

```

#include <stdio.h>
#include <thread>
using namespace std;
#include <signal.h>
#include <unistd.h>

#ifdef SIZE
#define SIZE 7
#endif

#define MTHREAD

#define BIT(a) (1ULL<<(a))
typedef unsigned long long int ULL;

ULL mask[SIZE*SIZE];

// 盤面の位置は 左下から右上まで 0 以上 SIZE*SIZE 未満の自然数で管理.
// しかし実際は, 例えばゴール位置は 1ULL<<((SIZE-1)*SIZE+(SIZE-1)) のように
// 対応するビットを用いている.

#define FULL      (BIT(SIZE*SIZE)-1ULL)      // 盤面全部埋まった状態
#define end_flag  BIT((SIZE+1)*(SIZE-1))      // ゴール
const ULL nwcorner_flag = BIT(SIZE*(SIZE-1)); // 左上隅
const ULL secorner_flag = BIT(SIZE-1);        // 右下隅
#define nw_1_flag BIT((SIZE-2)*SIZE+2)
#define nw_2_flag BIT((SIZE-3)*SIZE+1)
#define ne_1_flag BIT((SIZE-2)*SIZE+(SIZE-3))
#define ne_2_flag BIT((SIZE-3)*SIZE+(SIZE-2))
#define se_1_flag BIT(1*SIZE+(SIZE-3))
#define se_2_flag BIT(2*SIZE+(SIZE-2))

// bit_to_mask ( 1<<p ) = ( p 番のマス目から 1 段階で進めるマス目の一覧)
// x86_64 ではこれが実質的に bsfq という CPU の1 命令で済んでしまう!
#ifdef __GNUC__ // GCC

```

```

ULL inline bit_to_mask(const ULL a) {
    return mask [__builtin_ctzll(a)];
}
#else
#ifdef _MSC_VER // Visual Studio (未確認)
#include <intrin.h>
#pragma intrinsic(_BitScanReverse)
ULL inline bit_to_mask(const ULL a){
    int x;
    _BitScanReverse64(&x,a);
    return mask[x];
}
#endif
#endif

```

```

// 最下位ビット
#define rmb(a) ((a)&-(a))
// a (>0) は 1 ビットしか立っていないか?
#define is_2power(a) (rmb(a)==(a))
// while(x) { xr = rmb(x); ... x-=xr; } で
// x の最下位ビットから 1 ビットずつ xr に取り出してループできる

```

```

void main_loop(const ULL p, const ULL state, ULL * const sum);

```

```

// pos: x から 1 回で進めるマス一覧. pos_rmb: その最上位ビット
// pos==pos_rmb, つまり x から進めるマスが 1 つだけなら, そこへ飛ぶ
// (main_loop を呼び出すのではなく, 末尾再帰している)
// そうでなければ,
// * x から進める各マス a で, 「a から進めるマスが 1 つしかない」ものを数える
// そのような a の位置は, next_p に格納される.
// * そういう a が 1 つしかない ( is_2power(next_p) )なら a の唯一の進行先へ
// * a が存在しない場合は, 素直に x から進める位置を探していく
// (高々 8 通りなので, 手動でループ展開しています)
// * それ以外の場合は, ここで終わり

```

// 「そうでなければ」以降の枝刈方法は長尾さんのプログラムと同様のものです.

```
#define loop_aux(x) { \
    ULL pos = bit_to_mask(x) & ~state; \
    if (pos) { \
        ULL pos_rmb = rmb(pos); \
        if (is_2power(pos)) { p = pos; state|= pos; goto BEGIN; } \
        else { \
            ULL pos2 = pos; ULL next_p = 0; ULL next_next_p, temp; \
            while (pos2) { \
                temp = bit_to_mask(rmb(pos2)) & ~(state|rmb(pos2)); \
                if ( temp && is_2power(temp) ) { next_p|=rmb(pos2); next_next_p=temp; } \
                pos2-=rmb(pos2); \
            } \
            if ( !next_p ) { \
                main_loop(pos_rmb, state|pos_rmb, sum); pos-=pos_rmb; \
                inn_loop(); inn_loop(); inn_loop(); inn_loop(); \
                inn_loop(); inn_loop(); p = pos; state|= pos; goto BEGIN; \
            } else if (is_2power(next_p)) { \
                p = next_next_p; state|= next_p|next_next_p; goto BEGIN; \
            } \
        } \
    } \
} \
/*
*/
#define inn_loop() \
{ const ULL pos_rmb = rmb(pos); \
  main_loop(pos_rmb, state|pos_rmb, sum); \
  pos-=pos_rmb; } \
if(!pos) return;
```

// 角周囲の処理. c: 角, x, q: 角に 1 回で行ける 2 マス (現在位置は x) .

// もし c に未到達で, q も未到達ならば x -> c -> q と飛ぶしかない.

// もし c に未到達で q に到達済みなら, x -> c で止まってしまうのでカット

// もし c に到達済みなら, 再帰に回す

```
#define corner_aux(x, q, c) {\
    if ( !(state&c) ) { if ( !(state&q) ) { p=q; state|=q|c; goto BEGIN; } } \
    else loop_aux(x); \
}

void main_loop(ULL p, ULL state, ULL * const sum) {
    BEGIN:
        switch (p) {
        #if (SIZE >= 6)
            case ne_1_flag:
                if ( state&ne_2_flag ) { if (state==FULL-end_flag) ++(*sum); }
                else loop_aux(ne_1_flag);
                break;
            case ne_2_flag:
                if ( state&ne_1_flag ) { if (state==FULL-end_flag) ++(*sum); }
                else loop_aux(ne_2_flag);
                break;
        #else // SIZE == 5
            case end_flag:
                if (state==FULL) ++(*sum); break;
        #endif
            case nw_1_flag:
                corner_aux(nw_1_flag, nw_2_flag, nwcorner_flag); break;
            case nw_2_flag:
                corner_aux(nw_2_flag, nw_1_flag, nwcorner_flag); break;
            case se_1_flag:
                corner_aux(se_1_flag, se_2_flag, secorner_flag); break;
            case se_2_flag:
                corner_aux(se_2_flag, se_1_flag, secorner_flag); break;
            default:
                loop_aux(p);
        }
}
```

```

ULL sum[5];

// SIGINT (C-c), SIGTERM (kill) への割り込み：それまでの合計を表示
static void sigint_handler(int sig) {
    ULL gt = 0;
    for (int i=0;i<5;i++) gt += sum[i]<<1;
    printf("!_interrupt_(%s)._%lld\n",
           (sig==SIGINT)?"SIGINT":((sig==SIGTERM)?"SIGTERM":"???"),
           gt);
    exit(sig);
}

#define main_loop_call(a,b,c,d) main_loop(BIT(a+b), c+BIT(a+b), sum+d)

#define setmask(dx, dy) \
    if( ((unsigned)(i+dx)<SIZE)&&((unsigned)(j+dy)<SIZE) ) \
        mask[i+j*SIZE] |= BIT( (i+dx)+SIZE*(j+dy) );

int main(void) {
    signal(SIGINT, sigint_handler);
    signal(SIGTERM, sigint_handler);
    printf("size_%d\n", SIZE);

    { // mask 初期化
        for (int i=0;i<SIZE*SIZE;i++) mask[i]=0;
        for (int i=0;i<SIZE;i++)
            for (int j=0;j<SIZE;j++) {
                setmask(+2,+1); setmask(+2,-1); setmask(-2,+1); setmask(-2,-1);
                setmask(+1,+2); setmask(+1,-2); setmask(-1,+2); setmask(-1,-2);
            }
        for (int i=0;i<SIZE*SIZE;i++) mask[i]&= ~(1ULL+BIT(SIZE+2));
        mask[SIZE*SIZE-1] |= BIT(SIZE*SIZE);
    }
}

```

```

#ifdef MTHREAD
    auto th1 = thread([]{ main_loop_call ( 4,      0, 1ULL+BIT(SIZE+2), 0 );});
    auto th2 = thread([]{ main_loop_call ( 4, 2*SIZE, 1ULL+BIT(SIZE+2), 1 );});
    auto th3 = thread([]{ main_loop_call ( 3, 3*SIZE, 1ULL+BIT(SIZE+2), 2 );});
    auto th4 = thread([]{ main_loop_call ( 1, 3*SIZE, 1ULL+BIT(SIZE+2), 3 );});
    auto th5 = thread([]{ main_loop_call ( 0, 2*SIZE, 1ULL+BIT(SIZE+2), 4 );});
    th1.join(); th2.join(); th3.join(); th4.join(); th5.join();
#else
    main_loop_call ( 2, SIZE, 1ULL, 0 );
#endif

{ // 合計表示
    ULL gt = 0;
    for (int i=0;i<5;i++) {
        gt += sum[i]<<1;
        printf("%c:_sum_%lld\n", 'A'+i, sum[i]<<1);
    }
    printf("total_%lld\n", gt);
}
}

```