

# **“THEORY OF COMPUTATION FINAL PAPER”**

## **COMPUTER ENGINEERING(CO313)**



**“TOWER OF HANOI”**

**Project Report**

NAME	ROLL NO	EMAIL I'D
<b>Sristi Mitra</b>	<b>2K19/CO/389 (A-4)</b>	<a href="mailto:sristimitra_2k19co389@dtu.ac.in">(sristimitra_2k19co389@dtu.ac.in)</a>
<b>Vineet</b>	<b>2K19/CO/427 (A6)</b>	<a href="mailto:vineet_2k19co427@dtu.ac.in">(vineet_2k19co427@dtu.ac.in)</a>
<b>Zishnendu Sarker</b>	<b>2K19/CO/450 A6</b>	<a href="mailto:zishnendusarker_2k19co450@dtu.ac.in">(zishnendusarker_2k19co450@dtu.ac.in)</a>

*A comprehensive project report has been submitted in partial fulfillment  
of the requirements for the degree of*

**Bachelor of Technology**

in

**Computer Engineering**

*Under the supervision*

of

**[Maam Raveena Yadav](#)**

**Delhi Technological University, formerly**

**Delhi College of Engineering**

**Department of Computer Science Engineering**

**Delhi Technological University, Shahbad Daultpur, Main Bawana  
Road, Delhi-110042. India**

## **Acknowledgment:**

We would like to express my deepest appreciation for all the resources that have provided me with the possibility to make progress in our report. A special gratitude I give to our “Theory of Computation faculty” **Raveena Yadav Maam**, whose stimulating suggestions and encouragement helped to coordinate in writing this project. Since the Tower of Hanoi is a very interesting and ancient topic and incorporation with the theory of Computation will be a great learning opportunity. So, we planned to work on the recursive problem which is also played as a game; named “Tower of Hanoi”. We were inspired by our subject teacher who taught us the basics of the theory of computation and also gave us the option to choose this topic. According to the knowledge we gained in the class and from some online resources, we are able to finish this project paper. We are also grateful that the project enhances our knowledge of the algorithm behind the recursive game problem and also the whole concept of the theory of computation.

## **Abstract:**

The Tower of Hanoi is an ancient problem. This paper describes the theory tower of Hanoi and analyzes the use of this recursive problem for problem-solving in the Theory of Computation. We will explain the algorithm behind the recursive problem and also the time complexity. The paper also shows an evolutionary algorithm approach for searching for solutions to the problem. In the end, we will visualize the game with  $n$  pegs and also include codes to solve the problem. Tower of Hanoi shows recursive behavior which we are incorporating with the Theory of Computation in the overall paper.

# INDEX:

Topic name	Page
1. Introduction	04
→ History of Tower of Hanoi	05
→ Problem description	06
→ Tower of Hanoi in Star Graph	06
2. Theory Explained	
→ Pushdown Automata	07
→ Delta Function	09
→ Cyclic Tower of Hanoi	11
→ Classical Sequences are hidden behind the Hanoi Experiment	12
→ Period double Sequence	12
→ Double-Free Subset	12
→ Structural (Coding) of	13
→ Recursive Problem Solving	13
→ Relationship between stack And Tower of Hanoi	14
→ Making simple stack	14
→ Solving Tower Of Hanoi	15
→ The logic for Sample Output	16
3. Algorithm Explained	17
→ Example of Code in Algorithm	18
→ Output	19
4. Conclusion	20
5. References	20

## Introduction:

The tower of Hanoi (also called the Tower of Brahma or the Lucas tower) was invented by a French mathematician Édouard Lucas in the 19th century. The Tower of Hanoi puzzle has been studied extensively in problem-solving literature. In the game, there exist three vertical pegs, and a player is given a certain number of disks of mutually different diameters placed in small-on-large ordering on a peg. . The task is to get from a given initial state to a target state by moving a single disk from the top of the peg to the top of another possible one while obeying the following rules:

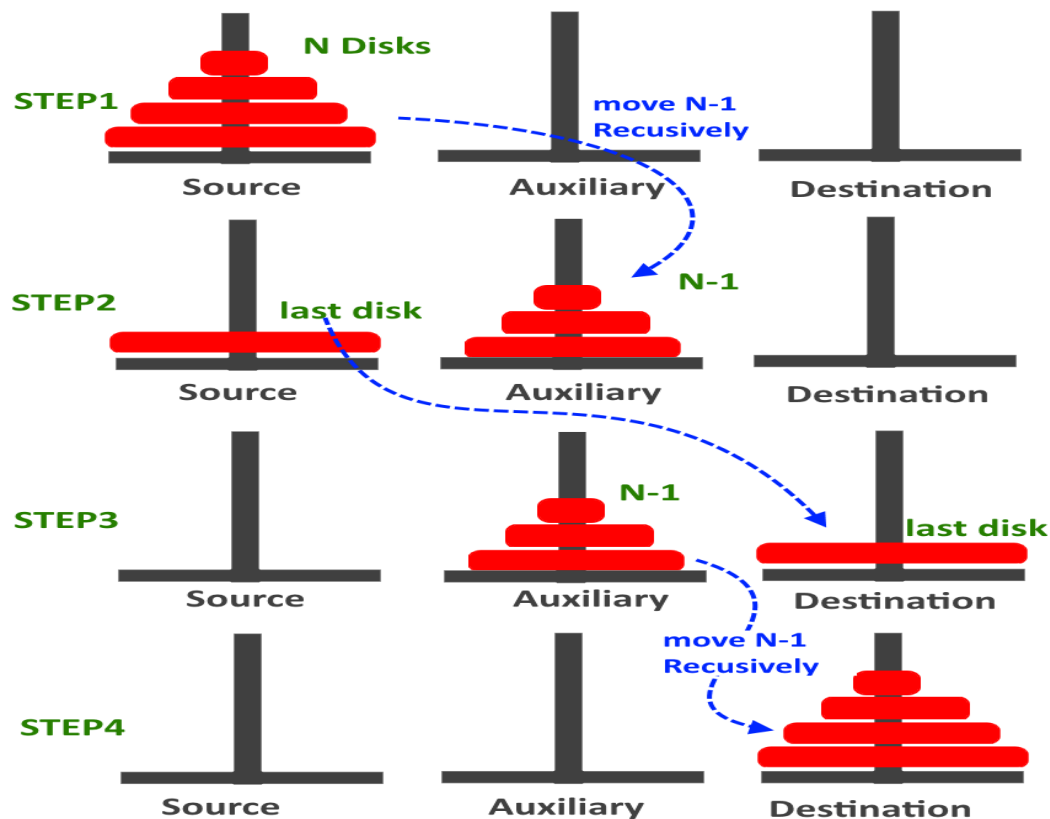


Fig 1: Introducing Tower of Hanoi

- Each time only one disk is moved;
- Only the topmost disk can be moved;
- At any moment, a disk cannot reside on a smaller one

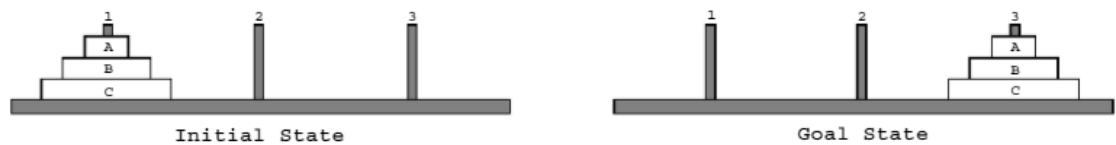
The Tower of Hanoi is a well-known recursive issue in computer science that has been debated and utilized in a variety of domains. Solving the Tower of Hanoi problem is a useful way to represent the planning process, such as robot task planning and unmanned vehicle path planning. In addition, the Tower of Hanoi is used to study how humans develop problem-solving abilities.

### → History of Tower of Hanoi:

According to the Tower of Hanoi's continuous mythology, the Hindu temple priests were given a stack of 64 fragile gold disks at the beginning of time, and they were assigned the responsibility of transporting the disks one by one from one pole in the temple's ground to the third pole on the opposite side of the temple. Because no two disks were the same size and each one was slightly smaller than the one below it, the largest disk was at the bottom of the pile and the tiniest disk was at the top. Because the disks were so delicate, the sole crucial restriction was that a larger disk should never be placed on a smaller disk, and there was only one intermediate pole where the disks might be temporarily placed. The priests worked day and night to complete the assignment, and tradition has it that when they did, the temple crumbled into dust and the world vanished before the priests could finish, but no one knows if this happened. When considering the legend situation, it appears that completing such a task is practically difficult. Despite the fact that the priests were unaware of it at the time, the mission was performed effectively thanks to the power of sequences and patterns. (PBS). Edouard Lucas popularized the narrative in the hopes of increasing the popularity of the Tower of Hanoi game. After a broken dinner plate tore his cheek and developed a systemic illness, Lucas died in 1891. His mathematical innovations were "as funny as they were informative," according to his obituary in the 1892 issue of "Popular Science Monthly".

### → Problem Description or gameplay:

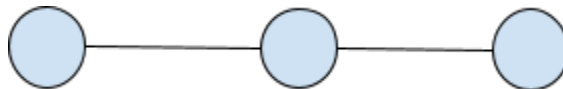
Three pegs, indicated as A, B, C, and  $n(>=1)$  disks of various sizes, are used in the traditional Tower of Hanoi problem. All disks are originally arranged in a tower on the source peg in small-on-large order, with the largest at the bottom, the second-largest above it, and so on, with the smallest at the top. The goal is to move the tower from the source to the target peg using legal maneuvers. The lawful move is defined as an operation that allows the topmost disk to be transferred from one peg to another while adhering to the constraints outlined above.



**Fig 2 : 3 peg tower of Hanoi problem**

The initial and objective states of a 3-disk classical Tower of Hanoi problem are shown in Fig. 1. D1, D2, and D3 are the three disks in order of increasing size. At first, all of the disks are on stake A, the source peg, while stake C is the target peg. The purpose of the game is the same for all versions of multi-peg Tower of Hanoi problems: move the tower from the source to the destination.

### → Tower of Hanoi in Star Graph:



**Fig 3: star graphs with 3 vertices**

The Hanoi Star Tower has the following two sorts of problems. The first is to move all of the disks from one leaf of the star graph to the next leaf. The second step is to move the disks from a leaf to the graph's center. The first type of problem is with 4 pegs and the second type is to transfer  $n$  disks from one leaf to the center.

In the case of figure 02, this graph is also known as the route graph. The following two types of Tower of Hanoi issues are considered on this graph. We move all the disks from the initial leaf to another, which is known as leaf-to-leaf, or to the center peg, which is known as leaf-to-center, given  $n$  disks on a leaf peg. It's worth noting that the leaf-to-center dilemma is the same as moving disks in the reverse direction, from the center to a leaf. Although the optimum methods and their analysis are simple, we outline them to serve as a foundation for further investigation.

## Theory Explained:

### Pushdown Automata(PDA):

Finite automata with extra memory (stack) are called pushdown automata. In simple words; Finite Automata(FA)+stack=PDA.

Context-free language recognizes PDA. It contains 7 tuples;

- $P=(Q, \Sigma, T, \delta, q, z, F)$

$Q$ = finite set of states like FA

$\Sigma$ = Input symbol like FA

$T$ =stack alphabet (which is pushed into the stack)

$\delta$ =transition function  $\rightarrow \delta:Q(\Sigma \cup \epsilon) \times T \rightarrow Q \times T^*$



q=initial state

z=stack start symbol

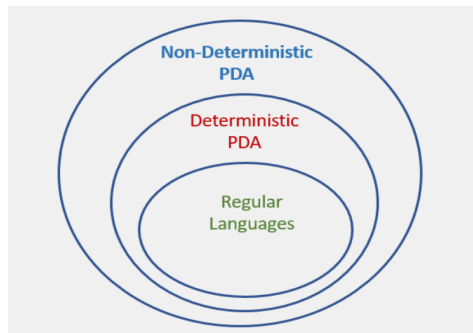
F=final state

- $\delta: Q(\Sigma \cup \epsilon) \times T \rightarrow Q \times T^*$ 
  - If T and T\* is same after transition then the stack will remain unchanged
  - If T\* become  $\epsilon$  or null, then the T will pop
  - If T and T\* is not same then T will push
- A stack does two operations –
  - Push – a new symbol is added at the top.
  - Pop – the top symbol is read and removed.
- Two types of pushdown automata:
  - Deterministic Automata
  - Non-deterministic Automata

Deterministic Pushdown Automata	Non-deterministic Pushdown Automata
<p>A deterministic PDA is one in which there is at most one possible transition from any state based on the current input. Formally, a deterministic PDA is a PDA where: for every state, input symbol, and stack symbol, there is at most one transition of the form for any state and stack symbol.</p> <p>A Deterministic PDA is 5 tuple -</p>	<p>A Non-deterministic PDA is used to generate a language that a deterministic automaton cannot generate. It is more powerful than a deterministic PDA. So, a pushdown automata is allowed to be non-deterministic.</p>

<p>-- equation:</p> <ul style="list-style-type: none"> <li>• <math>M = (\Sigma, \Gamma, Q, \delta, q)</math></li> </ul> <p>→ <math>\Sigma</math> - It is a finite set that does not contain a blank symbol,</p> <p>→ <math>\Gamma</math> - a finite set of stack alphabet,</p> <p>→ <math>Q</math> - set of states,</p> <p>→ <math>q</math> - start state,</p> <p>→ <math>\delta</math> - a transition function, denoted as  <math>\delta : Q \times (\Sigma \cup \{\square\}) \times \Gamma \rightarrow Q \times \{N, R\} \times \Gamma^*</math></p>	<p>-- equation:</p> <ul style="list-style-type: none"> <li>• <math>M = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)</math></li> </ul> <p>→ <math>Q</math> - It is the finite set of states,</p> <p>→ <math>\Sigma</math> - finite set of input alphabet,</p> <p>→ <math>\Gamma</math> - finite set of stack alphabet,</p> <p>→ <math>\delta</math> - transition function,</p> <p>→ <math>q_0</math> - initial state,</p> <p>→ <math>Z_0</math> - stack start symbol,</p> <p>→ <math>F</math> - finite states.</p>
---	---

**Table 1. Pushdown automata explanation**



**Fig 2. Types of PDA**

### **Delta Function:**

Delta Function is the transition function, the use of which will become more clear by taking a closer look at the Two Major operations done on Stack :-

1. Push
2. Pop

Equation of Delta Function:

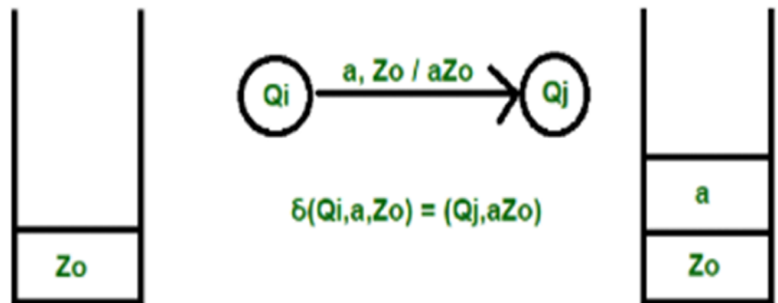
$$\delta: Q(\Sigma \cup \epsilon) \times T \rightarrow Q \times T^*$$

- If  $T$  and  $T^*$  is same after transition then the stack will remain unchanged
- If  $T^*$  become  $\epsilon$  or null, then the  $T$  will pop
- If  $T$  and  $T^*$  is not same then  $T$  will push

1

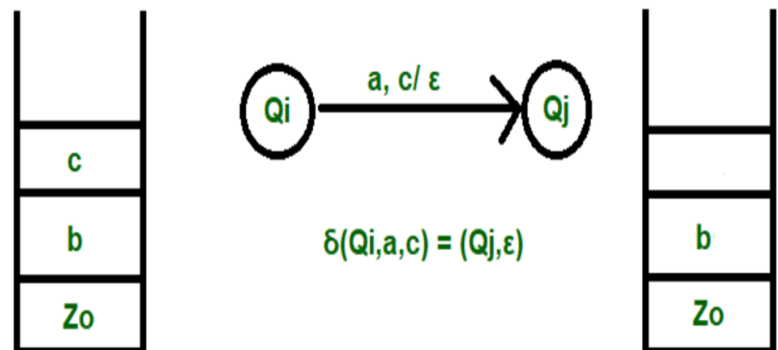
### 1. Push

- $a$  is the input element,
- which is inserted into the stack



### 2. Pop

- $a$  is the input
- $c$  is the element to be deleted



## ● Cyclic towers of Hanoi

In Cyclic Hanoi, we are given three pegs (A, B, C), which are arranged as a circle with the clockwise and the counterclockwise directions being defined as  $A \rightarrow B \rightarrow C \rightarrow A$  and  $A \rightarrow C \rightarrow B \rightarrow A$  respectively. The moving direction of the disk must be clockwise. It suffices to represent the sequence of disks to be moved. The solution can be found using two mutually recursive procedures:

→ To move  $n$  disks counterclockwise to the neighboring target peg:

1. move  $n - 1$  disks counterclockwise to the target peg
2. move disk # $n$  one step clockwise
3. move  $n - 1$  disks clockwise to the start peg
4. move disk # $n$  one step clockwise
5. move  $n - 1$  disks counterclockwise to the target peg

→ To move  $n$  disks clockwise to the neighboring target peg:

1. move  $n - 1$  disks counterclockwise to a spare peg
2. move disk # $n$  one step clockwise
3. move  $n - 1$  disks counterclockwise to the target peg

→ Let  $C(n)$  and  $A(n)$  represent moving  $n$  disks clockwise and counterclockwise, then we can write down both formulas:

$$C(n) = A(n-1) \text{ n } A(n-1) \quad \text{and} \quad A(n) = A(n-1) \text{ n } C(n-1) \text{ n } A(n-1).$$

$$\text{Thus } C(1) = 1 \quad \text{and} \quad A(1) = 1 \text{ } 1,$$

$$C(2) = 1 \text{ } 1 \text{ } 2 \text{ } 1 \text{ } 1 \quad \text{and} \quad A(2) = 1 \text{ } 1 \text{ } 2 \text{ } 1 \text{ } 2 \text{ } 1 \text{ } 1.$$

The solution for the Cyclic Hanoi has some interesting properties:

- 1) The move patterns of transferring a tower of disks from a peg to another peg are symmetric with respect to the center points.
- 2) The smallest disk is the first and last disk to move.

- 3) Groups of the smallest disk moves alternate with single moves of other disks.  
 4) The number of disk moves specified by  $C(n)$  and  $A(n)$  is minimal.

### ● Classical sequences are hidden behind the Hanoi sequence

Several classical sequences are linked to the Hanoi sequence. We will describe some of them in this report.

#### → Period-doubling sequence

A binary sequence  $T$  can be deduced from  $S_\infty$  by replacing each of  $a, b, c$  by 1 and each of  $\bar{a}, \bar{b}, \bar{c}$  by 0 (i.e., the non-barred letters by 1 and the barred letters by 0), thus obtaining

$$\begin{array}{l} \mathcal{S}_\infty = a \bar{c} b a c \bar{b} a \bar{c} b \bar{a} c b a \bar{c} b \dots \\ \mathcal{T} = 1 \ 0 \ 1 \ 1 \ 1 \ 0 \ 1 \ 0 \ 1 \ 0 \ 1 \ 1 \ 1 \ 0 \ 1 \dots \end{array}$$

It is not difficult to prove that  $T$  is the iterative fixed point of the morphism  $\omega$  on  $\{0,1\}^*$  defined by  $\omega(1) := 10$ ,  $\omega(0) := 11$ . This iterative fixed point is known as the period-doubling sequence. It was introduced in the study of iterations of unimodal continuous functions in relation to Feigenbaum cascades

#### → Double-free subsets

Define a sequence of integers  $U$  by counting for each term of  $S_\infty$  the cumulative number of the non-barred letters up to this term

$$\begin{array}{l} \mathcal{S}_\infty = a \bar{c} b a c \bar{b} a \bar{c} b \bar{a} c b a \bar{c} b \dots \\ \mathcal{U} = 1 \ 1 \ 2 \ 3 \ 4 \ 4 \ 5 \ 5 \ 6 \ 6 \ 7 \ 8 \ 9 \ 9 \ 10 \dots \end{array}$$

The sequence  $U$  is equal to the sequence of maximal sizes of a subset  $S$  of  $\{1,2,\dots,n\}$  with the property that if  $x$  is in  $S$  then  $2x$  is not. (The sequences  $T$  in above and  $U$  are respectively called A035263 and A050292 in Sloane's Encyclopedia

## ● Structural (Coding) of Tower of Hanoi:

Here if we write a function Hanoi (n, start, end) that outputs a sequence of steps to move n disks from the start rod to end rod. Suppose 3 pegs,  
 Hanoi (3,1,3)=hanoi (n,start,end)

Assumptions=

- $1 \leq \text{start} \leq 3$
- $1 \leq \text{end} \leq 3$
- Start (not equal to) end;  $n \geq 1$

A solution sequence for a problem is a sequence of legal moves that transform the starting configuration into the ending configuration. An optimal solution sequence is one of minimal length. The optimal solution sequence for an N-disk problem is no longer than  $2N - 1$  moves. An N-disk TOH problem can have any one of  $3N$  starting configurations and  $3N - 1$  ending configurations, and therefore there are a total of  $3N(3N - 1)$  problems. It is useful to group TOH problems into five classes based on the spatial properties of their starting and ending configurations. All tower-to-tower problems (including the standard one) require the maximum  $2N - 1$  moves to solve. Tower-to-spread problems have all N disks stacked on one peg in the starting configuration; in the ending configuration, the disks are distributed over two or more pegs. Spread-to-tower problems are just the opposite: the disks are spread over two or more pegs in the starting configuration and stacked on one peg in the ending configuration. Finally, spread-to-spread problems distribute the disks over two or more pegs in both the starting and end configurations. A variable number of moves – between 1 and  $2N - 1$  – are required to solve problems of the last three classes. (The exact number depends on the specific starting and ending configurations.)

## ● Recursive Problem solving

Let,  $f(n)$  be a recursive f.

- Show  $f(1)$  works (base case)
- Assume  $f(n-1)$  works

➤ Show  $f(n)$  works  $F(n-1)$  (keeping  $f(n)$  close to  $f(n-1)$ )

```

hanoi (n, start, end)= pm(start, end) if n=1
other= 6-(start+end)
hanoi(n-1), start, other)
pm (start, end)
hanoi( n-1, other, end)
pm(start, end)=print (start→ end)

```

## ● Relationship between Stack and tower of Hanoi:

### → Making simple stack

In case of initiate the solution of the Tower of Hanoi problem, we have to firstly,

#### 1. Creating a simple stack:

- empty: stack

Returns a stack with no elements.

- height : stack -> int

Returns the number of elements in the stack.

- isEmpty : stack -> bool

Returns *true* if the stack is empty.

- push : stack \* disc -> stack

Puts the specified disc on the top of the stack.

- pop : stack -> disc \* stack

Removes the topmost element of the stack, and returns it together with the 'leftover' stack.

- top : stack -> disc

Returns the topmost element of the stack without removing it.

After checking our stack is implemented properly, we will move towards the solution of the actual problem,

Here, pegs can be considered as a tuple containing string and stack. A move is a single step in the Hanoi puzzle and consists of taking a single disc from one peg and placing it onto another. We define a move as a tuple of (disc, source peg, destination peg).

### → Solving Tower of Hanoi

Each disc is represented by the integer number corresponding to its size (see introduction). A disc with a higher number is larger than a disc with a lower number, and hence higher number disc cannot be above a lower number disc in a peg.

The source and destination pegs are represented by their string names. The pegs are named "A", "B", and "C".

- `makePeg : string * int -> peg`

Returns a new peg with the name, provided in the first argument. This peg should have the first  $n$  discs already on it, where  $n$  is specified as an argument.

- `isValid : peg -> bool`

Indicates whether the given peg is in a valid configuration. We define a valid configuration as one in which no disc is under a disc with a larger size (i.e. for all  $i \geq 1$ ,  $\text{stack}(i-1) > \text{stack}(i)$ ), where  $\text{stack}(k)$  denotes the  $k$ th element on the stack.

Note: a peg is not guaranteed to be in a consistent state. The underlying stack that you implemented does not enforce this validity condition; the condition is a property of this specific application of your stack. Therefore, this function is necessary to ensure that your code is working with valid stacks.

- `isLegalMove: disc * peg -> bool`

Returns true if the specified disc can be placed on top of the specified peg without creating an invalid configuration on the peg.



- `moveDisc : peg * peg -> peg * peg`

Moves the top disc from the first peg onto the second peg (and returns the resulting new pegs).

- `solve: towers -> towers * move list`

This function, given an initial configuration specified in the input, outputs a series of moves IN ORDER (i.e. the first element of the list is the first move to be made) that will cause all discs initially on the first peg to be moved onto the second peg. It should also return the final configuration of pegs that would result from applying the list of moves generated.

### → The logic for Sample output

Here is some sample output that your program is expected to produce. In this case we have 3 discs on peg *A* initially and generate a move list that shows how to move them onto peg *B*:

- `val a = makePeg("a", 3);`

`val a = ("a",[1,2,3]) : peg`

- `val b = makePeg("b", 0);`

`val b = ("b",[]) : peg`

- `val c = makePeg("c", 0);`

`val c = ("c",[]) : peg`

→ `solve(a,b,c);`

`val it =`

`((("a",[]),("b",[1,2,3]),("c",[])),`

`[(1,"a","b"),(2,"a","c"),(1,"b","c"),(3,"a","b"),(1,"c","a"),(2,"c","b"),`

`(1,"a","b")) : towers * move list`

## Algorithm Explained:

As we know that only one disk can be moved among the towers at any given time, only the top disk can be removed, and lastly, no large disk can sit over a small disk. According to its theory, the puzzle of the Tower of Hanoi with  $n$  disks can be solved in a minimum of  $2^n - 1$  steps. So the puzzle with 3 disks has taken  $2^3 - 1 = 7$  steps.

To design an algorithm for Tower of Hanoi, at first, we must understand how to solve the problem with fewer discs, such as one or two. Name, source, destination, and aux are all expressed on three towers (only to help move the disks). If we simply have one disc, we can easily move it from the source to the destination peg.

If we only have two pegs then first, we will move the smaller that means the top disk to the aux peg and then we will move the larger that means the bottom peg to the destination peg. And finally, we will move the smaller disk from aux to destination peg.

We can now create an algorithm for the Tower of Hanoi using more than two discs. The stack of discs is divided into two parts. The largest disc (the  $n$ th disc) is in one portion, while the remaining  $(n-1)$  discs are in the other.

Our final goal is to transfer disc  $n$  from source to destination and then copy all of the remaining  $(n-1)$  discs to it. We may consider doing the same for all of the drives in a recursive manner.

To get the final state for the tower of Hanoi problem for more than 2 disks or  $n$  discs

- Create a function Tower with:-
  - int 'n' – for number of disks,
  - char 'from' – for from peg,
  - char 'aux' – for a secondary peg,
  - char 'to' – for destination peg

- Put 'if' loop
- If (n=1) i.e. if the number of disk = 1, move it from 'initial peg' to the 'destination peg'
- Else, call function tower for 'n-1' i.e. the number of disk -1, recall function tower for n-1 disk, and move it 'from' to 'to'
- Recall function again using recursion until an or number of the disk is 1.

**→ Example of code in algorithm :**

```
void tower(int n,char from,char aux,char to){
if(n==1){
cout<<"\t\tMove disc 1 from "<<from<<" to "<<to<<"\n";
return;
}
else{
tower(n-1,from,to,aux);
cout<<"\t\tMove disc "<<n<<" from "<<from<<" to "<<to<<"\n";
tower(n-1,aux,from,to);
}
}
void main(){
clrscr();
int n;
cout<<"\n\t\t*****Tower of Hanoi*****\n";
cout<<"\t\tEnter number of discs : ";
cin>>n;
cout<<"\n\n";
tower(n,'A','B','C');
getch(); }
```

```

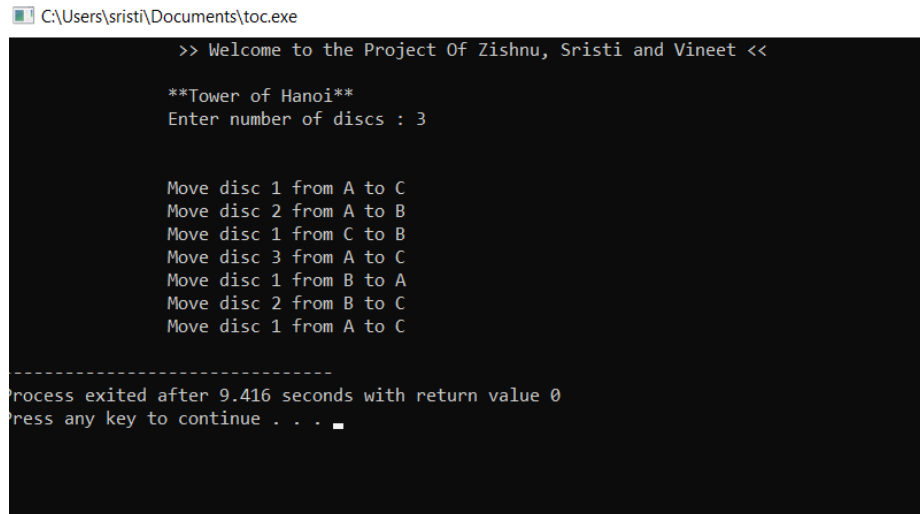
void tower(int n, char from, char aux, char to) {
if(n==1) {
std::cout<<"\t\tMove disc 1 from "<<from<<" to "<<to<<"\n";
return;
}
else{
tower(n-1, from, to, aux);
std::cout<<"\t\tMove disc "<<n<<" from "<<from<<" to "<<to<<"\n";
tower(n-1, aux, from, to);
}
}

main() {
int n;
std::cout<<"\n\t\t*****Tower of Hanoi*****\n";
std::cout<<"\n\t\tEnter number of discs : ";
std::cin>>n;
std::cout<<"\n\n";
tower(n, 'A', 'B', 'C');
}

```

Fig: code of tower of hanoi

## Output :



```

C:\Users\sristi\Documents\toc.exe
>> Welcome to the Project Of Zishnu, Sristi and Vineet <<

**Tower of Hanoi**
Enter number of discs : 3

Move disc 1 from A to C
Move disc 2 from A to B
Move disc 1 from C to B
Move disc 3 from A to C
Move disc 1 from B to A
Move disc 2 from B to C
Move disc 1 from A to C

-----
process exited after 9.416 seconds with return value 0
press any key to continue . . .

```

Fig: output for 3 pegs

## Conclusion

By tackling the Tower of Hanoi problem from the representation viewpoint, we have come up with a different algorithm. By making the disc moving directions explicit in the representation, we have shown that the bit-string encodes all the essential information and is appropriate for driving the algorithm. More importantly, with the help of bit-string, we can make precise statements about the behavior of the solution which is hidden in other approaches. For instance, moving direction and number of steps taken by each disc, optimality and others can be made precise. The bit-string gains us tremendous insight into the Tower of Hanoi problem which is otherwise obscure.

## References:

1. Hinz, Andreas M. "The tower of Hanoi." *Ensign. Math* 35.2 (1989): 289-321.
2. Er, M. C. "A representation approach to the tower of Hanoi problem." *The Computer Journal* 25.4 (1982): 442-447.
3. Hinz, Andreas M. "Shortest paths between regular states of the tower of Hanoi." *Information sciences* 63.1 (1992): 173-181.
4. Klavžar, Sandi, and Uroš Milutinović. "Graphs  $S(n, k)$  and a variant of the Tower of Hanoi problem." *Czechoslovak Mathematical Journal* 47.1 (1997): 95-104.
5. Hayes, P. J. "Discussion and correspondence A note on the Towers of Hanoi problem." *The Computer Journal* 20.3 (1977): 282-285
6. Chi, Michelene TH, and Robert Glaser. *Problem-solving ability. Learning Research and Development Center, University of Pittsburgh, 1985.*

7. Spitz, Herman H., Nancy A. Webster, and Suzanne V. Borys. "Further studies of the Tower of Hanoi problem-solving performance of retarded young adults and nonretarded children." *Developmental Psychology* 18.6 (1982): 922.
8. Cambon, Stéphane, Fabien Gravot, and Rachid Alami. "A robot task planner that merges symbolic and geometric reasoning." *ECAI*. Vol. 16. 2004.
9. McDermott, Patricia L., Thomas F. Carolan, and Mark R. Gronowski. "Application of Worked Examples to Unmanned Vehicle Route Planning." *The Interservice/Industry Training, Simulation & Education Conference (I/ITSEC)*. Vol. 2012. No. 1. National Training Systems Association, 2012.
10. Jie Li, and Shitan Huang. "Evolving in extended hamming distance space: hierarchical mutation strategy and local learning principle for EHW." *Evolvable Systems: From Biology to Hardware*. Springer Berlin Heidelberg, 2007. 368-378.
11. Miller, Julian F., and Peter Thomson. "Cartesian genetic programming." *Genetic Programming*. Springer Berlin Heidelberg, 2000. 121-132.
12. Jie Li, and Shitan Huang. "Adaptive salt-&-pepper noise removal: a function level evolution based approach." *Adaptive Hardware and Systems*, 2008. AHS'08. NASA/ESA Conference on. IEEE, 2008.
13. Romik, Dan. "Shortest paths in the Tower of Hanoi graph and finite automata." *SIAM Journal on Discrete Mathematics* 20.3 (2006): 610-622.
14. Klavžar, Sandi, Uroš Milutinović, and Ciril Petr. "On the Frame–Stewart algorithm for the multi-peg Tower of Hanoi problem." *Discrete applied mathematics* 120.1 (2002): 141-157.
15. Houston, Ben, and H. Masun. *Explorations in 4-peg Tower of Hanoi*. Technical Report TR-04-10, 2004.
16. Klavzar, Sandi, Uros Milutinovic, and Ciril Petr. "Combinatorics of topmost discs of multi-peg tower of Hanoi problem." *Ars Combinatoria* 59 (2001): 55-64.