Tower of Hanoi

Year 2019 – 2020

Aprilia Moldovan, Teodora Pașca, students in the 11thgrade

Ioana Gabor, Adelin Bojan, Luca Apahidean, students in the 10th grade

Teachers: Ariana-StancaVăcărețu, Valentina Vasilescu, Adrian Andrea

School: Colegiul Național "Emil Racoviță"

Researcher : Lorand Parajdi, *Babeș-Bolyai University, 1 Kogălniceanu Str., Cluj-Napoca, Romania*

1. The Tower of Hanoi problem

The Tower of Hanoi (also called 'Tower of Brahma' or 'Lucas' Tower' and sometimes pluralized as 'Towers'), invented by the French mathematician Édouard Lucas in 1883, is a mathematical game or puzzle, which consists of three rods and a number of disks of different sizes, which can be moved onto any rod. The puzzle starts with the disks in a neat stack in ascending order of size on one rod, the smallest at the top, thus making a conical shape, the purpose of the game being the replacement of the disks from a rod to another one, following the next rules:

i. Only one disk can be moved at a time.
ii. Each move consists of taking the upper disk from one of the stacks and placing it on top of another stack or on an empty rod.
iii. No larger disk may be placed on top of a smaller disk.

The research topic consisted in the mathematical determination of the number of minimum displacements required to move n disks from any situation A to any other situation B. Additionally, the situation of displacement in which a disk can be placed only on the neighbouring pillar was also taken into consideration.

2. Methods used and research results

Two methods have been used to provide the mathematical result of the minimum number of movements required to displace n disks from any given situation to another one.
Firstly, the method of induction, after observing a certain numerical pattern in the studied examples, followed by a C++ program created for the situation in which the disks can be placed only on the neighbouring pillar.

After using the induction method to verify the correctness of the formula, a statement was made, more exactly that the nr. of required moves = $2^n - 1$, where n = number of disks (for the general case), and that the nr. of required moves $=\frac{3^{n-1}}{2}$, where n = number of disks (following the additional rule, which states that disks can be placed only on the neighbouring pillar), whereas the algorithm provided the result for the second case, featuring the additional rule.

3. Research work

After explaining the purpose of the game of the Hanoï towers, we worked on determining the number of minimum displacements required to move n disks from any situation A to any other situation B. In addition to the rules given, a new restriction was added: one can only move a disk to the nearest rod.

The induction method

After analysing the variation of the number of movements depending on the number of disks, the base step of the induction method led to the following observed pattern in numbers. As one may observe, the general rule of the game is: **nr. of required moves = $2^n - 1$**, where n = number of disks.

Number of DISKS	Number of MOVES
1	1
2	3
3	7
4	15
5	31
6	63
7	127
8	255
9	511
10	1023

By using the information in the table above, the base step of the induction method was completed.

- **$a_n = 2^n - 1$**

I. If P(n): $a_n = 2^n - 1$, $\forall n \in \mathbb{N}^*$

P(1): $a_1 = 1$ „T”

P(2): $a_2 = 3$ „T”

For the inductive step, a form of writing the general number of moves was pointed out, which is $a_n= 2\ a_{n-1} + 1$, where a_n stands for the number of moves (for example, $15 = 2\cdot7+1$ or $31 = 2\cdot15+1$), a_{n-1} being the nr. of moves needed for the number of disks smaller with one unit than the one in discussion. Hence, for the inductive step, $a_{n+1}= 2\ a_n + 1$.

In order to verify whether the base step proved the statement to be true for the initial value, the inductive step for the first case (in the 1st case, we considered the general rules of the game, where a disk can be moved on any of the 2 pillars next to it) followed as it seems:

II. If P(1), P(2), ..., P(n) „T” $\Rightarrow(?)$ P(n+1) „T”

$a_{n+1} =(?)\ 2^{n+1} -1$

$a_{n+1}= 2\ a_n + 1$

$= 2\ (2^n - 1\ ) + 1$

$= 2^{n+1} - 2 + 1$

$= 2^{n+1} -1$

I, II $\Rightarrow a_n = 2^n - 1\ ,\ \forall\ n \in \mathbb{N}^*$

If the second situation is to be taken into consideration, where the additional rule introduced by the task specifies that a disk can only be moved on the pillar right next to it, the numerical pattern that occured in the base step was the next one:

Number of DISKS	Number of MOVES
1	1
2	4
3	13
4	40
5	121

For the inductive step to be easier to go through, a form of writing the general number of movements was pointed out, which is $a_n = 3a_{n-1} +1$, where a_n stands for the number of moves (for example, $13 = 3\cdot4+1$ or $40 = 3\cdot13+1$), a_{n-1} being the nr. of moves needed for the number of disks smaller with one unit than the one in discussion. The inductive step went in the following manner:

- $\mathbf{a_n = \frac{3^n-1}{2}}$

  P(n): $a_n=\frac{3^n-1}{2},\ \forall\ n \in \mathbb{N}^*$

$$
\begin{aligned}
a_n &= 3a_{n-1} + 1 \\
a_{n-1} &= 3a_{n-2} + 1 \quad |\cdot 3 \\
&\ldots \\
a_3 &= 3a_2 + 1 \quad |\cdot 3^{n-3} \\
a_2 &= 3a_1 + 1 \quad |\cdot 3^{n-2}
\end{aligned}
$$

$+$ ————

$$a_n + 3a_{n-1} + \ldots + a_3 \cdot 3^{n-3} + a_2 \cdot 3^{n-2} =$$
$$3a_{n-1} + a_{n-2} \cdot 3^2 + \ldots + a_2 \cdot 3^{n-2} + a_1 \cdot 3^{n-1}$$
$$+ 1 + 3 + \ldots + 3^{n-2}$$

$$\Rightarrow a_n = a_1 \cdot 3^{n-1} + 1 \cdot \frac{3^{n-1} - 1}{2}$$
$$a_n = 3^{n-1} + \frac{3^{n-1} - 1}{2}$$
$$a_n = \frac{2 \cdot 3^{n-1} + 3^{n-1} - 1}{2}$$
$$a_n = \frac{3^{n-1}}{2}, \ \forall n \in \mathbb{N}^*$$

C++ Algorithm

A program in C++ was created with the purpose of establishing the number of moves needed for the displacement of n disks, considering the additional rule added by the task. The program allows one to calculate the number of movements for the second case, where a disk can be moved only on the neighbouring pillar.

```
#include <iostream>
#define NDISKS 15
using namespace std;
int n,state1[NDISKS+5],state2[NDISKS+5],powers[NDISKS+5],sol;
int index(int state[NDISKS+5]){
    bool reversed=false;
    intind=0;
    for(inti=n;i>=1;i--){
        if(!reversed){
            if(state[i]==2){
                ind+=2*powers[i-1];
            }else if(state[i]==1){
```

```
            ind+=powers[i-1];
            reversed=true;
          }
      }else{
          if(state[i]==0){
            ind+=2*powers[i-1];
          }else if(state[i]==1){
            ind+=powers[i-1];
            reversed=false;
          }
      }
  }
  returnind;
}

int main(){
  cin>>n;
  powers[0]=1;
  for(inti=1;i<=n;i++){
    powers[i]=3*powers[i-1];
  }
  for(inti=1;i<=n;i++){
    cin>>state1[i];
    state1[i]--;
  }
  for(inti=1;i<=n;i++){
    cin>>state2[i];
    state2[i]--;
  }
  sol=index(state1)-index(state2);
  if(sol<0){
    sol=-sol;
  }
  cout<<sol;
}
```

Proving the algorithm correctness

The input contains an integer n ≤ 15, the number of disks, and two arrays, state1[] and state2[], state1[i] denotes the rod on which the i-th disk is situated in the first state, state2[i] denotes the rod on which the i-th disk is situated in the second state; state1[i], state2[i] ϵ{1,2,3}.

The output contains a single number sol, the number of steps needed to reach state 2 from state 1.

Let's encode any given state as a base 3 number (because there are three rods, the first one will be represented as 0, the second one as 1 and the third one as 2), with n digits, one digit for each disk.

For instance, if there are 4 disks, the first(the lightest) is on the second rod, the second and the fourth (the heaviest) ones are on the first rod and the third one is on the third rod. If we write this representation from right to left, the state can be encoded as "0201". Observe that the i-th leftmost digit in the base 3 number shows us on which rod is the n-i+1-th disk placed, the i-th digit is state[n-i+1]-1. In our algorithm, we subtract 1 from each element of both arrays, although this isn't needed, to maintain the base 3 number analogy.

To reach state "222..2" from state "000..0" we need to pass through exactly 3^n different states (this can be proved with mathematical induction). As there are exactly 3^n base 3 numbers with n digits, there is an unique solution to reach any given state (base 3 number) from another given state (base 3 number). Therefore, each base 3 number has an unique index, that indicates the number of steps needed to reach that state from state "000..0". Index is a bijective function.

For example, index("000")=0, index("111")=13,index("222")=26, index("001")=1.In this example, the function parameter is a base 3 number, but in our algorithm, the parameter is an array that contains the reversed digits of a base 3 number.

The number of steps between two states is |index (state1) - index (state2)|. In the presented algorithm, index(state) computes the index of any given state in O(n) time complexity.

In the following, we are going to explain how the index function works. The state[] parameter is an array that contains the reversed digits of a base 3 number (as we previously stated, the i-th digit is state[n-i+1]-1). We iterate through the elements of the array, from the biggest disk to the smallest one (from the beginning of the base 3 number to the end of the base 3 number), and we count all the previous states. The Boolean variable "reversed" is true if the current number of "1" is odd and is false otherwise. If the current number of "1" is even, we add 2*3^(i-1) for a "2", because there are 2*3^(i-1) previous states for which the first n-i+1 digits of the base 3 numbers are equal to the first n-i digits of the base 3 number corresponding to the given state and the n-i+1 digit is smaller than "2". We add 3^(i-1) for a "1" and 0 for a "0", with similar explanations. If the current number of "1" is odd, the meanings of "2" and "0" are reversed: we add 2*3^(i-1) for a "0" and nothing for a "2". To make this clearer, we will show the indexes of all the 3-digit base 3 numbers. Remember, in our algorithm the function parameter is an array with reversed base 3 number digits, but in this example, the numbers are not reversed!

index(000)=0	index(122)=9	index(200)=18
index(001)=1	index(121)=10	index(201)=19
index(002)=2	index(120)=11	index(202)=20
index(012)=3	index(110)=12	index(212)=21
index(011)=4	index(111)=13	index(211)=22
index(010)=5	index(112)=14	index(210)=23
index(020)=6	index(102)=15	index(220)=24
index(021)=7	index(101)=16	index(221)=25
index(022)=8	index(100)=17	index(222)=26

Alternative representation

The game can be represented by a graph, the nodes representing distributions of disks and the edges representing moves. The graph of 3 disks, as represented in Fig.1 and Fig.2, resembles a fractal (Sierpiński triangle) in the repetitive case of the game (if distributions from one side to another were approved within a move, as if it would be a circular system), but our case can be visualized by erasing the unused edges. The graph shows that from every arbitrary distribution of disks, there is exactly one shortest way to move all disks onto one of the three pegs.

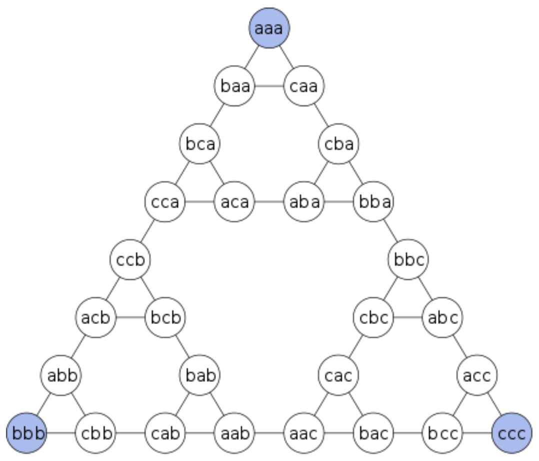

Fig.1 Undirected graph, the nodes representing distributions of disks and the edges representing moves, for three disks, in the first case

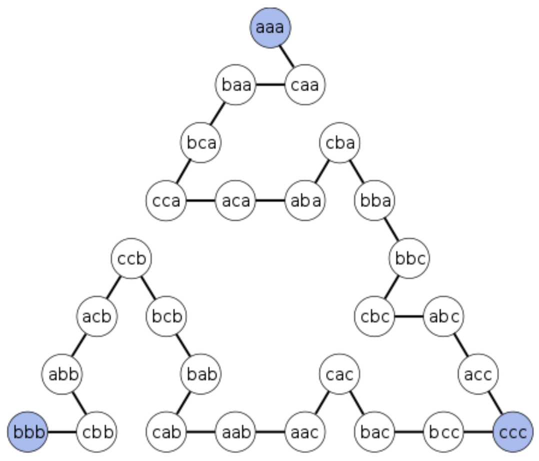

Fig.2Undirected graph, the nodes representing distributions of disks and the edges representing moves, for three disks, in the second case

4. Conclusion

Therefore, the solution to the problem can be seen from 2 angles. Both mathematically and computer-wise, solving the problem involves analysing moves of the disks, as well as streamlining this process. Processing the situation, mathematically we have proved that the smallest number of steps is rendered by the formula $\mathbf{2^n - 1}$. From a computer point of view, we transposed the mathematical reasoning into the C++ language, so that the computer can calculate the number of steps from the position on which the disks are, to the final position. If the mathematical part fixed our idea, the computer part made our work easier, giving the correct answer through a virtual interface.