CHARUSAT
CHAROTAR UNIVERSITY OF SCIENCE AND TECHNOLOGY

Accredited with Grade A by NAAC
Accredited with Grade A by KCG

cspit
Chandubhai S Patel Institute of Technology

CE245 : Data Structure and Algorithms

Topic: Stack Data Structure

Prepared By,
Dr. Nikita Bhatt

Stack: Data Structure

- Described as a "Last In First Out" (LIFO) OR First In Last Out (FILO) data structure.
- Operations
  - push (add item to stack)
  - pop (remove top item from stack)
- Linear Data Structure





Approaches to Implement Stack Data Structure

Array Implementation

```
int stack[max]; int top = −1
void push(int item)
{
        if(top == max − 1)
                printf("overflow");
        else
        {
                top = top + 1;
                stack[top] = item;
        }
}
```

Array Implementation

```
int stack[max]; int top = −1
int pop( )
{
        int temp;
        if(top == −1){
         printf(Underflow);        return  − 1; }
         else
         {
                temp = stack[top];
                top = top − 1;
         }
return temp;
}
```

Approaches to Implement Stack Data Structure(Continue)

Linked List Implementation

```
struct node
{
int data; struct node  * link;
}
void push(int item)
{
struct node p=(struct node *)malloc(sizeof(struct node));
if(p == NULL)
     {
          printf("Memory Error");
          return;
     }
     p → data = item;
     p → link = head;
   head = p;
 }
```

Approaches to Implement Stack Data Structure(Continue)

Linked List Implementation

```
int pop( )
{
int item;
struct node  * p;
      if(head == NULL)
      {
            printf("Underflow");
            return  − 1;
      }
      item = head → data;
      p = head;
      head = head → link;
     free(p);
     return item;
}
```

Approaches to Implement Stack Data Structure

Array Implementation

```
int stack[max]; int top = −1
void push(int item)
{
        if(top == max − 1)
                printf("overflow");
        else
        {
                top = top + 1;
                stack[top] = item;
        }
}
```

Linked List Implementation

```
struct node
{
int data; struct node  * link;
}
void push(int item)
{
struct node p=(struct node *)malloc(sizeof(struct node));
if(p == NULL)
     {
             printf("Memory Error");
             return;
     }
     p → data = item;
     p → link = head;
    head = p;
 }
```

Approaches to Implement Stack Data Structure(Continue)

Array Implementation	Linked List Implementation

int stack[max]; int top = −1
int pop( )
{
int temp;
if(top == −1) {
printf(Underflow); return − 1; }
else {
temp = stack[top];
top = top − 1; }
return temp; }
	int pop( )
	{
	int item;
	struct node * p;
	if(head == NULL) {
	printf("Underflow"); return − 1; }
	item = head → data;
	p = head;
	head = head → link;
	free(p);
	return item;
	}

Algebraic Expression

- **INFIX Expression :** x + y, x + y * z
- **POSTFIX Expression:** Also Known as Reverse Polish Notation (RPN). Examples are xy+, xyz*+.
- **PREFIX:** Also Known as Polish notation. Examples are +xy, *+xyz.
- To our surprise INFIX notations are not as simple as they seem specially while evaluating them. To evaluate an infix expression we need to consider Operators' Priority and Associative property.

  For example expression 3+5*4 evaluate to

  (a)32 i.e. (3+5)*4

  (b)23 i.e. 3+(5*4).

- Postfix and prefix expression forms do not rely on operator priorities, a tie breaker, or delimiters. So, it is easier to evaluate expressions that are in these forms.

Algebraic Expression(Continue)

Infix	PostFix	Prefix
A+B	AB+	+AB
(A+B) * (C + D)	AB+CD+*	*+AB+CD
A-B/(C*D^E)	ABCDE^*/-	-A/B*C^DE

Application 1 of Stack: Infix to Postfix Conversion(Continu

- **Infix Expression: 2*3/(2-1)+5*3**

Expression	Stack	Output
2	Empty	2
*	*	2
3	*	23
/	/	23*
(	/(	23*
2	/(	23*2
-	/(-	23*2
1	/(-	23*21
)	/	23*21-
+	+	23*21-/
5	+	23*21-/5
*	+*	23*21-/53
3	+*	23*21-/53
	Empty	23*21-/53*+

Self Assessment-III

Infix	Reverse Polish (Postfix)
((a+b)/d^((e-f)+g))	ab+def-g+^/
z+(y*x-(w/v^u)*t)*s	zyx*wvu^/t*-s*+
(a+b)*(c-d)^e*f	ab+cd-e^*f*
(a+b)*(c^(d-e)+f)-g	ab+cde-^f+*g-
a+(((b-c)*(d-e)+f)/g)^(h-j)	abc-de-*f+g/hj-^+
a/b^c+d*e-a*c	abc^/de*+ac*-
(a+b)*c+d/(b+a*c)+d	ab+c*dbac*+/+d+
((((m^n^d)-(a^c-b)/(p-q)*e)+f)-x)	mnd^^ac^b-pq-/e*-f+x-
m+(n-(a^(b-c)*d)/f)	mnabc-^d*f/-+

Algorithm to convert infix to postfix

Symbol	Precedence f	Rank r
+, –	1	–1
*, /	2	–1
a,b,c...	3	1
#	0	-

1. [Initialize the stack]
   TOP ← 1
   S[TOP] ← '#'
2. [Initialize output string and rank count]
   POLISH ← ' '
   RANK ← 0
3. [Get first input symbol]
   NEXT ← NEXTCHAR(INFIX)
4. [Translate the infix expression]
   Repeat thru step 6 while NEXT ≠ '#'
5. [Remove symbols with greater or equal precedence from stack]
   Repeat while f(NEXT) ≤ f(S[TOP])
   TEMP ← POP(S,TOP)
   POLISH ← POLISH ○ TEMP
   RANK ← RANK + r(TEMP)
   If RANK < 1
   then Write ('INVALID')
   Exit
6. [Push current symbol onto stack & obtain next input symbol]
   Call PUSH (S,TOP,NEXT)
   NEXT ← NEXTCHAR(INFIX)
7. [Remove remaining elements from stack]
   Repeat while S[TOP] ≠ '#'
   TEMP ← POP(S,TOP)
   POLISH ← POLISH ○ TEMP
   RANK ← RANK + r(TEMP)
   If RANK < 1
   then Write ('INVALID')
   Exit
8. [Is the expression valid?]
   If RANK = 1
   then Write('VALID')
   else Write('INVALID')
   Exit

Application 2 of Stack: Evaluation of Expression

- 2 3 * 2 1 - / 5 4 1 - * +

	Operand1	Operand2	Value	Stack
2				2
3				2,3
*	2	3	6	6
2				6,2
1				6,2,1
-	2	1	1	6,1
/	6	1	6	6
5				6,5
4				6,5,4
1				6,5,4,1
-	4	1	3	6,5,3
*	5	3	15	6,15
+	6	15	21	21

Application 2 of Stack: Evaluation of Expression (continue)

- + / * 2 3 - 2 1 * 5 - 4 1

	Operand1	Operand2	Value	Stack
1				1
4				1,4
-	4	1	3	3
5				3,5
*	5	3	15	15
1				15,1
2				15,1,2
-	2	1	1	15,1
3				15,1,3
2				15,1,3,2
*	2	3	6	15,1,6
/	6	1	6	15,6
+	6	15	21	21

Application 2 of Stack: Evaluation of Expression (continue)

- **Algorithm**

WHILE more input items exist

{

If symb is an operand

then push (S, Top, symb)

else //symbol is an operator

{

Opnd2=pop(S, Top);

Opnd1=pop(S, Top);

Value = opnd1 operator opnd2 // perform operation as per theoperator

Push(S, Top, value);

} //End of else

} // end while

Result = pop (S, Top);

	Operand1	Operand2	Value	Stack
2				2
3				2,3
*	2	3	6	6
2				6,2
1				6,2,1
-	2	1	1	6,1
/	6	1	6	6
5				6,5
4				6,5,4
1				6,5,4,1
-	4	1	3	6,5,3
*	5	3	15	6,15
+	6	15	21	21

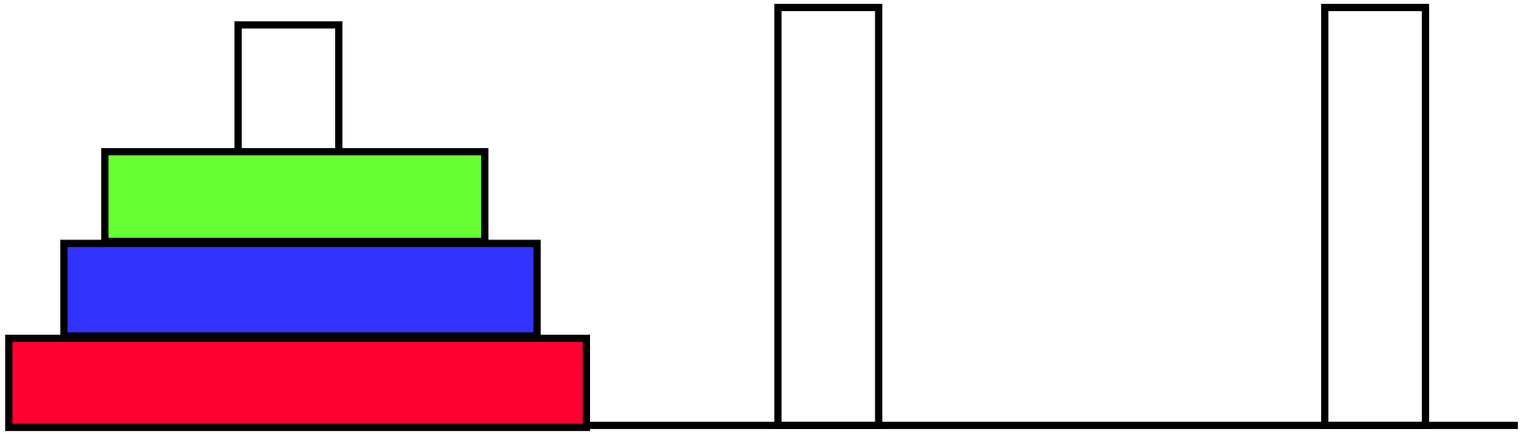
Application 3 of Stack: Function Call

```
public int f(int a){
        if (a==1)
           return(1);
        else
           return(a * f( a-1));
}
```

Application 4 of Stack: Tower of Hanoi

The problem is as follows:

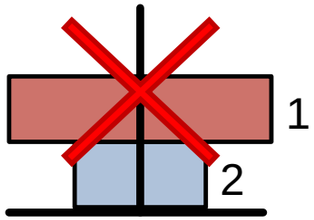
- *N* discs of decreasing size stacked on one needle & two empty needles are given.
- It is required to arrange all the discs onto a second needle in decreasing order of size.
- The Third needle may be used as temporary storage.

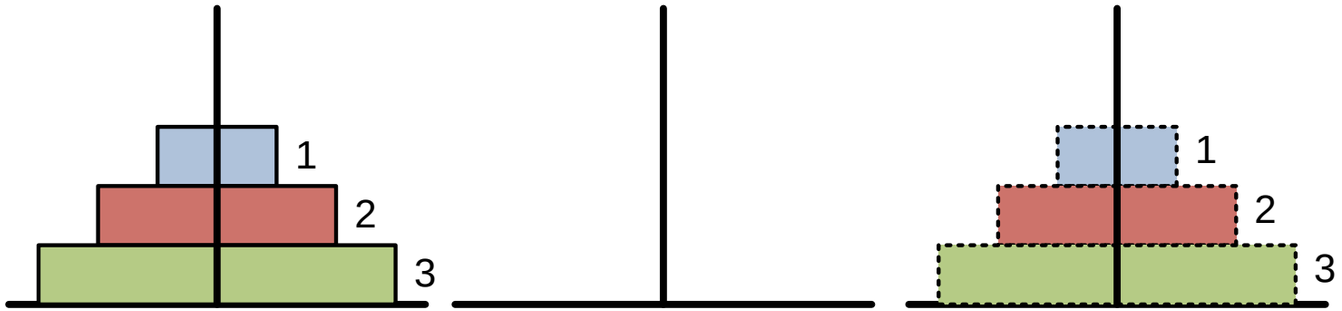


Application 4 of Stack: Tower of Hanoi (Continue)

The movement of the discs is restricted by the following rules:

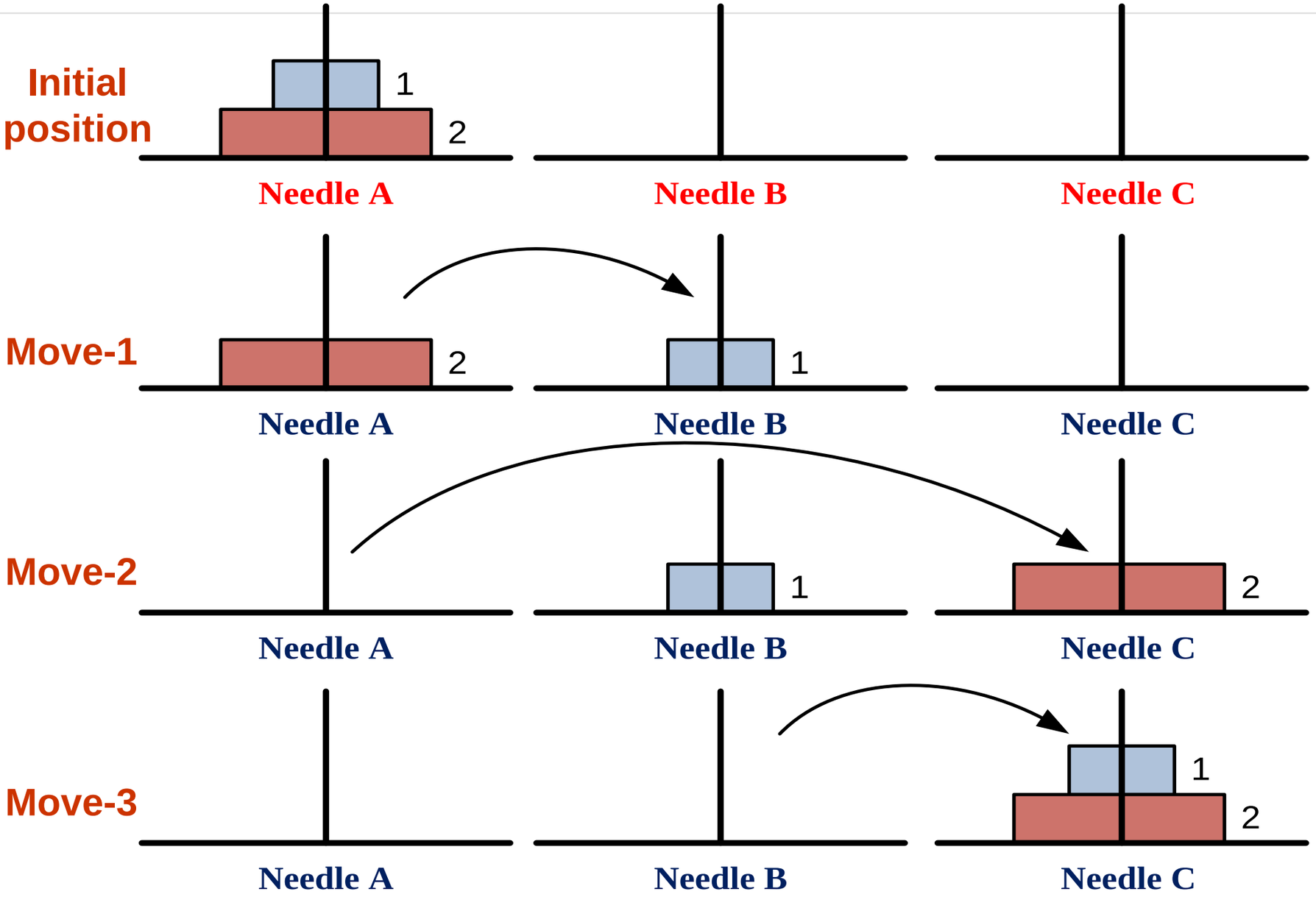
1. Only one disc may be moved at a time.
2. A disc may be moved from any needle to any other.
3. At no time may a larger disc rest upon a smaller disc.



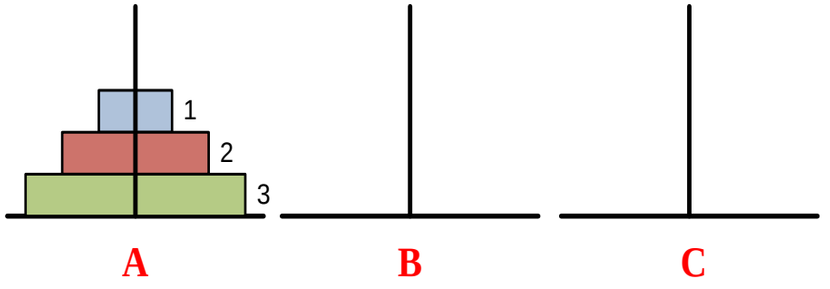


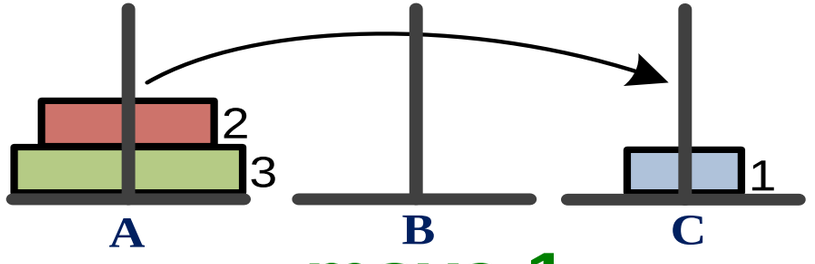
Needle A (start of problem) **Needle B (intermediate)** **Needle C (completion of problem)**

Solution of the problem (N=2)

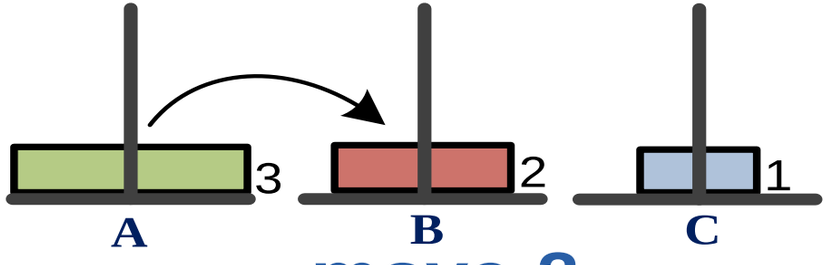


N = 3

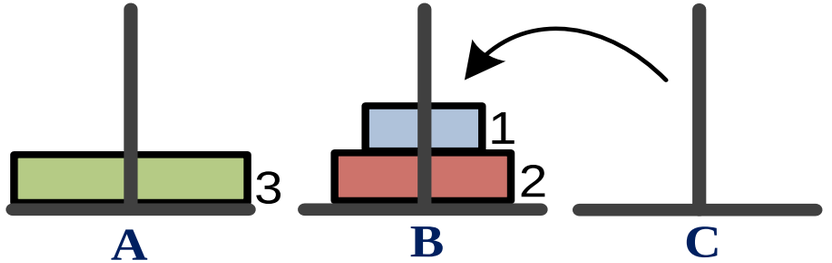




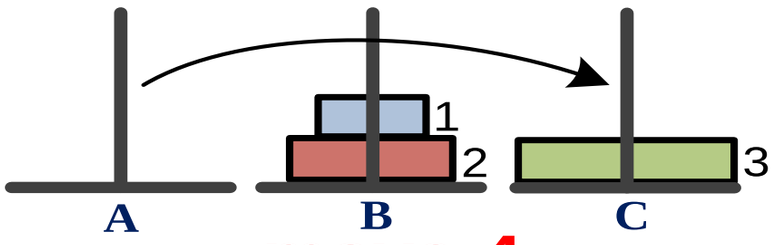
move-1



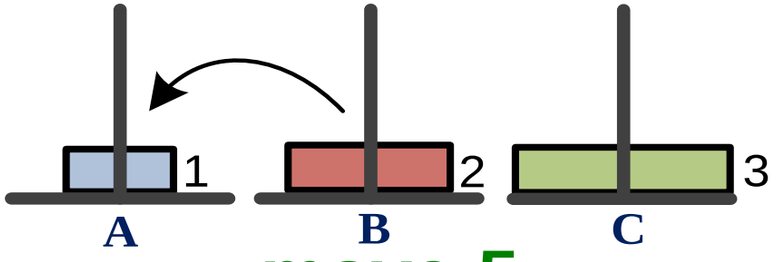
move-2



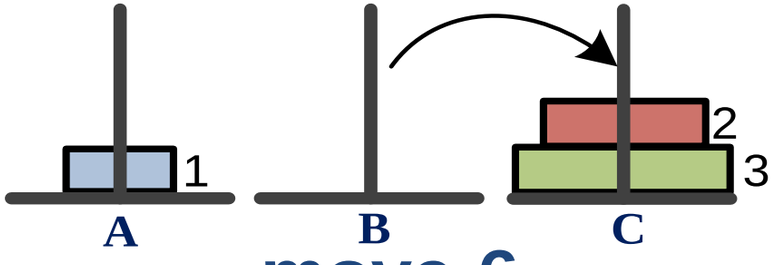
move-3



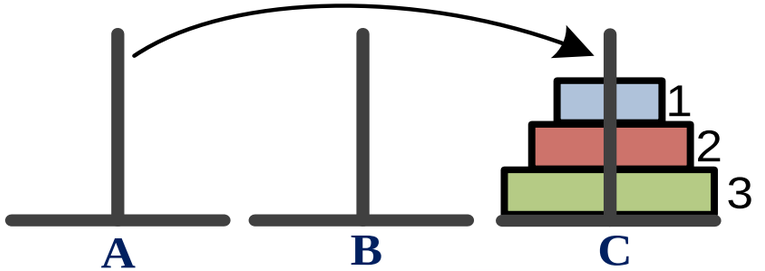
move-4



move-5



move-6



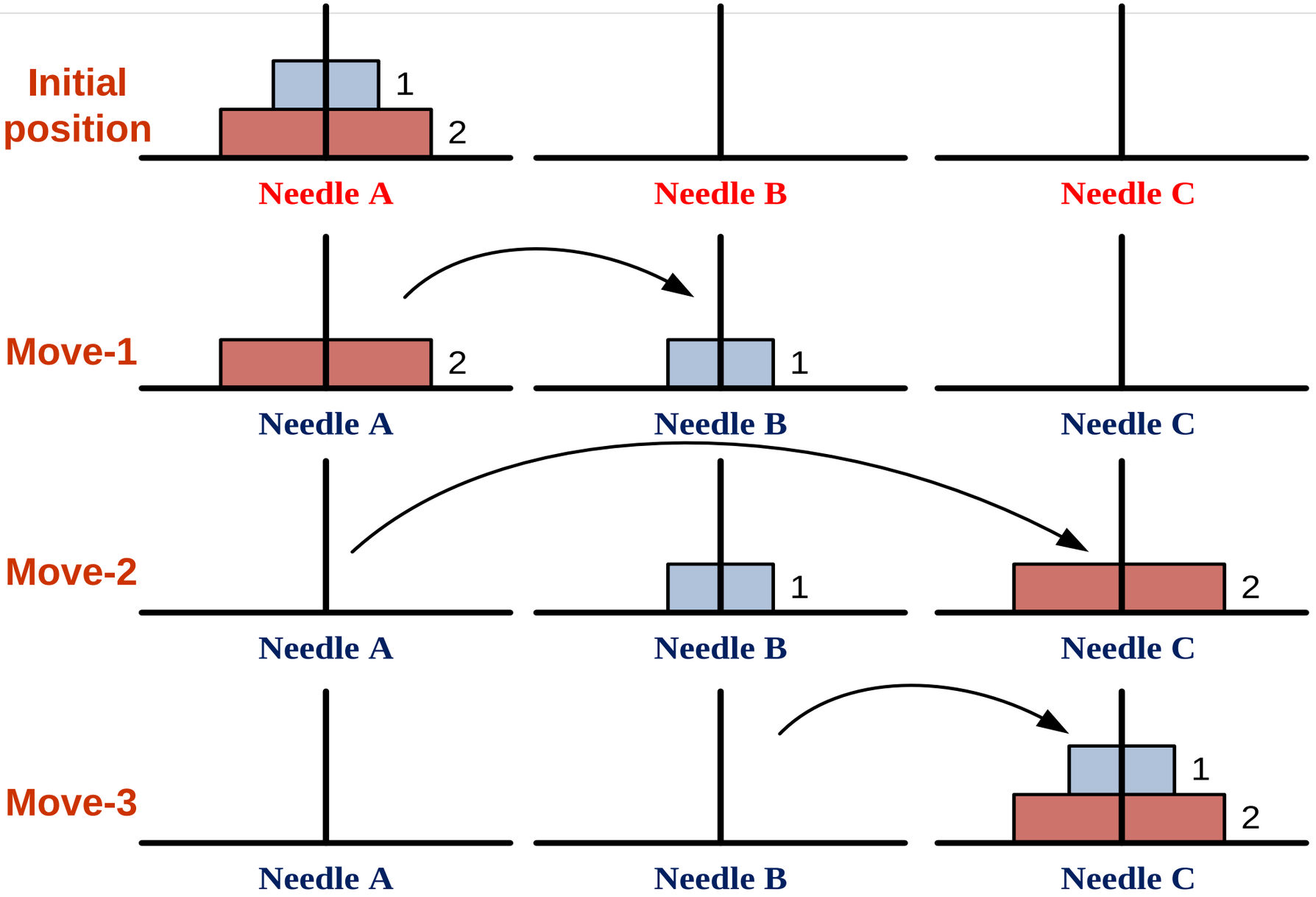
move-7

Solution

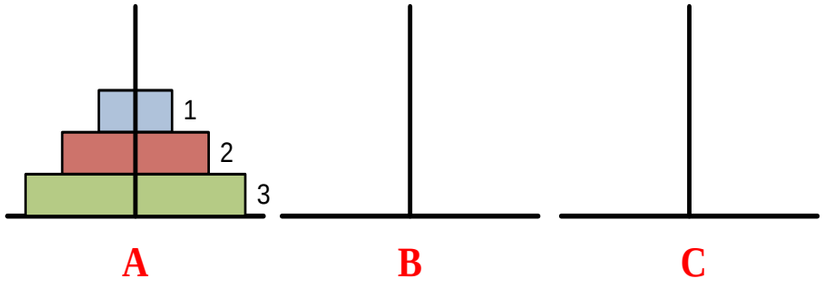
- **The solution of this problem is:**
- To move **one** disc, move it from A➞C.
- To move **two** discs, move the first disc A➞B, move the second from A➞C, then move the disc from B➞C.
- In general, the solution of the problem moving ***N*** discs from A to C has three steps:

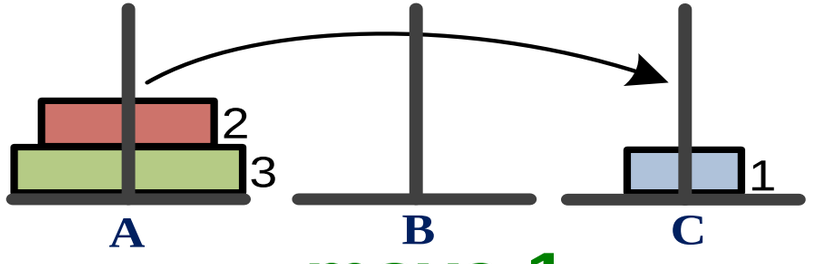
1. Move ***N* – 1** discs from A➞B. (**source➞interm**)
2. Move ***N*th** disc from A➞C. (**source➞destination**)
3. Move ***N* – 1** discs from B➞C. (**interm➞dest**)

Solution of the problem (N=2)

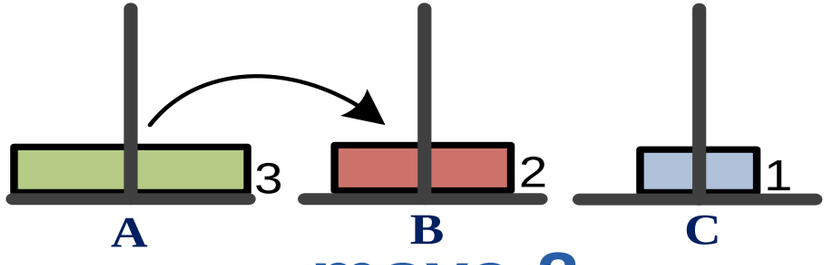


N = 3

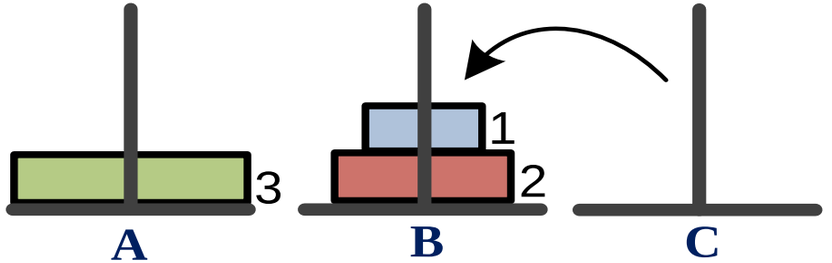




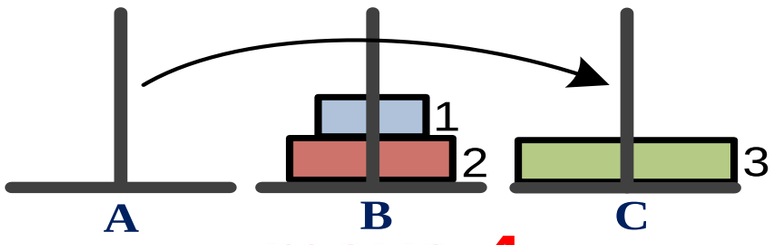
move-1



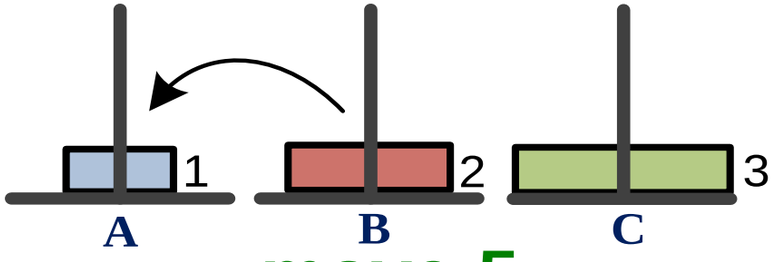
move-2



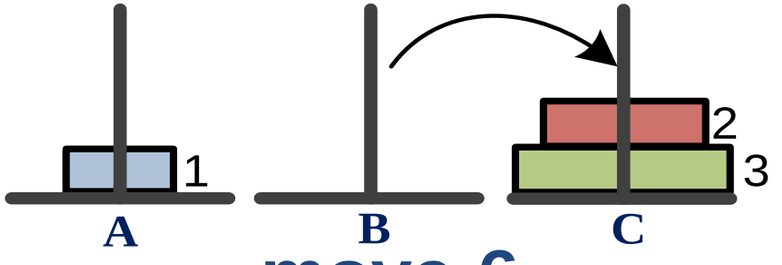
move-3



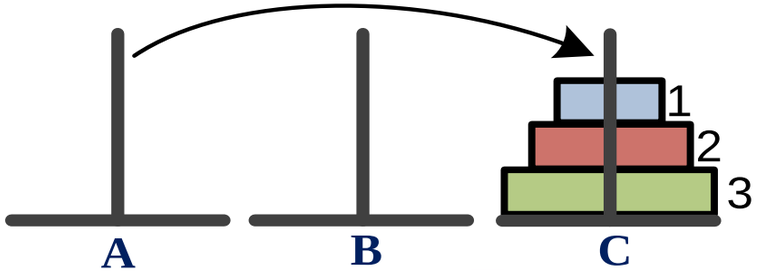
move-4



move-5

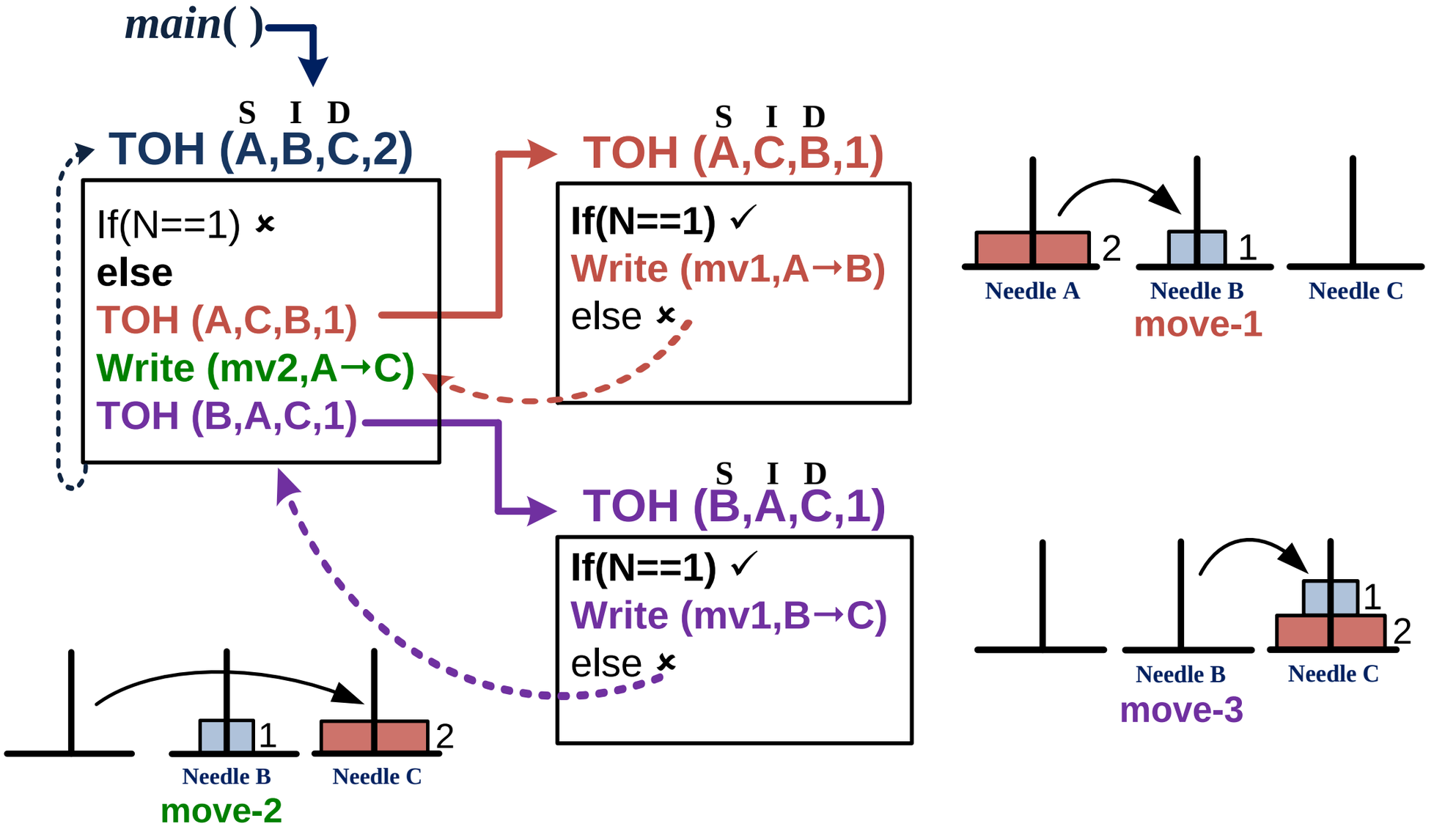


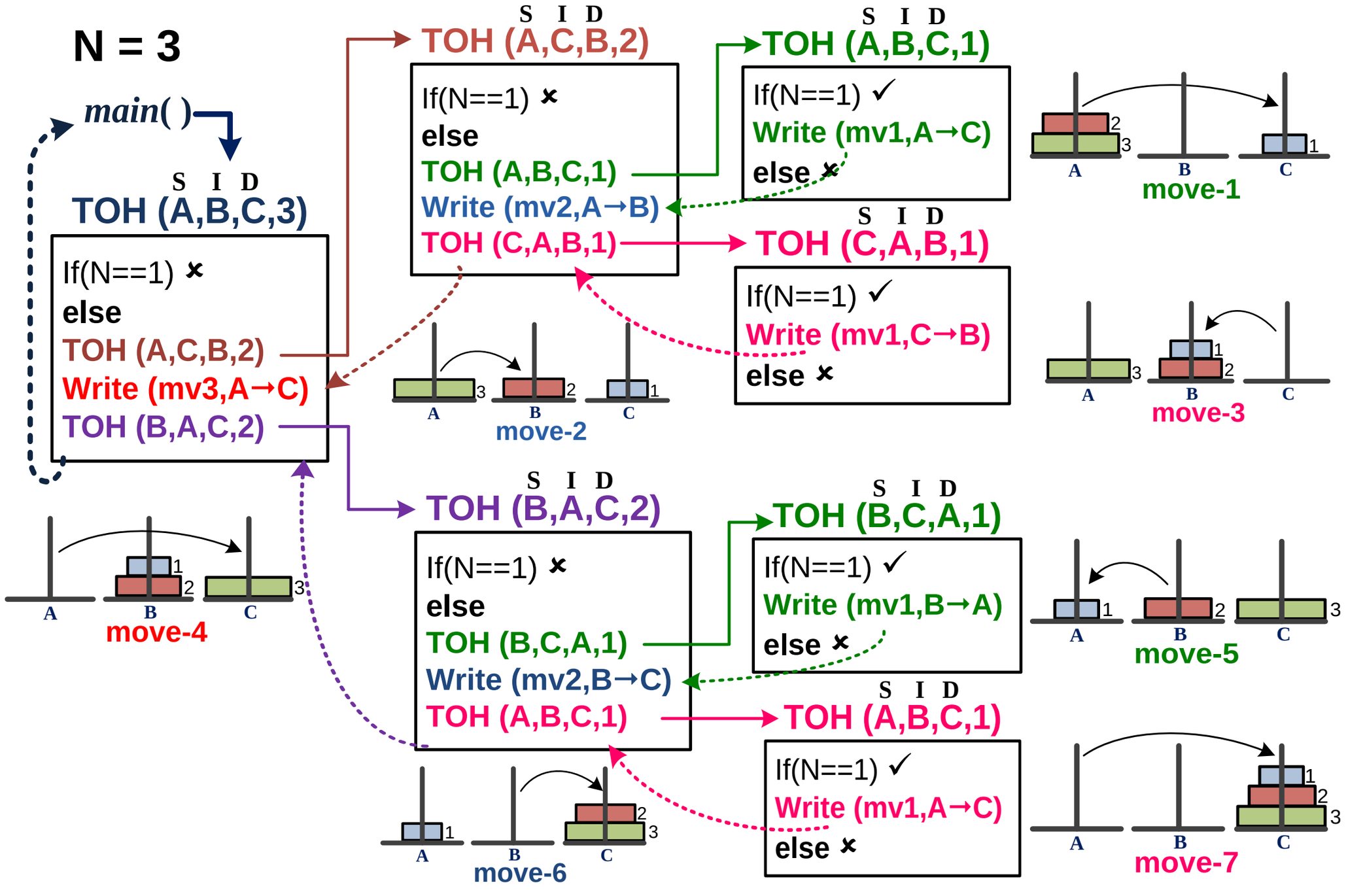
move-6



move-7

Graphical solution of the problem (*N*=2)




N = 3
main( )
S I D
TOH (A,B,C,3)
If(N==1) ✘
else
TOH (A,C,B,2)
Write (mv3,A→C)
TOH (B,A,C,2)
S I D
TOH (A,C,B,2)
If(N==1) ✘
else
TOH (A,B,C,1)
Write (mv2,A→B)
TOH (C,A,B,1)
S I D
TOH (A,B,C,1)
If(N==1) ✓
Write (mv1,A→C)
else ✘
S I D
TOH (C,A,B,1)
If(N==1) ✓
Write (mv1,C→B)
else ✘
S I D
TOH (B,A,C,2)
If(N==1) ✘
else
TOH (B,C,A,1)
Write (mv2,B→C)
TOH (A,B,C,1)
S I D
TOH (B,C,A,1)
If(N==1) ✓
Write (mv1,B→A)
else ✘
S I D
TOH (A,B,C,1)
If(N==1) ✓
Write (mv1,A→C)
else ✘
A B C
move-1
A B C
move-2
A B C
move-3
A B C
move-4
A B C
move-5
A B C
move-6
A B C
move-7

Algorithm

- **ToH(source, intermediate, destination, no.of disc)**

  S I D

- **ToH (A, B, C, N)**

```
if ( N==1 )
    Write (Move Disc N from S➞D);
else
{       S  I  D
   ToH (A, C, B, N-1);

      Write (Move Disc N from S➞D);
        S  I  D
   ToH (B, A, C, N-1);
}
```

Stack: Example of Operations

The following sequence of operations is performed on a stack:

push(10),push(20),pop(),push(10),push(20),pop(),pop(),pop(),push(20), pop(). The sequence of values popped out is:

(a) 20,10,20,10,20

(b) 20,20,10,10,20

(c) 10,20,20,10,20

(d) 20,20,10,20,10

Answer : (b)

Gate 1994

Which of the following permutations can be obtained in the output (in the same order) using a stack assuming that the input is the sequence 1, 2, 3, 4, 5 in that order?

(a) 3, 4, 5, 1, 2

(b) 3, 4, 5, 2, 1

(c) 1, 5, 2, 3, 4

(d) 5, 4, 3, 1, 2

Answer : (b)

Self Assessment-I

Consider the following pseudocode that uses a stack

```
 declare a stack of characters
 while ( there are more characters in the word to read)
{
   read a character
   push the character on the stack
}
 while ( the stack is not empty )
{
   pop a character off the stack
   write the character to the screen
}
```

What is output for input "geeksquiz"?

(a) geeksquizgeeksquiz (b) ziuqskeeg (c) geeksquiz (d) ziuqskeegziuqskeeg

Answer : (b)

Self Assessment-IV (Continue)

2. The prefix form of an infix expression p + q - r * t is

3. Consider the following operation performed on a stack of size 5. Push(1); Pop(); Push(2); Push(3); Pop(); Push(4); Pop(); Pop(); Push(5); After the completion of all operation, the no of element present on stack are :

Self Assessment-II

Consider a empty stack of integers. Let the numbers 1,2,3,4,5,6 be pushed on to this stack only in the order they appeared from left to right. Let S indicates a push and X indicate a pop operation. Can they be permuted in to the order 325641(output) and order 154623?

If a permutation is possible give the order string of operations.

(Hint: SSSSSSXXXXXX outputs 654321)

Solution:

- SSSXXSSXSXXX outputs 325641.

- 154623 cannot be output as 2 is pushed much before 3 so can appear only after 3 in output.

Self Assessment-IV (Continue)

4. Consider the usual implementation of parentheses balancing program using stack. What is the maximum number of parentheses that will appear on stack at any instance of time during the analysis of (( ) (( )) (( )))?

5. The following postfix expression with single digit operands in evaluated using a stack 8 2 3 ^ / 2 3 * 5 1 * -. The top two elements of the stack after the first * is evaluated are

Self Assessment-IV (Continue)

6. The following postfix expression, containing single digit operands and arithmetic operators + and *, is evaluated using a stack. 5 2 * 3 4 + 5 2 * * + Show the contents of the stack.

(i) After evaluating 5 2 * 3 4 +

(ii) After evaluating 5 2 * 3 4 + 5 2

(iii) At the end of evaluation

Self Assessment-IV (Continue)

7. The postfix expression for the infix expression a + b * (c + d) / f + d * e is

Self Assessment-IV (Continue)

8. Perform Postfix Conversion for following expression:

((((4^1^6)-(3^2-8)/(7-5)*2)+9)-3)

Self Assessment-IV (Continue)

9. Evaluation for following expression:

((((4^1^6)-(3^2-8)/(7-5)*2)+9)-3)

Gate 2004

The best data structure to check whether an arithmetic expression has balanced parentheses is a

(A) queue

(B) stack

(C) tree

(D) list

Answer: (B)

Self Assessment-IV

Following is C like pseudo code of a function that takes a number as an argument, and uses a stack S to do processing.

```
void fun(int n)
{
 Stack S;  // Say it creates an empty stack S
while (n >0)
{
        push(&S, n%2);
        n = n/2;
  }
while (!isEmpty(&S))
printf("%d ", pop(&S)); // pop an element from S and print it
}
```

Self Assessment-IV (Continue)

What does the above function do in general?

(A) Prints binary representation of n in reverse order

(B) Prints binary representation of n

(C) Prints the value of Logn

(D) Prints the value of Logn in reverse order

Answer : (B)

Applications of Stack

- To convert Infix expression to postfix and prefix notation.
- To evaluate postfix and prefix expressions.
- To perform undo and redo operations in Editors.
- To check the syntax in programing language.
  - Matching Parenthesis, Match in HTML tags < >, Matching if else
- To perform Forward – backward surfing in browser- check the history of visited websites.
- To use in virtual machines like JVM.
- Real life examples:
  - Message logs
  - Call logs
  - E-mails
  - Google photos' [any gallery]
  - YouTube downloads and Notifications (latest appears first)
  - Scratch card's earned after Google pay transaction