

UNIT-1

Prepared by:

Manjula L, Assistant Professor

Dept. of CSE, MSRIT

Text Book: Horowitz, Sahni, Anderson-Freed: Fundamentals of Data Structures in C, 2nd Edition, Universities Press, 2008.

Contents

Basic Concepts:

- Pointers and Dynamic Memory Allocation
- Algorithm Specification
- Data Abstraction.

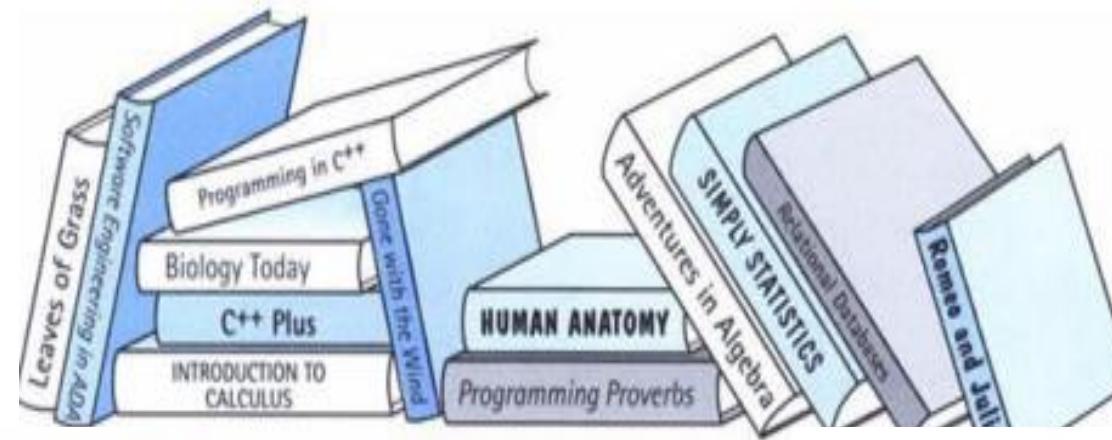
Arrays and Structures:

- Arrays
- Dynamically Allocated Arrays
- Structures and Unions
- Polynomials
- Sparse Matrices
- Representation of Multidimensional Arrays
- Strings

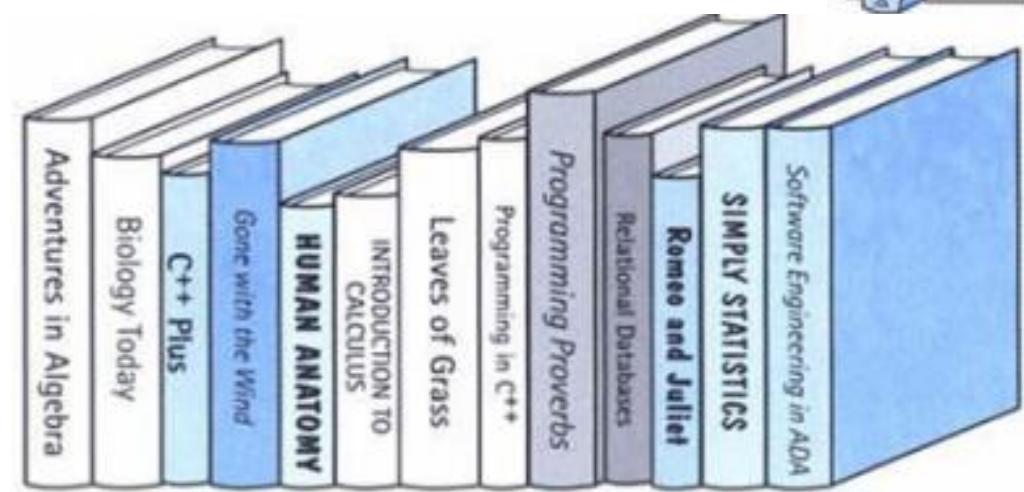
Data Structure

Storing, Organising, Manipulating, Retrieving.

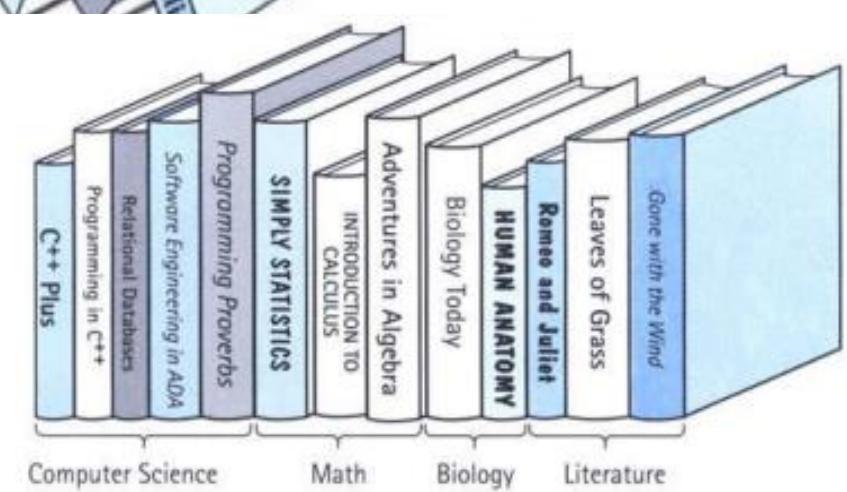
- A data structure is a specialized format for organizing, processing, retrieving and storing data. There are several basic and advanced types of data structures, all designed to arrange data to suit a specific purpose. Data structures make it easy for users to access and work with the data they need in appropriate ways.
- Ex: Library



All over the place (Unordered)



Alphabetical order by title



Ordered by subject

Pointers

- A pointer is a constant or variable that contains the address of another variable. It can be used to access data
- Once a pointer has been assigned the address of a variable, the value of the variable can be accessed using the indirection operator (*).
- * (dereference operator) is inverse of &

int a;

int *p;

p=&a;

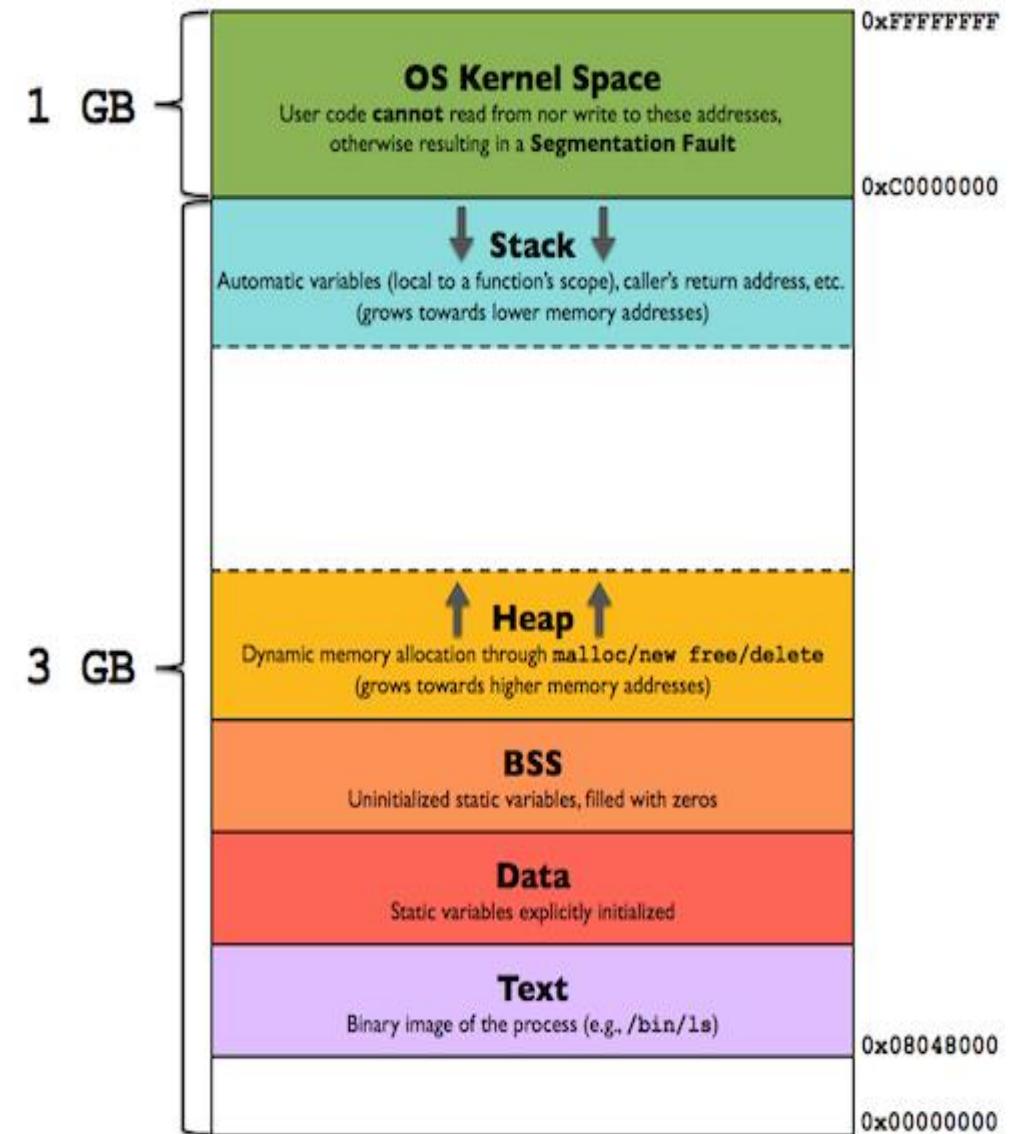
***p = * (&a) = a**

Problem : When you write a program you may not know how much space you will need (array size not known till run time) or you allocate some very large area that may never be required.

To solve this problem , C provides a mechanism called heap for allocating storage at run time.

How to use heap?

By dynamic memory allocation



Dynamic Memory Allocation

- Dynamic memory allocation is a process that allocates memory for variables and data structures at **runtime**, when the program requests it. This allows for flexibility and efficiency, as the size and location of memory blocks can be changed according to the program logic and data size.
- In C, dynamic memory is allocated from the heap using some standard library functions. The two key dynamic memory functions are malloc() and free(). The malloc() function takes a single parameter, which is the size of the requested memory area in bytes. It returns a pointer to the allocated memory.

malloc () – stdlib.h

- The “malloc” or “memory allocation” method in C is used to dynamically allocate a single large block of memory with the specified size. It returns a pointer of type void which can be cast into a pointer of any form.
- If space is insufficient, allocation fails and returns a NULL pointer
- Syntax:

ptr = (cast-type*) malloc(byte-size)

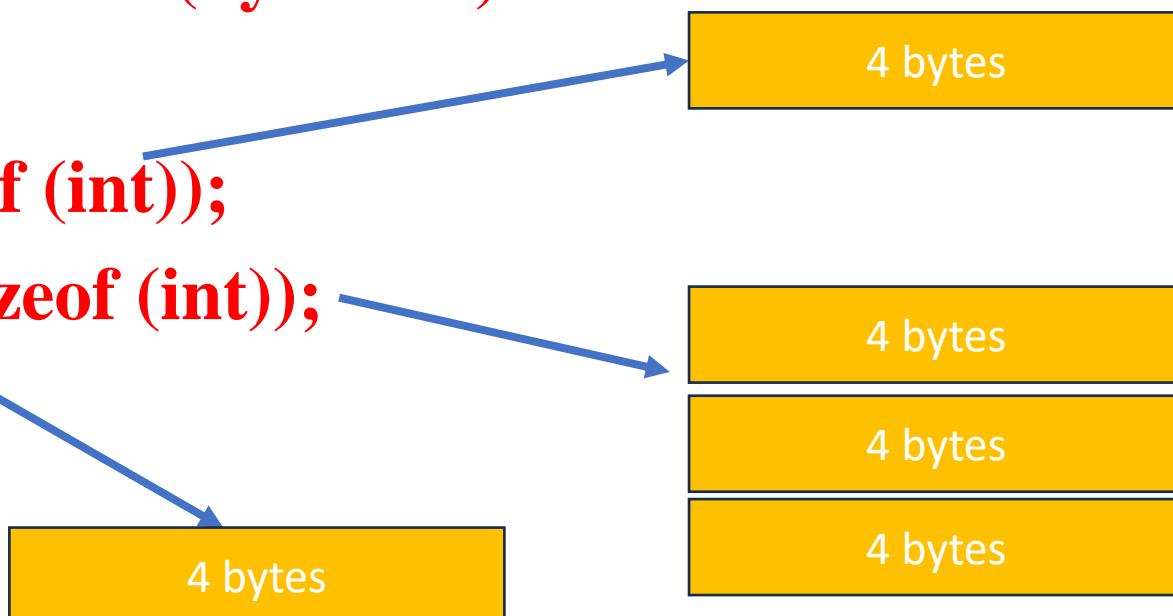
If int= 4 bytes

Examples:

ptr= (int *) malloc (sizeof (int));

ptr= (int *) malloc (3* sizeof (int));

ptr = malloc(sizeof (int));



```
int *pi;  
pi=(int *)malloc(sizeof(int));  
if(pi==NULL)  
{  
    printf("memory space is not avail!");  
    exit(0);  
}  
*pi=567;  
printf("%d",*pi);  
free(pi);
```

*Simple
example of
malloc()*

```
int* ptr;
int n, i;

// Get the number of elements for the array
printf("Enter number of elements:");
scanf("%d",&n);
printf("Entered number of elements: %d\n", n);

// Dynamically allocate memory using malloc()
ptr = (int*)malloc(n * sizeof(int));

// Check if the memory has been successfully allocated by malloc
if (ptr == NULL) {
    printf("Memory not allocated.\n");
    exit(0);
}
else { // Memory has been successfully allocated
    printf("Memory successfully allocated using malloc.\n");
}
```

Macro definition for memory allocation

```
#define MALLOC(p,s)
if(!((p)=malloc(s)))
{
printf("cant allocate memory");
exit(0);
}
// Calling macro in main function
int *pi;
MALLOC(pi,sizeof(int))
```

Exercises

1. Write a C program to add two numbers using pointers.
Assign variables to pointers and do addition.
2. Write a C program to add two numbers using pointers.
Allocate space to pointers and then assign the values and do addition.
3. Write a C program to do addition of two numbers using
pointers and Macro definition.

Algorithm

Definition

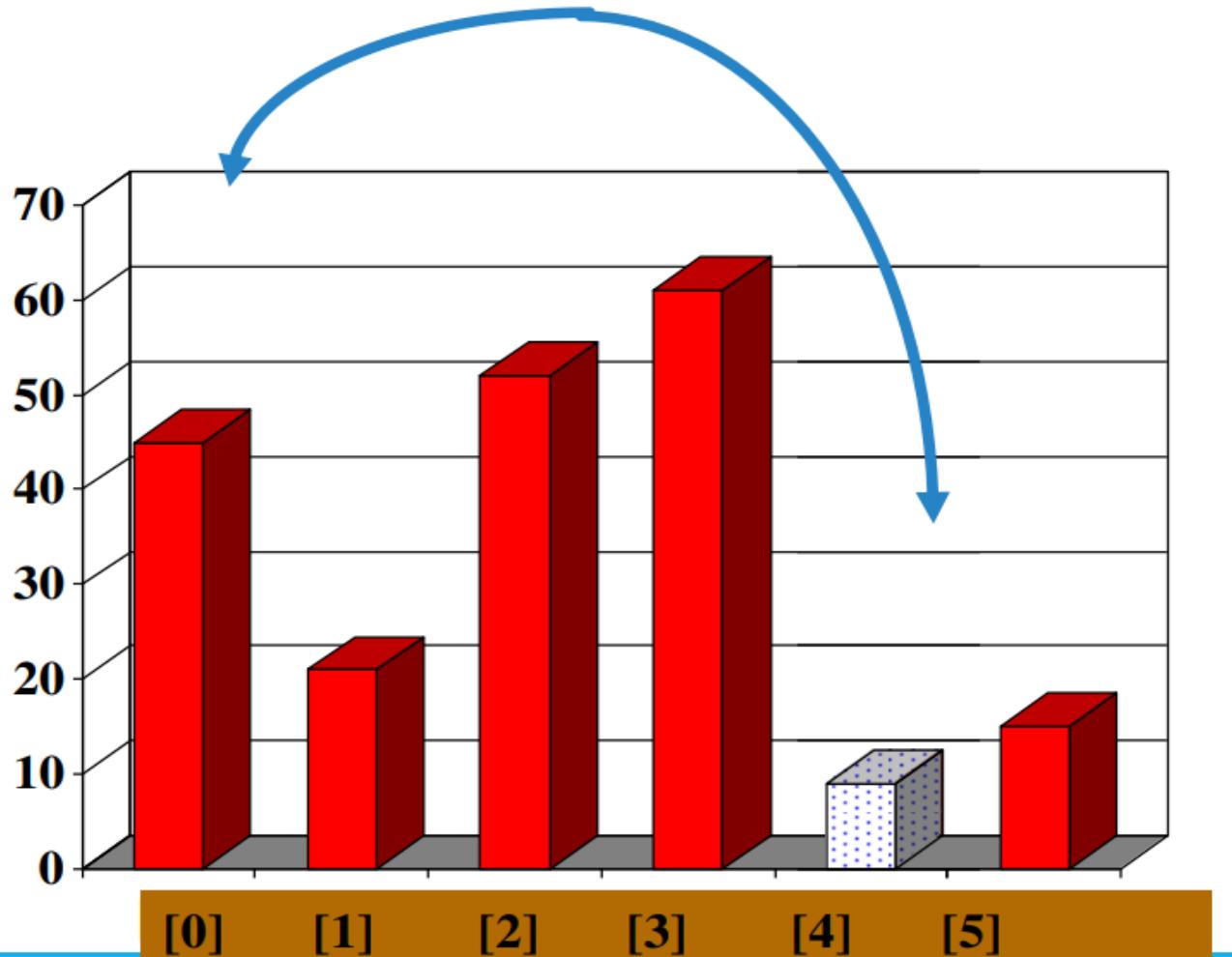
An algorithm is a finite set of instructions that accomplishes a particular task.

Criteria

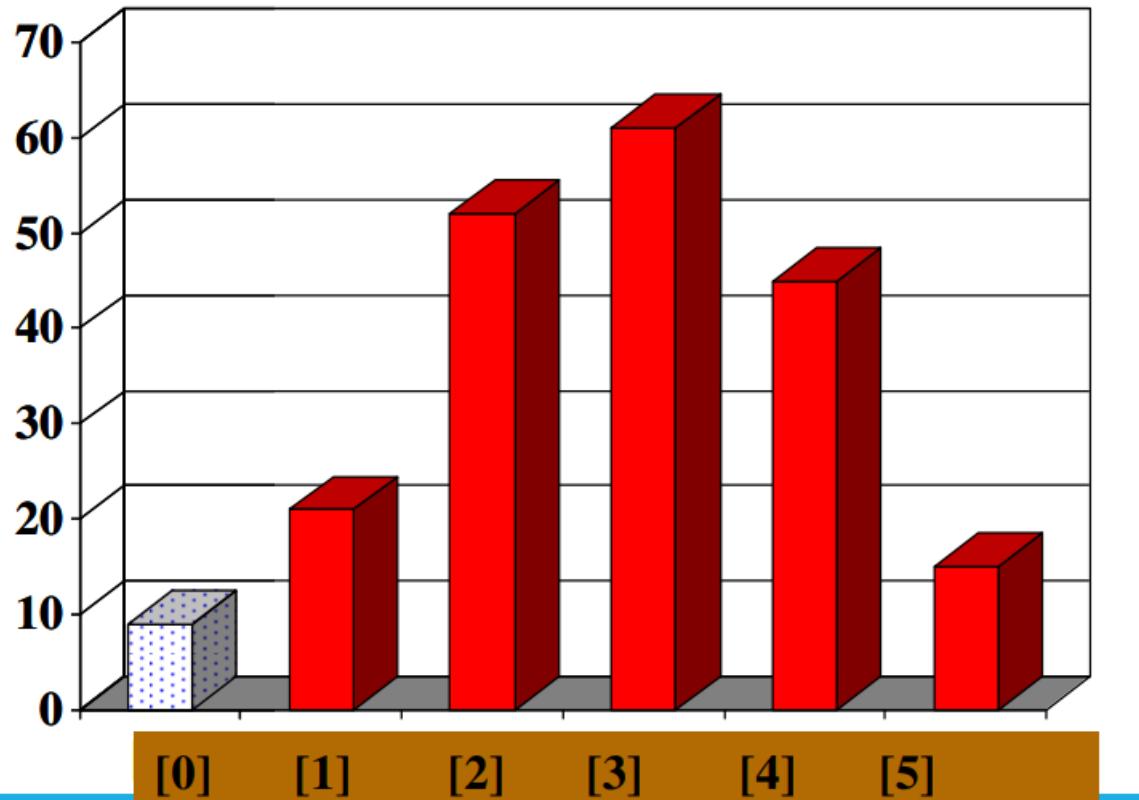
- **input** 0 or more I/Ps
- **output** Atleast 1 o/p is produced
- **definiteness:** clear and unambiguous
- **finiteness:** terminate after a finite number of steps
- **effectiveness:** instruction is basic enough to be carried out

The Selection Sort Algorithm

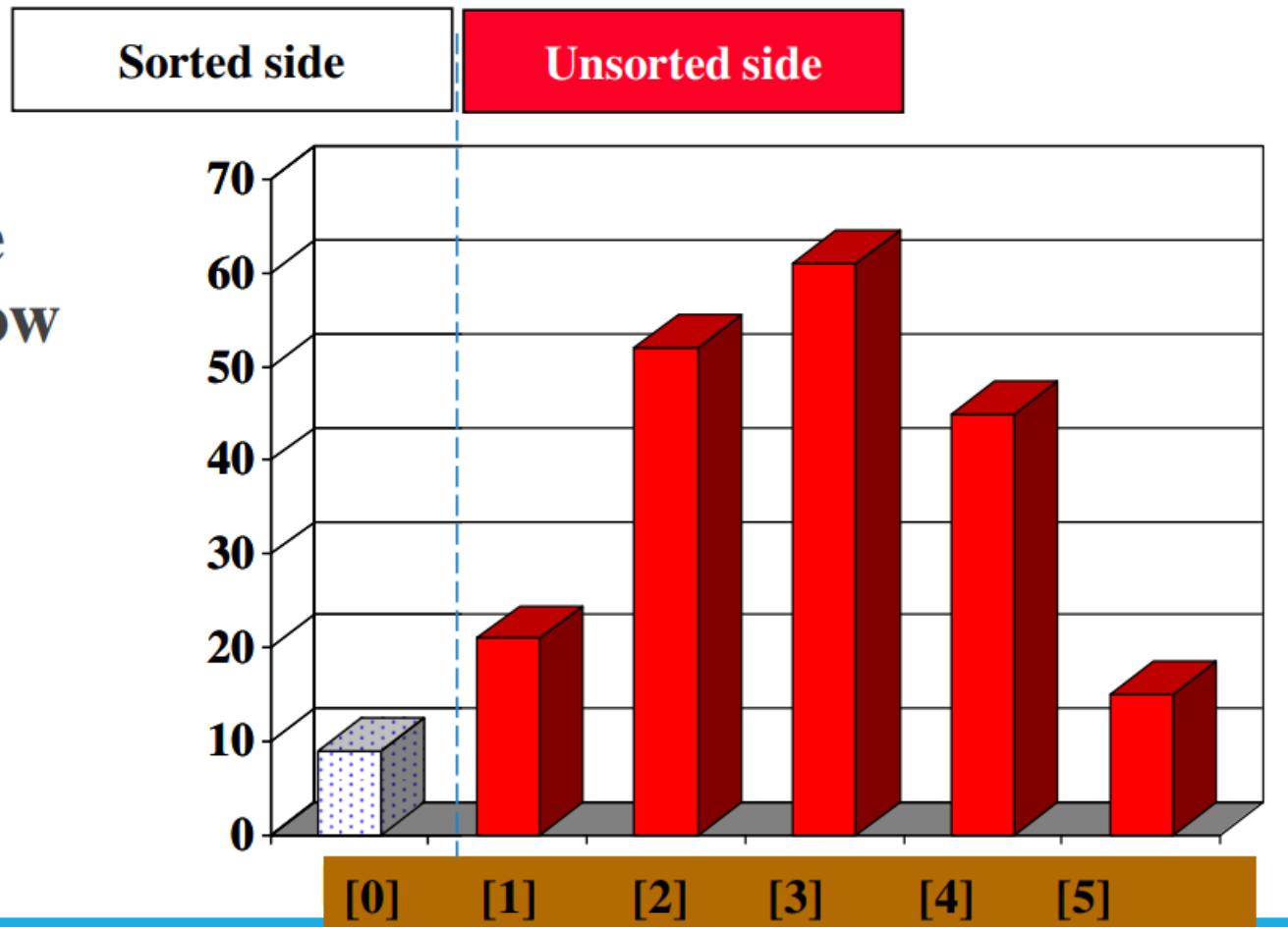
**Swap the
smallest entry
with the first
entry.**



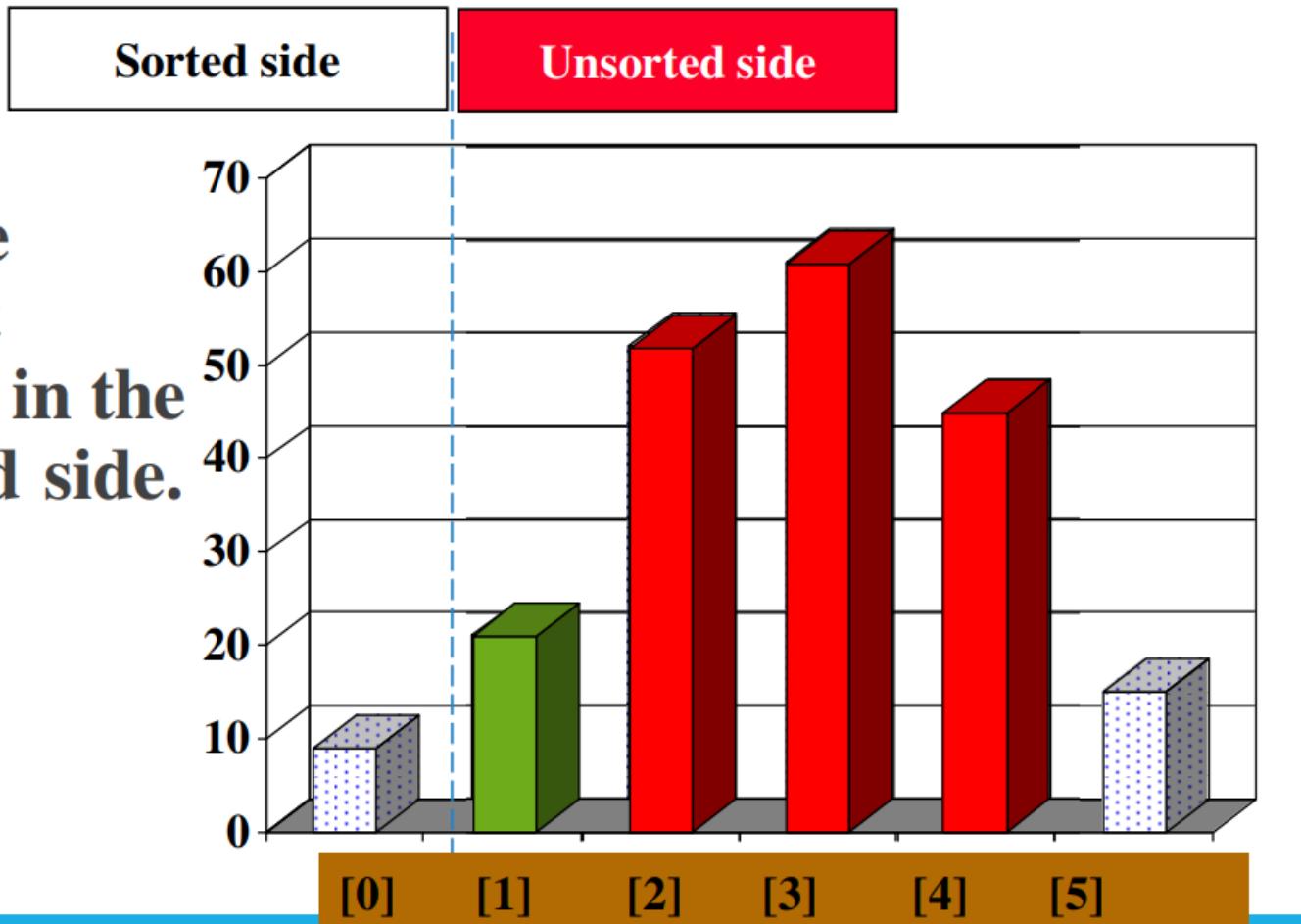
**Swap the
smallest entry
with the first
entry.**

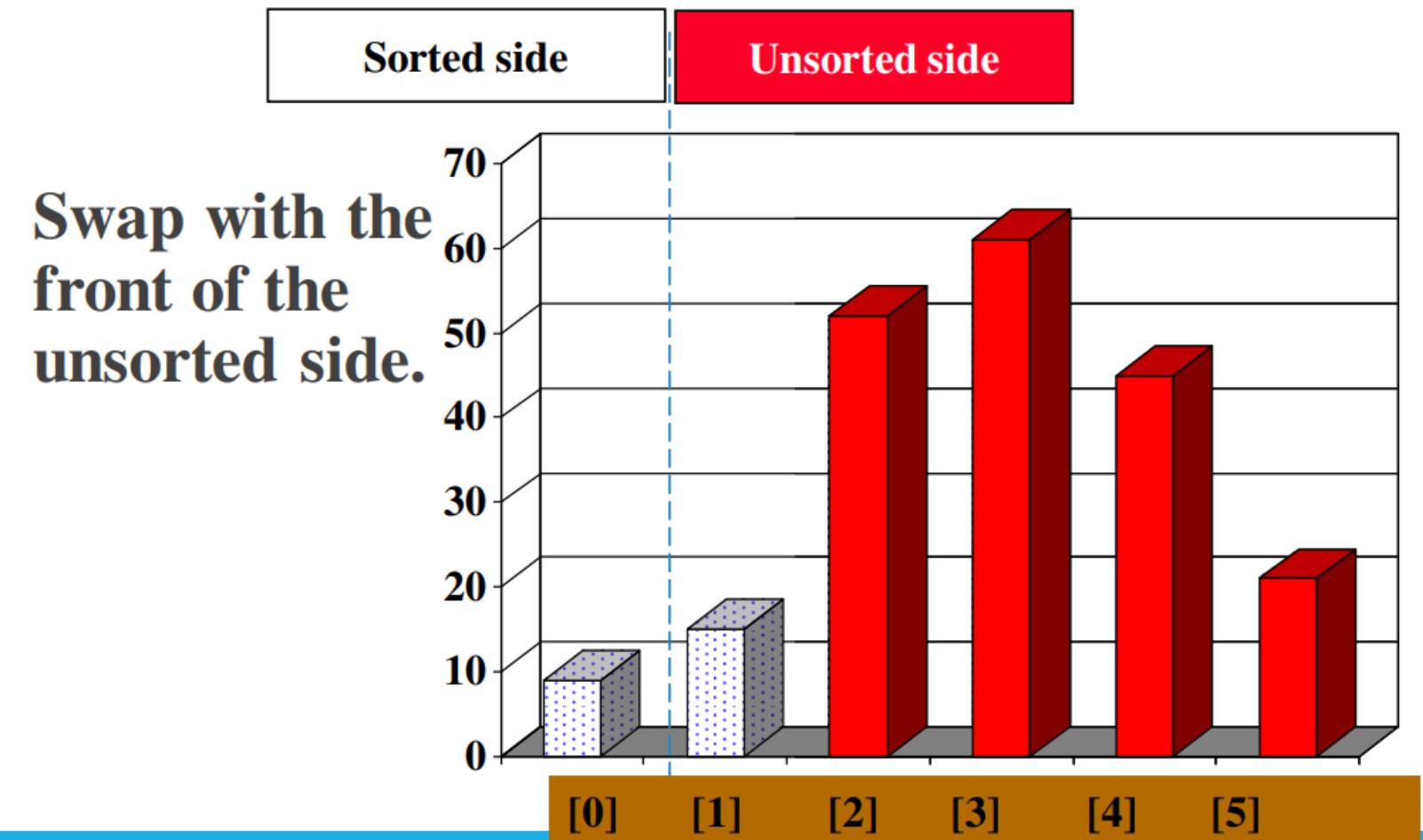


Part of the array is now sorted.

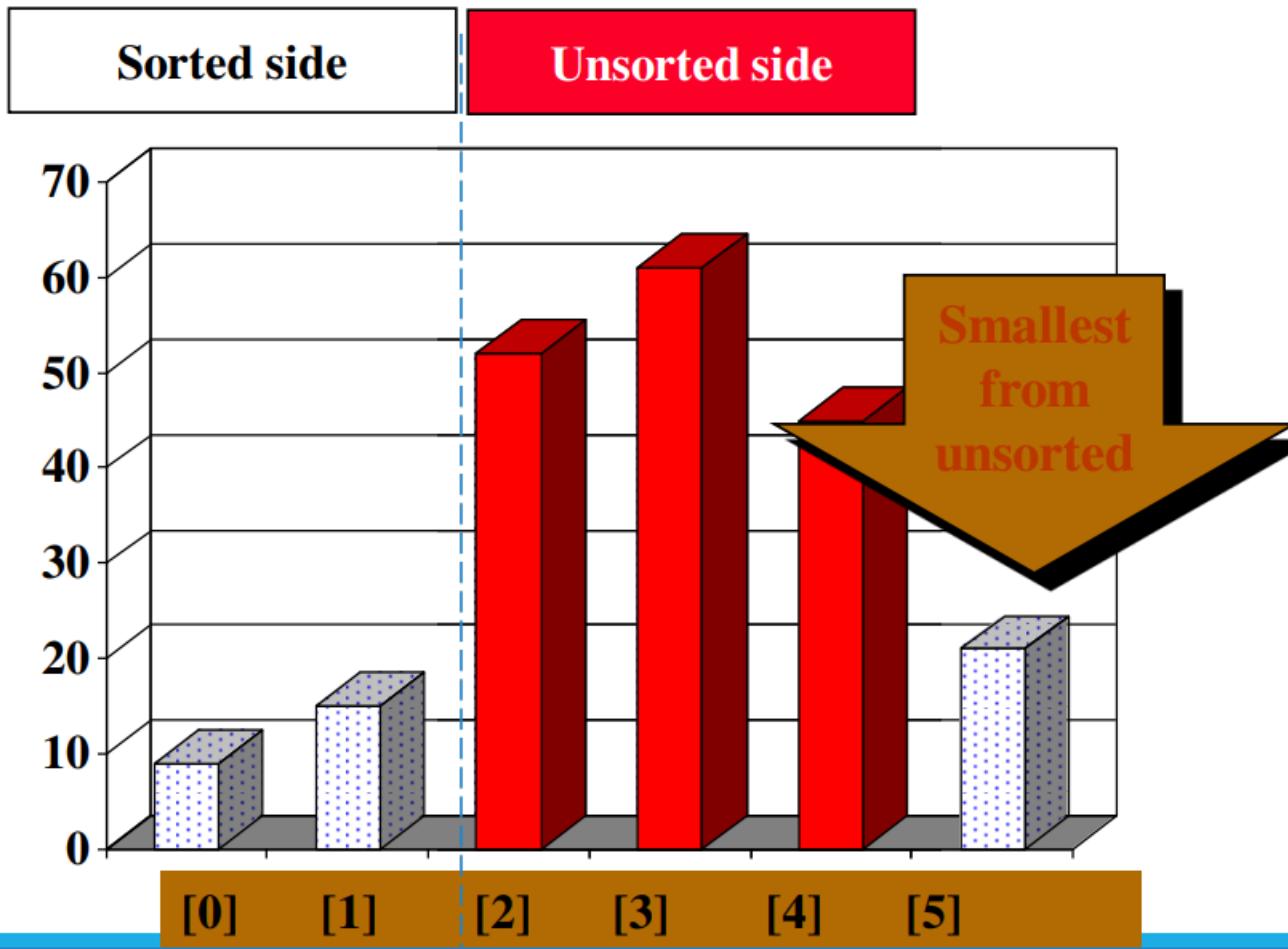


**Find the
smallest
element in the
unsorted side.**

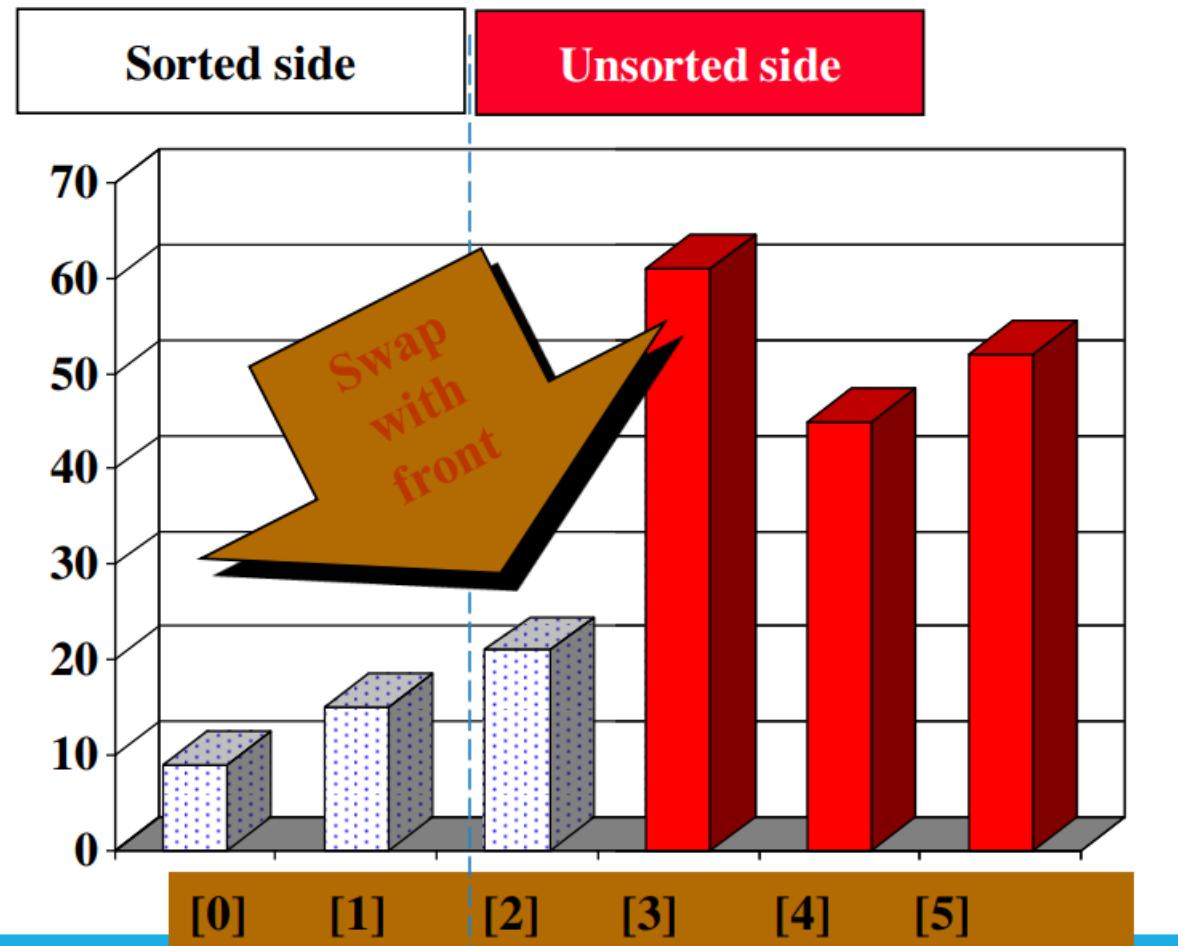




The process continues...

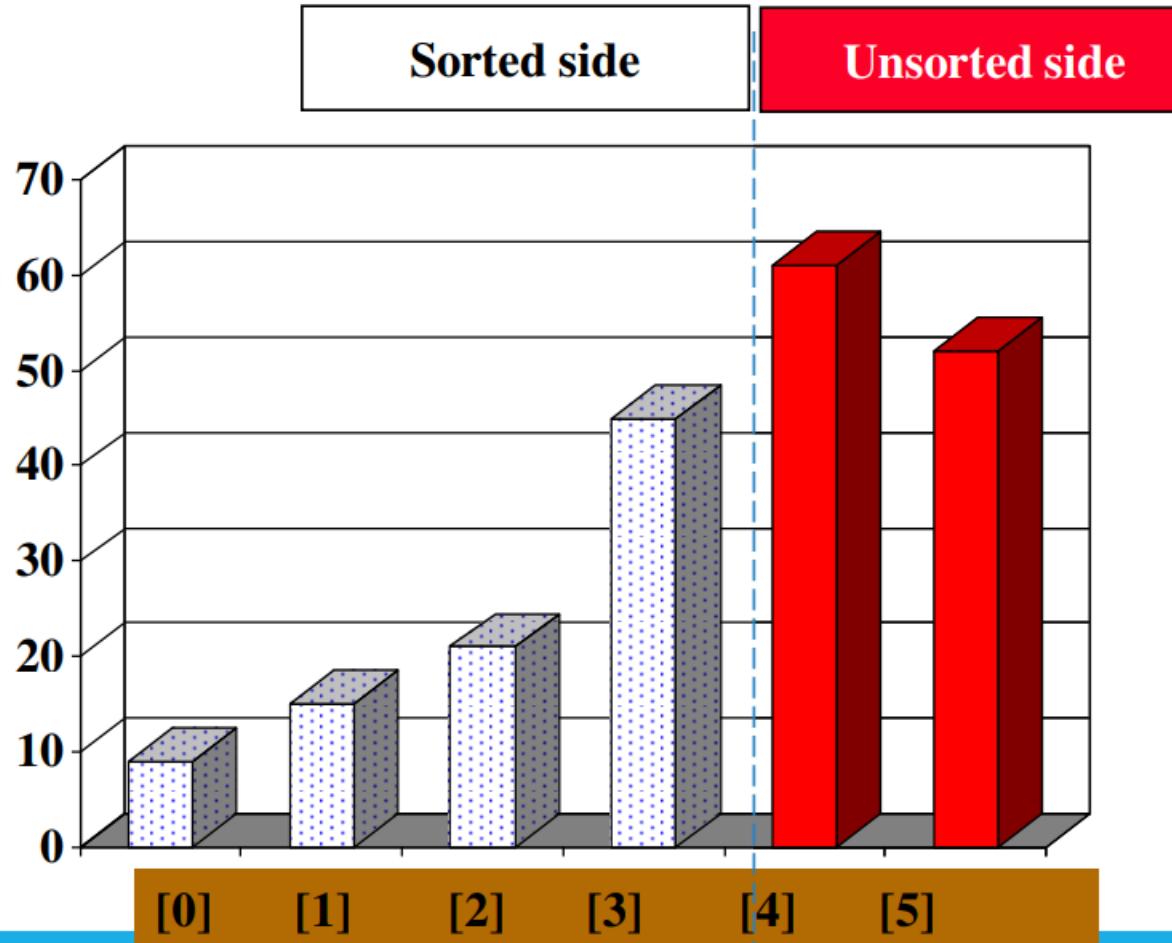


The process continues...

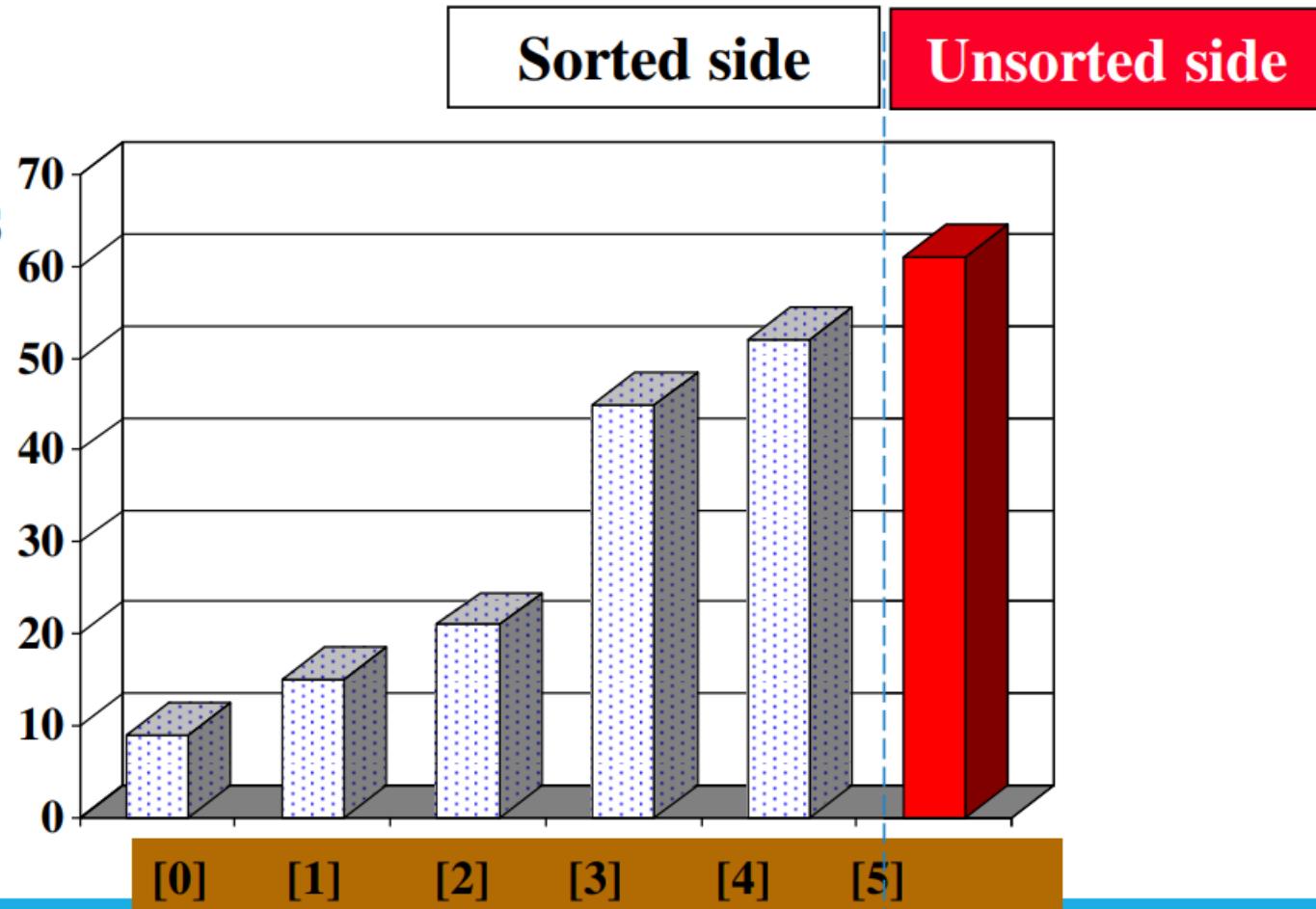


**The process
keeps adding one
more number to
the sorted side.**

**The sorted side
has the smallest
numbers,
arranged from
small to large.**

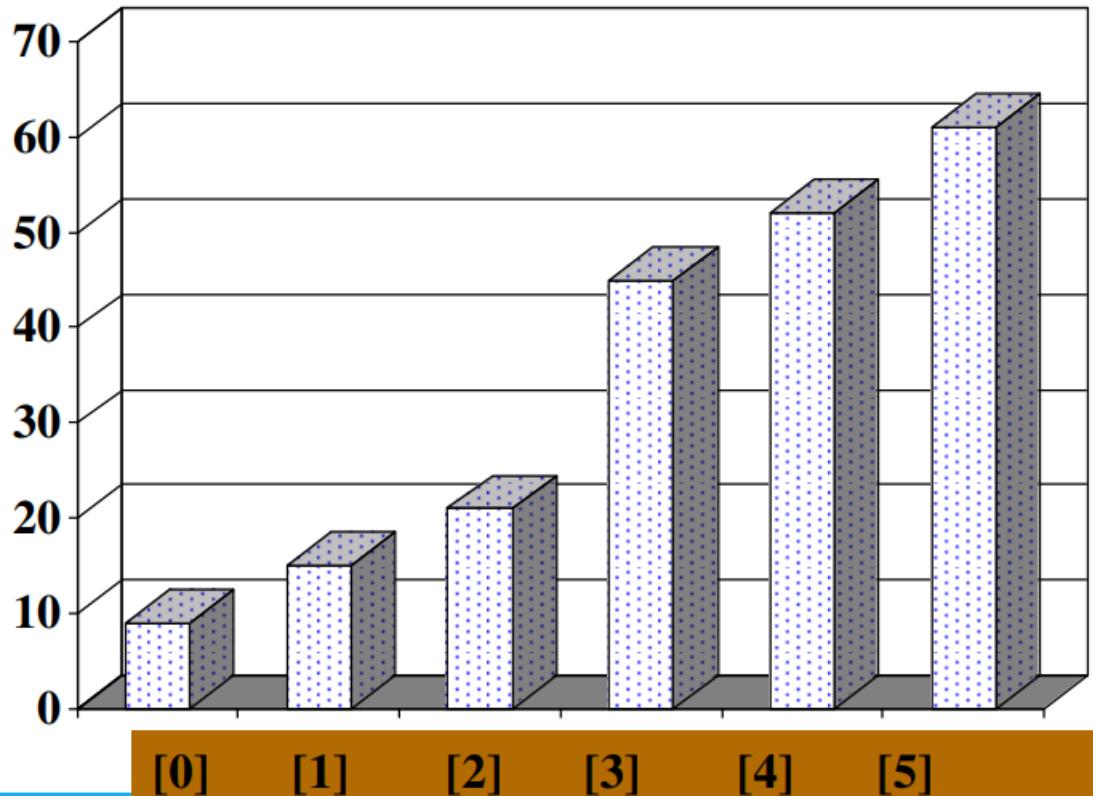


We can stop
when the
unsorted side has
just one number,
since that
number must be
the largest
number.

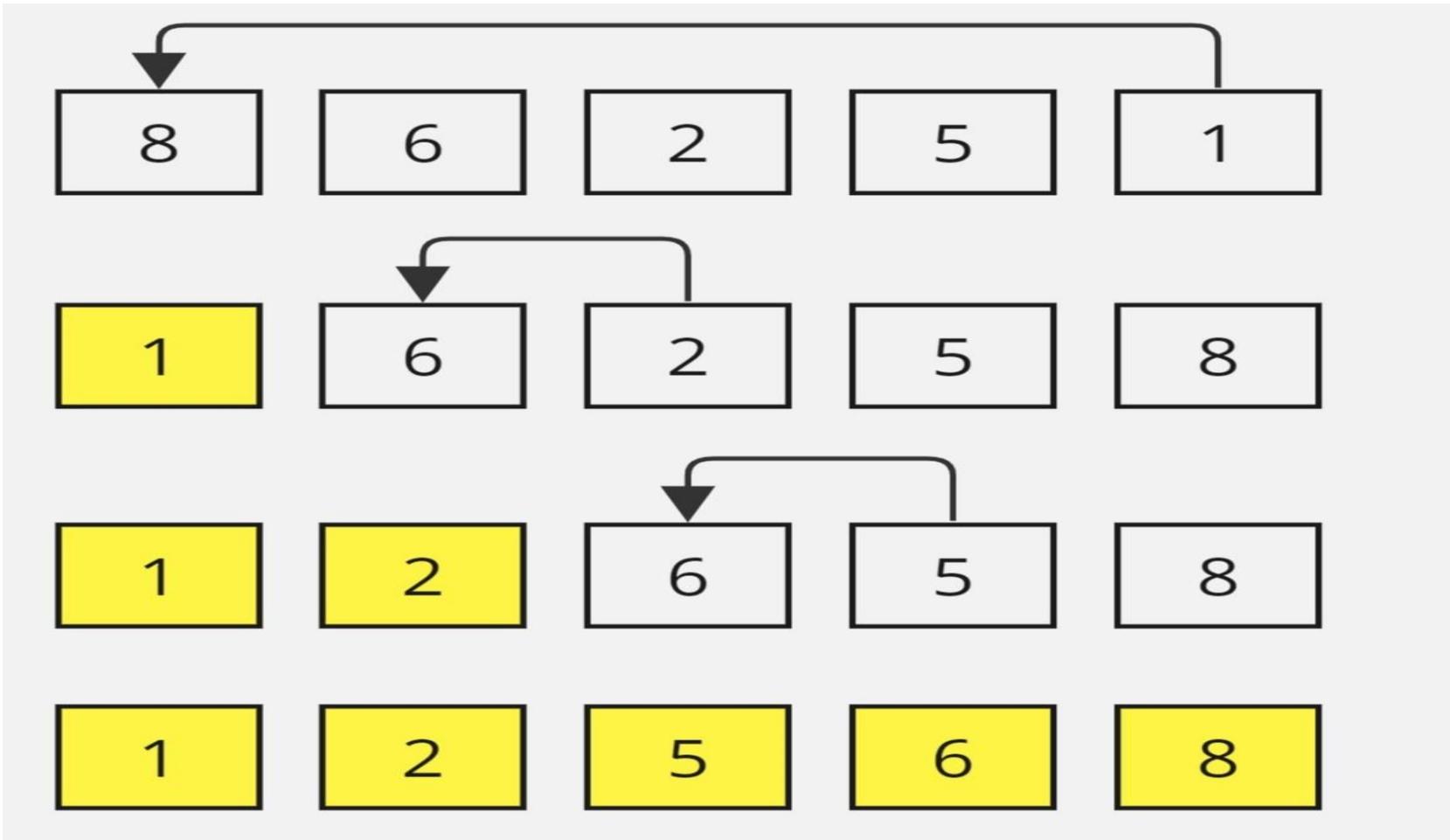


The array is now sorted.

We repeatedly selected the smallest element, and moved this element to the front of the unsorted side.



Selection sort:



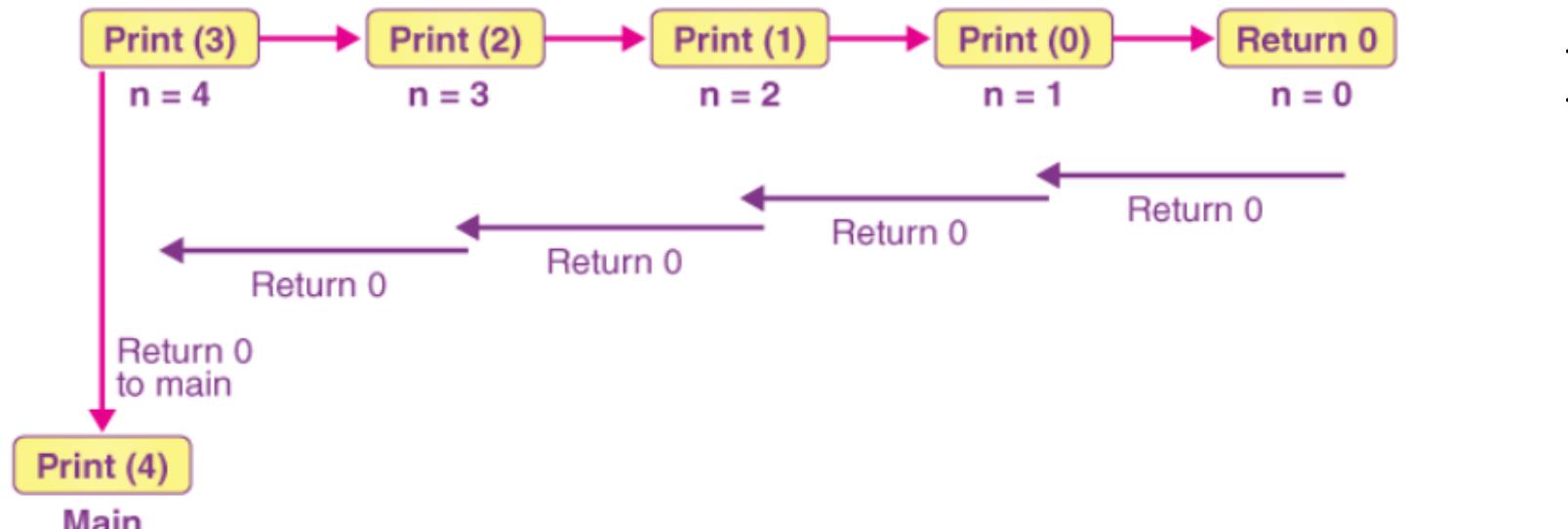
Selection Sort

```
void selection_sort(int arr[], int n)
{
    int i, j, min;
    for (i = 0; i < n - 1; i++)
    {
        min = i;
        for (j = i+1; j < n; j++)
        {
            if (arr[j] < arr[min])
                min = j;
        }
        swap(arr[i],arr[min]);
    }
}
```

Recursion

- Recursion is a technique that solves a problem by solving a smaller problem of the same type. It calls same procedure many a times.
- Recursion has also been described as the ability to place one component inside

```
int display (int x)
{
    if(x == 0)
        return 0;
    else
    {
        printf("%d",x);
        return display(x-1);
    }
}
```

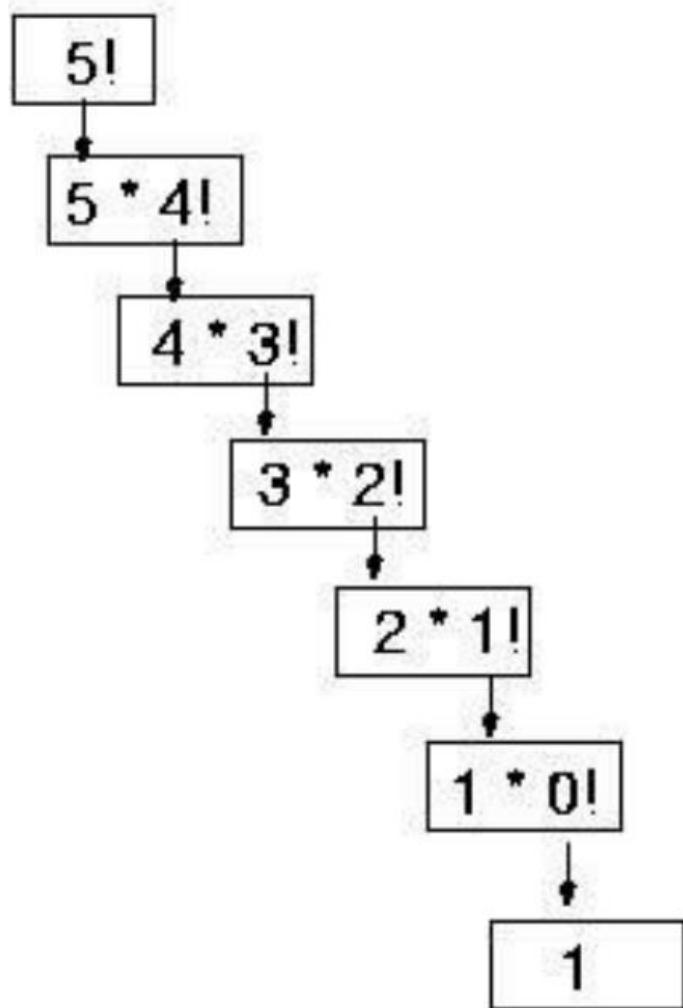


Stack tracing for recursive function call

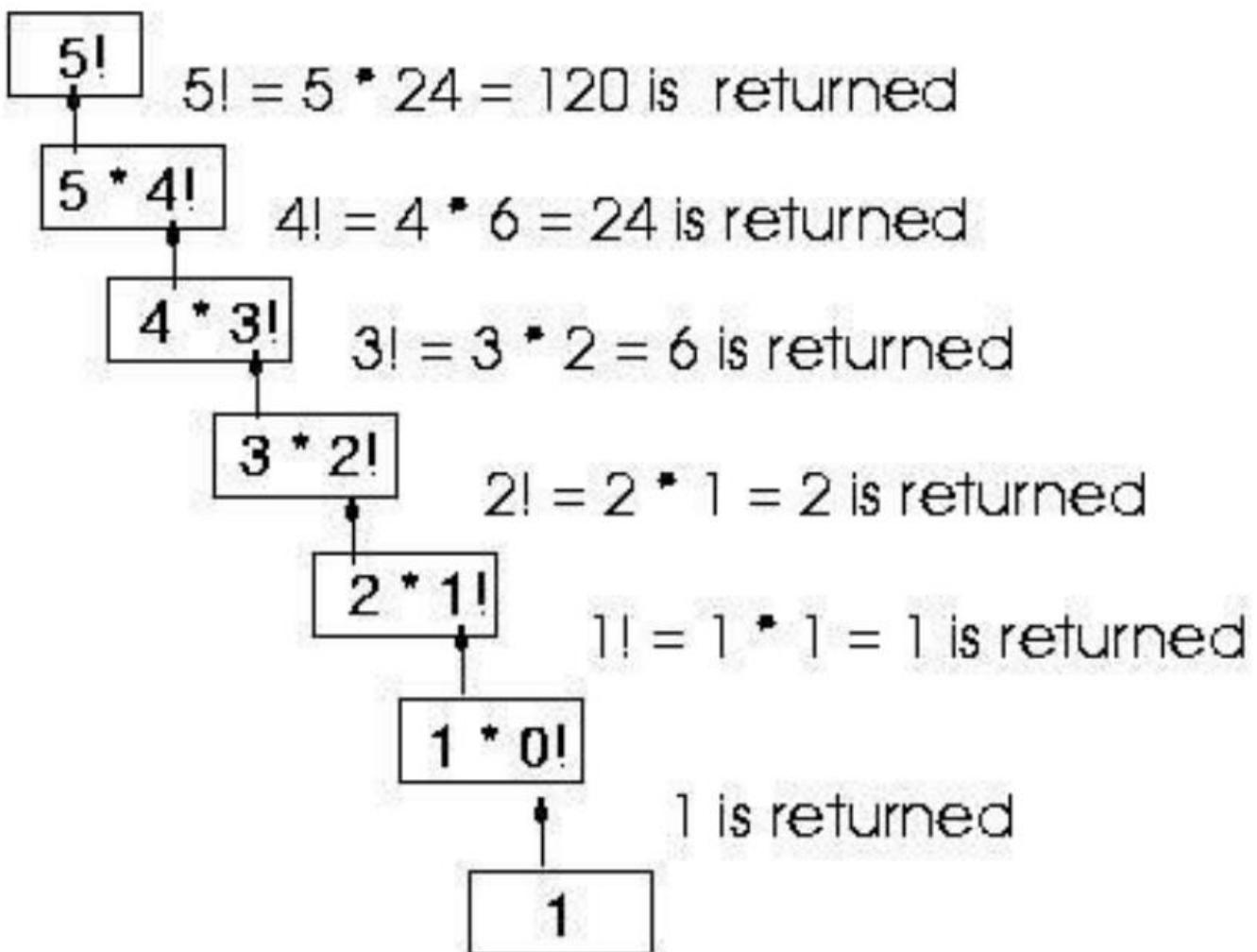
Coding the factorial function

Recursive implementation

```
int Factorial(int n)
{
    if (n==0)
        return 1;
    else
        return n * Factorial(n-1);
}
```

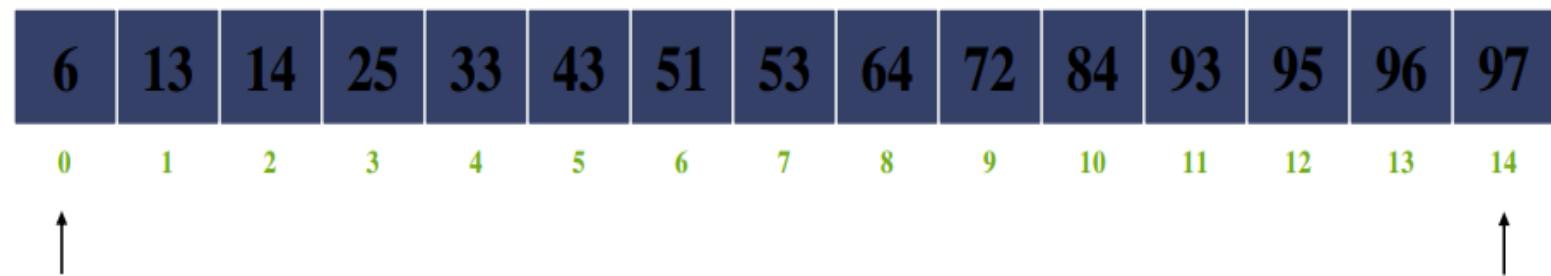


Final value = 120



Binary Search

Binary search. Given value and sorted array $a[]$, find index i such that $a[i] = \text{value}$, or report that no such index exists.

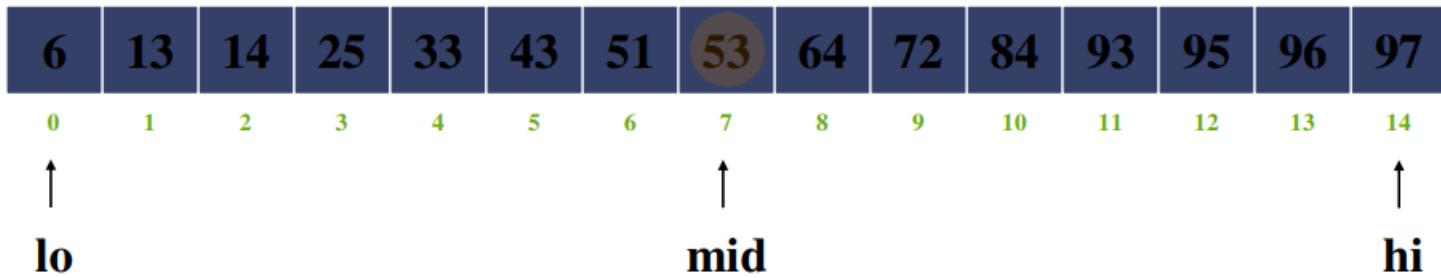


Ex. Binary ^{lo} search for 33. ^{hi}

lo =0;

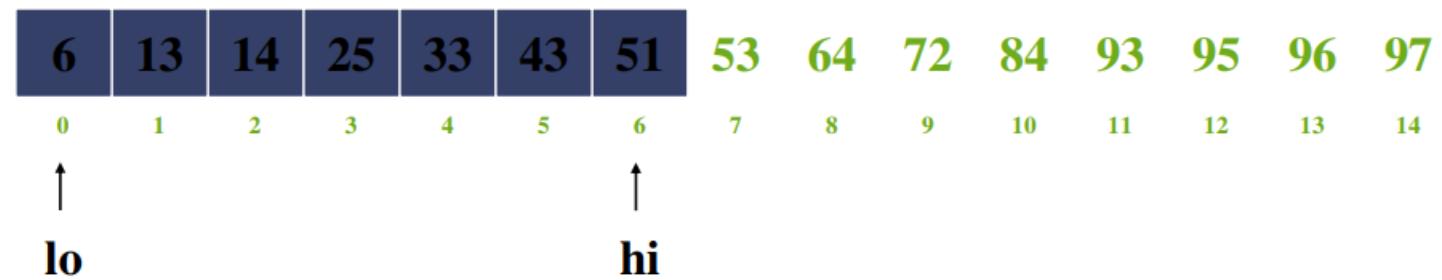
hi = n-1;

$$\text{mid} = (\text{lo}+\text{hi})/2 = (0+14)/2 = 7$$



Ex. Binary search for 33.

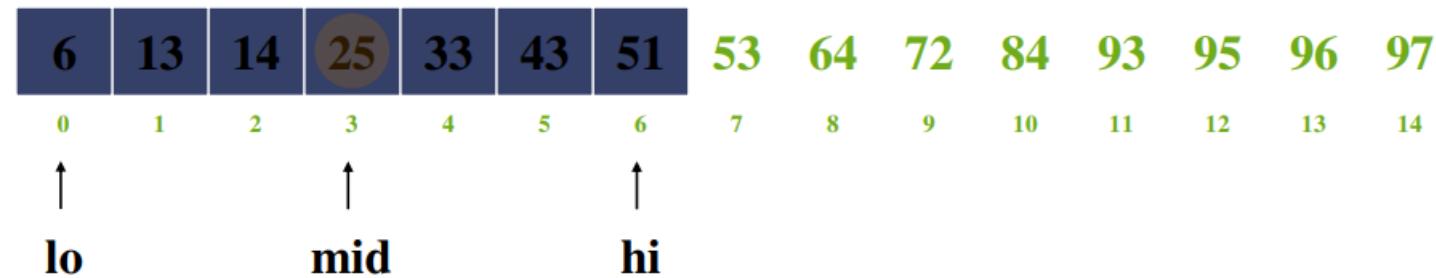
```
if (key == a[mid])
    return 0;
else if (key > a[mid])
    lo = mid + 1;
else
    hi = mid - 1
```



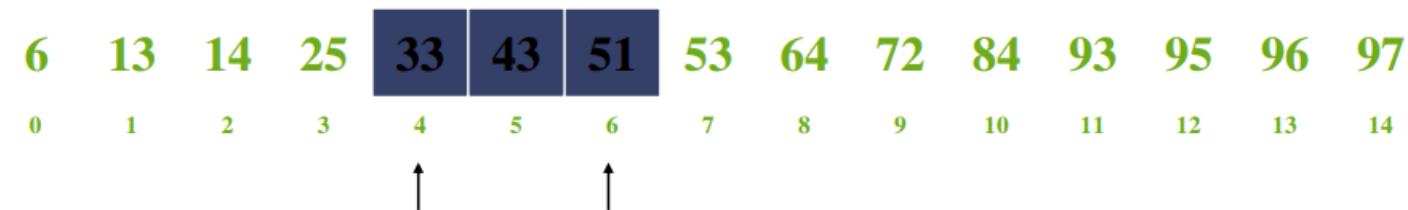
lo = 0

hi = mid-1 = 7-1 =6

mid = (lo+hi) /2 = (0+6)/2 = 3



```
if (key = a[mid])
    return 0;
else if (key > a[mid])
    lo= mid+1;
else
    hi=mid-1
```

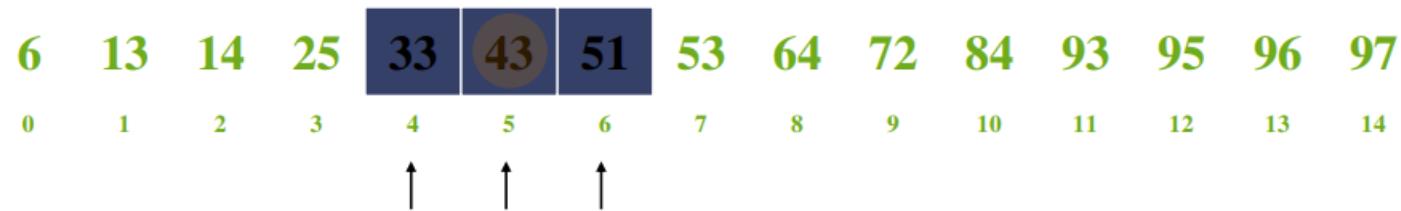


Ex. Binary search for 33.
lo hi

$$\text{lo} = \text{mid} + 1 = 3 + 1 = 4$$

$$\text{hi} = 6$$

$$\text{mid} = (\text{lo}+\text{hi})/2 = (4+6)/2 = 5$$



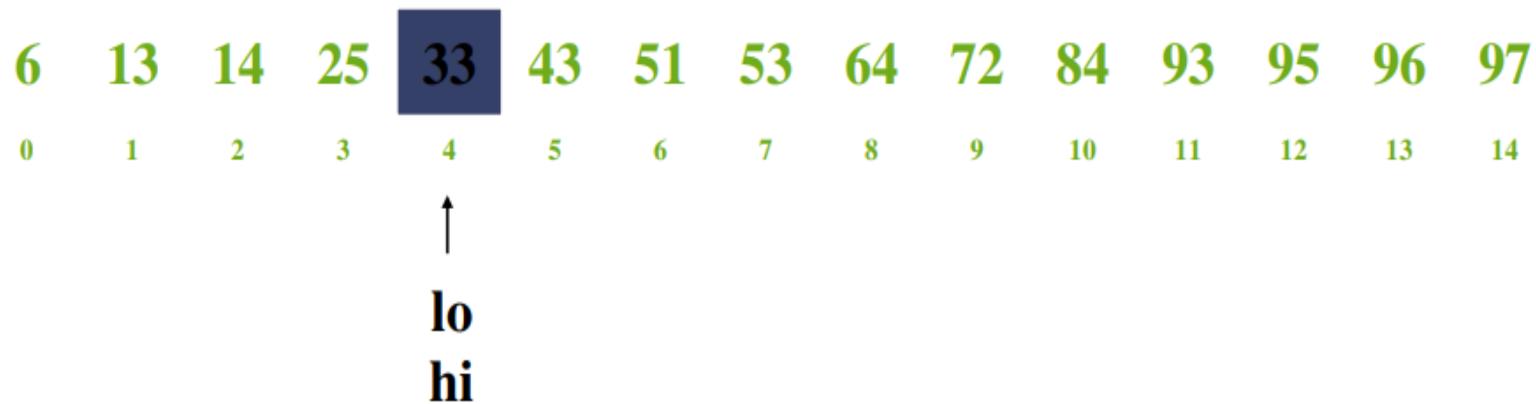
Ex. Binary search for 33.

```
if (key == a[mid])
    return 0;
else if (key > a[mid])
    lo = mid+1;
else
    hi = mid-1;
```

lo = 4

hi = mid -1 =5-1=4

mid = (lo+hi) /2 = (4+4)/2 = 4



Ex. Binary search for 33.

```
if (key = a[mid])
    return 0;
else if (key > a[mid])
    lo= mid+1;
else
    hi=mid-1
```

```
void binary_search(int list[], int lo, int hi, int key)
{
    int mid;
    if (lo > hi)
    {
        printf("Key not found\n");
        return;
    }
    mid = (lo + hi) / 2;
    if (list[mid] == key)
    {
        printf("Key found\n");
    }
    else if (list[mid] > key)
    {
        binary_search(list, lo, mid - 1, key);
    }
    else if (list[mid] < key)
    {
        binary_search(list, mid + 1, hi, key);
    }
}
```

Define Macro : COMPARE

```
#define COMPARE(x,y) (((x)<(y))?-1 : ((x) == (y)) ? 0:1)
```

Using COMPARE macro in Binary search

```
int binary_search(int list[], int lo, int hi, int key)
{
    int mid;
    if (lo > hi)
    {
        printf("Key not found\n");
        return;
    }
    mid = (lo + hi) / 2;
    switch(COMPARE(list[mid],key))
    {
        case 0: {printf("Key found\n");
        return mid;}
        case 1: // (list[mid] > key)
        {
            return binary_search(list, lo, mid - 1);
        }
        case -1:// (list[mid] < key)
        {
            return binary_search(list, mid + 1, hi);
        }
    }
}
```

Data Type

Definition

- A data type is a collection of objects and a set of operations that act on those objects

Example of "int"

- Objects: 0, +1, -1, ..., Int_Max, Int_Min
- Operations: arithmetic(+, -, *, /, and %), testing(equality/inequality), assigns, functions

Define operations

- Its name, possible arguments and results must be specified

The design strategy for representation of objects

- Transparent to the user

Data Abstraction

Definition

- An abstract data type(ADT) is a data type that is organized in such a way that the specification of the objects and the specification of the operations on the objects is separated from the representation of the objects and the implementation of the operation.#

Why abstract data type ?

- implementation-independent

Abstract Data type (ADT)

Abstract Data type (ADT) is a type (or class) for objects whose behavior is defined by a set of values and a set of operations. The definition of ADT only mentions what operations are to be performed but not how these operations will be implemented. It does not specify how data will be organized in memory and what algorithms will be used for implementing the operations. It is called “abstract” because it gives an implementation-independent view.

Example: List()

The **List ADT Functions** is given below:

- `get()` – Return an element from the list at any given position.
- `insert()` – Insert an element at any position of the list.
- `remove()` – Remove the first occurrence of any element from a non-empty list.
- `removeAt()` – Remove the element at a specified location from a non-empty list.
- `replace()` – Replace an element at any position by another element.
- `size()` – Return the number of elements in the list.
- `isEmpty()` – Return true if the list is empty, otherwise return false.
- `isFull()` – Return true if the list is full, otherwise return false.

Classifying the Functions of a Data Type

Creator/constructor:

- Create a new instance of the designated type

Transformers

- Also create an instance of the designated type by using one or more other instances

Observers/reporters

- Provide information about an instance of the type, but they do not change the instance

Note: An ADT definition will include at least one function from each of these three categories

An Example of the ADT

structure Natural_Number is

objects: an ordered subrange of the integers starting at zero and ending at the maximum integer (INT_MAX) on the computer

functions:

for all x, y is Nat_Number, TRUE, FALSE is Boolean and where
. +, -, <, and == are the usual integer

operations

Nat_NoZero() ::= 0

Boolean Is_Zero(x) ::= if (x) return FALSE

Cont....

```
Nat_No Add(x, y) ::= if ((x+y)<= INT_MAX) return x+ y  
                      else return INT_MAX
```

```
Boolean Equal(x, y) ::= if (x== y) return TRUE  
                           else return FALSE
```

```
Nat_No Successor(x) ::= if (x== INT_MAX) return x  
                           else return x+ 1
```

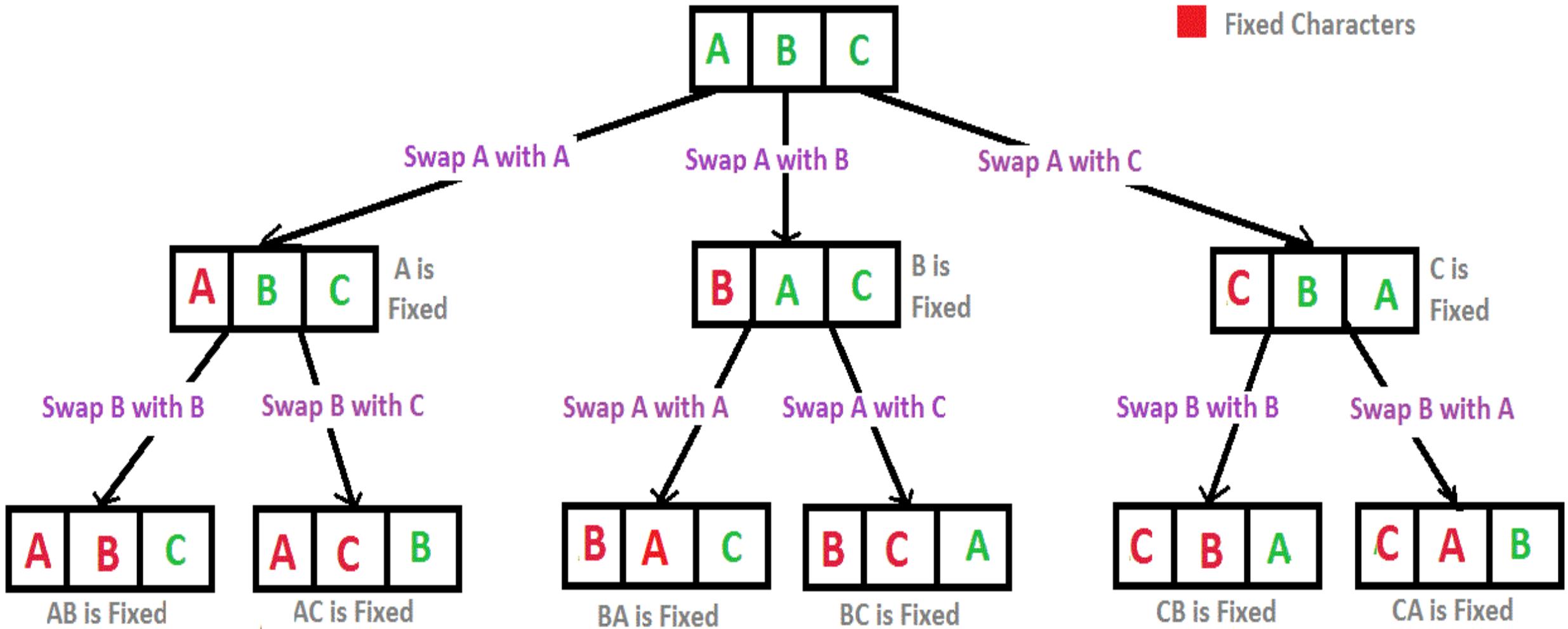
```
Nat_No Subtract(x, y) ::= if (x< y) return 0  
                           else return x-y
```

```
end Natural_Number
```

Permutation using Recursion

A permutation also called an “arrangement number” or “order,” is a rearrangement of the elements of an ordered list S into a one-to-one correspondence with S itself. A string of length n has $n!$ permutation.

■ Fixed Characters



Recursion Tree for Permutations of String "ABC"

```
void perm(char *list,int i,int n)
{
    int j,temp;
    if(i==n)
    {
        for{j=0;j<=n;j++)
            printf("%c",list[j]);
    }
    else
    {
        for(j=i;j<=n;j++)
        {
            swap(list[i],list[j],temp);
            perm(list,i+1,n);
            swap(list[i],list[j],temp);
        }
    }
}
```

Pointer to Arrays

```
#include<stdio.h>

int main()
{
    int arr[5] = { 1, 2, 3, 4, 5 };
    int *ptr = arr;

    printf("%p\n", ptr);
    return 0;
}
```

In this program,
we have a
**pointer *ptr* that
points to the
0th element of the
array.**

Similarly, we can also declare a pointer that can point to whole array instead of only one element of the array. This pointer is useful when talking about multidimensional arrays.

Pointer to Arrays

Syntax: data_type (*var_name)[size_of_array];

Ex: int (*ptr)[10];

```
int main()
{
    int *p;
    int (*ptr)[5];
    int arr[5];

    p = arr;
    ptr = &arr;

    printf("p = %p, ptr = %p\n", p, ptr);

    p++;
    ptr++;

    printf("p = %p, ptr = %p\n", p, ptr);

    return 0;
}
```

Macro definition for memory allocation

```
#define MALLOC(p,s)
if(!((p)=malloc(s)))
{
    printf("cant allocate memory");
    exit(0);
}
```

Calling Macro

```
int *pi;
MALLOC(pi,sizeof(int))
```

Dynamically Allocated Arrays

One dimensional arrays

```
int i,n,*list;  
        //read n  
        //if(n<1) error  
MALLOC(list, n*sizeof(int))
```

Two Dimensional Arrays



A= address i.e &A[0]

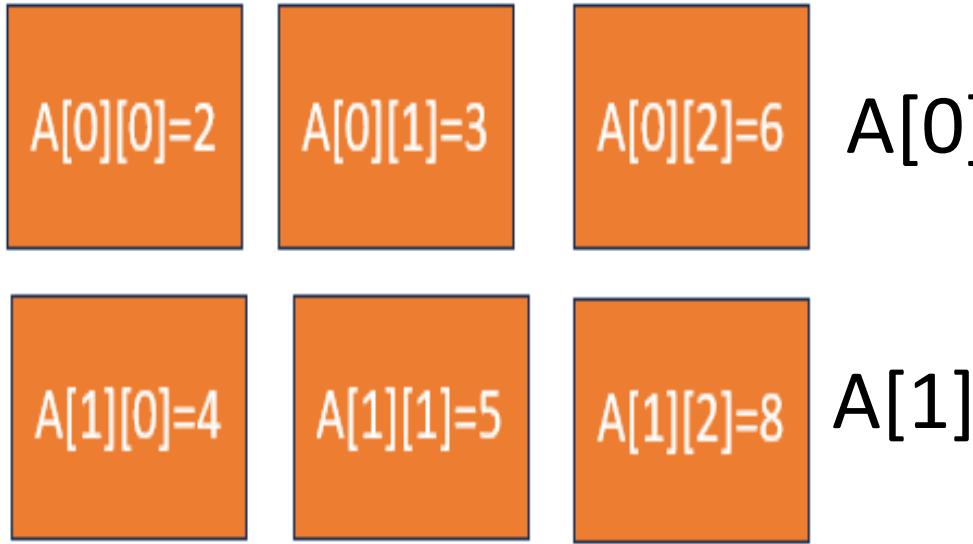
***A= A[0]
& A[0][0]**

Two Dimensional Arrays

A[0][0]=2	A[0][1]=3	A[0][2]=6
A[1][0]=4	A[1][1]=5	A[1][2]=8

A= address i.e &A[0]

Two Dimensional Arrays



$*(A + i) = ?$

$*(\&A[i]) = A[i]$

$*(*(arr + i) + j) =$

$*(*(\&a[i]) + j) =$

$(a[i]+j)=*(\&a[i][j])$

$arr[i][j]$

Two Dimensional Arrays

$*(*(\text{a} + \text{i}) + \text{j})$ = value of $\text{a}[\text{i}][\text{j}]$

$(*(\text{a} + \text{i}) + \text{j})$ = address of $\text{a}[\text{i}][\text{j}]$ = $\& \text{a}[\text{i}][\text{j}]$

Two Dimensional Arrays

```
#include<stdio.h>
int main()
{
    int a[3][5];
    int i,j;
    for(i=0;i<3;i++)
    {
        for(j=0;j<5;j++)
        {
            scanf("%d", (*(a+i)+j));
        }
    }
    for(i=0;i<3;i++)
    {
        for(j=0;j<5;j++)
        {
            printf("%d\t", *(*(a+i)+j));
        }
        printf("\n");
    }
    return 0;
}
```

Dynamic Memory Allocation for 2d- Arrays

1. Using Array of Pointers

```
int* x[r];  
for (i = 0; i < r; i++)  
    x[i] = (int*)malloc(c * sizeof(int));
```

2. Using Pointers to Pointers

```
int x[3][5];  int **pa;  
int r,c; //read r and c  
pa= create2d(r,c)
```

```
int **create2d(int r, int c)  
{  
    MALLOC(x,r*sizeof(*x));  
    for(i=0;i<r;i++)  
    {  
        MALLOC(x[i],c*sizeof(**x));  
    }  
    return x;  
}
```

Other Dynamic Memory Allocation Functions

Calloc()

calloc(): To allocate n blocks of memory and initialize to 0.

```
#define CALLOC(p,n,s)
if(!((p)=calloc(n,s)))
{
    printf("no memory");
    exit(0);
}
```

Realloc()

realloc() : Used to resize memory space allocated for a pointer

```
realloc(p,newsize)
```

Structures

A structure is a collection of data items, where each item is identified as to its type and name.

```
struct {  
    char name[10];  
    int age;  
    float salary;  
} person;  
  
strcpy(person.name, "james");  
person.age=10;  
person.salary=35000;
```

Create structure data type

```
typedef struct human_being
{
    char name[10];
    int age;
    float salary;
};
```

or

```
typedef struct
{
    char name[10];
    int age;
    float salary
} human_being;
human_being person1, person2;
```

Calloc()

calloc(): To allocate n blocks of memory and initialize to 0.

```
#define CALLOC(p,n,s)
```

```
if(!((p)=calloc(n,s)))
```

```
{
```

```
    printf("no memory");
```

```
    exit(0);
```

```
}
```

Realloc()

realloc() : Used to resize memory space allocated for a pointer

```
realloc(p,newsize)
```

Structures

struct

{

char name[10];

int age;

float salary;

} person;

strcpy(person.name, "james");

person.age=10;

person.salary=35000;

Create structure data type

```
typedef struct human_being {  
char name[10];  
int age;  
float salary;  
};
```

or

```
typedef struct {  
char name[10];  
int age;  
float salary  
} human_being;
```

Unions

Similar to struct, but only one field is active.

Example: Add fields for male and female.

```
typedef struct gender_type {  
    enum gender_field {female, male} gender;  
    union {  
        int children;  
        int beard;  
    } u;  
};
```

```
typedef struct human_being {  
    char name[10];  
    int age; float salary;  
    date dob; gender_type gender_info;  
}
```

```
human_being person1, person2;  
person1.gender_info.gender=male;  
person1.gender_info.u.beard=FALSE;
```

Sparse Matrix

	col 1	col 2	col 3
row 1	-27	3	4
row 2	6	82	-2
row 3	109	-64	11
row 4	12	8	9
row 5	48	27	47

row0	col1	col2	col3	col4	col5	col6
row1	15	0	0	22	0	-15
row2	0	11	3	0	0	0
row3	0	0	0	-6	0	0
row4	91	0	0	0	0	0
row5	0	0	28	0	0	0

(a) 15/15

(b) 8/36

sparse matrix
data structure?

Two matrices

SPARSE MATRIX ABSTRACT DATA TYPE

Structure Sparse_Matrix is

objects: a set of triples, <row, column, value>, where row and column are integers and form a unique combination, and value comes from the set item.

functions: for all $a, b \in \text{Sparse_Matrix}$, x is item, i, j , max_col, max_row in index

Sparse_Marix Create(max_row, max_col) :: =

return a Sparse_matrix that can hold up to max_items = max_row, max_col and whose maximum row size is max_row and whose maximum column size is max_col.

SPARSE MATRIX ABSTRACT DATA TYPE

Sparse_Matrix Transpose(a) ::= return the matrix produced by interchanging the row and column value of every triple.

Sparse_Matrix Add(a, b) ::= if the dimensions of a and b are the same return the matrix produced by adding corresponding items, namely those with identical row and column values.
else return error

Sparse_Matrix Multiply(a, b) ::=
if number of columns in a equals number of rows in b
return the matrix d produced by multiplying a by b according to the formula:
 $d[i][j] = (a[i][k] \cdot b[k][j])$ where d (i, j) is the (i,j)th element
else return error.

SPARSE MATRIX ABSTRACT DATA TYPE

(1) Represented by a two-dimensional array. Sparse matrix wastes space.

(2) Each element is characterized by <row, col, value>.

	row	col	value					
				# of rows	(columns)	# of nonzero terms	row	col
a[0]	6	6	8	b[0]	6	6	8	
	[1]	0	0		[1]	0	0	15
	[2]	0	3		[2]	0	4	91
	[3]	0	5		[3]	1	1	11
	[4]	1	1		[4]	2	1	3
	[5]	1	2		[5]	2	5	28
	[6]	2	3		[6]	3	0	22
	[7]	4	0		[7]	3	2	-6
	[8]	5	2		[8]	5	0	-15

(a) row, column in ascending order
*Figure Sparse matrix and its transpose stored as triples (b)

Sparse_matrix Create(max_row, max_col) ::=

```
#define MAX_TERMS 101 /* maximum number of terms +1*/  
typedef struct {  
    int col;  
    int row;  
    int value;  
} term;  
term a[MAX_TERMS]
```

of rows (columns)
of nonzero terms



Transpose a Matrix

(1) for each row i

take element $\langle i, j, \text{value} \rangle$ and store it
in element $\langle j, i, \text{value} \rangle$ of the transpose.

difficulty: where to put $\langle j, i, \text{value} \rangle$

$(0, 0, 15) \implies (0, 0, 15)$

$(0, 3, 22) \implies (3, 0, 22)$

$(0, 5, -15) \implies (5, 0, -15)$

$(1, 1, 11) \implies (1, 1, 11)$

Move elements down very often.

(2) For all elements in column j,

place element $\langle i, j, \text{value} \rangle$ in element $\langle j, i, \text{value} \rangle$

```
void transpose (term a[], term b[])
/* b is set to the transpose of a */
{
    int n, i, j, currentb;
    n = a[0].value; /* total number of elements */
    b[0].row = a[0].col; /* rows in b = columns in a */
    b[0].col = a[0].row; /*columns in b = rows in a */
    b[0].value = n;
    if (n > 0) {           /*non zero matrix */
        currentb = 1;
        for (i = 0; i < a[0].col; i++)
            /* transpose by columns in a */
            for(j = 1; j <= n; j++)
                /* find elements from the current column */
                if (a[j].col == i) {
                    /* element is in current column, add it to b */
```

```
columns
```

```
elements
```

```
b[currentb].row = a[j].col;  
b[currentb].col = a[j].row;  
b[currentb].value = a[j].value;  
currentb++;
```

```
}
```

```
}
```

Fast Transpose Matrix

a[0]	6	6	8						
a[1]	0	0	15						
a[2]	0	3	22						
a[3]	0	5	-15						
a[4]	1	1	11						
a[5]	1	2	3						
a[6]	2	3	-6						
a[7]	4	0	91						
a[8]	5	2	28	[0]	[1]	[2]	[3]	[4]	[5]
row_terms =	2	1	2	2	0	1			
starting_pos =	1	3	4	6	8	8			

```
void fast_transpose(term a[ ], term b[ ])
{
/* the transpose of a is placed in b */
    int row_terms[MAX_COL], starting_pos[MAX_COL];
    int i, j, num_cols = a[0].col, num_terms = a[0].value;
    b[0].row = num_cols; b[0].col = a[0].row;
    b[0].value = num_terms;
    if (num_terms > 0){ /*nonzero matrix*/
        for (i = 0; i < num_cols; i++)
columns        row_terms[i] = 0;
elements        for (i = 1; i <= num_terms; i++)
elements        row_term [a[i].col]++;
        starting_pos[0] = 1;
columns        for (i = 1; i < num_cols; i++)
columns        starting_pos[i]=starting_pos[i-1] +row_terms [i-1];
```

elements

```
for (i=1; i <= num_terms, i++) {  
    j = starting_pos[a[i].col]++;  
    b[j].row = a[i].col;  
    b[j].col = a[i].row;  
    b[j].value = a[i].value;  
}  
}  
}
```

***Program** Fast transpose of a sparse matrix

Polynomial Addition

$$\text{Polynomials } A(X) = 3X^{20} + 2X^5 + 4,$$

$$B(X) = X^{10} + 10X^5 + 3X + 1$$

Structure *Polynomial* is

objects: a set of ordered pairs of $\langle e_i, a_i \rangle$ where a_i in *Coefficients* and e_i in *Exponents*, e_i are integers ≥ 0

functions:

for all *poly*, *poly1*, *poly2* are *Polynomial*, *coef* stores *Coefficients*, *expon* stores *Exponents*

Polynomial Zero()

::= return the polynomial,

$$p(x) = 0$$

*Boolean IsZero(*poly*)*

::= if (*poly*) return *FALSE*
else return *TRUE*

Coefficient Coef(poly, expon)

::= if (expon is in poly) return
its coefficient else return Zero

Exponent Lead_Exp(poly)

::= return the largest exponent in poly

Polynomial Attach(poly,coef, expon)

::= if (expon is in poly) return error
else return the polynomial
poly with the term <coef, expon>
inserted

Polynomial Remove(poly, expon)

**::= if (expon is in poly) return the polynomial poly with the term whose exponent is expon deleted
else return error**

Polynomial SingleMult(poly, coef, expon) ::= return the polynomial

$\text{poly} \bullet \text{coef} \bullet x^{\text{expon}}$

Polynomial Add(poly1, poly2) ::= return the polynomial
 $\text{poly1} + \text{poly2}$

Polynomial Mult(poly1, poly2) ::= return the polynomial
 $\text{poly1} \bullet \text{poly2}$

Polynomial Addition : Implementation

```
MAX_TERMS 100 /* size of terms array */
typedef struct {
    float coef;
    int expon;
} polynomial;
polynomial terms[MAX_TERMS];
int avail = 0;
```

Polynomial Addition : Implementation

```
void padd (int starta, int finisha, int startb, int finishb, int * startd, int *finishd)
{
/* add A(x) and B(x) to obtain D(x) */
float coefficient;
*startd = avail;
while (starta <= finisha && startb <= finishb)
    switch (COMPARE(terms[starta].expon,
                    terms[startb].expon)) {
        case -1: /* a expon < b expon */
            attach(terms[startb].coef, terms[startb].expon);
            startb++;
            break;
```

Polynomial Addition : Implementation

```
case 0: /* equal exponents */
    coefficient = terms[starta].coef + terms[startb].coef;
    if (coefficient)
        attach (coefficient, terms[starta].expon);
    starta++;
    startb++;
    break;
case 1: /* a expon > b expon */
    attach(terms[starta].coef, terms[starta].expon);
    starta++;
}
```

Polynomial Addition : Implementation

```
/* add in remaining terms of A(x) */
for( ; starta <= finisha; starta++)
    attach(terms[starta].coef, terms[starta].expon);
/* add in remaining terms of B(x) */
for( ; startb <= finishb; startb++)
    attach(terms[startb].coef, terms[startb].expon);
*finishd =avail -1;
}
```

Polynomial Addition : Implementation

```
void attach(float coefficient, int exponent)
{
    /* add a new term to the polynomial */
    if (avail >= MAX_TERMS) {
        fprintf(stderr, "Too many terms in the polynomial\n");
        exit(1);
    }
    terms[avail].coef = coefficient;
    terms[avail++].expon = exponent;
}
```

Representation of Multidimensional Arrays:

As its name implies, row major order stores multidimensional arrays by rows. For instance, we interpret the two-dimensional array $A[upper_0][upper_1]$ as $upper_0$ rows, $row_0, row_1, \dots, row_{upper_0-1}$, each row containing $upper_1$ elements.

Row-major order

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix}$$

Column-major order

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix}$$

Address of $A[i] = \text{Base address} + i$
if $\alpha = \text{Address of } A[0][0]$

Then $A[i][0] = \alpha + i * \text{upper1}$

$A[i][j] = \alpha + i * \text{upper1} + j$

$A[i][j][k] = \alpha + i * \text{upper1} * \text{upper2} + j * \text{upper2} + k$

The address of $a[i_0][i_1][0] \dots [0]$ is:

$$\alpha + i_0 \cdot \text{upper}_1 \cdot \text{upper}_2 \dots \text{upper}_{n-1} + i_1 \cdot \text{upper}_2 \cdot \text{upper}_3 \dots \text{upper}_{n-1}$$

Repeating in this way the address for $A[i_0][i_1] \dots [i_{n-1}]$ is:

$$\begin{aligned} & \alpha + i_0 \cdot \text{upper}_1 \cdot \text{upper}_2 \dots \text{upper}_{n-1} \\ & + i_1 \cdot \text{upper}_2 \cdot \text{upper}_3 \dots \text{upper}_{n-1} \\ & + i_2 \cdot \text{upper}_3 \cdot \text{upper}_4 \dots \text{upper}_{n-1} \\ & \cdot \\ & \cdot \\ & \cdot \\ & + i_{n-2} \cdot \text{upper}_{n-1} \\ & + i_{n-1} \end{aligned}$$

∴ $= \alpha + \sum_{j=0}^{n-1} i_j a_j$ where: $\begin{cases} a_j = \prod_{k=j+1}^{n-1} \text{upper}_k & 0 \leq j < n-1 \\ a_{n-1} = 1 & \end{cases}$

Strings

In C , we represent strings as character arrays terminated with the null character \0.

```
char name[25]={“RIT”};
```

```
char cname[]={“RIT”};
```

Strings -ADT

ADT string is

Objects : a finite set of zero or more characters

Functions:

For all s,t belongs to string,

i,j,m belongs to non negative integers.

**String null(m) ::= return a string whose maximum length
is m characters, but is initially set to NULL**

Integer compare(s,t)::= if s equals t return 0

**else if s precedes t return -1
else return +1**

**Boolean isNull(s) ::= if compare(s,null) return False
else return True**

**Integer Length(S)::= if(compare(s,null) return the number
of characters in s.
else return 0.**

**String concat(s,t) ::= if(compare(t,null)) return a string whose
elements are those of s followed by
those of t,
else return s.**

**String substr(s,i,j) ::= if(j>0) &&(i+j-1)<length(s))
return the string containing the characters
of s at position i,i+1,...,i+j-1.
else return null.**

C string functions & Examples

Strcat(s,t)

Strncat(s,t,n)

Strcmp(s,t)

Strncmp(s,t,n)

Strcpy(s,t)

Strncpy(s,t,n)

Strlen(s)

Strchr(s,c) : return ptr to first occurrence of c in s

Strrchr(s,c) : return ptr to last occurrence of c in s

Strtok(s,delim) : return string surrounded by delim in s

Strstr(s,pat) : return ptr to start of pat in s

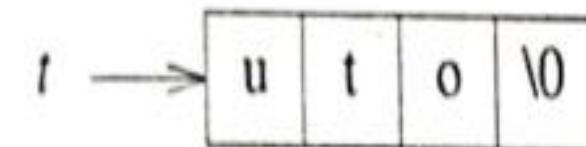
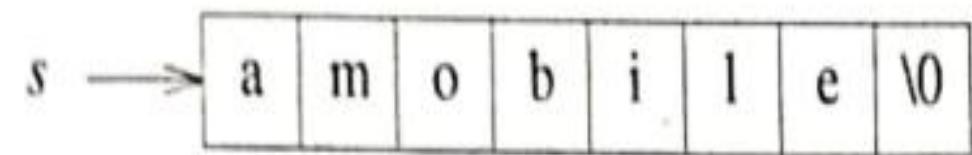
Strspn(s,spanset) : return length of span in s

Strcspn(s,spanset)

**Strpbrk(s,spanset): return ptr to first occurrence of char
from spanset**

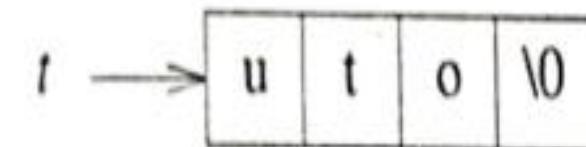
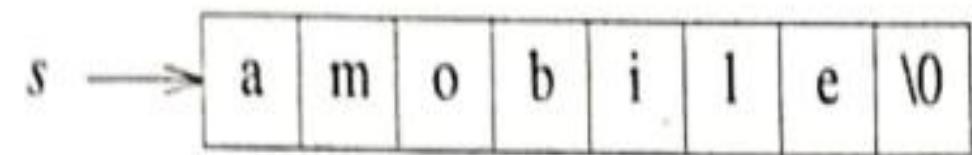
Strings

String insertion ?



Strings

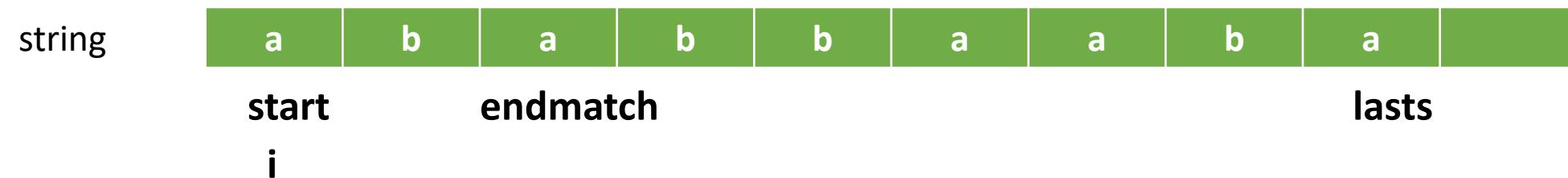
String insertion ?



```
void strnins(char *s, char *t, int i)
/* insert string t into string s at position i */
char string[MAX_SIZE], *temp = string;

if (i < 0 && i > strlen(s)) {
    fprintf(stderr,"Position is out of bounds \n");
    exit(EXIT_FAILURE);
}
if (!strlen(s))
    strcpy(s,t);
else if (strlen(t)) {
    strncpy(temp, s,i);
    strcat(temp,t);
    strcat(temp, (s+i));
    strcpy(s, temp);
}
}
```

Program 2.12: String insertion function

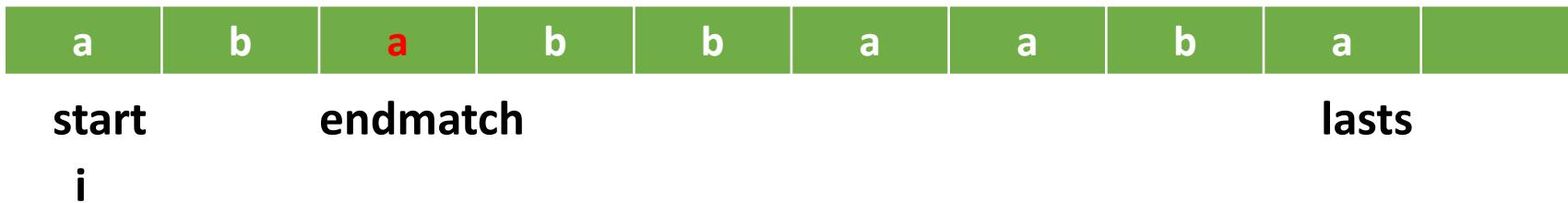


*N-Find Algorithm for
pattern matching*

pattern



string



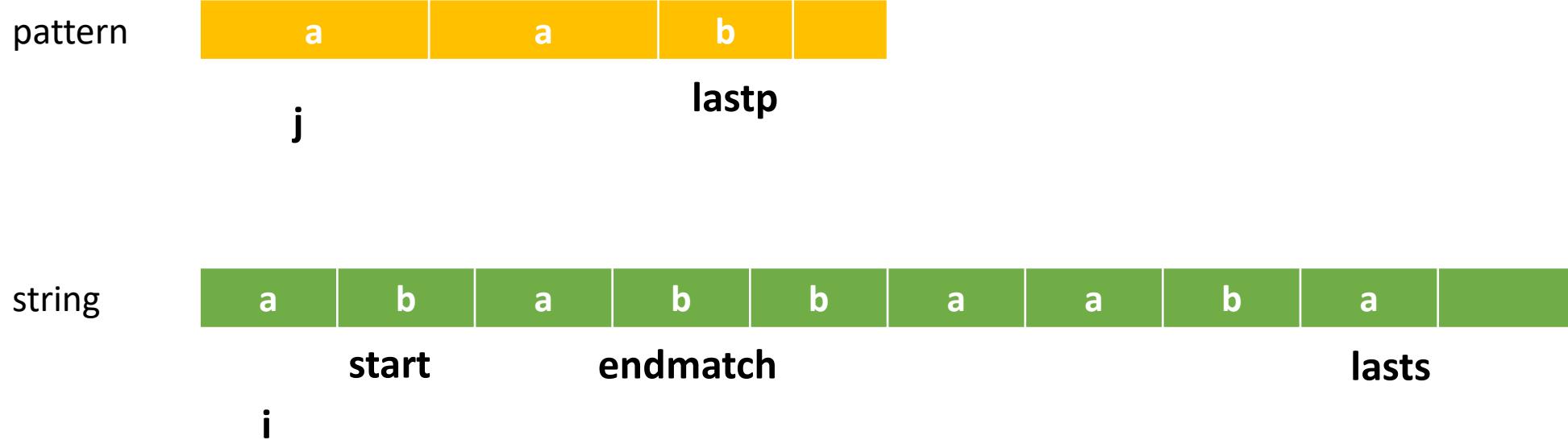
```
for (i=0;           endmatch<lasts;           endmatch++, start ++)  
{
```

```
    if (string[endmatch] == pat [lastp])  
        for(j=0, i=start ; j< lastp && string [i]==pat[j] ; i++ ,j++)
```

```
    ;
```

```
    if (j==lastp)  
        return start;
```

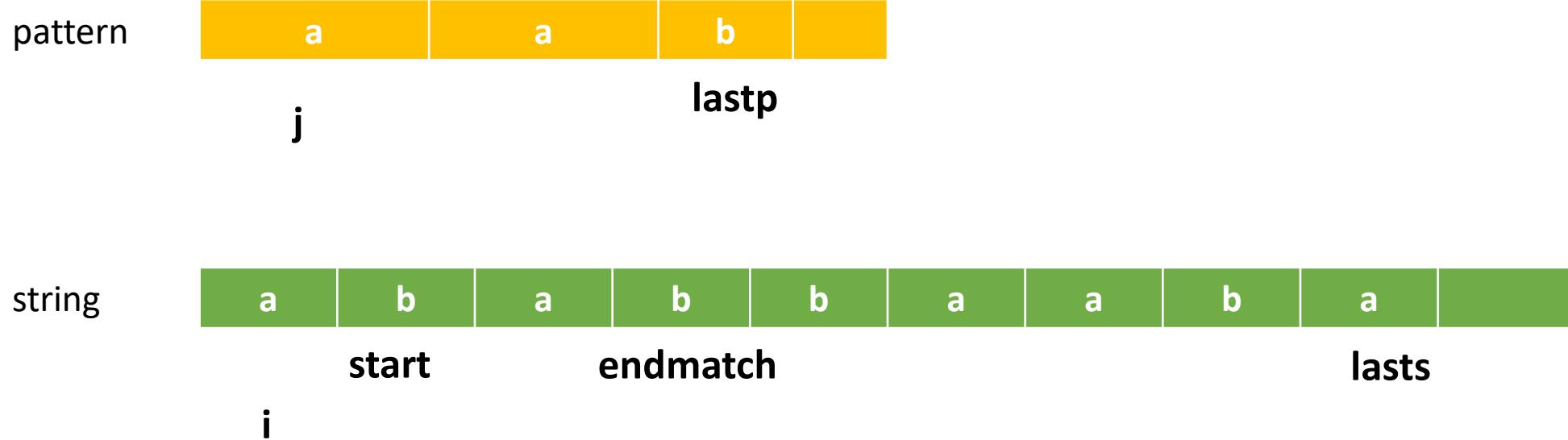
```
}
```



```

for (i=0;           endmatch<lasts;           endmatch++ , start ++)
{
    if (string[endmatch] == pat [lastp])
        for(j=0, i=start ; j< lastp && string [i]==pat[j] ; i++ ,j++)
    ;
    if (j==lastp)
        return start;
}

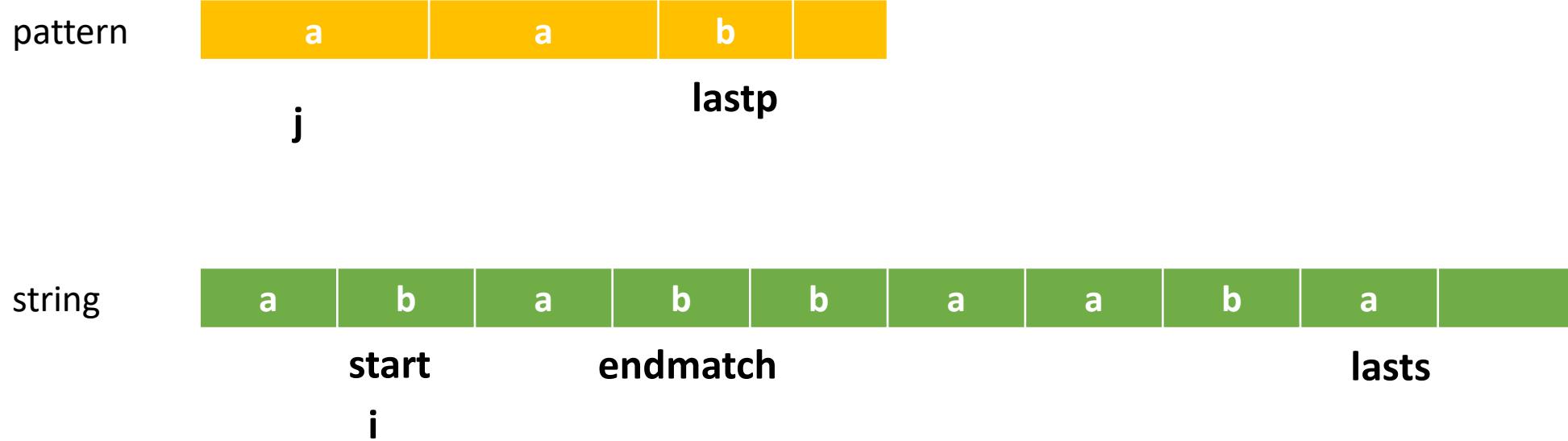
```



```

for (i=0;           endmatch<lasts;           endmatch++ , start ++)
{
    if (string[endmatch] == pat [lastp])
        for(j=0, i=start ; j< lastp && string [i]==pat[j] ; i++ ,j++)
            ;
        if (j==lastp)
            return start;
}

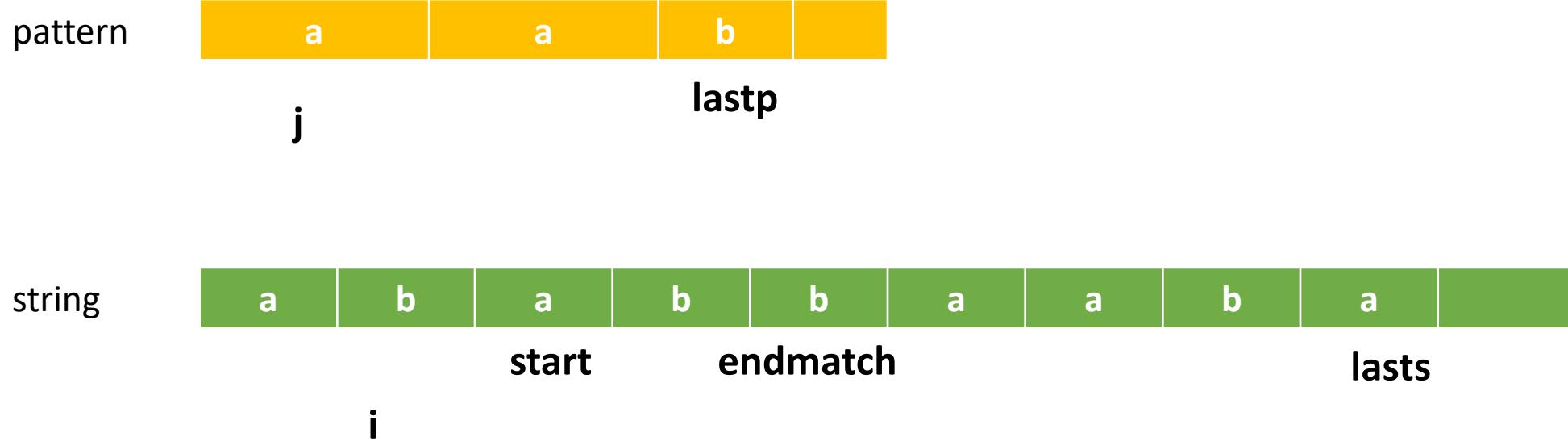
```



```

for (i=0;           endmatch<lasts;           endmatch++ , start ++)
{
    if (string[endmatch] == pat [lastp])
    for(j=0, i=start ; j< lastp && string [i]==pat[j] ; i++ ,j++)
        ;
    if (j==lastp)
        return start;
}

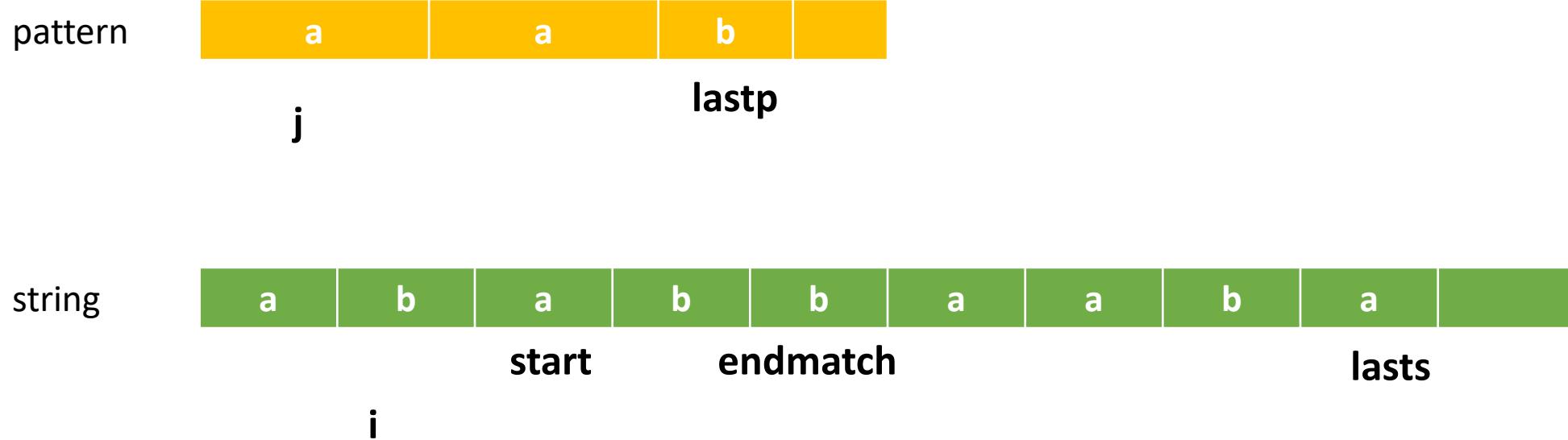
```



```

for (i=0;           endmatch<lasts;           endmatch++ , start ++)
{
    if (string[endmatch] == pat [lastp])
        for(j=0, i=start ; j< lastp && string [i]==pat[j] ; i++ ,j++)
    ;
    if (j==lastp)
        return start;
}

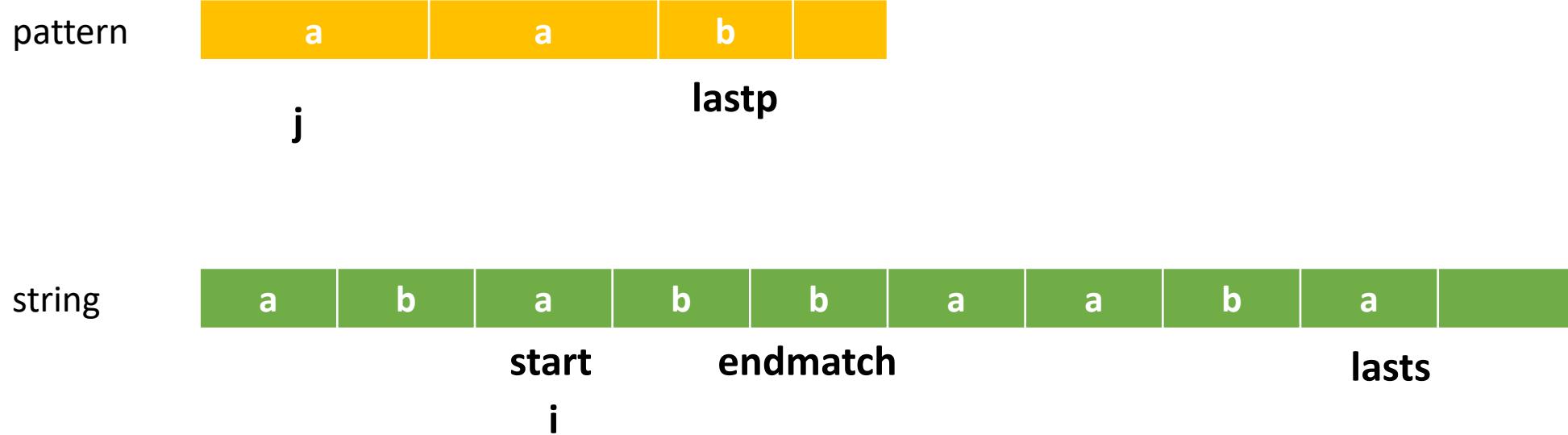
```



```

for (i=0;           endmatch<lasts;           endmatch++ , start ++)
{
    if (string[endmatch] == pat [lastp])
        for(j=0, i=start ; j< lastp && string [i]==pat[j] ; i++ ,j++)
            ;
        if (j==lastp)
            return start;
}

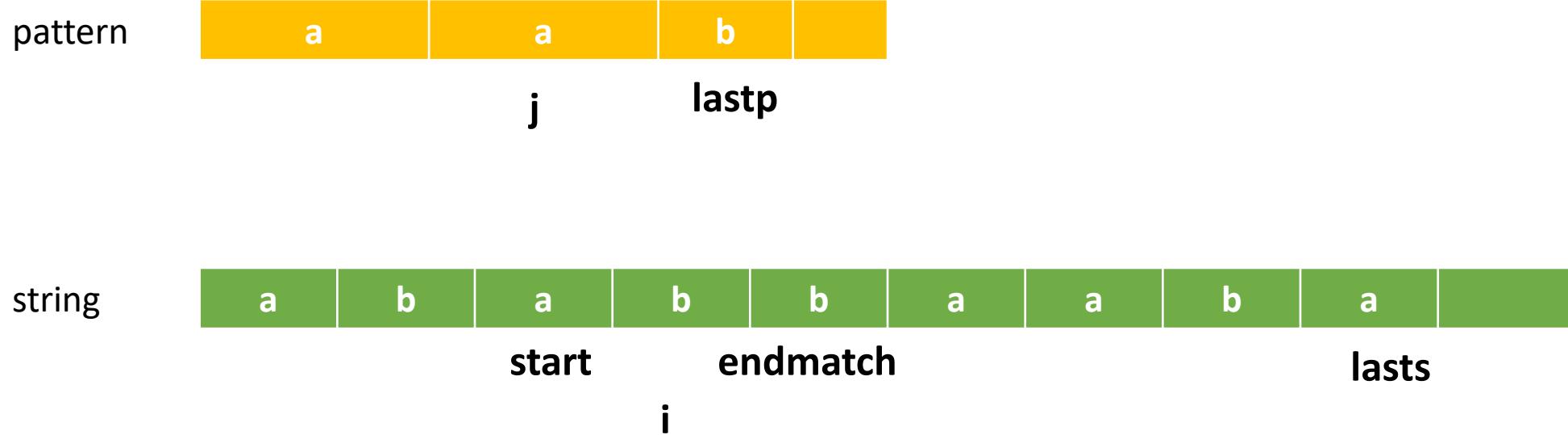
```



```

for (i=0;           endmatch<lasts;           endmatch++ , start ++)
{
    if (string[endmatch] == pat [lastp])
    for(j=0, i=start ; j< lastp && string [i]==pat[j] ; i++ ,j++)
        ;
    if (j==lastp)
        return start;
}

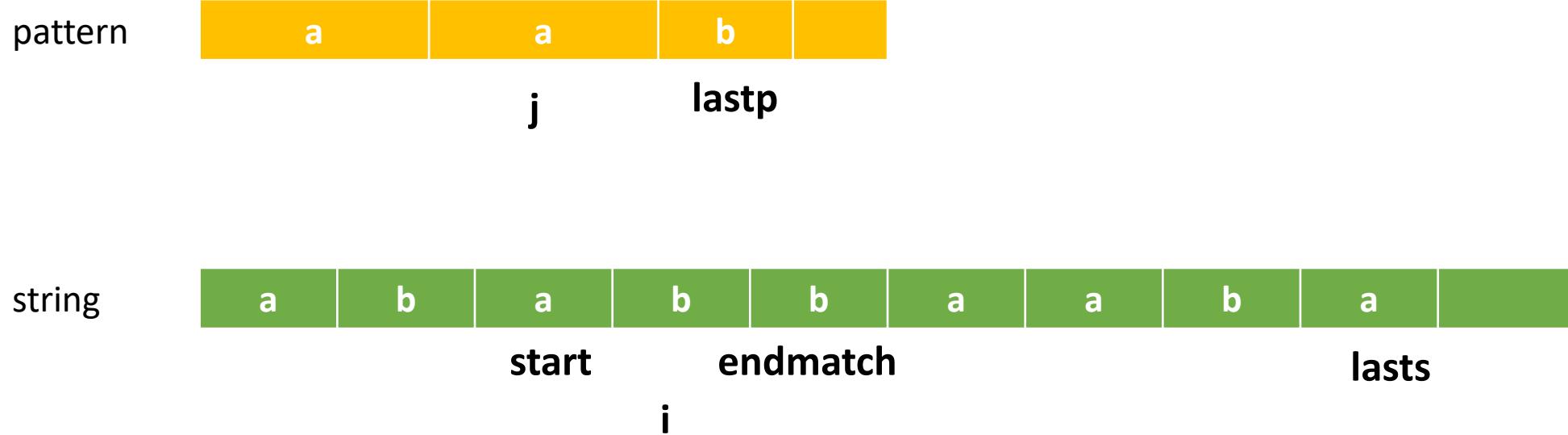
```



```

for (i=0;           endmatch<lasts;           endmatch++ , start ++)
{
    if (string[endmatch] == pat [lastp])
        for(j=0, i=start ; j< lastp && string [i]==pat[j] ; i++ ,j++)
            ;
        if (j==lastp)
            return start;
}

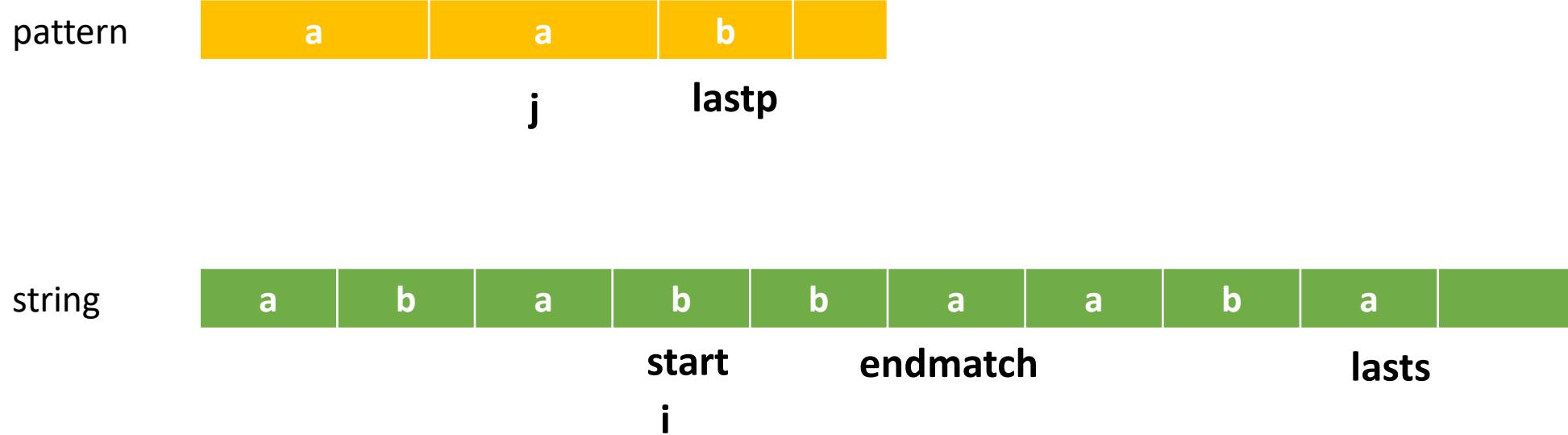
```



```

for (i=0;           endmatch<lasts;           endmatch++ , start ++)
{
    if (string[endmatch] == pat [lastp])
        for(j=0, i=start ; j< lastp && string [i]==pat[j] ; i++ ,j++)
            ;
        if (j==lastp)
            return start;
}

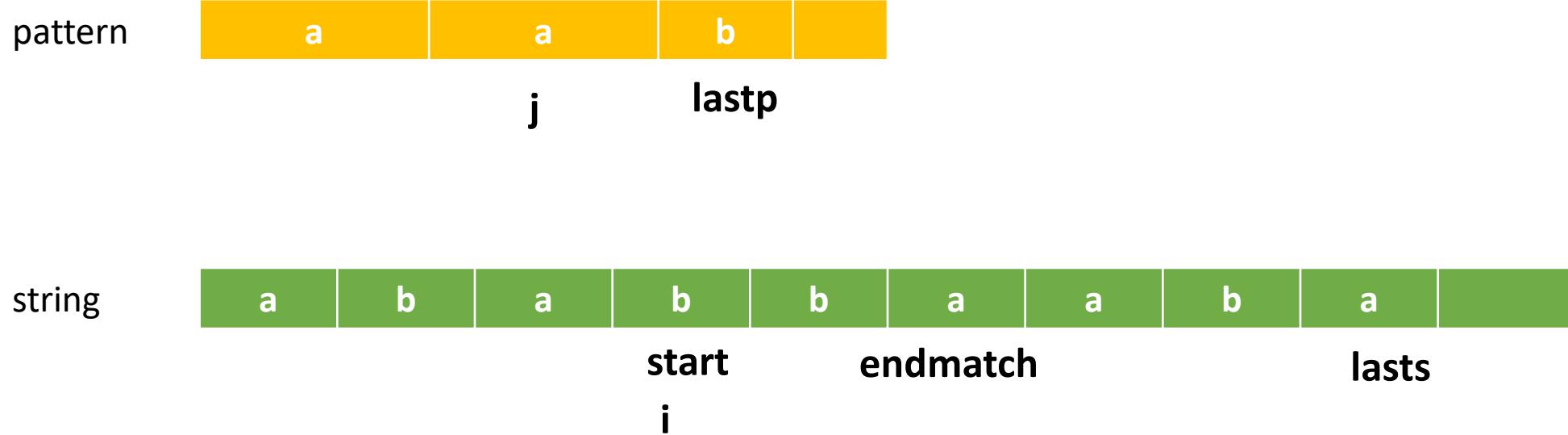
```



```

for (i=0;           endmatch<lasts;           endmatch++ , start ++)
{
    if (string[endmatch] == pat [lastp])
        for(j=0, i=start ; j< lastp && string [i]==pat[j] ; i++ ,j++)
            ;
        if (j==lastp)
            return start;
}

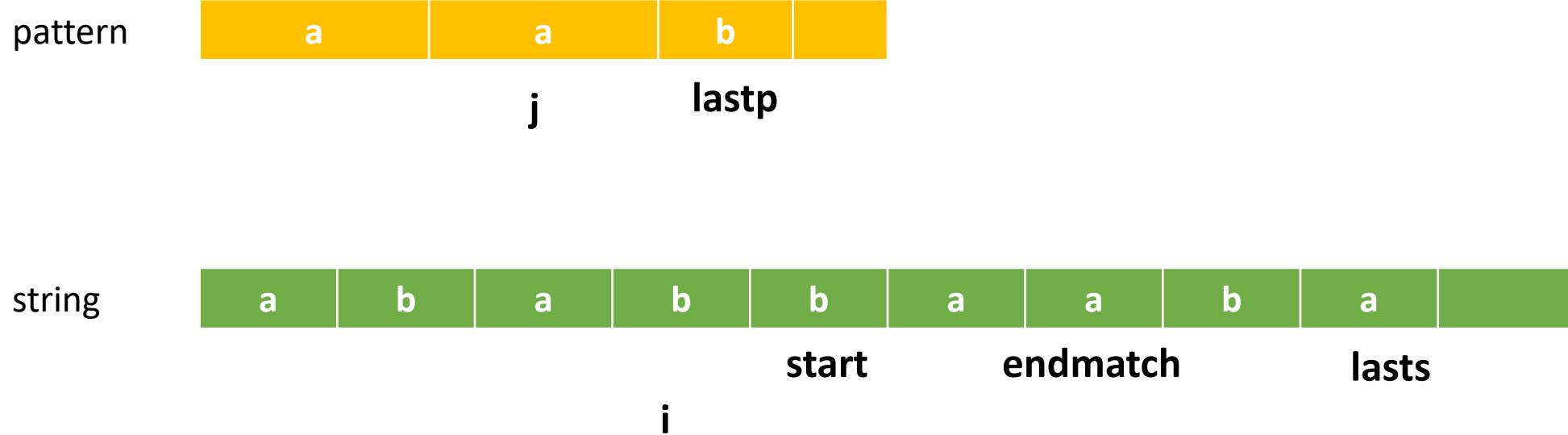
```



```

for (i=0;           endmatch<lasts;           endmatch++ , start ++)
{
    if (string[endmatch] == pat [lastp])
        for(j=0, i=start ; j< lastp && string [i]==pat[j] ; i++ ,j++)
        ;
        if (j==lastp)
            return start;
}

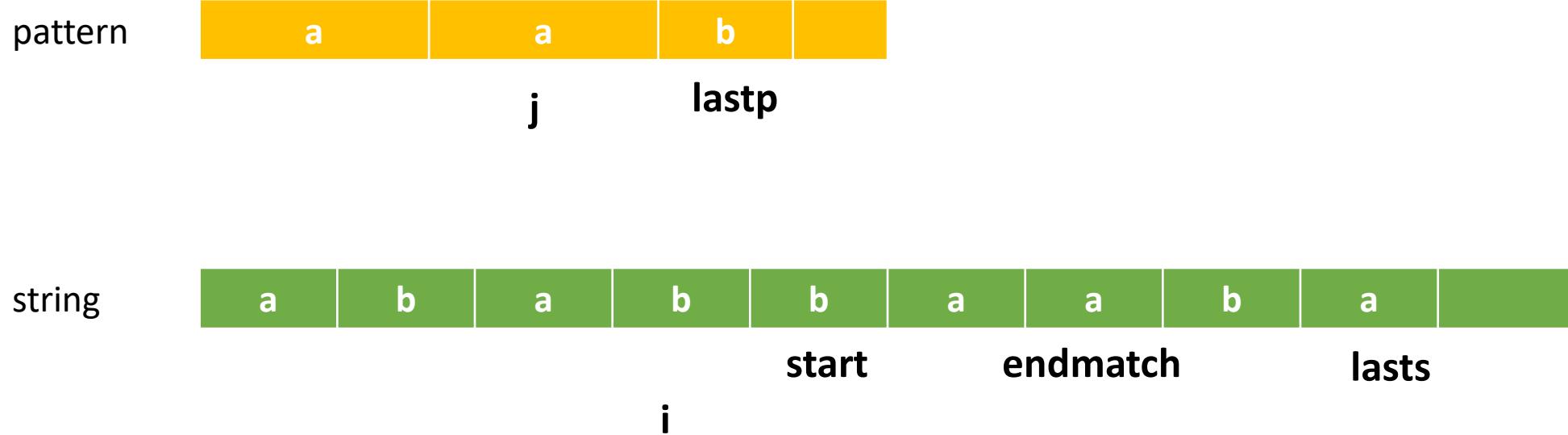
```



```

for (i=0;           endmatch<lasts;           endmatch++ , start ++)
{
    if (string[endmatch] == pat [lastp])
        for(j=0, i=start ; j< lastp && string [i]==pat[j] ; i++ ,j++)
    ;
        if (j==lastp)
            return start;
}

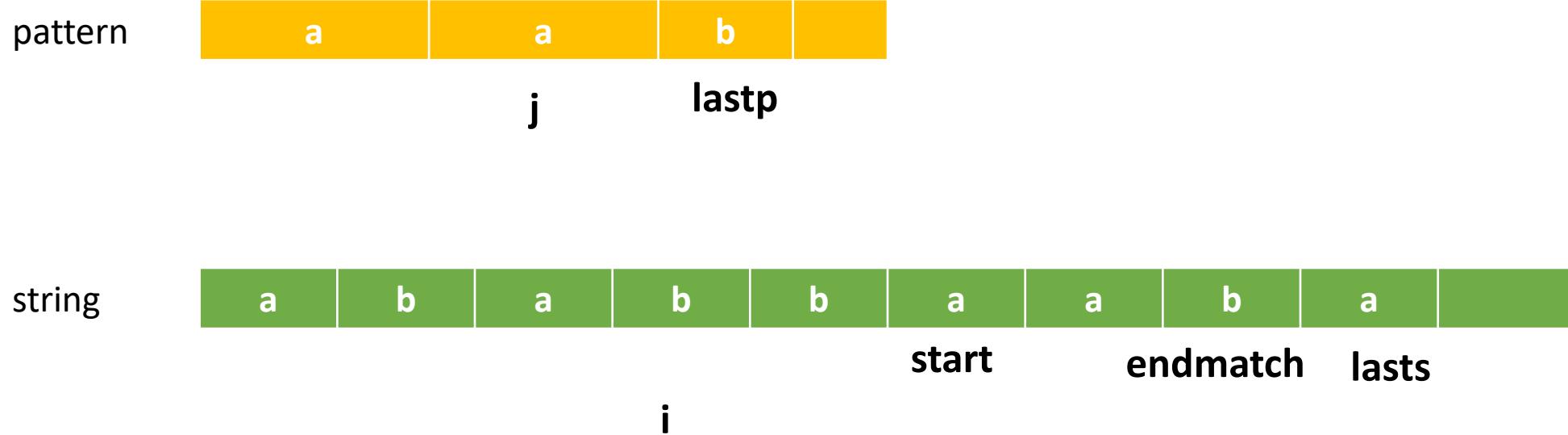
```



```

for (i=0;           endmatch<lasts;           endmatch++ , start ++)
{
    if (string[endmatch] == pat [lastp])
        for(j=0, i=start ; j< lastp && string [i]==pat[j] ; i++ ,j++)
        ;
        if (j==lastp)
            return start;
}

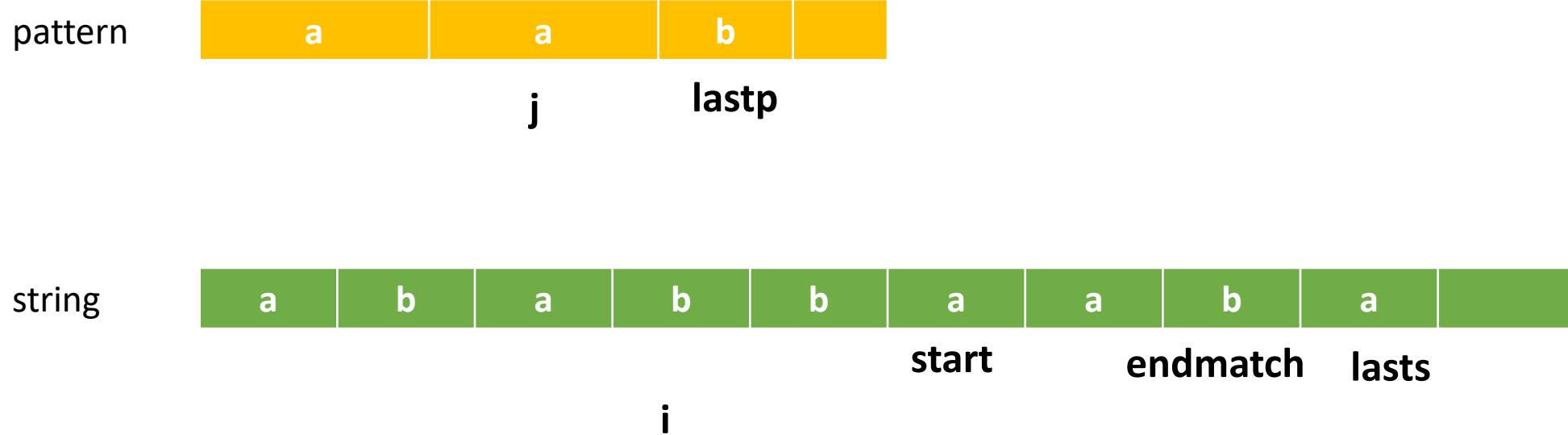
```



```

for (i=0;           endmatch<lasts;           endmatch++ , start ++)
{
    if (string[endmatch] == pat [lastp])
        for(j=0, i=start ; j< lastp && string [i]==pat[j] ; i++ ,j++)
    ;
    if (j==lastp)
        return start;
}

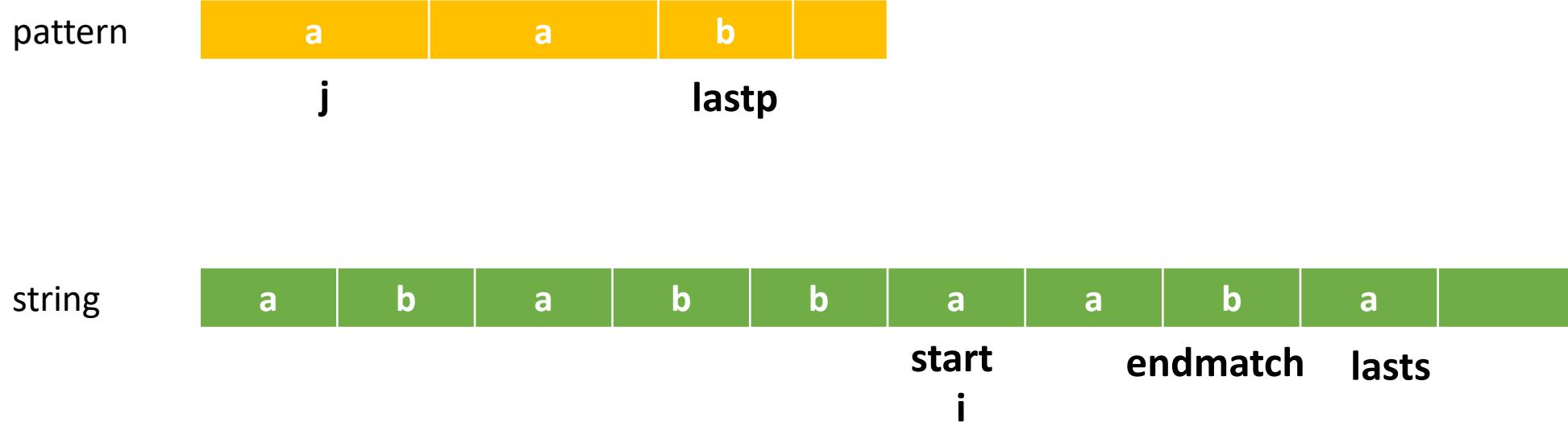
```



```

for (i=0;           endmatch<lasts;           endmatch++ , start ++)
{
    if (string[endmatch] == pat [lastp])
        for(j=0, i=start ; j< lastp && string [i]==pat[j] ; i++ ,j++)
            ;
        if (j==lastp)
            return start;
}

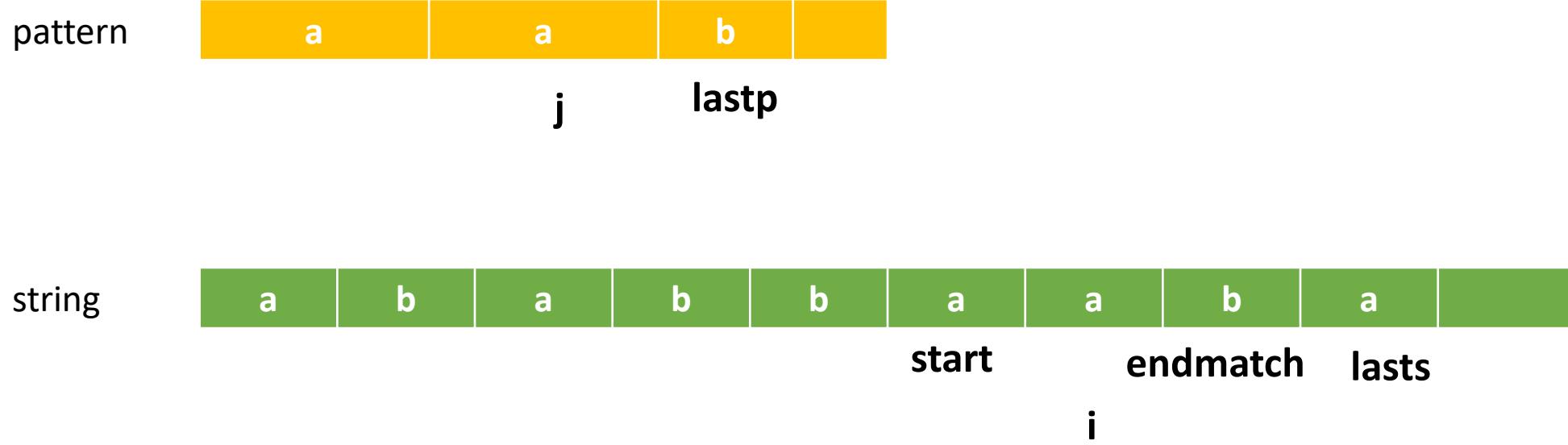
```



```

for (i=0;           endmatch<lasts;           endmatch++ , start ++)
{
    if (string[endmatch] == pat [lastp])
for(j=0, i=start ; j< lastp && string [i]==pat[j] ; i++ ,j++)
    ;
    if (j==lastp)
        return start;
}

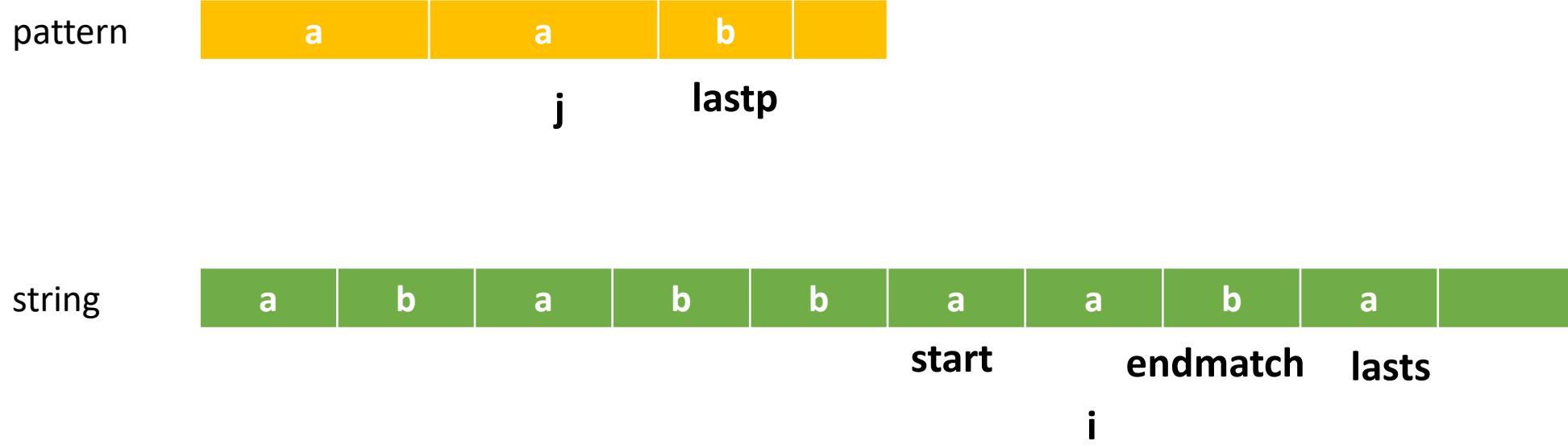
```



```

for (i=0;           endmatch<lasts;           endmatch++ , start ++)
{
    if (string[endmatch] == pat [lastp])
        for(j=0, i=start ; j< lastp && string [i]==pat[j] ; i++ ,j++)
            ;
        if (j==lastp)
            return start;
}

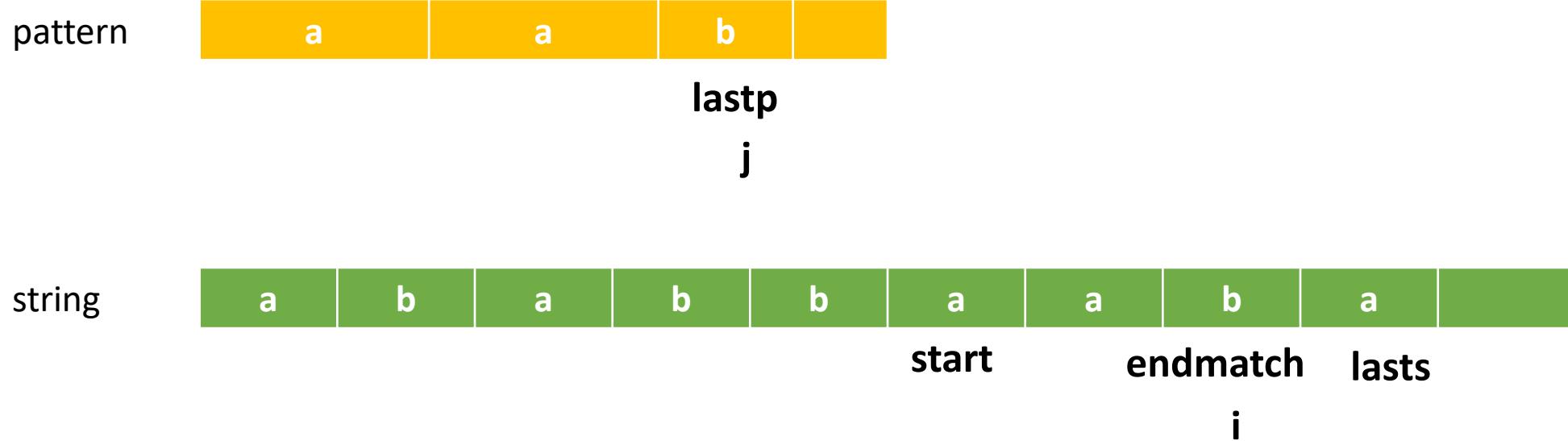
```



```

for (i=0;           endmatch<lasts;           endmatch++ , start ++)
{
    if (string[endmatch] == pat [lastp])
        for(j=0, i=start ; j< lastp && string [i]==pat[j] ; i++ ,j++)
            ;
        if (j==lastp)
            return start;
}

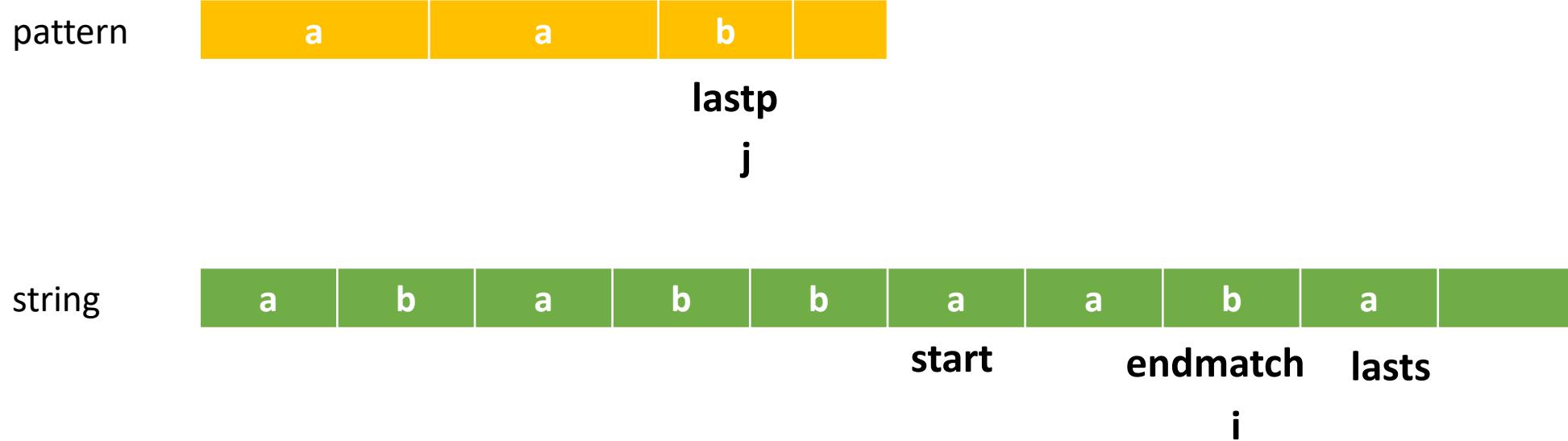
```



```

for (i=0;           endmatch<lasts;           endmatch++ , start ++)
{
    if (string[endmatch] == pat [lastp])
        for(j=0, i=start ; j< lastp && string [i]==pat[j] ; i++ ,j++)
            ;
        if (j==lastp)
            return start;
}

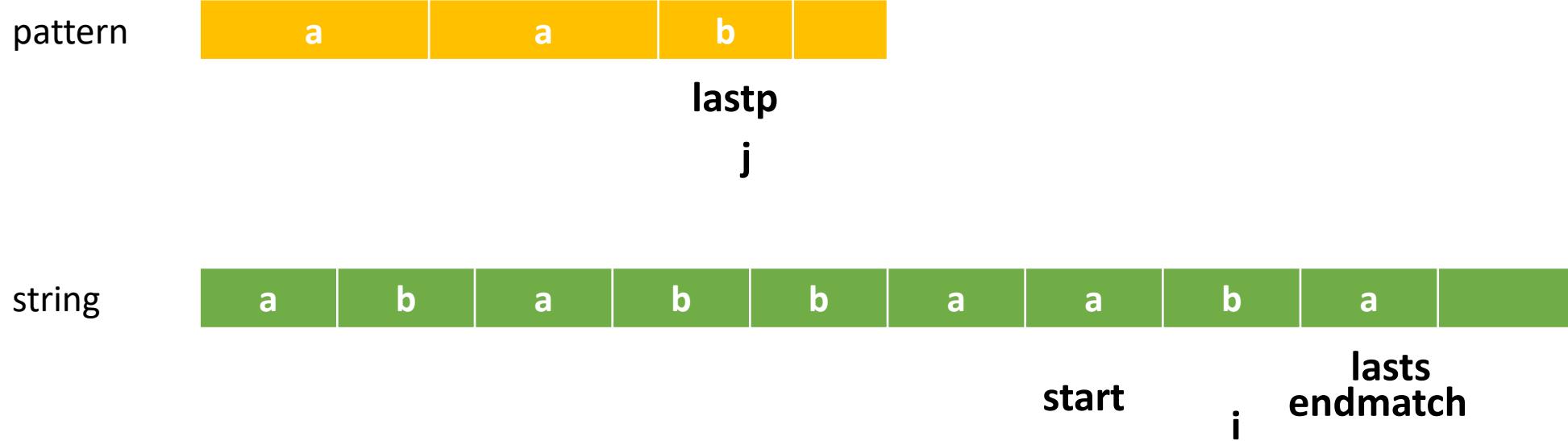
```



```

for (i=0;           endmatch<lasts;           endmatch++ , start ++)
{
    if (string[endmatch] == pat [lastp])
        for(j=0, i=start ; j< lastp && string [i]==pat[j] ; i++ ,j++)
            ;
        if (j==lastp)
            return start;
}

```



```
for (i=0;           endmatch<lasts;           endmatch++ , start ++)
{

```

```
    if (string[endmatch] == pat [lastp])
        for(j=0, i=start ; j< lastp && string [i]==pat[j] ; i++ ,j++)

```

```
    ;
```

```
    if (j==lastp)
        return start;
```

```
}
```

*N*find algorithm – Pattern Matching

```
int nfind(char *string; char *pat)
{ /* match the last character of pattern first, and
   then match from the beginning */
  int i,j,start = 0;
  int lasts = strlen(string)-1;
  int lastp = strlen(pat)-1;
  int endmatch = lastp;

  for (i = 0; endmatch <= lasts; endmatch++, start++) {
    if (string[endmatch] == pat[lastp])
      for (j = 0, i = start; j < lastp &&
           string[i] == pat[j]; i++,j++)
        ;
    if (j == lastp)
      return start; /* successful */
  }
  return -1;
}
```

KMP – Algorithm for pattern Matching

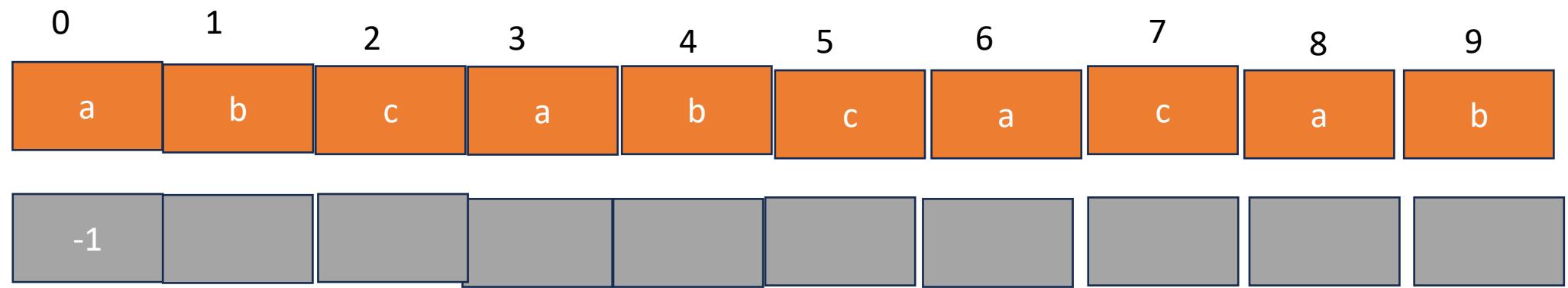
Definition: If $p = p_0p_1 \cdots p_{n-1}$ is a pattern, then its *failure function*, f , is defined as:

$$f(j) = \begin{cases} \text{largest } i < j \text{ such that } p_0p_1 \cdots p_i = p_{j-i}p_{j-i+2} \cdots p_j & \text{if such an } i \geq 0 \text{ exists} \\ -1 & \text{otherwise} \end{cases} \quad \square$$

For the example pattern, $pat = abcabcacab$, we have:

j	0	1	2	3	4	5	6	7	8	9
pat	a	b	c	a	b	c	a	c	a	b
f	-1	-1	-1	0	1	2	3	-1	0	1

<i>j</i>	0	1	2	3	4	5	6	7	8	9
<i>pat</i>	a	b	c	a	b	c	a	c	a	b
<i>f</i>	-1	-1	-1	0	1	2	3	-1	0	1

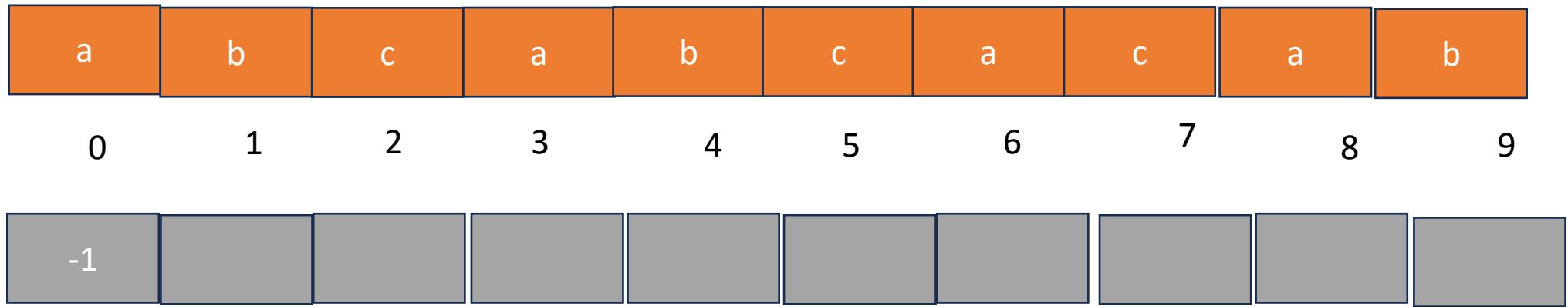


```

for (j=1; j<n ; j++)
{
    i=failure[j-1]; // i should have the previous value , i= -1
}

```

<i>j</i>	0	1	2	3	4	5	6	7	8	9
<i>pat</i>	a	b	c	a	b	c	a	c	a	b
<i>f</i>	-1	-1	-1	0	1	2	3	-1	0	1

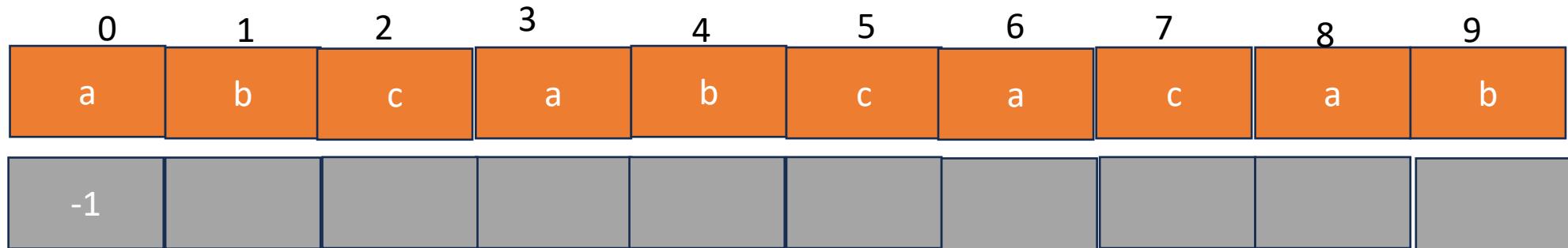


```

for (j=1; j<n ; j++)
{
    i=failure[j-1]; // i should have the previous value i= -1
    while((pat[j]!= pat[i+1]) && i>=0)
        i=failure[i];
}

```

<i>j</i>	0	1	2	3	4	5	6	7	8	9
<i>pat</i>	a	b	c	a	b	c	a	c	a	b
<i>f</i>	-1	-1	-1	0	1	2	3	-1	0	1

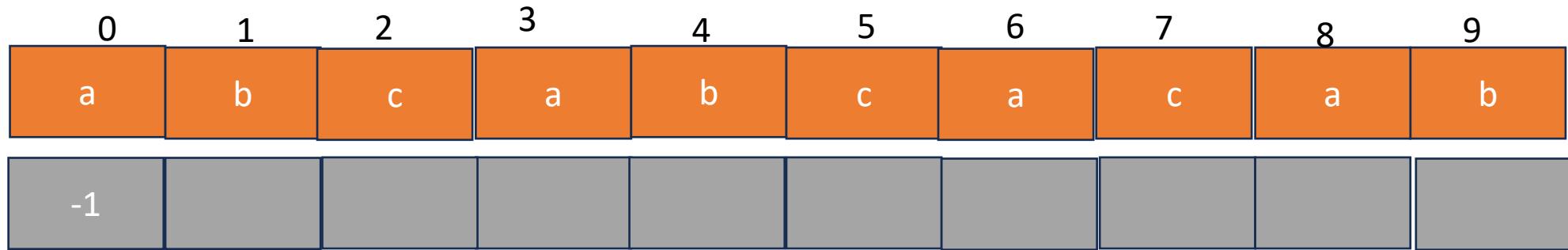


```

for (j=1; j<n ; j++)
{
    i=failure[j-1]; // i should have the previous value i= -1,j=1
    while((pat[j]!= pat[i+1]) && i>=0)
        i=failure[i];
    if(pat[j]==pat[i+1])
        failure[j]= i+1;
    else
        failure[j]= -1;
}

```

<i>j</i>	0	1	2	3	4	5	6	7	8	9
<i>pat</i>	a	b	c	a	b	c	a	c	a	b
<i>f</i>	-1	-1	-1	0	1	2	3	-1	0	1



```
for (j=1; j<n ; j++)
```

```
{
```

```
i=failure[j-1]; // i should have the previous value i= 0,j=1
```

```
while((pat[j]!= pat[i+1]) && i>=0)
```

```
    i=failure[i];
```

```
if(pat[j]==pat[i+1])
```

```
    failure[j]= i+1;
```

pat[1]= =pat[0]

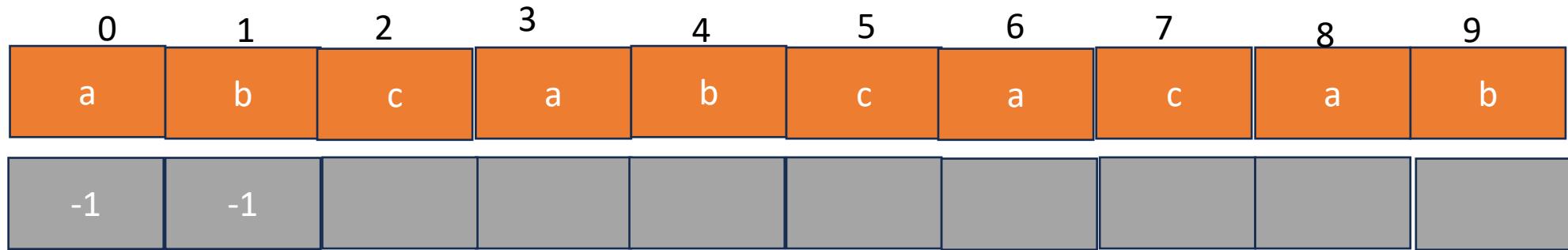
a==b

```
else
```

```
    failure[j]= -1;
```

```
}
```

<i>j</i>	0	1	2	3	4	5	6	7	8	9
<i>pat</i>	a	b	c	a	b	c	a	c	a	b
<i>f</i>	-1	-1	-1	0	1	2	3	-1	0	1

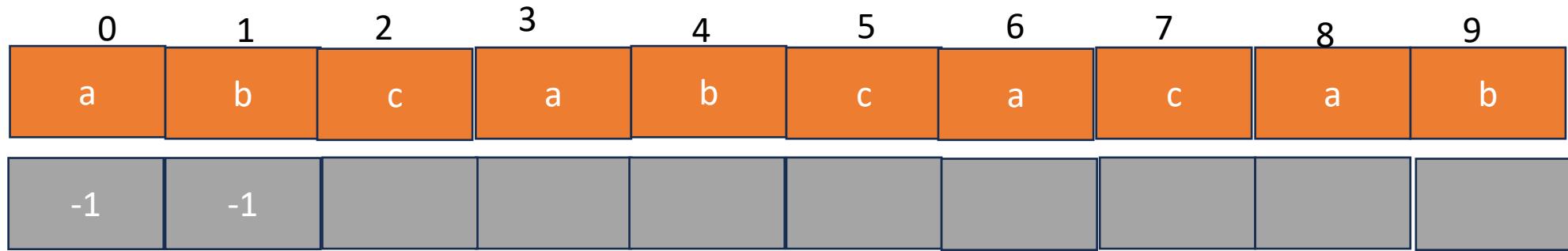


```

for (j=1; j<n ; j++)
{
    i=failure[j-1]; // i should have the previous value i= 0,j=1
    while((pat[j]!= pat[i+1]) && i>=0)
        i=failure[i];
    if(pat[j]==pat[i+1])
        failure[j]= i+1;           pat[1]= =pat[0]
    else                               a==b
        failure[j]= -1;          failure[j]=-1;
}

```

<i>j</i>	0	1	2	3	4	5	6	7	8	9
<i>pat</i>	a	b	c	a	b	c	a	c	a	b
<i>f</i>	-1	-1	-1	0	1	2	3	-1	0	1

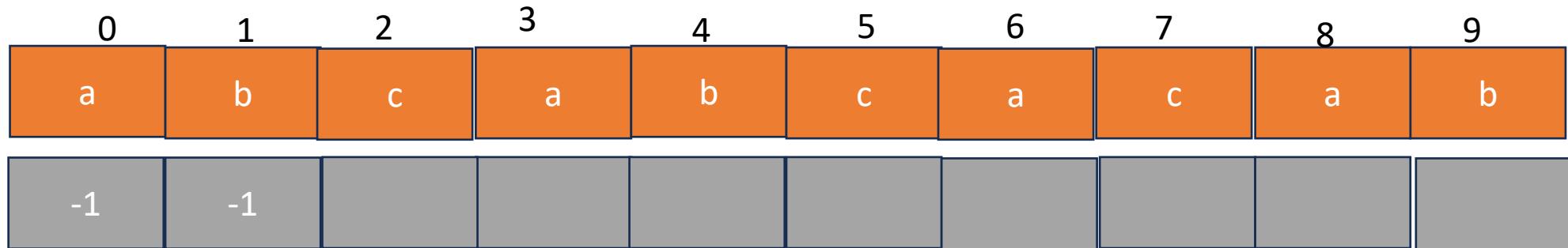


```

for (j=1; j<n ; j++)
{
    i=failure[j-1]; // i should have the previous value i= -1 ,j=2
    while((pat[j]!= pat[i+1]) && i>=0)
        i=failure[i];
    if(pat[j]==pat[i+1])
        failure[j]= i+1;
    else
        failure[j]= -1;
}

```

<i>j</i>	0	1	2	3	4	5	6	7	8	9
<i>pat</i>	a	b	c	a	b	c	a	c	a	b
<i>f</i>	-1	-1	-1	0	1	2	3	-1	0	1

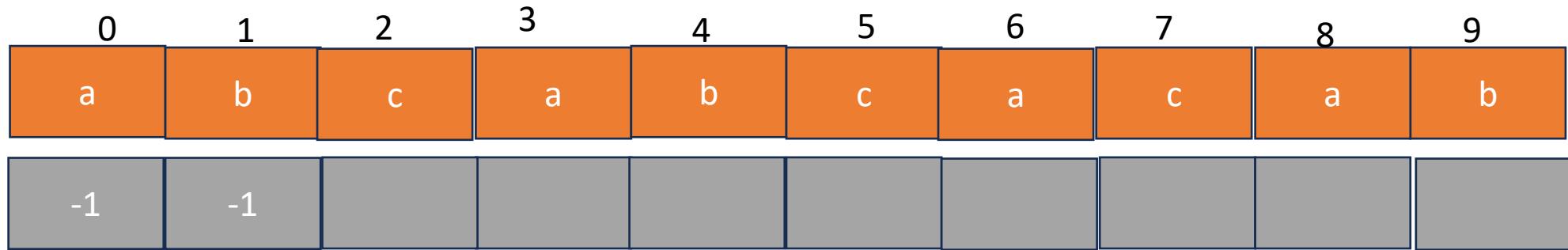


```

for (j=1; j<n ; j++)
{
    i=failure[j-1]; // i should have the previous value i= -1 ,j=2
    while((pat[j]!= pat[i+1]) && i>=0)
        i=failure[i];
    if(pat[j]==pat[i+1])
        failure[j]= i+1;
    else
        failure[j]= -1;
}

```

<i>j</i>	0	1	2	3	4	5	6	7	8	9
<i>pat</i>	a	b	c	a	b	c	a	c	a	b
<i>f</i>	-1	-1	-1	0	1	2	3	-1	0	1



```
for (j=1; j<n ; j++)
```

```
{
```

```
i=failure[j-1]; // i should have the previous value i= -1 ,j=2
```

```
while((pat[j]!= pat[i+1]) && i>=0)
```

```
    i=failure[i];
```

```
if(pat[j]==pat[i+1])
```

pat[2]= =pat[0]

```
    failure[j]= i+1;
```

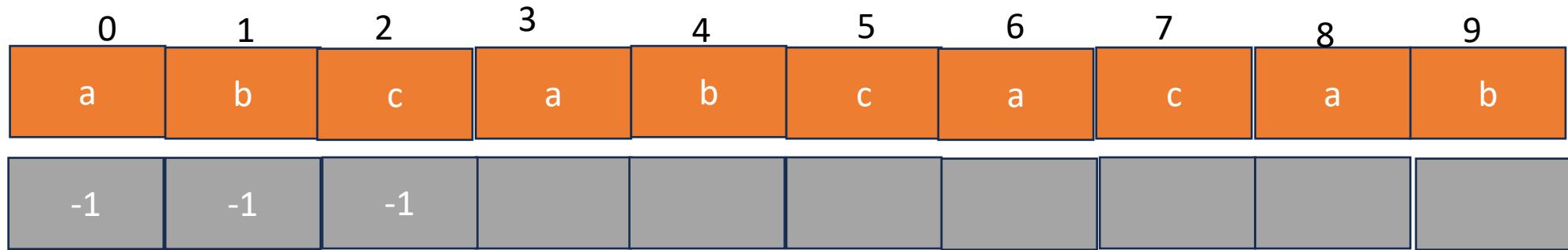
e==a

```
else
```

```
    failure[j]= -1;
```

```
}
```

<i>j</i>	0	1	2	3	4	5	6	7	8	9
<i>pat</i>	a	b	c	a	b	c	a	c	a	b
<i>f</i>	-1	-1	-1	0	1	2	3	-1	0	1



```
for (j=1; j<n ; j++)
```

```
{
```

```
    i=failure[j-1]; // i should have the previous value i= -1 ,j=2
```

```
    while((pat[j]!= pat[i+1]) && i>=0)
```

```
        i=failure[i];
```

```
        if(pat[j]==pat[i+1])
```

pat[2]= =pat[0]

```
            failure[j]= i+1;
```

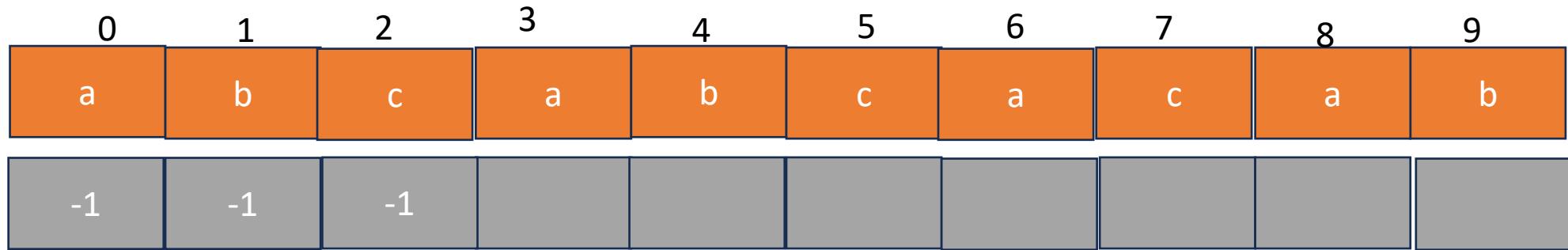
e==a

```
    else
```

```
        failure[j]= -1;
```

```
}
```

<i>j</i>	0	1	2	3	4	5	6	7	8	9
<i>pat</i>	a	b	c	a	b	c	a	c	a	b
<i>f</i>	-1	-1	-1	0	1	2	3	-1	0	1



```
for (j=1; j<n ; j++)
```

```
{
```

```
    i=failure[j-1]; // i should have the previous value i= -1 ,j=2
```

```
    while((pat[j]!= pat[i+1]) && i>=0)
```

```
        i=failure[i];
```

```
        if(pat[j]==pat[i+1])
```

pat[2]= =pat[0]

```
            failure[j]= i+1;
```

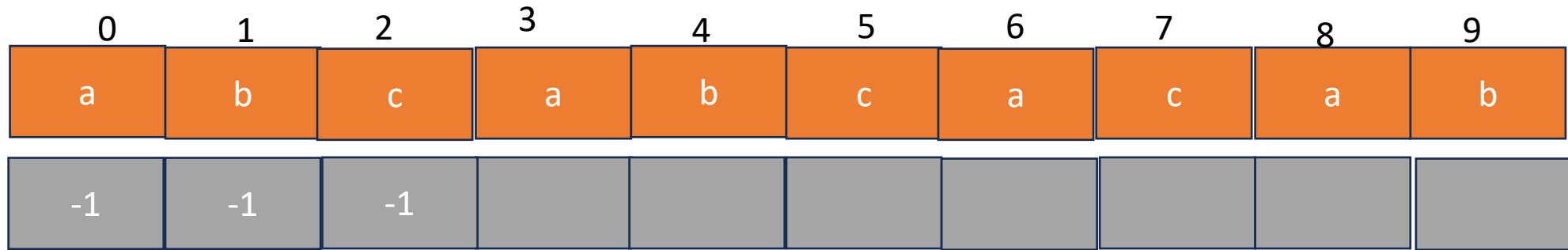
e==a

```
    else
```

```
        failure[j]= -1;
```

```
}
```

<i>j</i>	0	1	2	3	4	5	6	7	8	9
<i>pat</i>	a	b	c	a	b	c	a	c	a	b
<i>f</i>	-1	-1	-1	0	1	2	3	-1	0	1

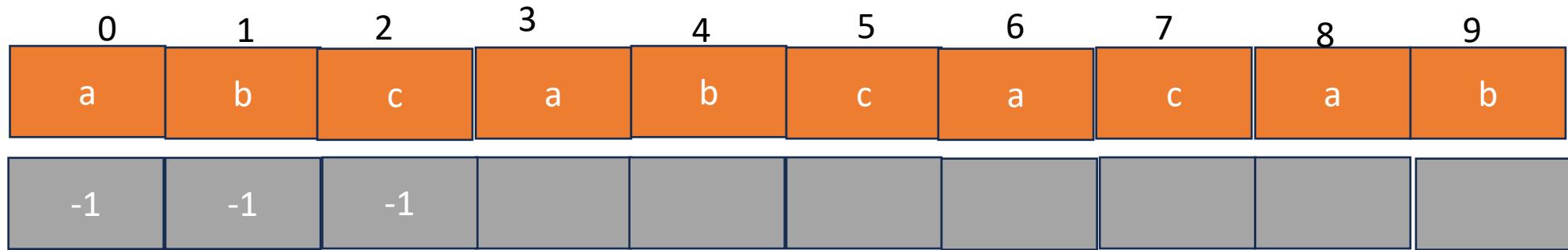


```

for (j=1; j<n ; j++)
{
    i=failure[j-1]; // i should have the previous value i= -1 ,j=3
    while((pat[j]!= pat[i+1]) && i>=0)
        i=failure[i];
    if(pat[j]==pat[i+1])
        failure[j]= i+1;
    else
        failure[j]= -1;
}

```

<i>j</i>	0	1	2	3	4	5	6	7	8	9
<i>pat</i>	a	b	c	a	b	c	a	c	a	b
<i>f</i>	-1	-1	-1	0	1	2	3	-1	0	1



```
for (j=1; j<n ; j++)
```

```
{
```

```
    i=failure[j-1]; // i should have the previous value i= -1 ,j=3
```

```
    while((pat[j]!= pat[i+1]) && i>=0)
```

```
        i=failure[i];
```

```
        if(pat[j]==pat[i+1])
```

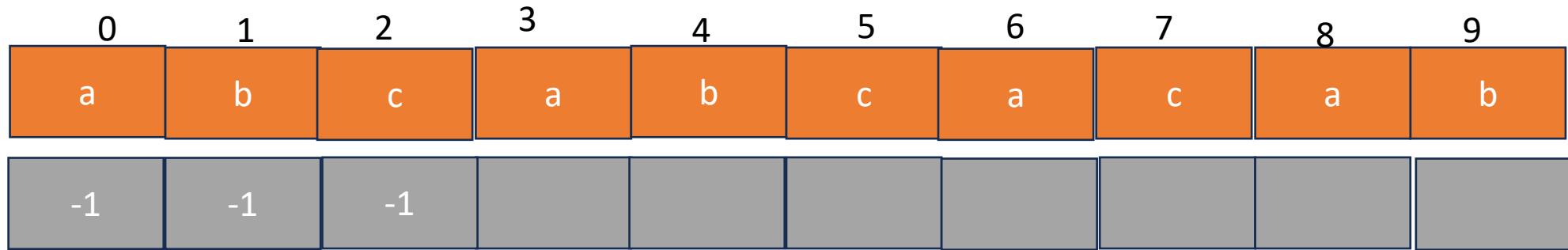
```
            failure[j]= i+1;
```

```
        else
```

```
            failure[j]= -1;
```

```
}
```

<i>j</i>	0	1	2	3	4	5	6	7	8	9
<i>pat</i>	a	b	c	a	b	c	a	c	a	b
<i>f</i>	-1	-1	-1	0	1	2	3	-1	0	1

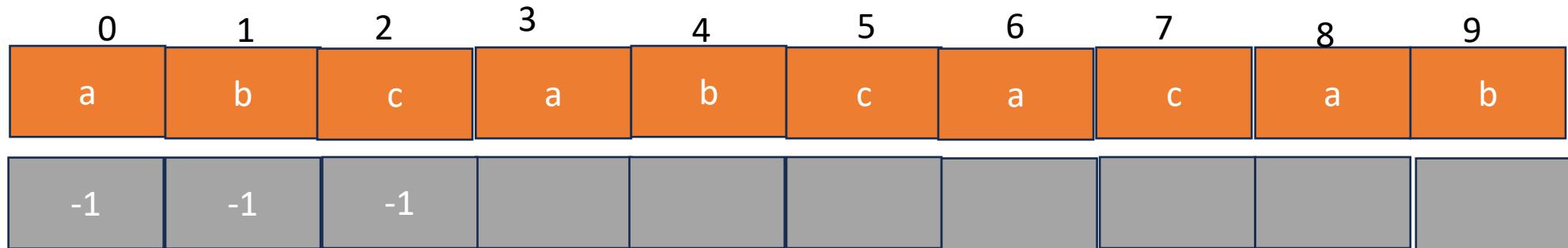


```

for (j=1; j<n ; j++)
{
    i=failure[j-1]; // i should have the previous value i= -1 ,j=3
    while((pat[j]!= pat[i+1]) && i>=0)
        i=failure[i];
    if(pat[j]==pat[i+1])
        failure[j]= i+1;
    else
        failure[j]= -1;
}

```

<i>j</i>	0	1	2	3	4	5	6	7	8	9
<i>pat</i>	a	b	c	a	b	c	a	c	a	b
<i>f</i>	-1	-1	-1	0	1	2	3	-1	0	1



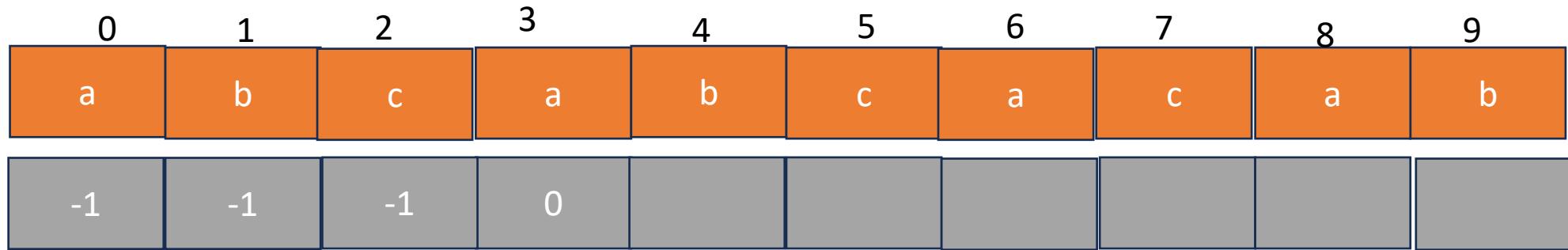
```

for (j=1; j<n ; j++)
{
    i=failure[j-1]; // i should have the previous value i= -1 ,j=3
    while((pat[j]!= pat[i+1]) && i>=0)
        i=failure[i];
    if(pat[j]==pat[i+1])
        failure[j]= i+1;
    else
        failure[j]= -1;
}

```

pat[3]= =pat[0]
a==a

<i>j</i>	0	1	2	3	4	5	6	7	8	9
<i>pat</i>	a	b	c	a	b	c	a	c	a	b
<i>f</i>	-1	-1	-1	0	1	2	3	-1	0	1

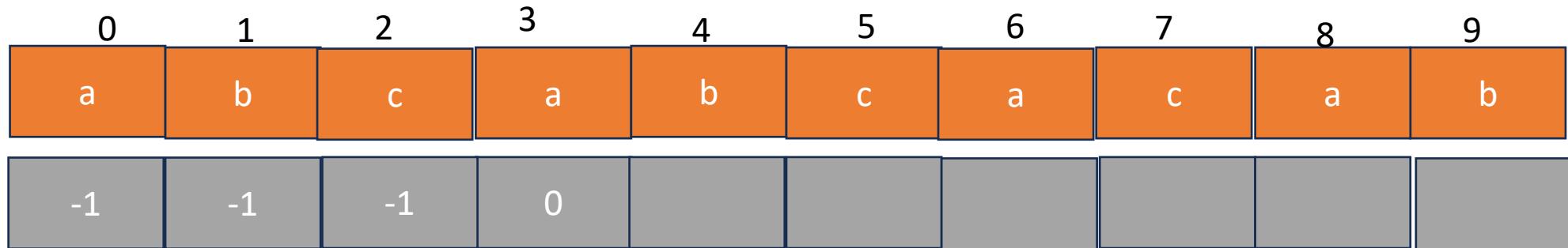


```

for (j=1; j<n ; j++)
{
    i=failure[j-1]; // i should have the previous value i= -1 ,j=3
    while((pat[j]!= pat[i+1]) && i>=0)
        i=failure[i];
    if(pat[j]==pat[i+1])
        failure[j]= i+1;
    else
        failure[j]=-1;
}

```

<i>j</i>	0	1	2	3	4	5	6	7	8	9
<i>pat</i>	a	b	c	a	b	c	a	c	a	b
<i>f</i>	-1	-1	-1	0	1	2	3	-1	0	1

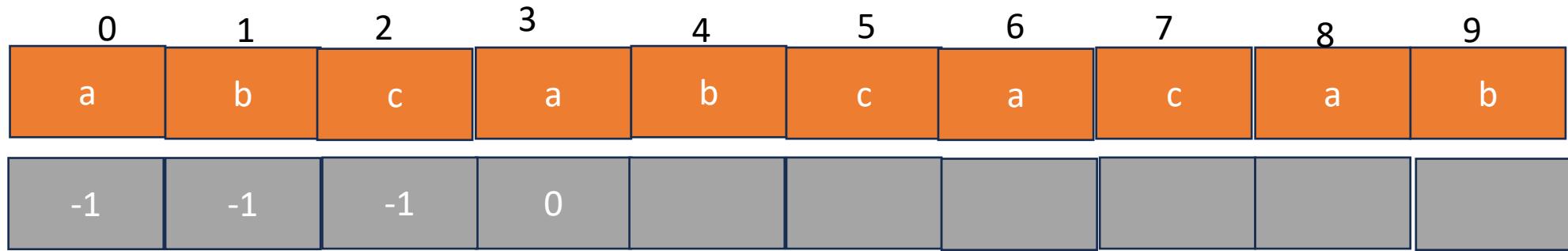


```

for (j=1; j<n ; j++)
{
    i=failure[j-1]; // i should have the previous value i= -1 ,j=3
    while((pat[j]!= pat[i+1]) && i>=0)
        i=failure[i];
    if(pat[j]==pat[i+1])
        failure[j]= i+1;
    else
        failure[j]= -1;
}

```

<i>j</i>	0	1	2	3	4	5	6	7	8	9
<i>pat</i>	a	b	c	a	b	c	a	c	a	b
<i>f</i>	-1	-1	-1	0	1	2	3	-1	0	1



```
for (j=1; j<n ; j++)
```

```
{
```

i=failure[j-1]; // i should have the previous value i= 0 ,j=4

```
while((pat[j]!= pat[i+1]) && i>=0)
```

```
    i=failure[i];
```

```
    if(pat[j]==pat[i+1])
```

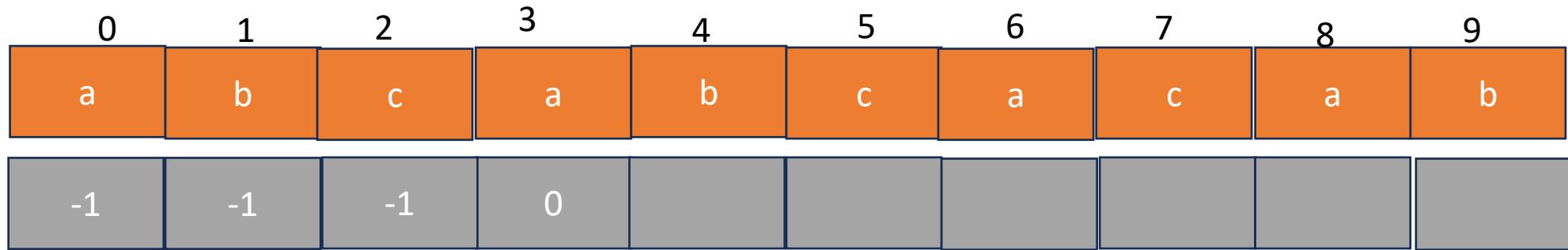
```
        failure[j]= i+1;
```

```
    else
```

```
        failure[j]= -1;
```

```
}
```

<i>j</i>	0	1	2	3	4	5	6	7	8	9
<i>pat</i>	a	b	c	a	b	c	a	c	a	b
<i>f</i>	-1	-1	-1	0	1	2	3	-1	0	1

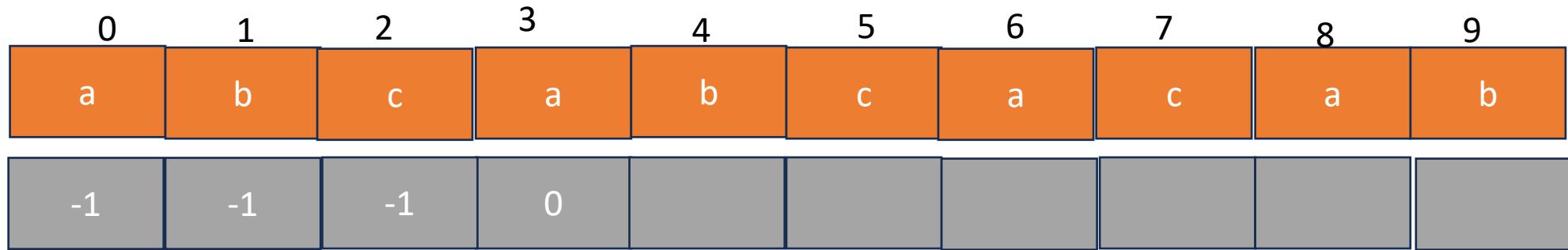


```

for (j=1; j<n ; j++)
{
    i=failure[j-1];           // i should have the previous value i= 0 ,j=4
    while((pat[j]!= pat[i+1]) && i>=0)           b!=b
        i=failure[i];
        if(pat[j]==pat[i+1])
            failure[j]= i+1;
        else
            failure[j]= -1;
}

```

<i>j</i>	0	1	2	3	4	5	6	7	8	9
<i>pat</i>	a	b	c	a	b	c	a	c	a	b
<i>f</i>	-1	-1	-1	0	1	2	3	-1	0	1

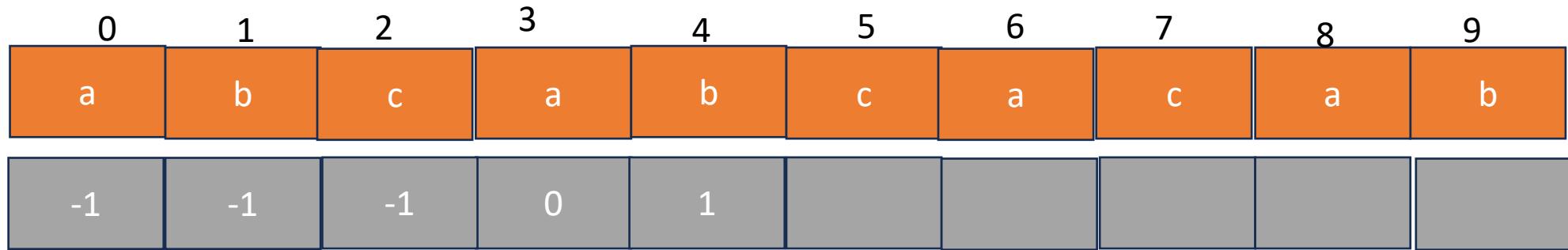


```

for (j=1; j<n ; j++)
{
    i=failure[j-1];           // i should have the previous value i= 0 ,j=4
    while((pat[j]!= pat[i+1]) && i>=0)
        i=failure[i];
    if(pat[j]==pat[i+1])      pat[4]==pat[1]
        failure[j]= i+1;       b=b
    else
        failure[j]=-1;
}

```

<i>j</i>	0	1	2	3	4	5	6	7	8	9
<i>pat</i>	a	b	c	a	b	c	a	c	a	b
<i>f</i>	-1	-1	-1	0	1	2	3	-1	0	1

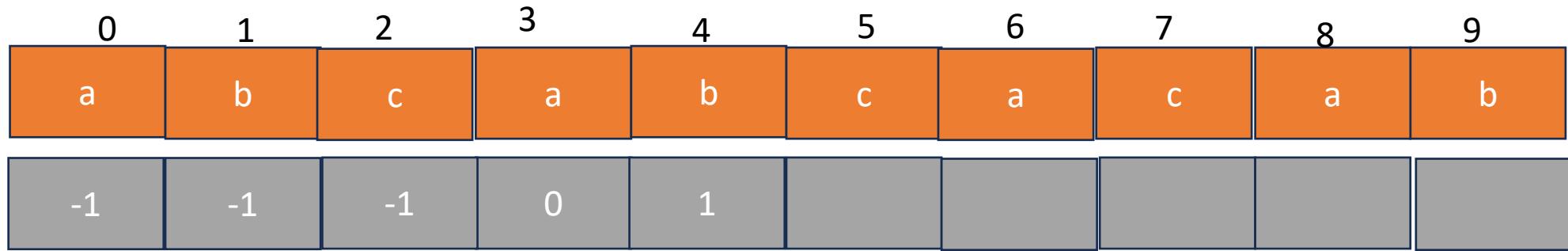


```

for (j=1; j<n ; j++)
{
    i=failure[j-1];           // i should have the previous value i= 0 ,j=4
    while((pat[j]!= pat[i+1]) && i>=0)
        i=failure[i];
    if(pat[j]==pat[i+1])
        failure[j]= i+1;
    else
        failure[j]=-1;
}

```

<i>j</i>	0	1	2	3	4	5	6	7	8	9
<i>pat</i>	a	b	c	a	b	c	a	c	a	b
<i>f</i>	-1	-1	-1	0	1	2	3	-1	0	1

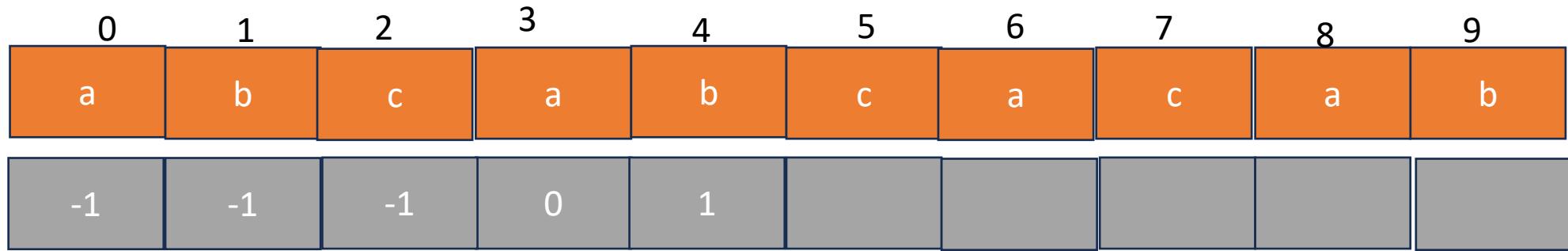


```

for (j=1; j<n ; j++)
{
    i=failure[j-1];           // i should have the previous value i= 0 ,j=4
    while((pat[j]!= pat[i+1]) && i>=0)
        i=failure[i];
    if(pat[j]==pat[i+1])
        failure[j]= i+1;
    else
        failure[j]= -1;
}

```

<i>j</i>	0	1	2	3	4	5	6	7	8	9
<i>pat</i>	a	b	c	a	b	c	a	c	a	b
<i>f</i>	-1	-1	-1	0	1	2	3	-1	0	1

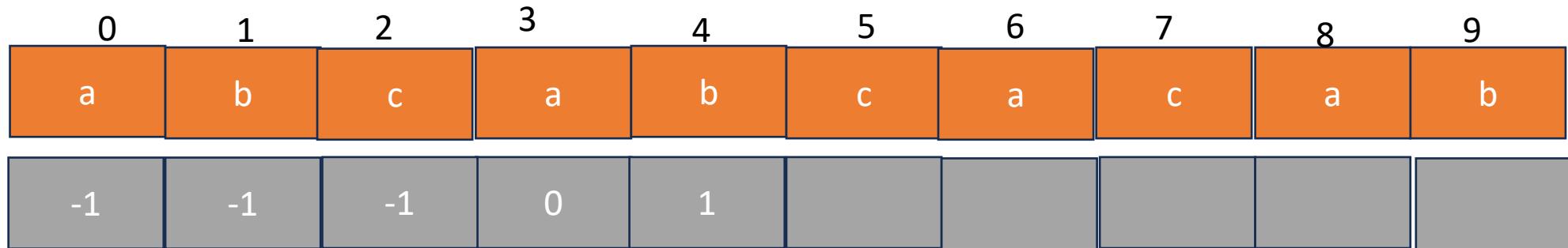


```

for (j=1; j<n ; j++)
{
    i=failure[j-1];          // i should have the previous value i= 1 ,j=5
    while((pat[j]!= pat[i+1]) && i>=0)
        i=failure[i];
    if(pat[j]==pat[i+1])
        failure[j]= i+1;
    else
        failure[j]= -1;
}

```

<i>j</i>	0	1	2	3	4	5	6	7	8	9
<i>pat</i>	a	b	c	a	b	c	a	c	a	b
<i>f</i>	-1	-1	-1	0	1	2	3	-1	0	1

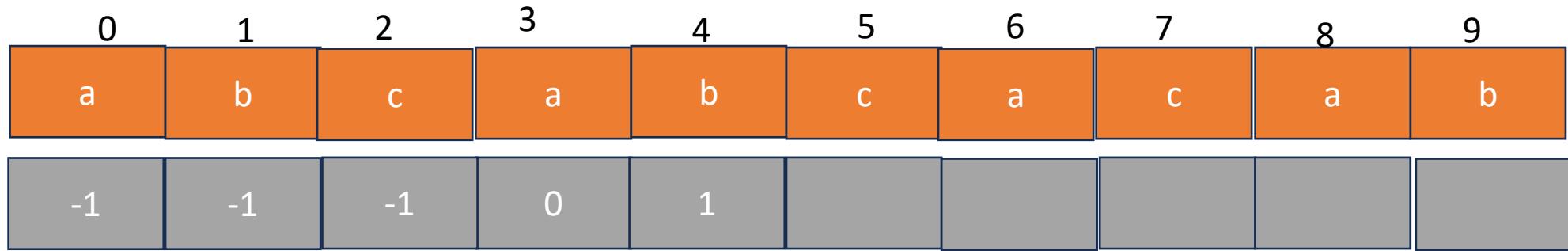


```

for (j=1; j<n ; j++)
{
    i=failure[j-1];          // i should have the previous value i= 1 ,j=5
    while((pat[j]!= pat[i+1]) && i>=0)           pat[5] !=pat[2]
        i=failure[i];
    if(pat[j]==pat[i+1])
        failure[j]= i+1;
    else
        failure[j]= -1;
}

```

<i>j</i>	0	1	2	3	4	5	6	7	8	9
<i>pat</i>	a	b	c	a	b	c	a	c	a	b
<i>f</i>	-1	-1	-1	0	1	2	3	-1	0	1

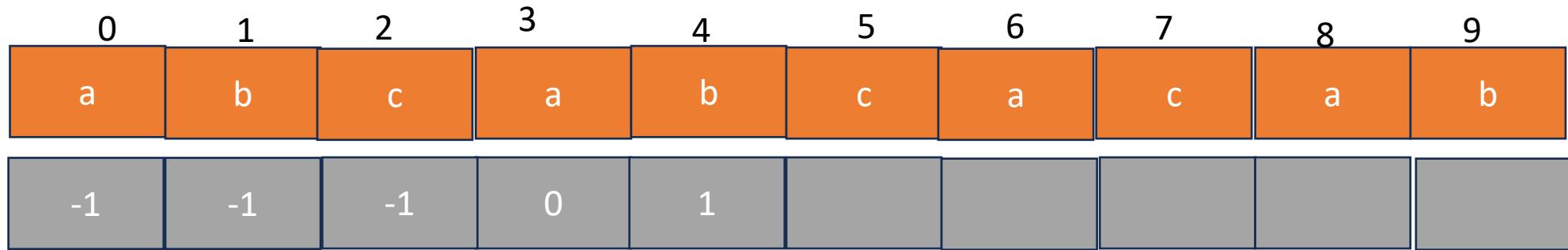


```

for (j=1; j<n ; j++)
{
    i=failure[j-1];           // i should have the previous value i= 1 ,j=5
    while((pat[j]!= pat[i+1]) && i>=0)
        i=failure[i];
    if(pat[j]==pat[i+1])      pat[5]==pat[2]
        failure[j]= i+1;
    else
        failure[j]= -1;
}

```

<i>j</i>	0	1	2	3	4	5	6	7	8	9
<i>pat</i>	a	b	c	a	b	c	a	c	a	b
<i>f</i>	-1	-1	-1	0	1	2	3	-1	0	1

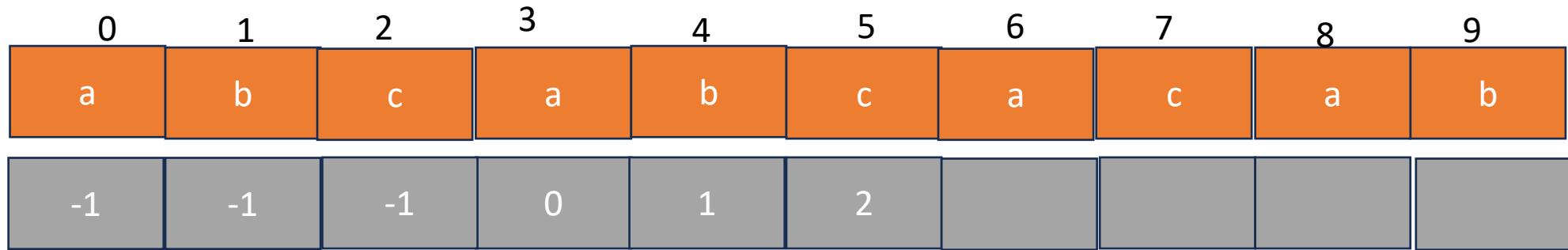


```

for (j=1; j<n ; j++)
{
    i=failure[j-1];           // i should have the previous value i= 1 ,j=5
    while((pat[j]!= pat[i+1]) && i>=0)
        i=failure[i];
    if(pat[j]==pat[i+1])
        failure[j]= i+1;
    else
        failure[j]=-1;
}

```

<i>j</i>	0	1	2	3	4	5	6	7	8	9
<i>pat</i>	a	b	c	a	b	c	a	c	a	b
<i>f</i>	-1	-1	-1	0	1	2	3	-1	0	1

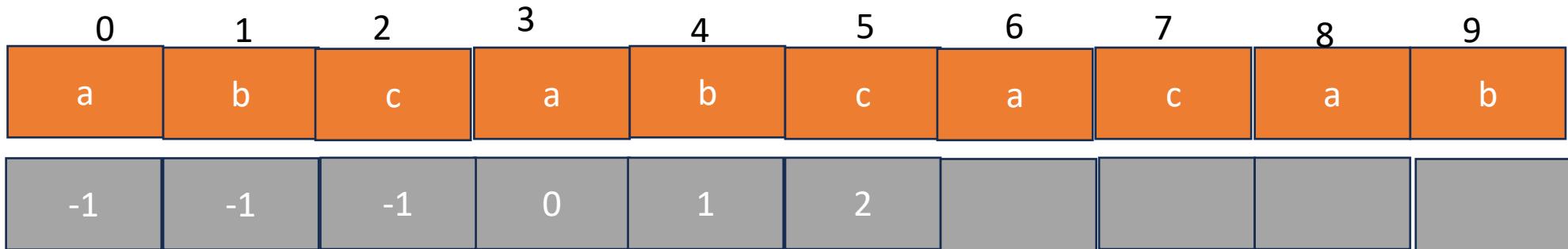


```

for (j=1; j<n ; j++)
{
    i=failure[j-1];           // i should have the previous value i= 1 ,j=5
    while((pat[j]!= pat[i+1]) && i>=0)
        i=failure[i];
    if(pat[j]==pat[i+1])
        failure[j]= i+1;
    else
        failure[j]= -1;
}

```

<i>j</i>	0	1	2	3	4	5	6	7	8	9
<i>pat</i>	a	b	c	a	b	c	a	c	a	b
<i>f</i>	-1	-1	-1	0	1	2	3	-1	0	1

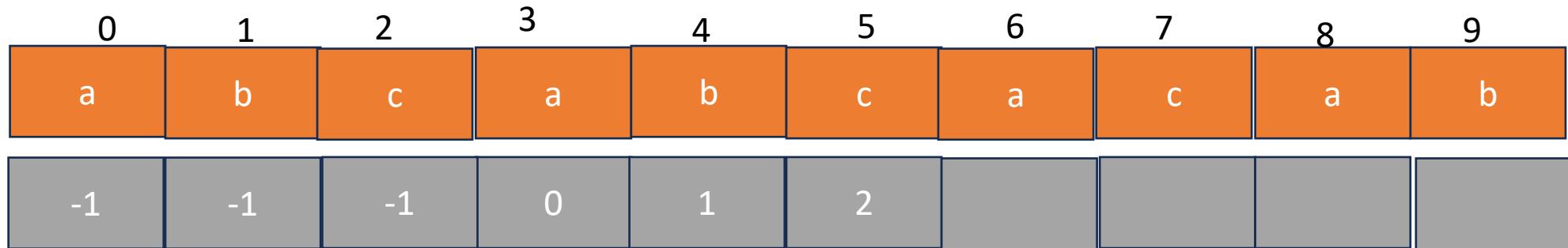


```

for (j=1; j<n ; j++)
{
    i=failure[j-1];          // i should have the previous value i= 2 ,j=6
    while((pat[j]!= pat[i+1]) && i>=0)
        i=failure[i];
    if(pat[j]==pat[i+1])
        failure[j]= i+1;
    else
        failure[j]= -1;
}

```

<i>j</i>	0	1	2	3	4	5	6	7	8	9
<i>pat</i>	a	b	c	a	b	c	a	c	a	b
<i>f</i>	-1	-1	-1	0	1	2	3	-1	0	1

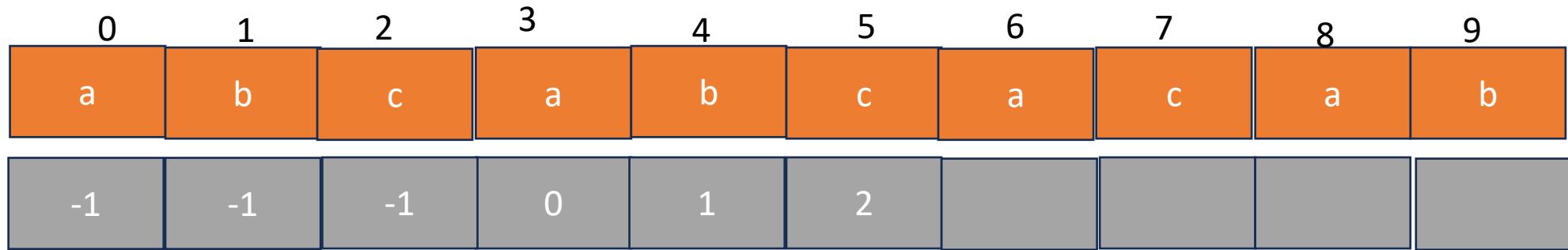


```

for (j=1; j<n ; j++)
{
    i=failure[j-1];           // i should have the previous value i= 2 ,j=6
    while((pat[j]!= pat[i+1]) && i>=0)      pat[6]!=pat[3]
        i=failure[i];
    if(pat[j]==pat[i+1])
        failure[j]= i+1;
    else
        failure[j]= -1;
}

```

<i>j</i>	0	1	2	3	4	5	6	7	8	9
<i>pat</i>	a	b	c	a	b	c	a	c	a	b
<i>f</i>	-1	-1	-1	0	1	2	3	-1	0	1

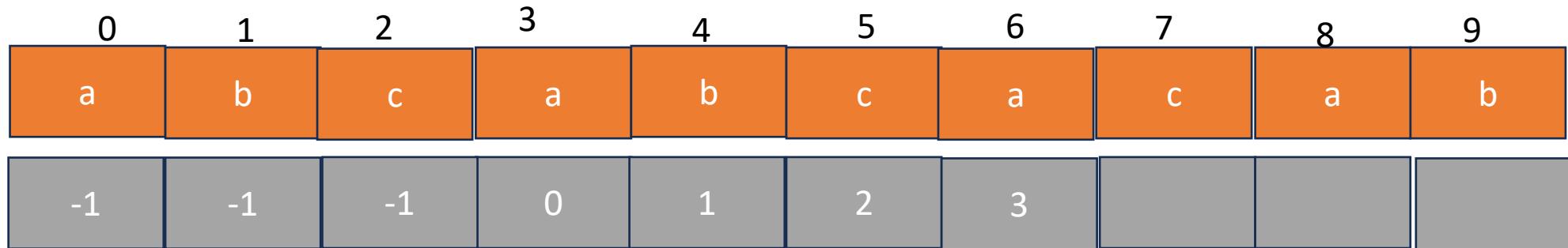


```

for (j=1; j<n ; j++)
{
    i=failure[j-1];           // i should have the previous value i= 2 ,j=6
    while((pat[j]!= pat[i+1]) && i>=0)
        i=failure[i];
    if(pat[j]==pat[i+1])      pat[6]==pat[3]
        failure[j]= i+1;
    else
        failure[j]= -1;
}

```

<i>j</i>	0	1	2	3	4	5	6	7	8	9
<i>pat</i>	a	b	c	a	b	c	a	c	a	b
<i>f</i>	-1	-1	-1	0	1	2	3	-1	0	1

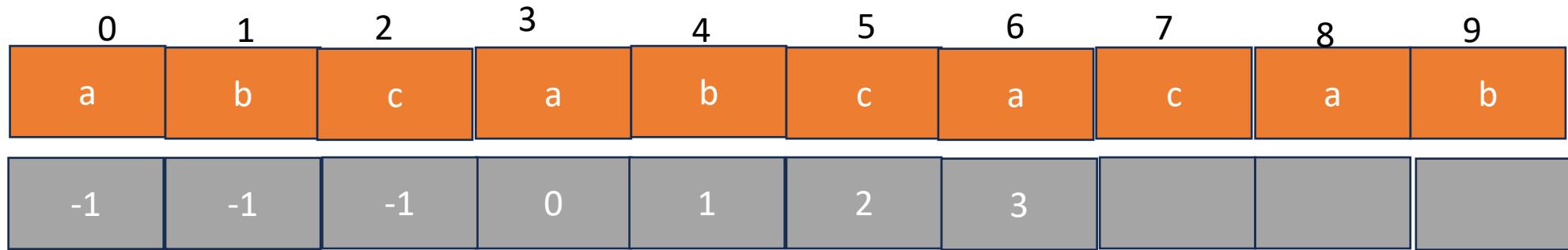


```

for (j=1; j<n ; j++)
{
    i=failure[j-1];           // i should have the previous value i= 2 ,j=6
    while((pat[j]!= pat[i+1]) && i>=0)
        i=failure[i];
    if(pat[j]==pat[i+1])
        failure[j]= i+1;
    else
        failure[j]=-1;
}

```

<i>j</i>	0	1	2	3	4	5	6	7	8	9
<i>pat</i>	a	b	c	a	b	c	a	c	a	b
<i>f</i>	-1	-1	-1	0	1	2	3	-1	0	1

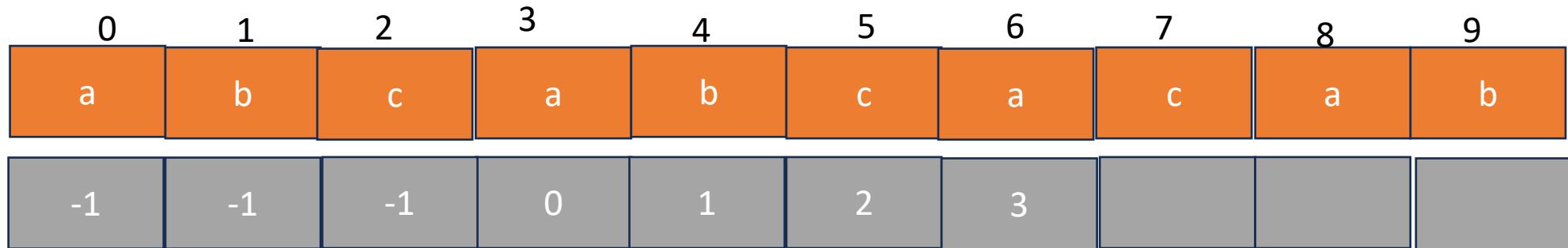


```

for (j=1; j<n ; j++)
{
    i=failure[j-1];           // i should have the previous value i= 2 ,j=6
    while((pat[j]!= pat[i+1]) && i>=0)
        i=failure[i];
    if(pat[j]==pat[i+1])
        failure[j]= i+1;
    else
        failure[j]= -1;
}

```

<i>j</i>	0	1	2	3	4	5	6	7	8	9
<i>pat</i>	a	b	c	a	b	c	a	c	a	b
<i>f</i>	-1	-1	-1	0	1	2	3	-1	0	1

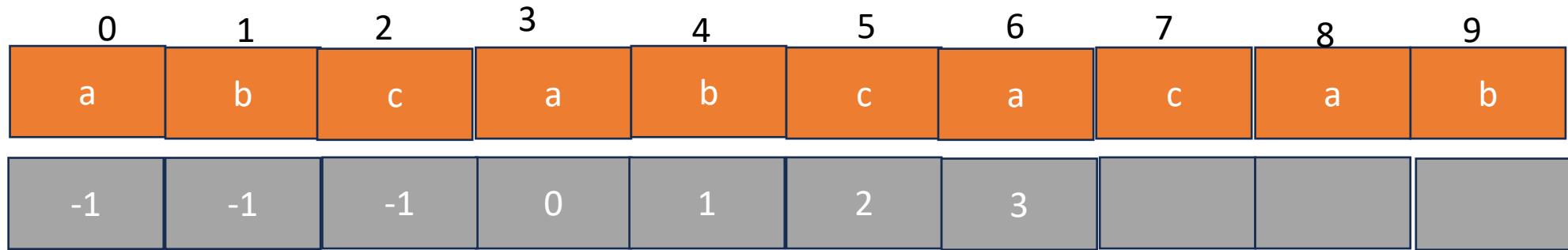


```

for (j=1; j<n ; j++)
{
    i=failure[j-1];          // i should have the previous value i= 3 ,j=7
    while((pat[j]!= pat[i+1]) && i>=0)
        i=failure[i];
    if(pat[j]==pat[i+1])
        failure[j]= i+1;
    else
        failure[j]= -1;
}

```

<i>j</i>	0	1	2	3	4	5	6	7	8	9
<i>pat</i>	a	b	c	a	b	c	a	c	a	b
<i>f</i>	-1	-1	-1	0	1	2	3	-1	0	1

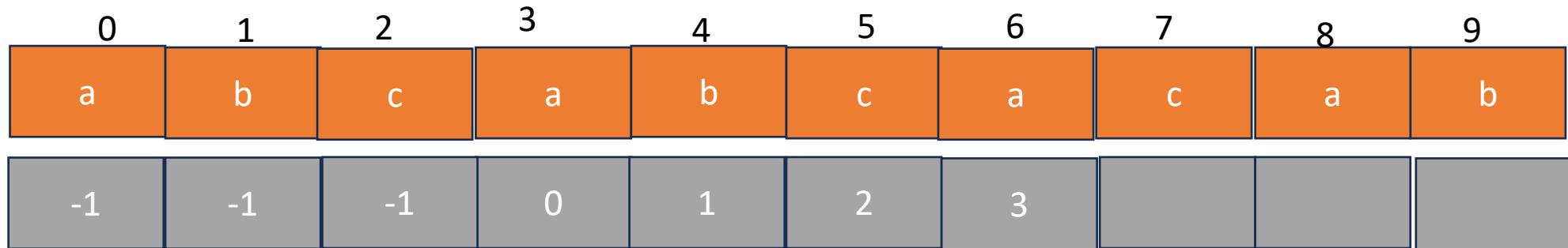


```

for (j=1; j<n ; j++)
{
    i=failure[j-1];           // i should have the previous value i= 3 ,j=7
    while((pat[j]!= pat[i+1]) && i>=0)           pat[7] != pat[4]
        i=failure[i];
    if(pat[j]==pat[i+1])
        failure[j]= i+1;
    else
        failure[j]= -1;
}

```

<i>j</i>	0	1	2	3	4	5	6	7	8	9
<i>pat</i>	a	b	c	a	b	c	a	c	a	b
<i>f</i>	-1	-1	-1	0	1	2	3	-1	0	1

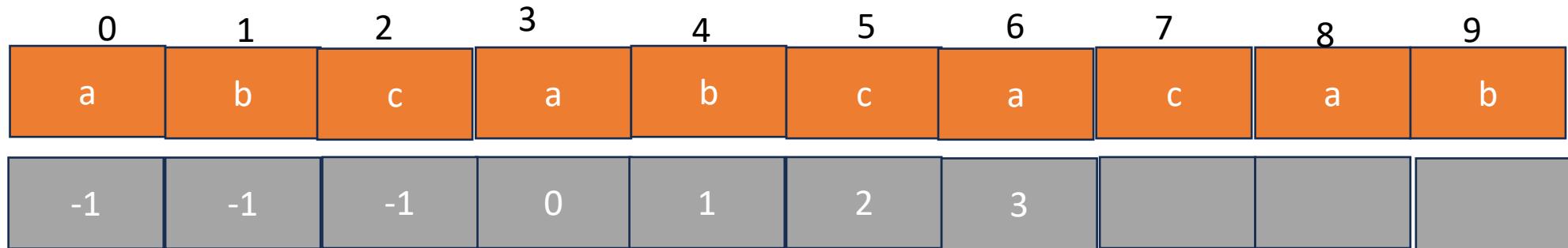


```

for (j=1; j<n ; j++)
{
    i=failure[j-1];           // i should have the previous value i= 3 ,j=7
    while((pat[j]!= pat[i+1]) && i>=0)           pat[7] != pat[4]
        i=failure[i];           i=0
    if(pat[j]==pat[i+1])
        failure[j]= i+1;
    else
        failure[j]= -1;
}

```

<i>j</i>	0	1	2	3	4	5	6	7	8	9
<i>pat</i>	a	b	c	a	b	c	a	c	a	b
<i>f</i>	-1	-1	-1	0	1	2	3	-1	0	1

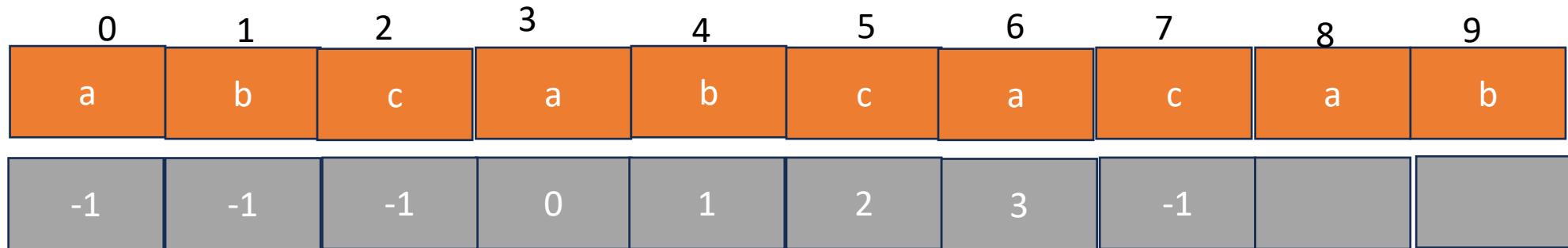


```

for (j=1; j<n ; j++)
{
    i=failure[j-1];                                // i should have the previous value i= 3 ,j=7
    while((pat[j]!= pat[i+1]) && i>=0)
        i=failure[i];                               i=0
    if(pat[j]==pat[i+1])                           pat[7]==pat[1]
        failure[j]= i+1;
    else
        failure[j]= -1;
}

```

<i>j</i>	0	1	2	3	4	5	6	7	8	9
<i>pat</i>	a	b	c	a	b	c	a	c	a	b
<i>f</i>	-1	-1	-1	0	1	2	3	-1	0	1

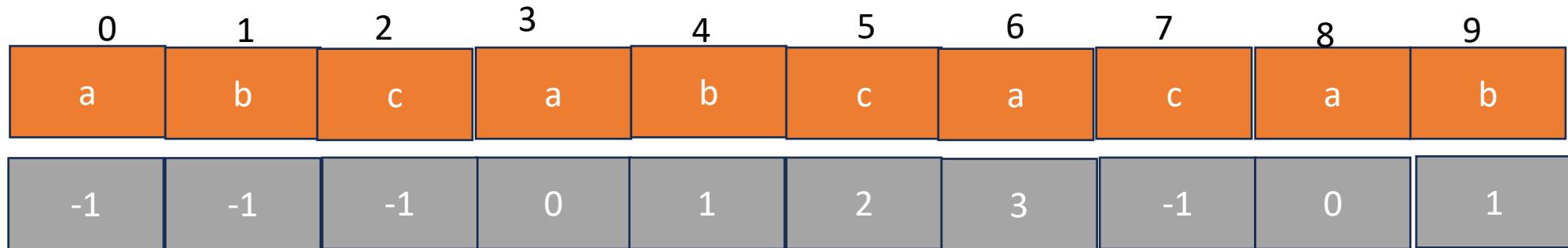


```

for (j=1; j<n ; j++)
{
    i=failure[j-1];                                // i should have the previous value i= 3 ,j=7
    while((pat[j]!= pat[i+1]) && i>=0)          // to reinitialize when pattern does not match
        i=failure[i];
    if(pat[j]==pat[i+1])
        failure[j]= i+1;
    else
        failure[j]= -1;
}

```

<i>j</i>	0	1	2	3	4	5	6	7	8	9
<i>pat</i>	a	b	c	a	b	c	a	c	a	b
<i>f</i>	-1	-1	-1	0	1	2	3	-1	0	1



```
for (j=1; j<n ; j++)
```

```
{
```

```
    i=failure[j-1];
```

// i should have the previous value i= 3 ,j=7

```
    while((pat[j]!= pat[i+1]) && i>=0)
```

// to reinitialize when pattern does not match

```
        i=failure[i];
```

```
    if(pat[j]==pat[i+1])
```

```
        failure[j]= i+1;
```

```
    else
```

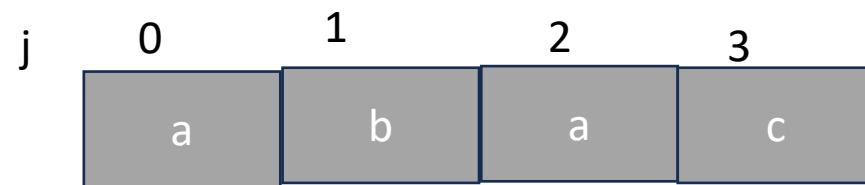
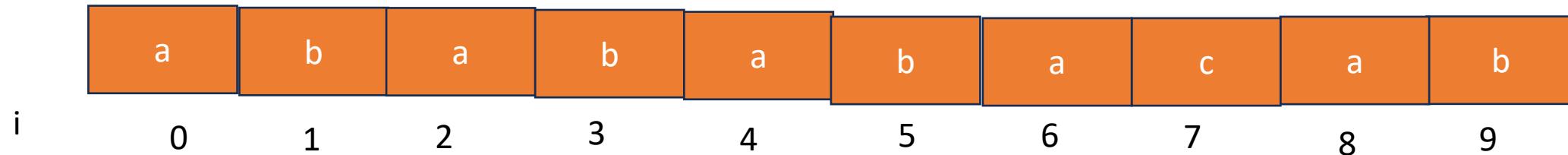
```
        failure[j]= -1;
```

```
}
```

KMP – Algorithm for pattern Matching

```
void fail(char *pat)
{ /* compute the pattern's failure function */
    int n = strlen(pat);
    failure[0] = -1;
    for (j=1; j < n; j++) {
        i = failure[j-1];
        while ((pat[j] != pat[i+1]) && (i >= 0))
            i = failure[i];
        if (pat[j] == pat[i+1])
            failure[j] = i+1;
        else failure[j] = -1;
    }
}
```

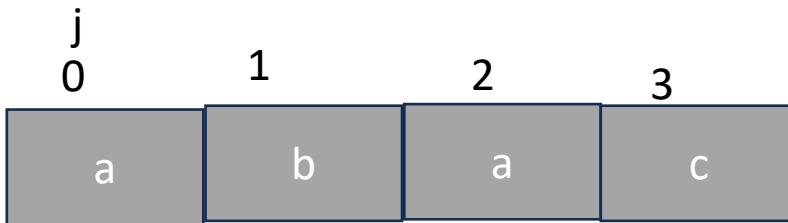
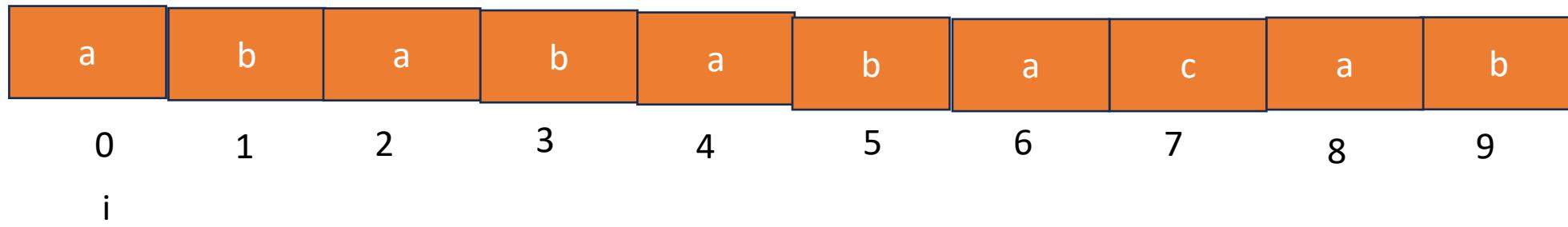
Program 2.15: Computing the failure function



```

While(i < lens && j < lenp)
    if( string[i] == pat [j])
        i++; j++;
    else if (j==0)
        i++;
    else
        j= failure[j-1]+1
    
```

a	b	a	c
-1	-1	0	-1



While(*i* < *lens* && *j* < *lenp*)

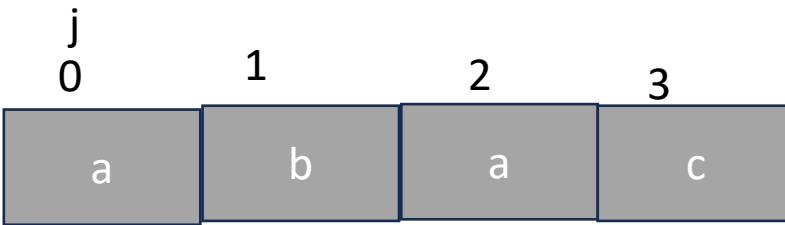
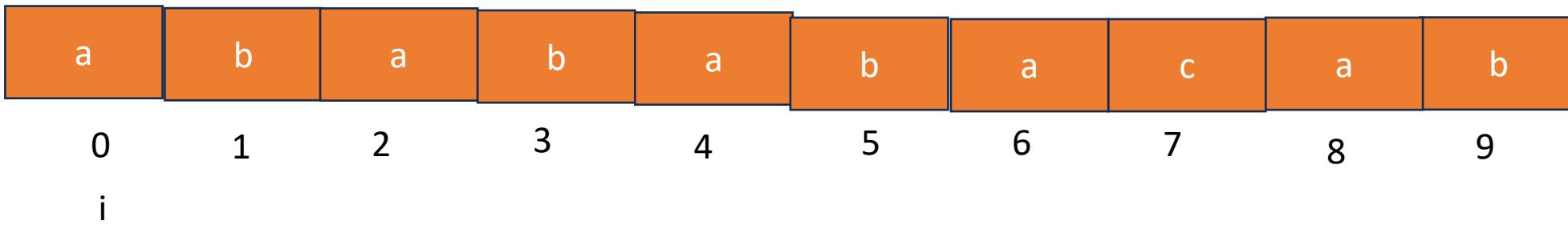
```
if( string[i] == pat [j])
    i++; j++;
```

```
else if (j==0)
    i++;
```

```
else
```

```
    j= failure[j-1]+1
```

a	b	a	c
-1	-1	0	-1

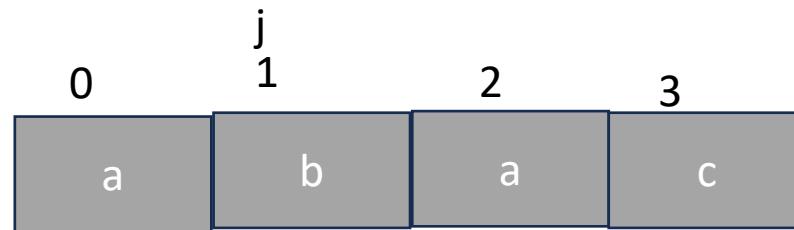
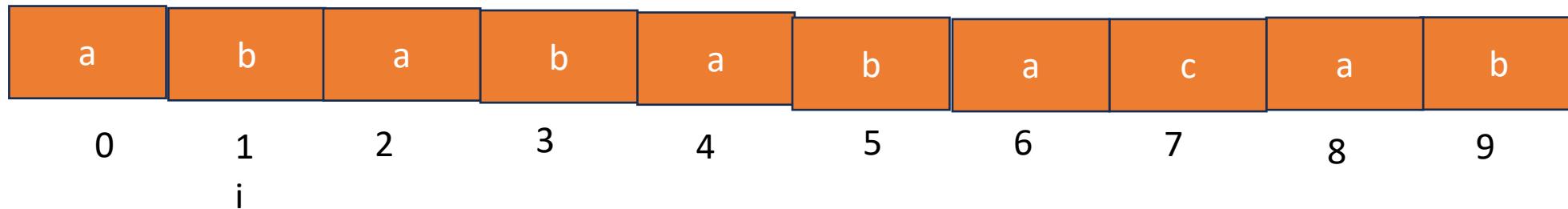


```

While(i < lens && j < lenp)
    if( string[i] == pat [j])
        i++; j++;
    else if (j==0)
        i++;
    else
        j= failure[j-1]+1

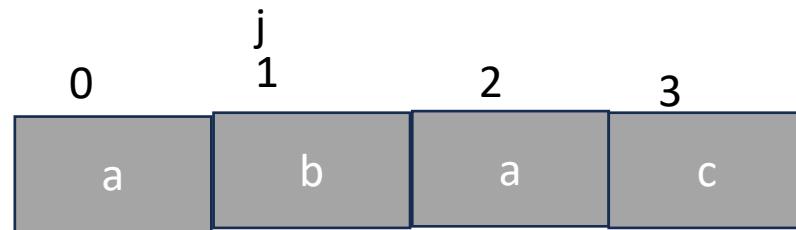
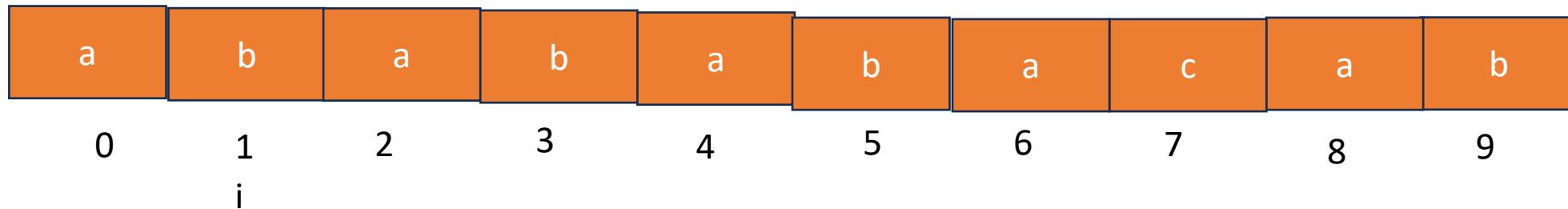
```

a	b	a	c
-1	-1	0	-1



```
While(i < lens && j < lenp)
    if( string[i] == pat [j])
        i++; j++;
    else if (j==0)
        i++;
    else
        j= failure[j-1]+1
```

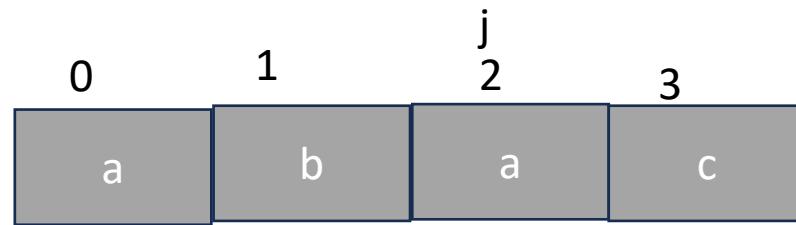
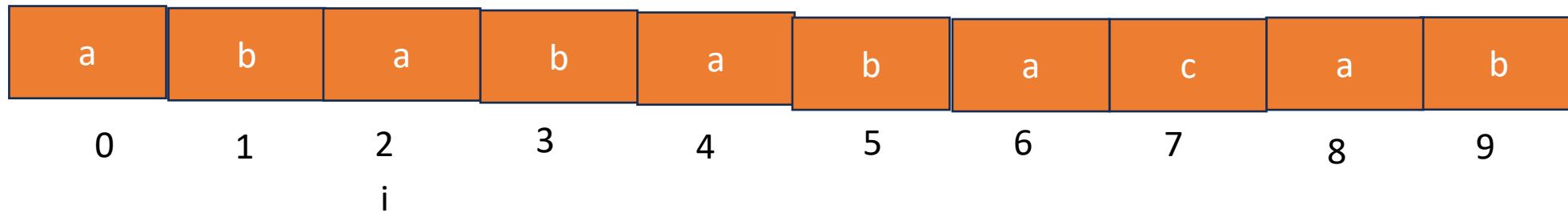
a	b	a	c
-1	-1	0	-1



```

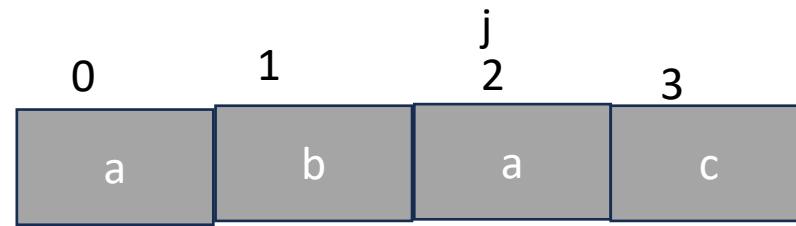
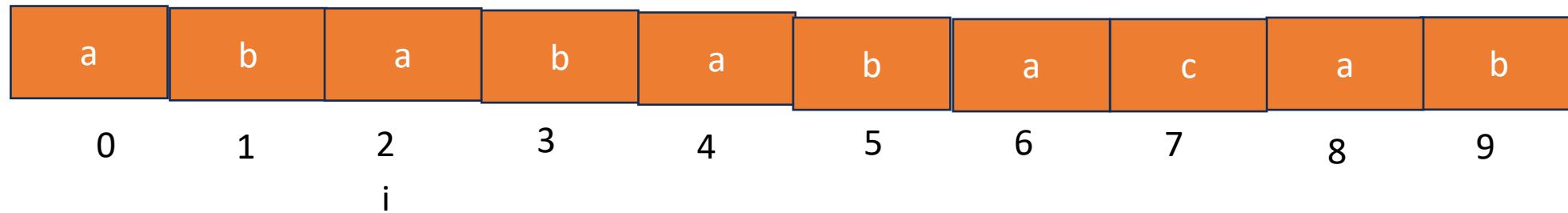
While(i < lens && j < lenp)
    if( string[i] == pat [j])
        i++; j++;
    else if (j==0)
        i++;
    else
        j= failure[j-1]+1

```



```

While(i < lens && j < lenp)
    if( string[i] == pat [j])
        i++; j++;
    else if (j==0)
        i++;
    else
        j= failure[j-1]+1
    
```



While(*i* < *lens* && *j* < *lenp*)

 if(*string*[*i*] == *pat* [*j*])

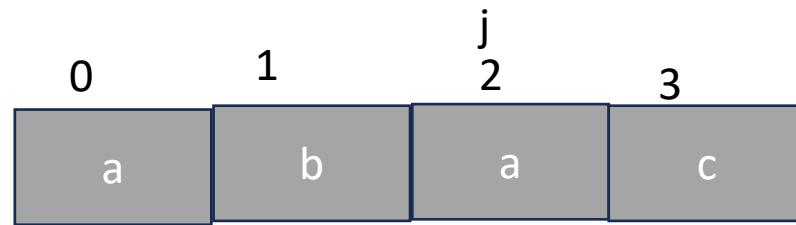
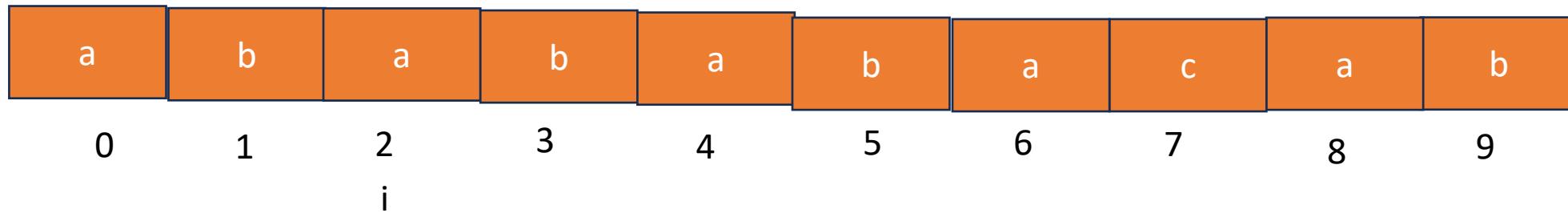
i++; *j*++;

 else if (*j*==0)

i++;

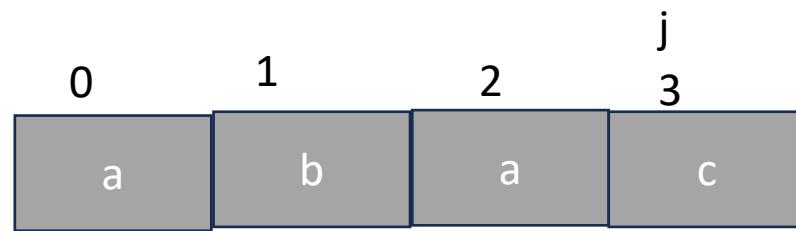
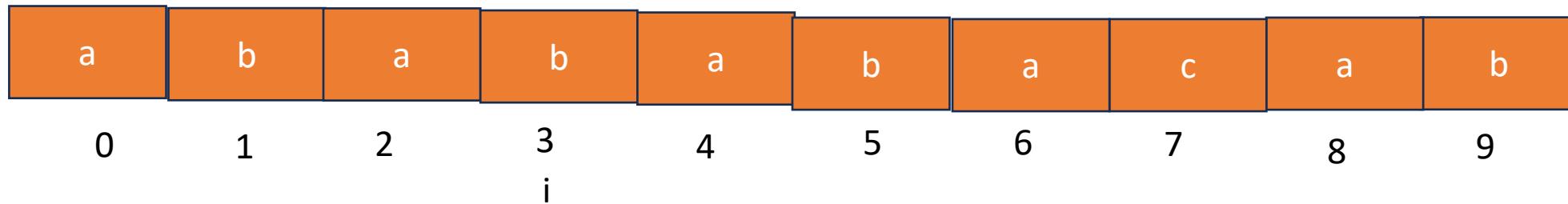
 else

j= *failure*[*j*-1]+1



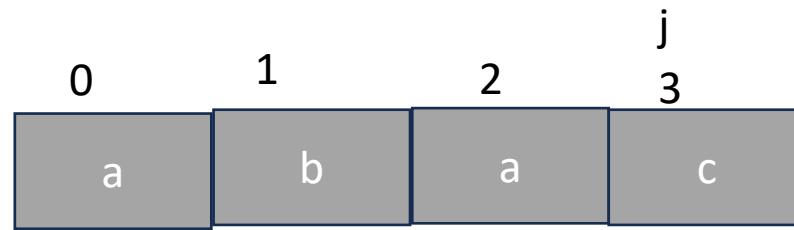
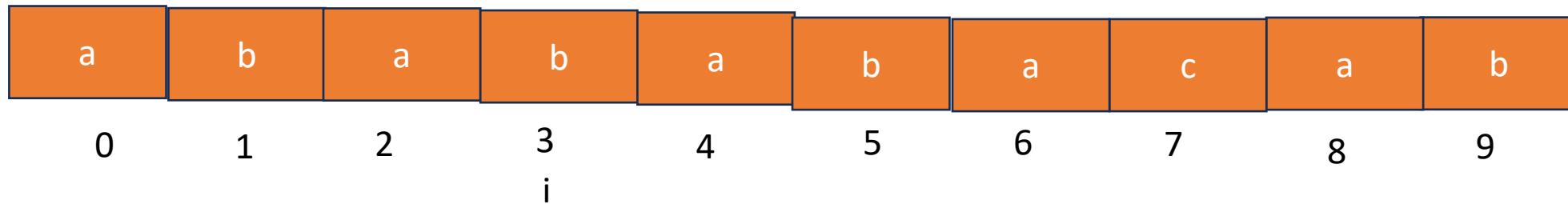
```

While(i < lens && j < lenp)
    if( string[i] == pat [j])
        i++; j++;
    else if (j==0)
        i++;
    else
        j= failure[j-1]+1
    
```



```

While(i < lens && j < lenp)
    if( string[i] == pat [j])
        i++; j++;
    else if (j==0)
        i++;
    else
        j= failure[j-1]+1
    
```



While(*i* < *lens* && *j* < *lenp*)

 if(*string*[*i*] == *pat* [*j*])

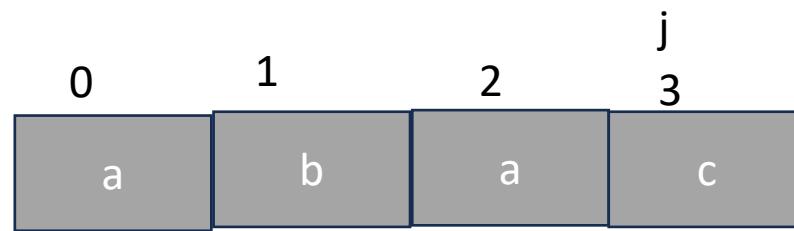
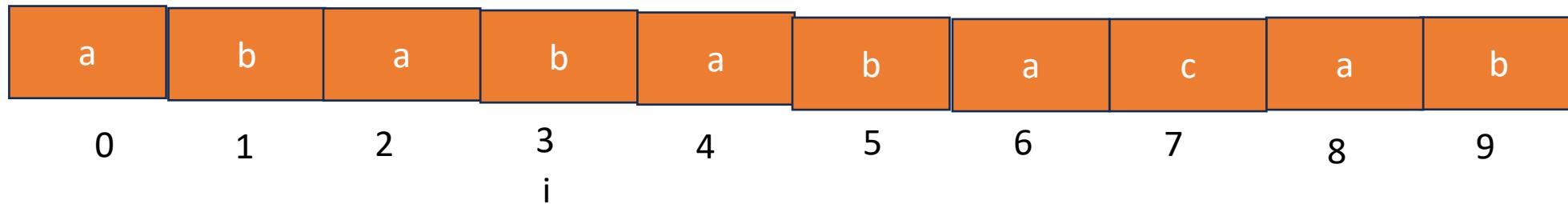
i++; *j*++;

 else if (*j*==0)

i++;

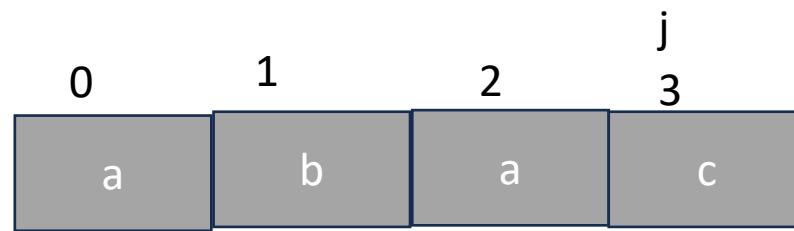
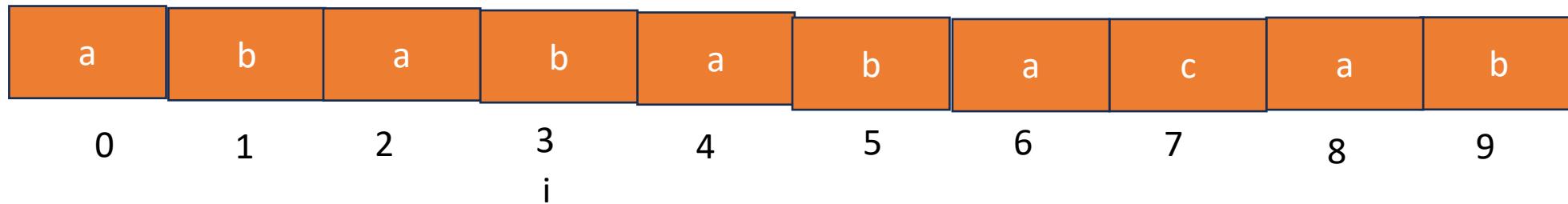
 else

j= *failure*[*j*-1]+1



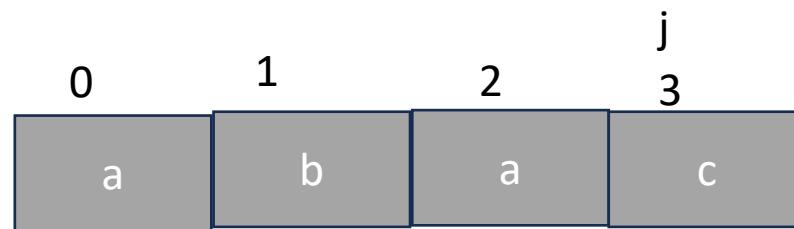
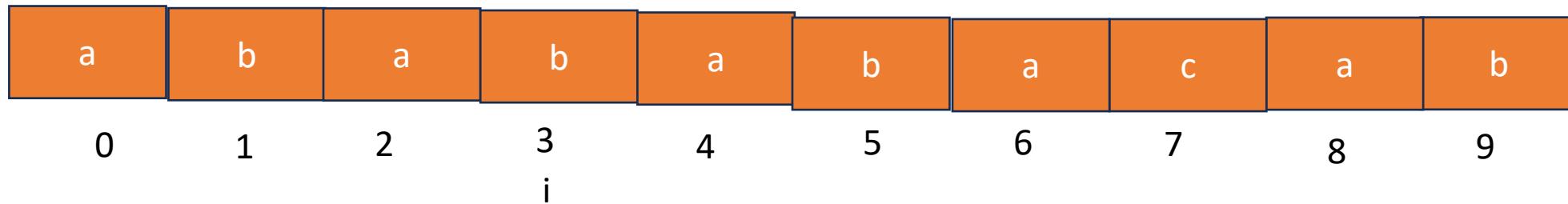
```

While(i < lens && j < lenp)
    if( string[i] == pat [j])
        i++; j++;
    else if (j==0)
        i++;
    else
        j= failure[j-1]+1
    
```



```

While(i < lens && j < lenp)
    if( string[i] == pat [j])
        i++; j++;
else if (j==0)
    i++;
else
    j= failure[j-1]+1
  
```



```

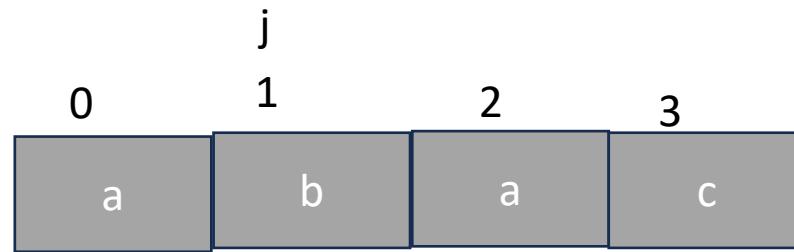
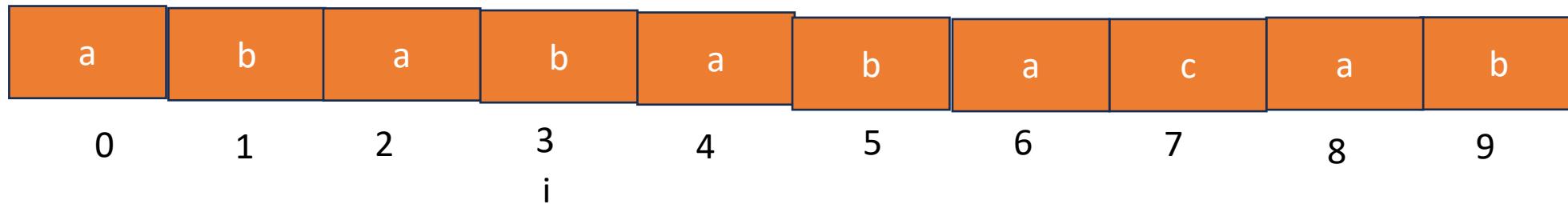
While(i < lens && j < lenp)
    if( string[i] == pat [j])
        i++; j++;
    else if (j==0)
        i++;
    else

```

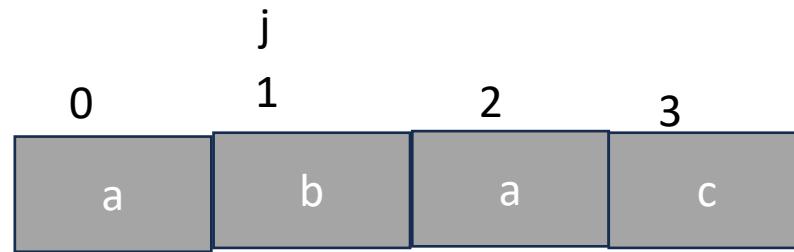
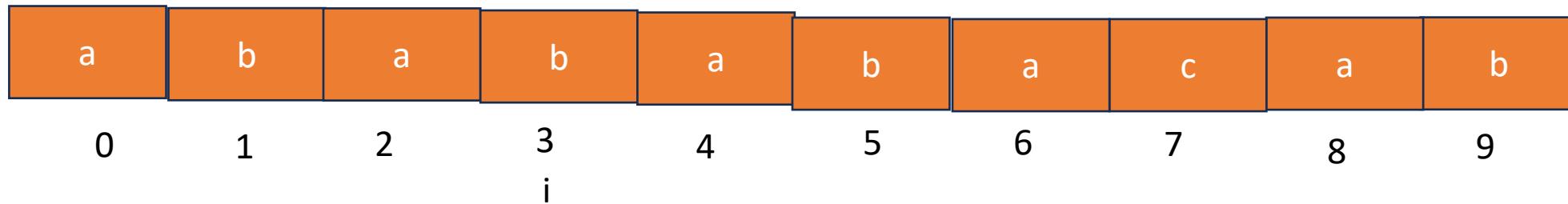
j= failure[j-1]+1

a	b	a	c
-1	-1	0	-1

j= failure[2] +1 = 0+1 =1



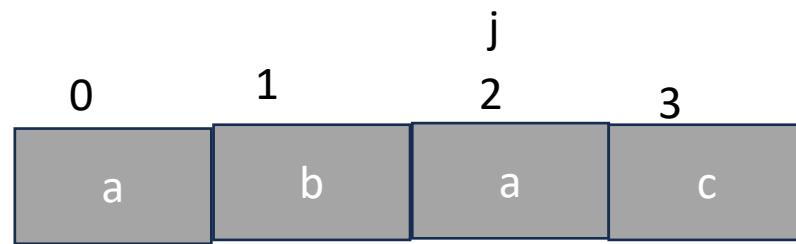
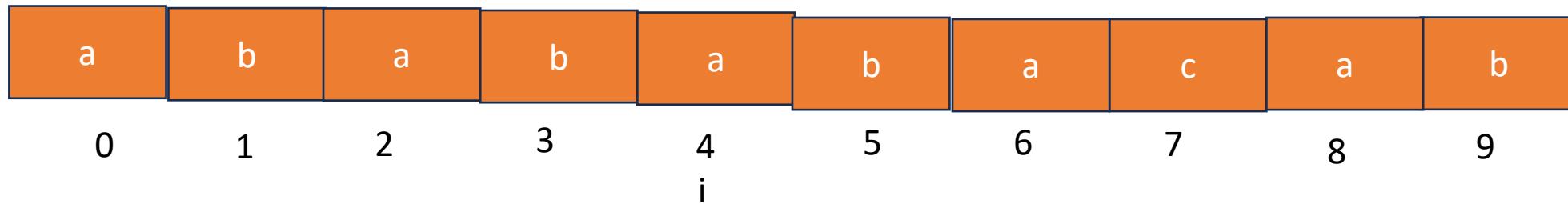
```
While(i < lens && j < lenp)
    if( string[i] == pat [j])
        i++; j++;
    else if (j==0)
        i++;
    else
        j= failure[j-1]+1
```



```

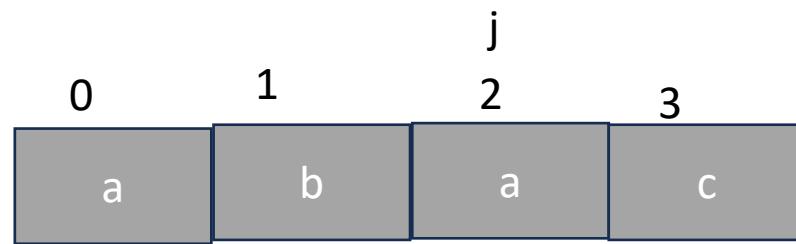
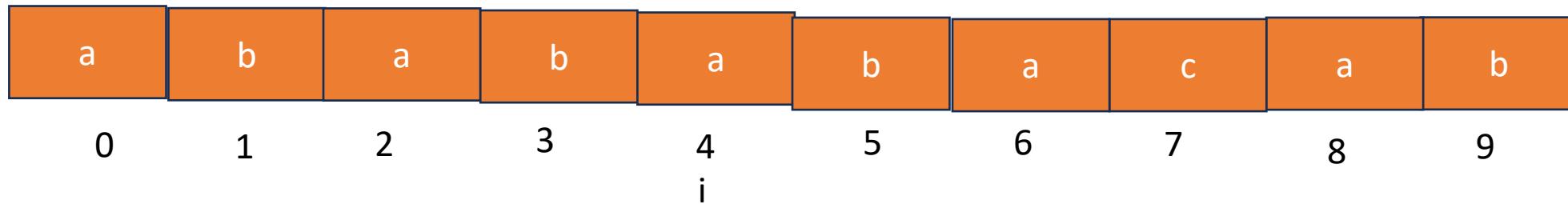
While(i < lens && j < lenp)
    if( string[i] == pat [j])
        i++; j++;
    else if (j==0)
        i++;
    else
        j= failure[j-1]+1

```



```

While(i < lens && j < lenp)
    if( string[i] == pat [j])
        i++; j++;
    else if (j==0)
        i++;
    else
        j= failure[j-1]+1
    
```



While(*i* < *lens* && *j* < *lenp*)

 if(*string*[*i*] == *pat* [*j*])

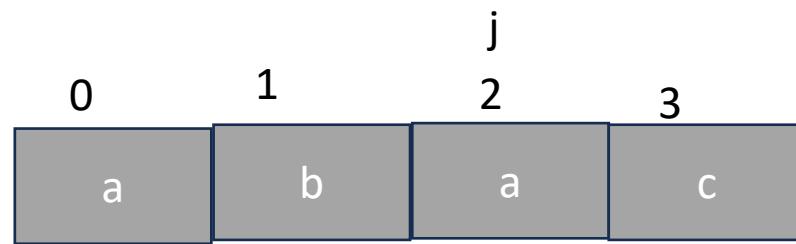
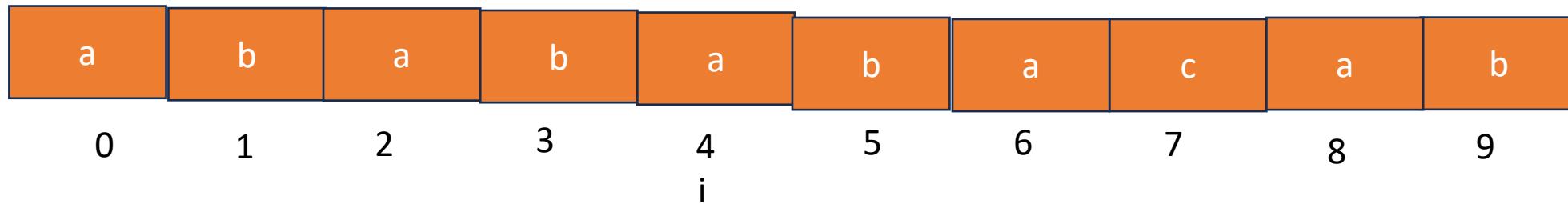
i++; *j*++;

 else if (*j*==0)

i++;

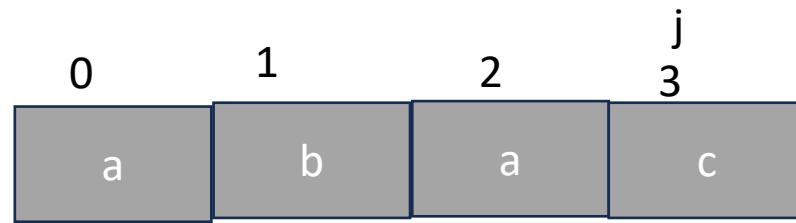
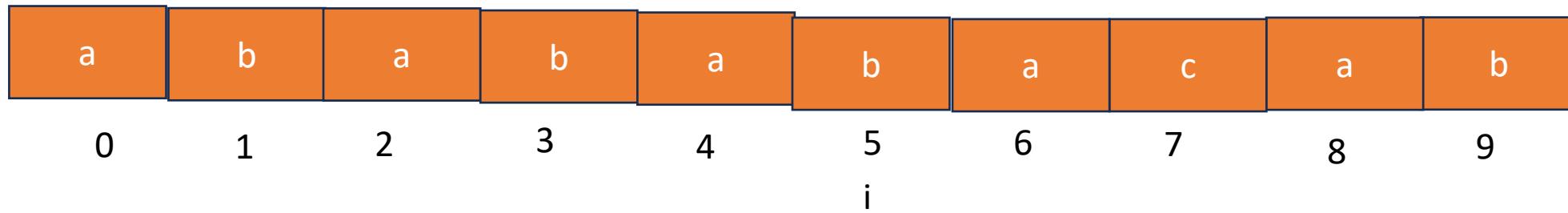
 else

j= *failure*[*j*-1]+1



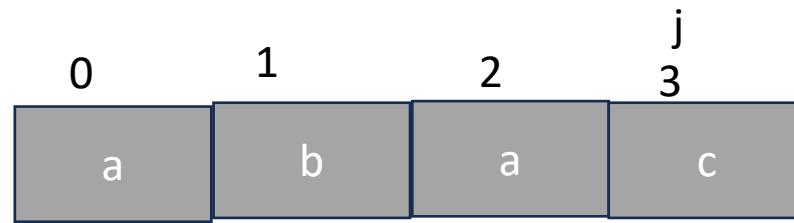
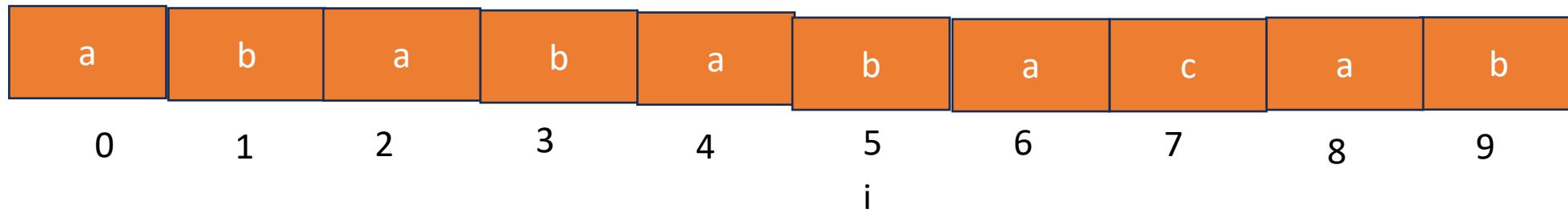
```

While(i < lens && j < lenp)
    if( string[i] == pat [j])
        i++; j++;
    else if (j==0)
        i++;
    else
        j= failure[j-1]+1
    
```

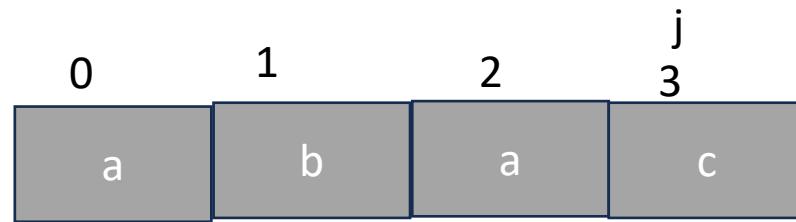
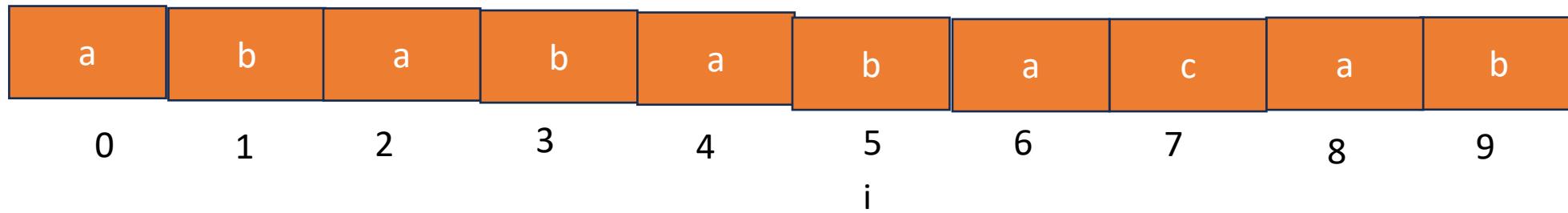


```

While(i < lens && j < lenp)
    if( string[i] == pat [j])
        i++; j++;
    else if (j==0)
        i++;
    else
        j= failure[j-1]+1
    
```

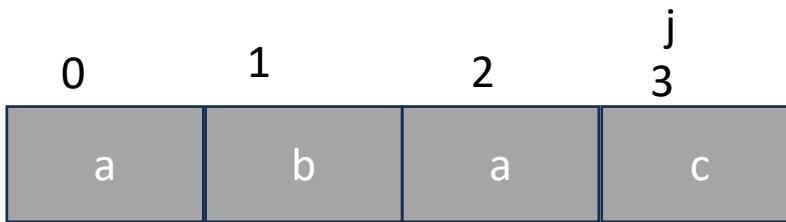
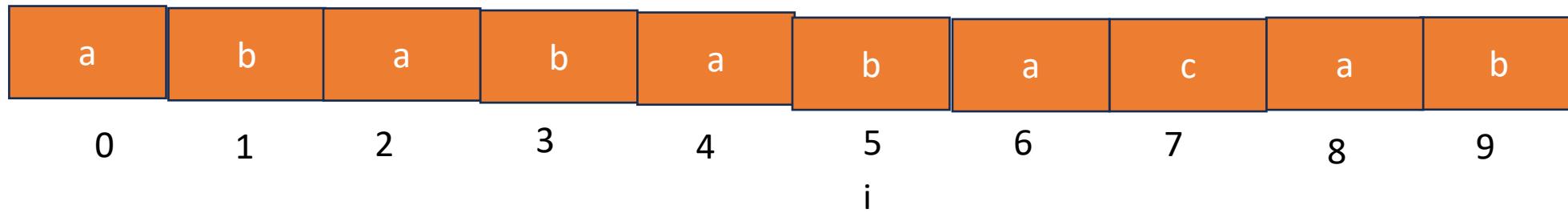


```
While(i < lens && j < lenp)
    if( string[i] == pat [j])
        i++; j++;
    else if (j==0)
        i++;
    else
        j= failure[j-1]+1
```



```

While(i < lens && j < lenp)
    if( string[i] == pat [j])
        i++; j++;
    else if (j==0)
        i++;
    else
        j= failure[j-1]+1
    
```

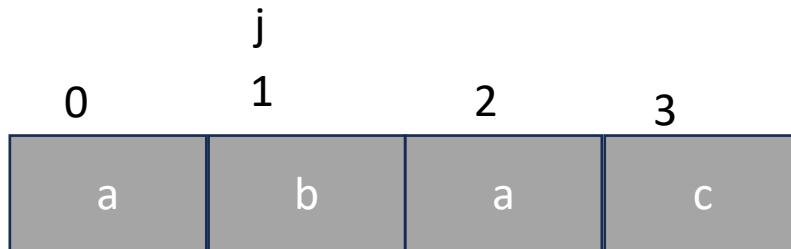
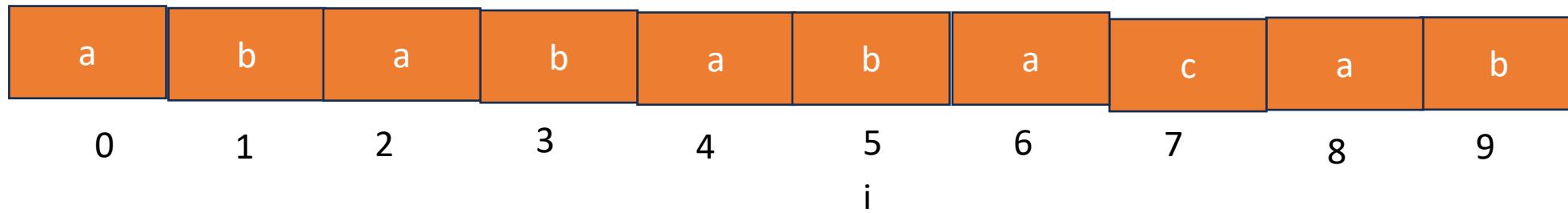


```

While(i < lens && j < lenp)
    if( string[i] == pat [j])
        i++; j++;
    else if (j==0)
        i++;
    else
        j= failure[j-1]+1
    
```

a	b	a	c
-1	-1	0	-1

$$j = \text{failure}[2] + 1 = 0 + 1 = 1$$

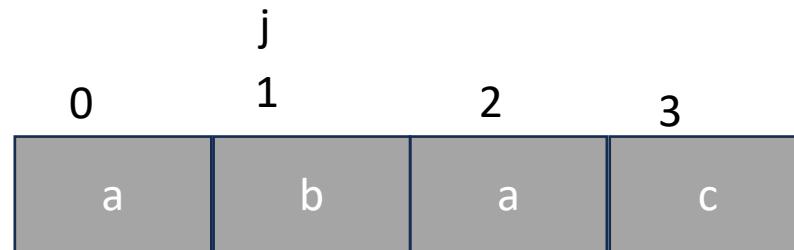
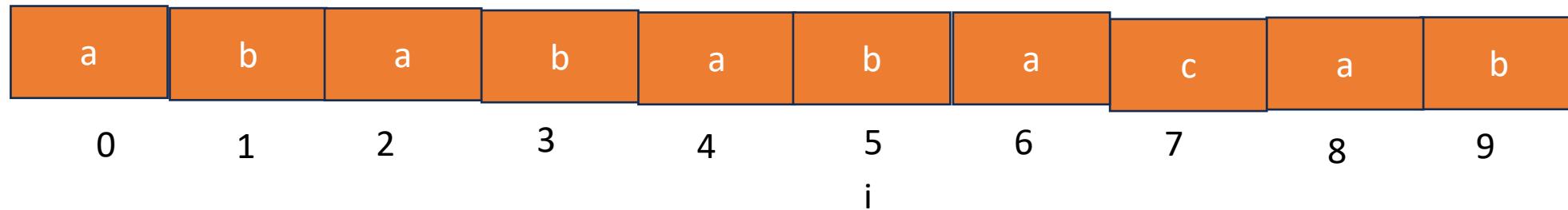


```

While(i < lens && j < lenp)
    if( string[i] == pat [j])
        i++; j++;
    else if (j==0)
        i++;
    else
        j= failure[j-1]+1
    
```

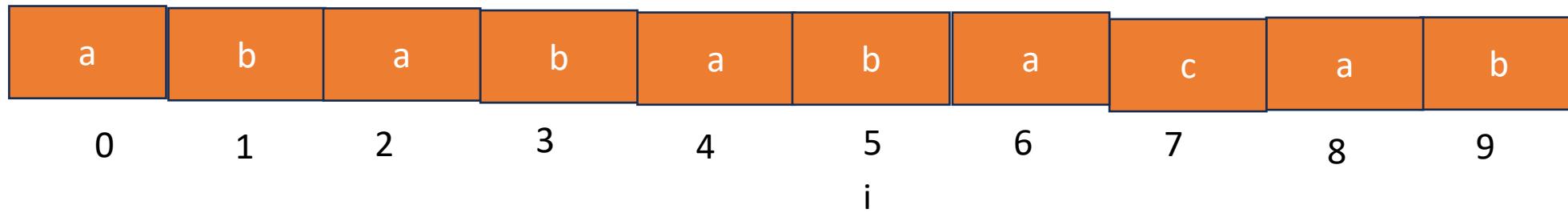
a	b	a	c
-1	-1	0	-1

j= failure[2]+1=0+1 =1



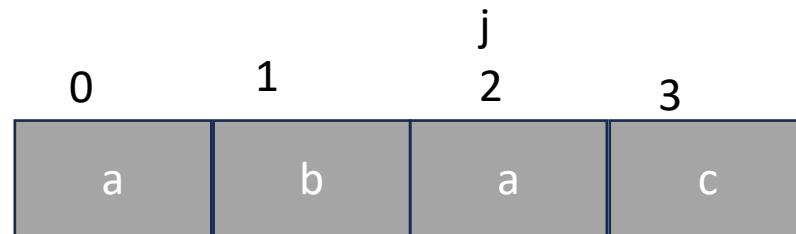
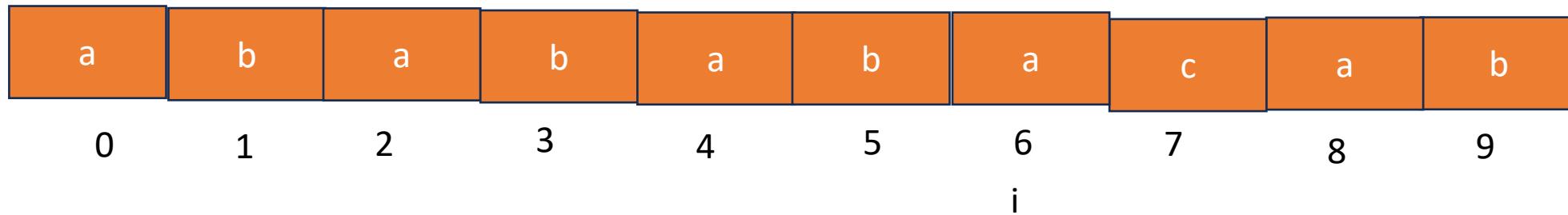
```

While(i < lens && j < lenp)
    if( string[i] == pat [j])
        i++; j++;
    else if (j==0)
        i++;
    else
        j= failure[j-1]+1
    
```



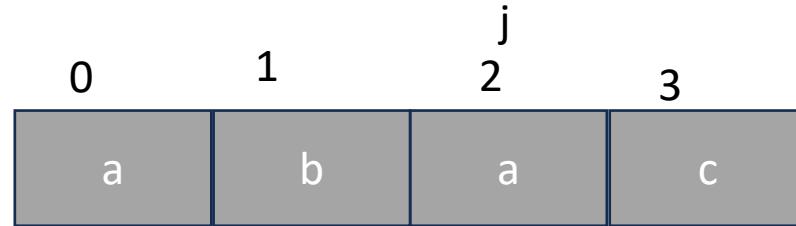
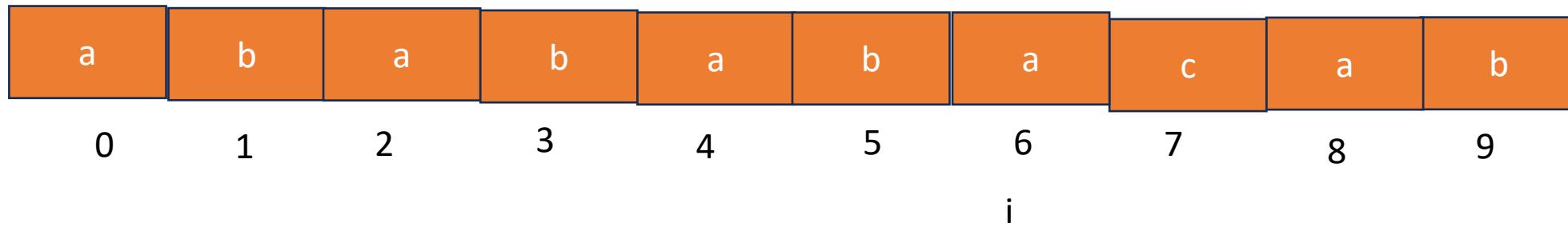
```

While(i < lens && j < lenp)
    if( string[i] == pat [j])
        i++; j++;
    else if (j==0)
        i++;
    else
        j= failure[j-1]+1
    
```



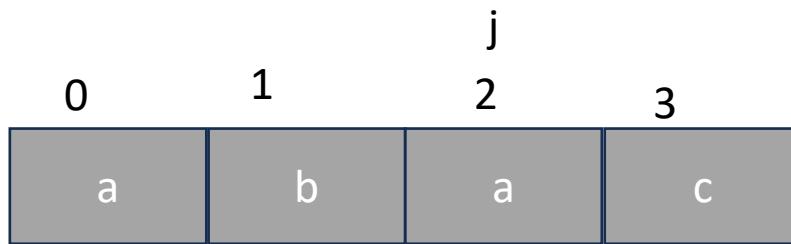
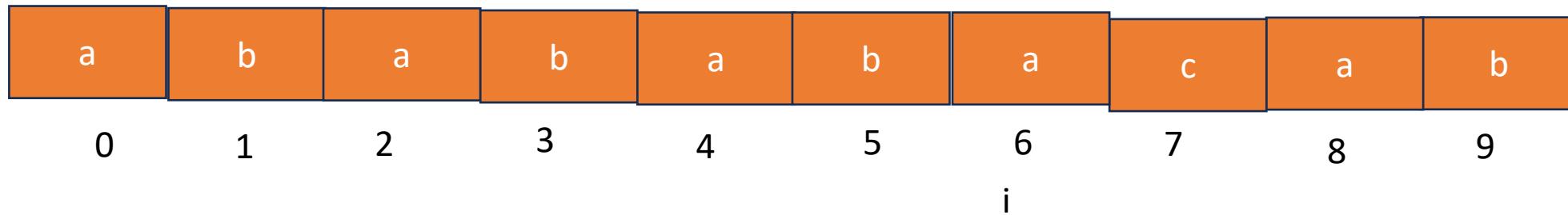
```

While(i < lens && j < lenp)
    if( string[i] == pat [j])
        i++; j++;
    else if (j==0)
        i++;
    else
        j= failure[j-1]+1
    
```



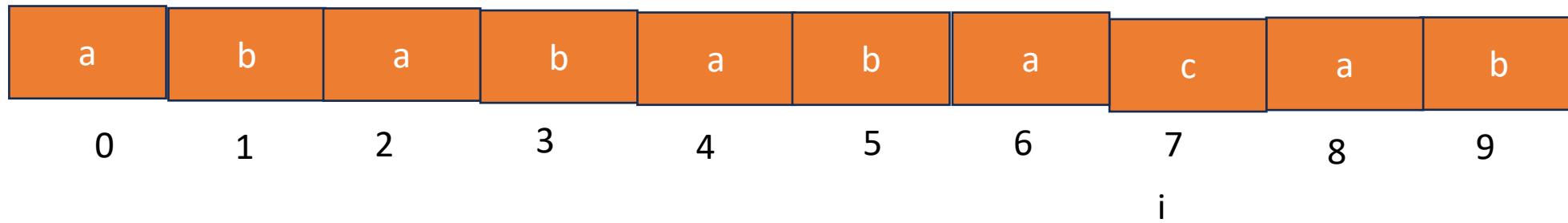
```

While(i < lens && j < lenp)
    if( string[i] == pat [j])
        i++; j++;
    else if (j==0)
        i++;
    else
        j= failure[j-1]+1
    
```



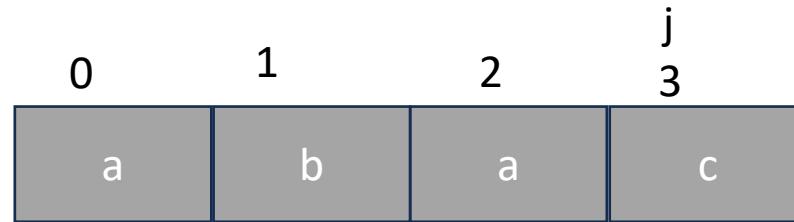
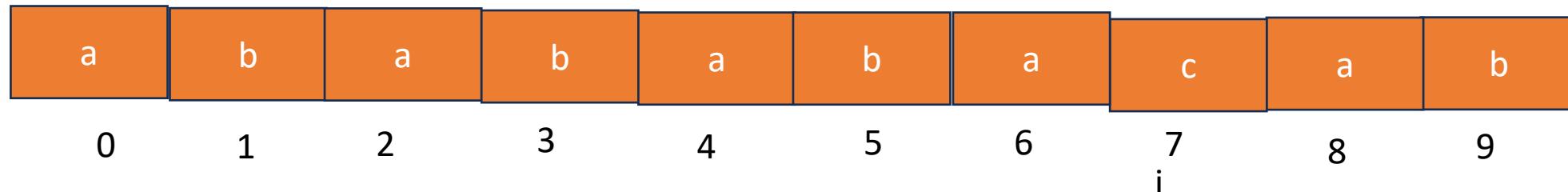
```

While(i < lens && j < lenp)
    if( string[i] == pat [j])
        i++; j++;
    else if (j==0)
        i++;
    else
        j= failure[j-1]+1
    
```



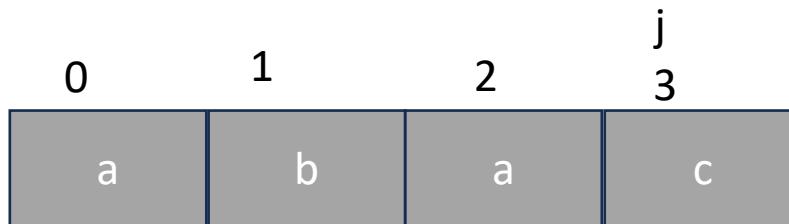
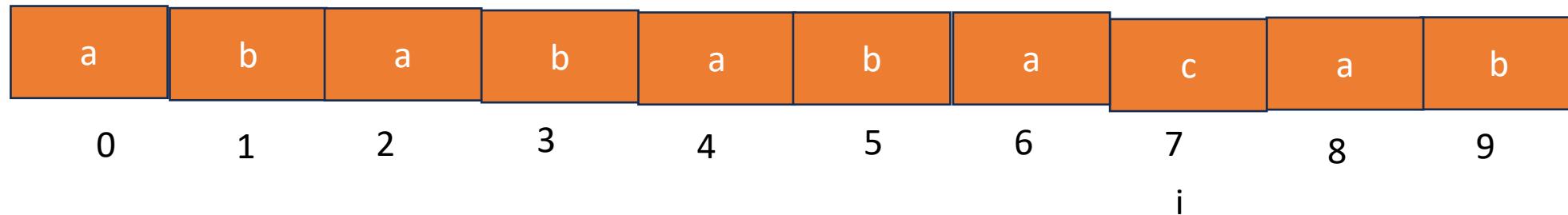
```

While(i < lens && j < lenp)
    if( string[i] == pat [j])
        i++; j++;
    else if (j==0)
        i++;
    else
        j= failure[j-1]+1
    
```



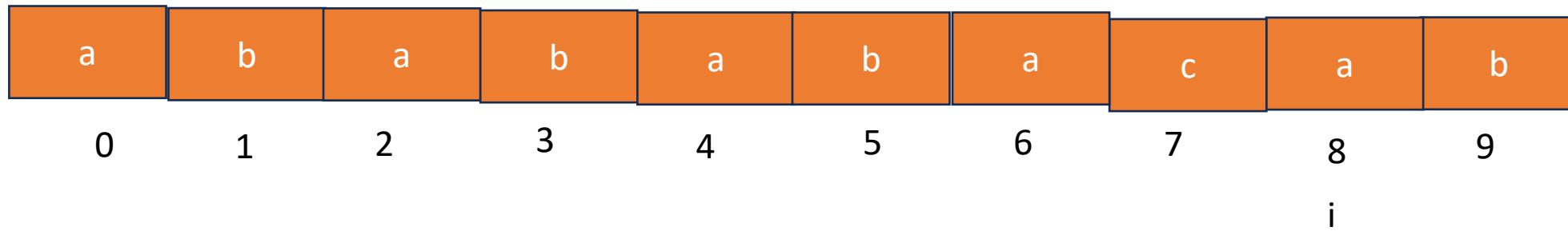
```

While(i < lens && j < lenp)
    if( string[i] == pat [j])
        i++; j++;
    else if (j==0)
        i++;
    else
        j= failure[j-1]+1
    
```



```

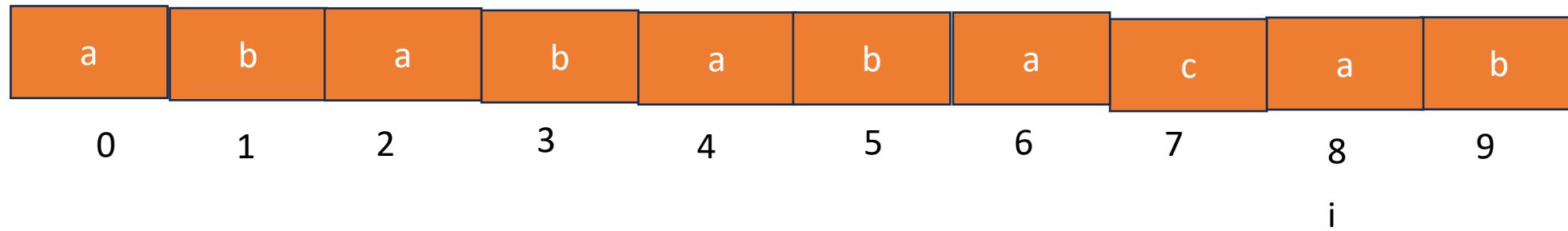
While(i < lens && j < lenp)
    if( string[i] == pat [j])
        i++; j++;
    else if (j==0)
        i++;
    else
        j= failure[j-1]+1
    
```



```

While(i < lens && j < lenp)
    if( string[i] == pat [j])
        i++; j++;
    else if (j==0)
        i++;
    else
        j= failure[j-1]+1

```



```

While(i < lens && j < lenp)
    if( string[i] == pat [j])
        i++; j++;
    else if (j==0)
        i++;
    else
        j= failure[j-1]+1
    
```

KMP – Algorithm for pattern Matching

```
int pmatch(char *string, char *pat)
{ /* Knuth, Morris, Pratt string matching algorithm */
    int i = 0, j = 0;
    int lens = strlen(string);
    int lenp = strlen(pat);
    while ( i < lens && j < lenp ) {
        if (string[i] == pat[j]) {
            i++; j++; }
        else if (j == 0) i++;
            else j = failure[j-1]+1;
    }
    return ( (j == lenp) ? (i-lenp) : -1);
}
```

Program 2.14: Knuth, Morris, Pratt pattern matching algorithm