2013
Gaviota	Jan 21, 2018
General Setwise Operations	Feb 25, 2018
Gunnar Andersson	Jun 23, 2011
Influence Quantity of Pieces	Jul 21, 2017
John L. Jerz	Jul 15, 2013
Kogge-Stone Algorithm	Sep 17, 2016
Lachex	Jan 7, 2016
Little Wing	Oct 26, 2017
Mobility	Jan 17, 2018
Othello	Jan 4, 2018
Pieces versus Directions	Oct 6, 2016
Sequential Logic	May 10, 2017
Shifted Bitboards	Jan 9, 2011
SIMD and SWAR Techniques	Jun 27, 2017
Sliding Piece Attacks	May 27, 2016
More Links	

Up one Level