Final Project of Theory of Computation

"TOWER OF HANOI"

Submitted by:

Participant 01: Sristi Mitra

Roll: 2K19/CO/389

Participant 02: Vineet

Roll: 2K19/CO/427

Participant 03: Zishnendu Sarker

Roll: 2K19/CO/450

Submitted to:

Raveena Yadav Maam

Theory Of Computation(TOC)

Delhi Technological University

Contents

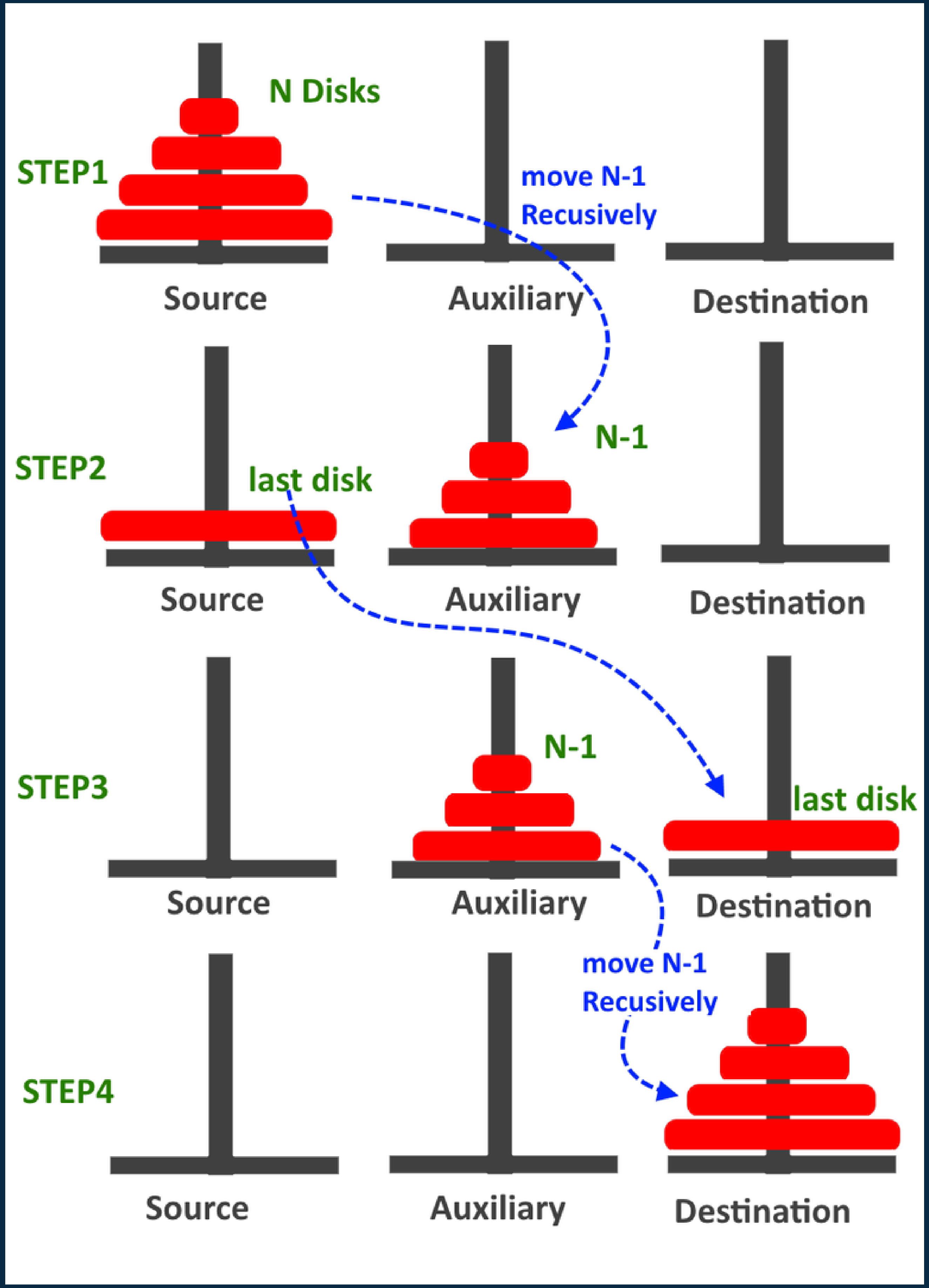
Explaining Tower of Hanoi

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 THEORY WHICH IT RELATES TO PUSHDOWN AUTOMATA AND ALSO DETERMINISTIC AND NONE DETERMINISTIC PUSHDOWN AUTOMATA; 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. TOWER OF HANOI SHOWS RECURSIVE BEHAVIOR WHICH WE ARE INCORPORATING WITH THE THEORY OF COMPUTATION IN THE OVERALL PAPER.

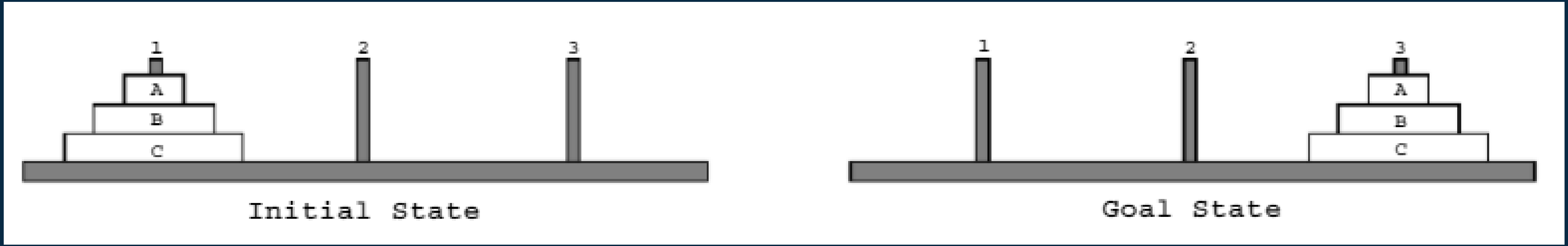
INTRODUCTION

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.



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 initial and objective states of a 3-disk classical Tower of Hanoi problem are shown in Fig. D1, D2, and D3 are the three disks in order of increasing size. At first, all of the disks are on stake 1, the source peg, while stake 3 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.



THEORY EXPLAINED:

HERE, WE ARE SHOWING THE RELATION BETWEEN TOWER OF HANOI AND THEORY OF COMPUTATION. FIRSTLY EXPLAINING THE THEORY BEHIND THE RELATIONSHIO:

PUSHDOWN AUTOMATA:

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 like;

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

Q= finite set of states like FA

Σ = Input symbol like FA

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

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

q=initial state

z=stack start symbol

F=final state

- Add a little bit ofTwo types of pushdown automata:

1.Deterministic Automata

2.Non-deterministic Automata

Delta Function

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

1. Push
2. Pop
3. Skip

Equation of Delta Function:

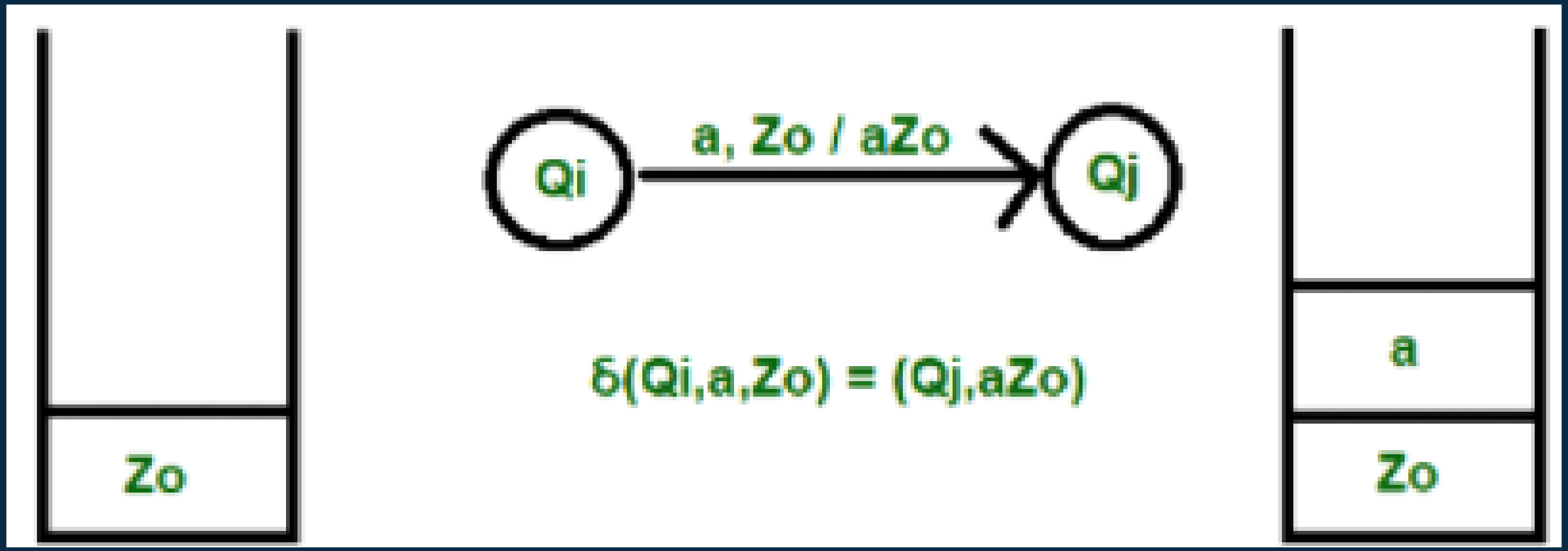
- $\delta : Q(\Sigma \cup \varepsilon) \times T \rightarrow Q \times T^*$

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

Push

-Input, Topmost Element / Final List

-a is the input elemen, which is inserted into the stack

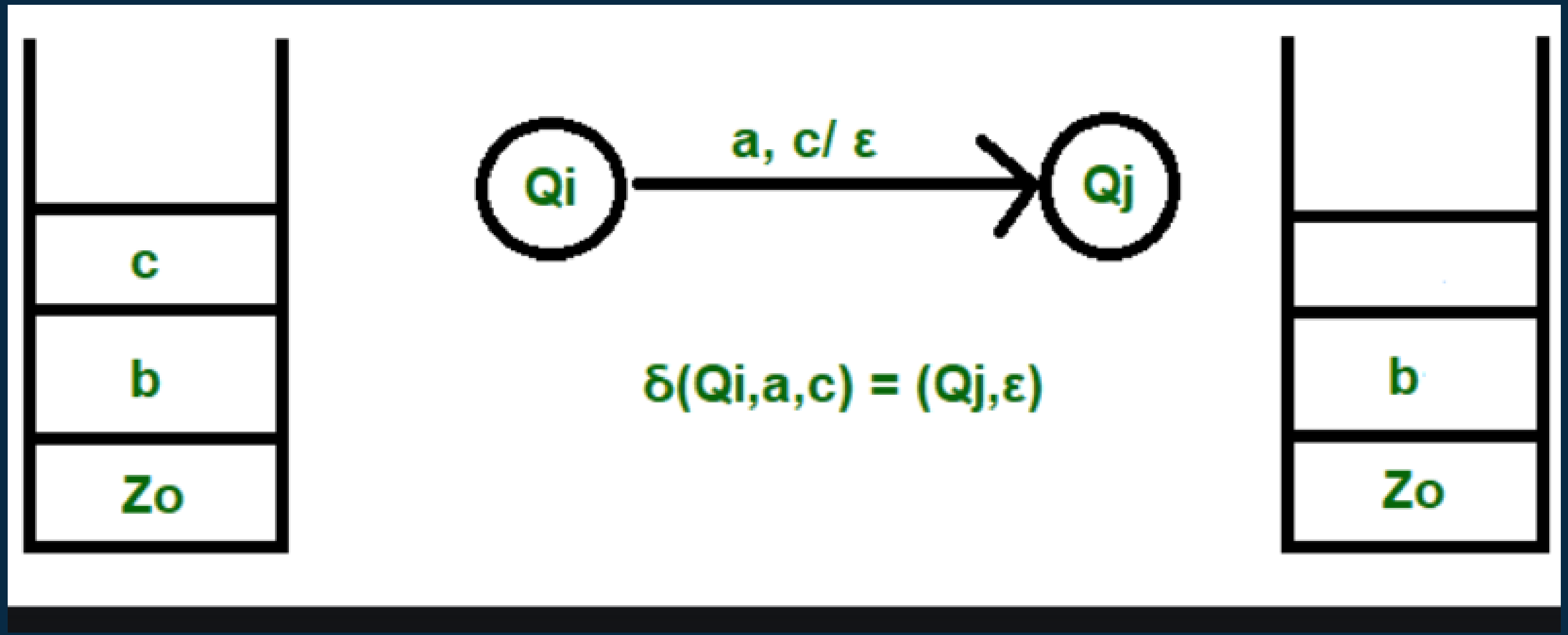


δ(Qi,a,Zo) = (Qj,aZo)

Pop

-Input, Topmost Element / Final List

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



δ(Qi,a,c) = (Qj,ε)

Algorithm of Tower of Hanoi for Code

The stack of discs is divided into two parts.

- The largest disc (the nth disc) is in one portion,
- The remaining (n-1) discs are in the other.

To get the final state for the tower of Hanoi problem for more than 2 disks or n discs for 3 pegs

- Create a function Tower with:-

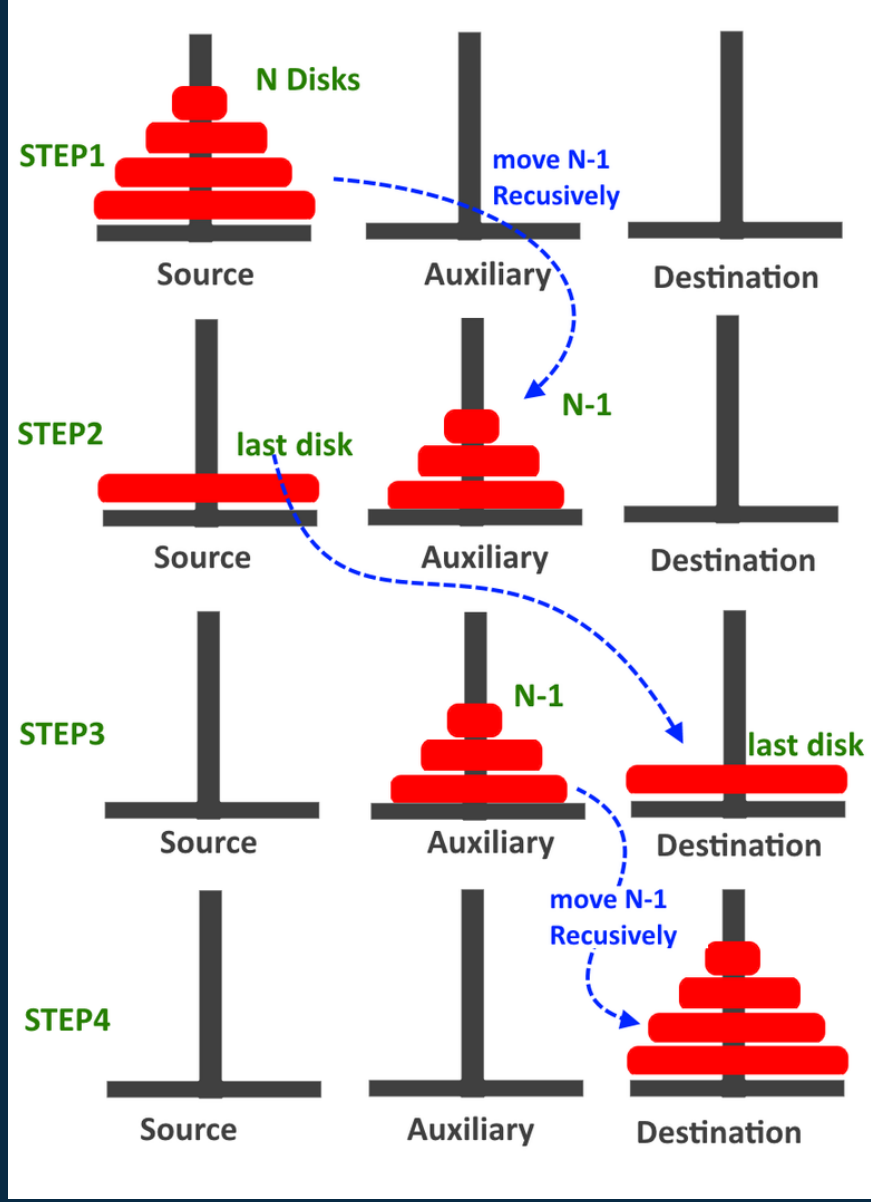
  int 'n' – for number of disks,

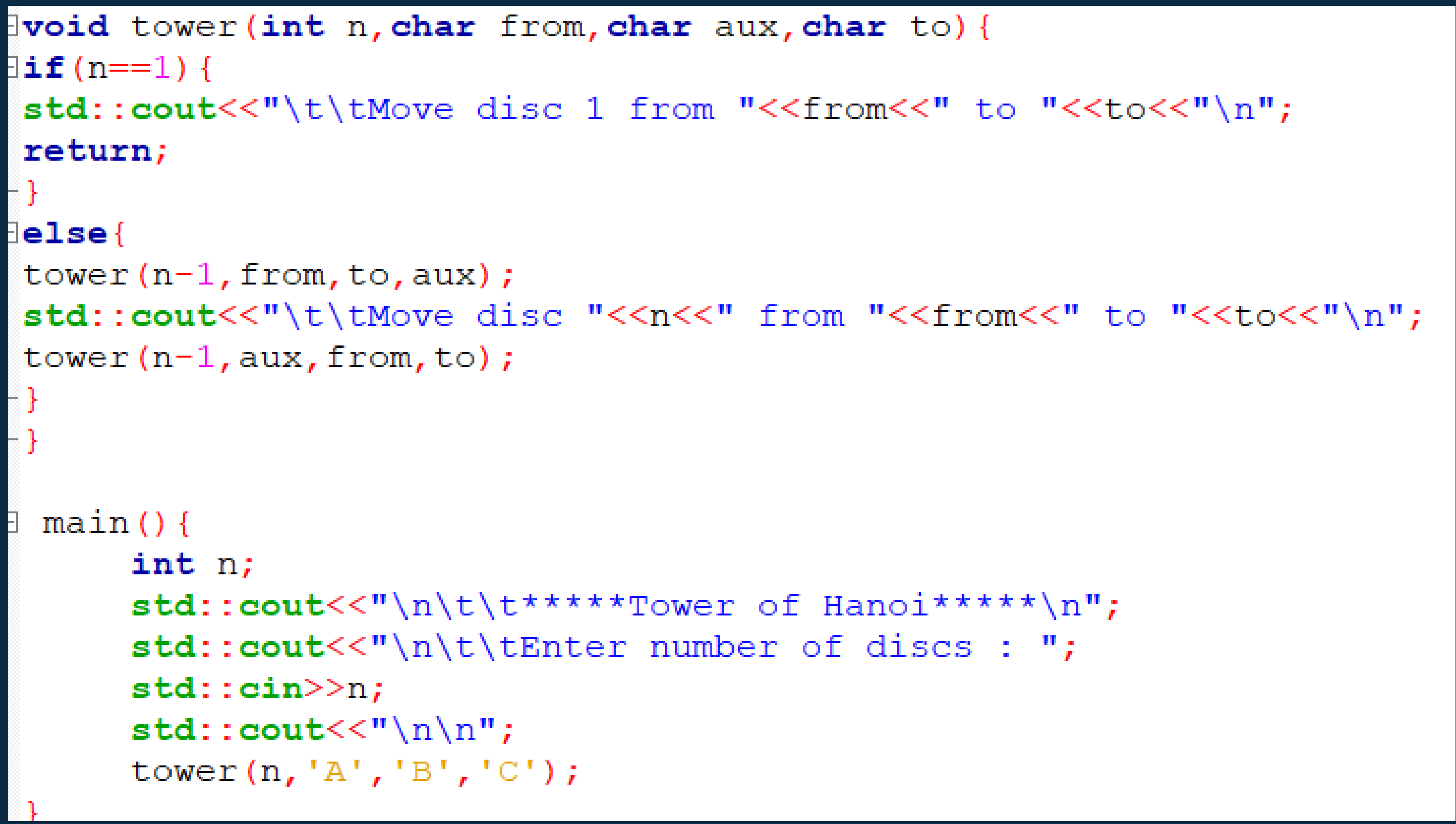
  char 'from' – for from peg,

  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.





```
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');
}
```

Relation with stack

STEP 01: MAKE 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.

Relation with stack

STEP 02: SOLVING THE TOWER OF HANOI

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 >= 1, stack(i-1) > stack(i)), where stack(k) denotes the kth element on the 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

Relation with stack

STEP 03: 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
```

CYCLIC TOWER 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 – B – C – A AND A – C – B – 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

Cyclic Hanoi has some interesting properties:

- THE MOVE PATTERNS OF TRANSFERRING A TOWER OF DISKS FROM A PEG TO ANOTHER PEG ARE SYMMETRIC WITH RESPECT TO THE CENTER POINTS.
- THE SMALLEST DISK IS THE FIRST AND LAST DISK TO MOVE.
- GROUPS OF THE SMALLEST DISK MOVES ALTERNATE WITH SINGLE MOVES OF OTHER DISKS.
- THE NUMBER OF DISK MOVES SPECIFIED BY C(N) AND A(N) IS MINIMAL.

Classical sequences are hidden behind the Hanoi sequence

PERIOD-DOUBLING SEQUENCE

A binary sequence T can be deduced from S∞ by replacing each of a, b, c by 1 and each of a, b, c by 0 (i.e., the non-barred letters by 1 and the barred letters by 0), thus obtaining

$$\begin{array}{lcccccccccccccccc} \mathscr{S}_\infty = & a & \bar{c} & b & a & c & \bar{b} & a & \bar{c} & b & \bar{a} & c & b & a & \bar{c} & b & \cdots \\ \mathscr{T} = & 1 & 0 & 1 & 1 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 1 & \cdots \end{array}$$

DOUBLE-FREE SUBSETS

Define a sequence of integers U by counting for each term of S∞ the cumulative number of the non-barred letters up to this term The sequence U is equal to the sequence of maximal sizes of a subset S of {1,2,...,n} with the property that if x is in S then 2x is not

$$\begin{array}{lcccccccccccccccc} \mathscr{S}_\infty = & a & \bar{c} & b & a & c & \bar{b} & a & \bar{c} & b & \bar{a} & c & b & a & \bar{c} & b & \cdots \\ \mathscr{U} = & 1 & 1 & 2 & 3 & 4 & 4 & 5 & 5 & 6 & 6 & 7 & 8 & 9 & 9 & 10 & \cdots \end{array}$$

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 behaviour 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

http://www.cs.cornell.edu/courses/cs312/2004fa/hw/ps2/ps2.html
http://www.cs.cornell.edu/courses/cs312/2004fa/hw/ps2/ps2.html
https://www.geeksforgeeks.org/detailed-study-of-pushdown-automata/
https://www.researchgate.net/publication/45849192_The_Tower_of_Hanoi_and_finite_automat
ahttps://arxiv.org/pdf/0905.0015.pdf

THANK YOU!!