Input File: `hanoi.in` **100 Points**
Output File: `hanoi.out` **Time limit:** 3 s
Source File: `hanoi.pas/.c/.cpp` **Memory limit:** 16 MB

Solution

First subtask: destination pile

The first subtask is quite simple, because it is always optimal to leave disc N (the largest disc) on the pile on which it lies at the beginning. Otherwise, one would have to move disc N source pile to some other pile that must be free at that time. This move would then not change the state of the game (aside from pile renumbering) and thus can't be a part of an optimal solution. From here on, we will denote that destination pile $dest$.

Second subtask: number of moves

A basic algorithmic approach can be derived from a number of simple observations about the problem:

Obs. 1 When moving disc i, the position of all larger discs $N, N-1, \ldots, i+1$ does not matter. In particular, if discs $N, N-1, \ldots, i$ already lie on $dest$, it never makes sense to move any one of them again, because there is no move that would be made possible by them being located on any other pile.

Obs. 2 In order to move disc i to its final location on $dest$, all discs $N, N-1, \ldots, i+1$ must already be present. Therefore, the discs must be moved to there final location in decreasing size.

Obs. 3 If discs $N, N-1, \ldots, i+1$ already lie on $dest$, all discs $i-1, \ldots, 1$ must not lie on the same pile as disc i, and also not on pile $dest$ due to the move rules. Therefore, they must first be moved to the other pile.

All these restrictions directly leed to the following recursive algorithm:

```
1  procedure moveDisc(disc, dstPile):
2    srcPile := pileOfDisc[disc];
3    auxPile := 1 + 2 + 3 - srcPile - dstPile;
4    for otherDisc := disc - 1 downto 1 do begin
5      moveDisc(otherDisc, auxPile);
6    end
7    numMoves := (numMoves + 1) MOD 1000000;
8    pileOfDisc[disc] := dstPile; // direct move
```

```
 9  end
10
11  // main:
12  numMoves := 0;
13  for disc := N-1 downto 1 do begin
14    moveDisc(disc, pileOfDisc[N])
15  end;
// numMoves now contains the solution
```

From the observations above, it should be obvious that this algorithm generates the right order of moves. However, it is also quite apparent that it is not yet fast enough. The number `numMoves` of moves is never increased by more than one, so the procedure `moveDisc` is obviously called once for each move. Because it is widely known that the number of moves in a Hanoi games grows exponentially with $O(2^N)$, the runtime will be at least $O(2^N)$, which is far too much because $N \leq 100.000$.

As most contestants do probably already know and is also suggested by the problem statement, it requires exactly $2^i - 1$ moves to move discs $i, i-1, \ldots 1$ from one pile to another (same piles for all discs). Therefore, instead of carrying out all these $2^i - 1$ moves individually, we can directly move these i discs to the destination pile and increase the number of moves by $2^i - 1$.

When we move disc i to a pile X, we do it either to move out of the way for a larger disc or because X is the destination pile, $dest$. In both cases we want to move all smaller discs $i-1, i-2, \ldots, 1$ to X subsequently. In the procedure `moveDisc`, all these smaller discs lie on the same pile after line 6. Because they will be moved to `dstPile` anyway, we can change `moveDisc` to a new procedure `moveDiscs` that moves them, too, and exploits their ordering to directly calculate the number of moves necessary. This leads to the following improved solution:

```
 1  // moves all discs 1, 2, ..., maxDisc to dstPile
 2  procedure moveDiscs(maxDisc, dstPile):
 3    srcPile := pileOfDisc[maxDisc];
 4    auxPile := 1 + 2 + 3 - srcPile - dstPile;
 5    if maxDisc < 1 then return;
 6    if (srcPile == dstPile) then begin
 7      moveDiscs(maxDisc-1, dstPile)
 8    end
 9    else begin
10      moveDiscs(maxDisc-1, auxPile);
11      // now maxDisc can be moved directly from srcPile to
12      // dstPile (one move), and all smaller discs can be
13      // moved from auxPile to dstPile (2^maxDisc - 1 moves)
14      // -> 2^maxDisc moves in total
```

```
15      for i := 1 to maxDisc do pileOfDisc[i] := dstPile;
16      numMoves := (numMoves + powerOfTwo(disc)) MOD 1000000;
17    end
18  end
19
20  // main:
21  numMoves := 0;
22  moveDiscs(N-1, pileOfDisc[N]);
```

In this improved version, the procedure `moveDiscs` never calls itself more than once. More precisely, it can be easily seen that the procedure is called exactly once for each `maxDisc` value $N, N-1, \ldots, 1$, i.e. N times in total. For each recursion depth, the runtime is $O(N)$ on average due to the `for` loop in line 15. After linear preprocessing and storing the powers of two in an array, they can be calculated in constant time. Of course they should also be calculated modulo 10^6 to avoid integers larger than 32bit. Thus the total runtime of this algorithm is $O(N^2)$, fast enough for the first twelve testcases.

By explicitely testing whether discs $1, \ldots, i$ that are to be moved all lie on the same pile with a `for` loop, one can also achieve a $O(N^3)$ solution that will solve the first six to eight test cases. However, this alternative solution has been omitted here.

However, we can still improve the $O(N^2)$ solution to achieve a linear time solution. To achieve this, one needs to notice that the loop in line 14 is completely unncessary and can be omitted. This is because the values of `pileOfDisc[]` that are written in this loop are never read. The `pileOfDisc[]` values are only read *before* the recursive calls, so it does not matter whether they are changed during or after these cals. Because we are only interested in the number of required moves and we know that all discs will lie on the destination pile in the end, anyway, this update of the discs' positions can be omitted. Because this loop is the only thing that consumed more than constant time within the procedure `moveDiscs`, and because this procedure is only called N times, this will yield a $O(N)$ solution. Obviously no solution can be better (except for a constant factor) because reading the source file already takes $O(N)$ time[1], and writing the source file also takes $O(N)$ time because the solution number is on average $O(2^N)$, i.e. $O(N)$ digits long.

The $O(N)$ solution is:

```
1   // moves all discs 1, 2, ..., maxDisc to dstPile
2   procedure moveDiscs(maxDisc, dstPile):
3     srcPile := pileOfDisc[maxDisc];
4     auxPile := 1 + 2 + 3 - srcPile - dstPile;
5     if maxDisc < 1 then return;
6     if (srcPile == dstPile) then begin
```

[1] Or even $O(N \log N)$ because the length of the numbers is $O(\log N)$ digits.

```
 7       moveDiscs(maxDisc-1, dstPile)
 8     end
 9     else begin
10       moveDiscs(maxDisc-1, auxPile);
11       // now disc maxDisc can be moved directly from srcPile
12       // to dstPile (one move), and all smaller discs can be
13       // moved from auxPile to dstPile (2^maxDisc - 1 moves)
14       // -->  2^maxDisc moves in total
15       numMoves := (numMoves + powerOfTwo[disc]) MOD 1000000;
16     end
17   end
18
19   // main:
20   numMoves := 0;
21   powerOfTwo[0] := 1;
22   for i := 1 to N
23     do powerOfTwo[i] := (2 * powerOfTwo[i-1]) MOD 1000000;
24   moveDiscs(N-1, pileOfDisc[N]);
```