# Arrays and Abstract Data Type

## • ADTs or Abstract Data Type:

Abstract data types are the ways of classifying data structures by providing a minimal expected interface and some set of methods. It is very similar to when we make a blueprint before actually getting into doing some job.

ADT → Minimal required functionality
ADT → Operations.

An array ADT holds the collection of given elements (int, bit, char, float etc) accessible by their index.

### 1. Minimal required functionality

We have two basic functionalities of an array, a get function to retrive the element at index i and a set function to assign an element to some index in the array.

- get (i) — get element i
- set (i, num) — set element i to num

### 2. Operations

Some basic operations are:

- Max ()
- Min ()
- search (num)
- Insert (i, num)
- Append/(add)

| | | | | | | |
|---|---|---|---|---|---|---|
| Index | 0 | 1 | 2 | 3 | 4 | ← Index |
| Value | 4 | 7 | 13 | 2 | 6 | |
| Address | 10 | 14 | 18 | 22 | 26 | ← Address |

Static arrays: size can't be changed

Dynamic arrays: size can be changed

Array: Array is a collection of items of same data types stored at contagious memory location.

Array

- Onedimensional array/linear array
- Twodimensional array
- multi dimensional array

• You can't resize an array.
• You can resize an array by coping all the elements of in a large array.

## Operations in Array

### Linear search:

Linear search is done by traversing array (going to every element) to find the fixed element and after finding the element traverse is stopped.

| Index → | 0 | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|---|
| | 2 | 7 | 9 | 8 | 10 | 13 |

• find 8

• To find 8 at first we will go to index 0 when the element will not match we will go to index 1. Continuously we will go to every index. When we will go to index 3 and the element will match with 8 the traversing will stop.

* It can be done in both sorted and unsorted arrays.

Code (Linear Search):

```
# include<stdio.h>
int main (){
int a[15], n, t, i;
printf(" Number of elements: ");
scanf(" %d", &n);
printf(" Searching element: ");
scanf(" % d", &t);
for(i = 0; i<n; i++){
 if (a[i] == t){
   break;
 }
}
printf("Element %d was found at index %d", t, i);
return 0;
}
```

Input:

Number of elements: 8
Searching element: 9
2 3 4 9 5 6 7 8

Output:

Element 9 was found at index 3.

Binary search:

* It can be done only in sorted arrays.

In binary searching the index[0] is taken as low and index[n-1] as high and then we find the mid = (low+high)/2. After finding the mid we see that if the searching element is less than the mid then

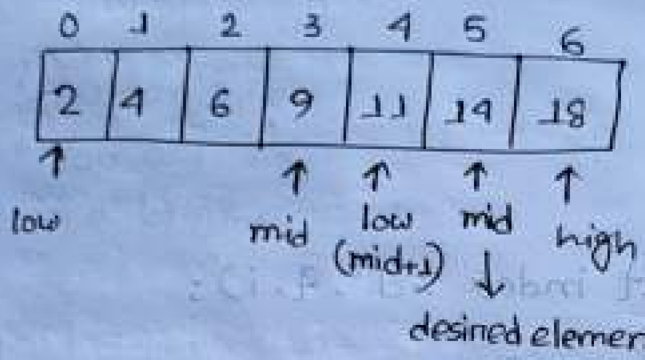
Case:1. low = index[0]
high = mid-1

If the searching element is greater than the mid than.

$\boxed{\text{Case:2}}$ high = index[n-1]

low = mid +1

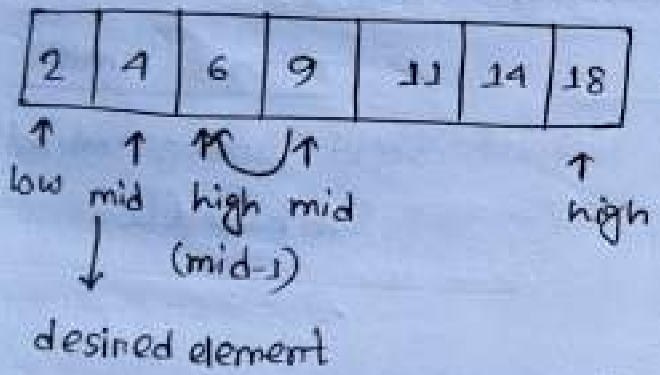
After doing these continuously we can find the desired element.




mid = (low+high)/2

Searching element 14

As mid is 9 and it is less than the searching element 14 so we will follow case:2




Code:

```
#include <stdio.h>
int main (){
int a[15], i, n, element, low, mid, high;
printf(" Number of elements : ");
scanf("%d", &n);
printf (" Searching element : ");
scanf (" %d ", &element);
for(i=0; i<n ; i++)
scanf ("%d", &a[i]);
```

```
low=0;
high=n-1;
while (low <=high){
mid=(low+high)/2;
if(a[mid]==element){
 printf("Index is : %d\n", mid);
 }
if(a[mid]<element){
   low=mid+1;
  }
else{
  high=mid-1;
  }
}
return 0;
}
```

Input:

Number of elements: 8

Searching element: 9

2 3 4 9 5 6 7 8

Output:

Index is : 3

How to find out the size of a given array:

```
int a[] = {2,3,4,5,6,7,8,9}
int size = sizeof(a)/sizeof(int);
printf("%d", size);
```

Output:

8

## Traversing

Visiting every element of an array once is known as traversing the array.

It is used for sorting, printing and updating elements.

### For printing

```
#include<stdio.h>
int main(){
int a[15],i;
for(i=0;i<10;i++)
scanf("%d",&a[i]);
for(i=0;i<10;i++)
printf("%d",a[i]);
return 0;
}
```

Input:

1 5 7 4 6 8 9 10

3 2

Output:

1 5 7 4 6 8 9 10 3 2

### For updating an element:

```
#include<stdio.h>
int main(){
int a[15],i,t;
scanf("%d",&t);
for(i=0;i<10;i++)
scanf("%d",&a[i]);
~~for(i=0;~~ a[5]=t;
for(i=0;i<10;i++)
printf("%d ",a[i]);
}
```

Input:

7

1 2 3 4 5 6 8 9 10 12

1 2 3 4 5 7 8 9 10 12

## Sorting

Sorting means arranging an array in an orderly fashion (ascending or descending).

```
#include<stdio.h>
int main (){
int a[20], i, j, n, temp;
scanf("%d", &n);
for(i=0; i<n; i++)
scanf("%d", &a[i]);
for(i=0; i<n-1; i++){
for(j=0; j<n-i-1; j++){
  if(a[j] >a[j+1]){
   temp = a[j];
   a[j] = a[j+1];
   a[j+1] = temp;
  }
 }
}
for(i=0; i<n; i++)
printf("%d ", a[i]);
}
```

**Input:**

7

2 8 1 5 9 3 15

**Output:**

1 2 3 5 8 9 15

## Insertion

An element can be inserted in an array at a specific position.

### Forward Method:

```
#include<stdio.h>
int main(){
int a[20], i, t, n, x, b;
printf("How many elements you want to insert:");
scanf("%d", &n);
for(i=0; i<n; i++)
scanf("%d", &a[i]);
printf("% Enter the index:");
scanf("%d", &x);
printf(" Enter the value:");
scanf("%d", &b);
t=a[x];
a[x]=b;
for(i=x+1; i<n+1; i++){
int temp=a[i];
    a[i]=t;
    t=temp;
    }
for(i=0; i<n+1; i++)
printf("%d", a[i]);
}
```

Input:

How many elements you want to inse

2 3 4 5 7 8 9

Enter the index: 3

Enter the Value: 15

2 3 4 15 5 7 8 9

## Backward Method:

```
#include <stdio.h>
int main (){
Int i,n,r,a[15],t;
printf(" How many elements:");
scanf ("%d", &n);
for(i=0;i<n;i++)
scanf ("%d", &a[i]);
printf(" Enter the index:");
scanf("%d", &r);
printf("Enter the value:");
scanf ("%d", &t);
a[r]=t;
for(i=n-1; i>=r; i++){
  a[i+1] = a[i];
  }
printf (" New value :");
for(i=0; i<n+1;i++){
  printf("%d ", a[i]);
}
return 0;
}
```

### Input:

How many elements: 7

2 3 4 5 6 7 8

Enter the index: 3

Enter the value: 12

### Output:

New Value: 2 3 4 12 5

6 7 8

## Deletion

An element at a specified position can be deleted, creating a void that needs to be fixed by shifting all the elements to their adjacent left.

## Code:

```
#include<stdio.h>
 int main(){
 int a[30], pos, i,n;
 printf("Number of elements:");
 scanf("%d", &n);
 for(i=0;i<n;i++)
 scanf("%d", &a[i]);
 printf("Deleting array position:");
 scanf("%d", &pos);
 for(i=pos;i<n;i++)
    a[i]=a[i+1];
 for(i=0;i<n;i++)
 printf("%d  ", a[i]);
}
```

## Input:

Number of elements: 7

2 3 4 5 6 78 8

Deleting array position: 3

## Output:

23 2 3 4 6 7 8

To delete all the odd numbers from an array:

```
#include<stdio.h>
int main(){
int a[15], i, j=0, n;
scanf("%d", &n);
for(i=0; i<n; i++){
a[i] = rand()%70;
printf("%d ", a[i]);
}
printf("\n");
for(i=0; i<n; i++){
if(a[i]%2!=0){
a[j] =a[i];
j++;
}
}
for(i=0; i<j; i++){
printf("%d ", a[i]);
}
return 0;
}
```

Input:

8

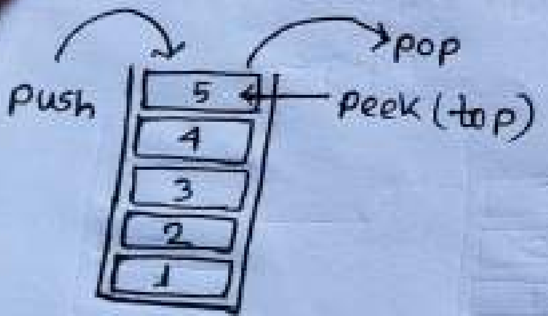
43 46 37 5 43 45 66 2

Output:

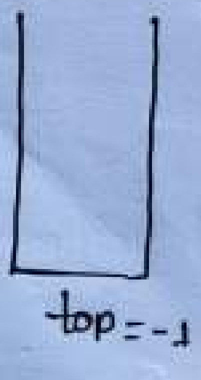
46 66 2

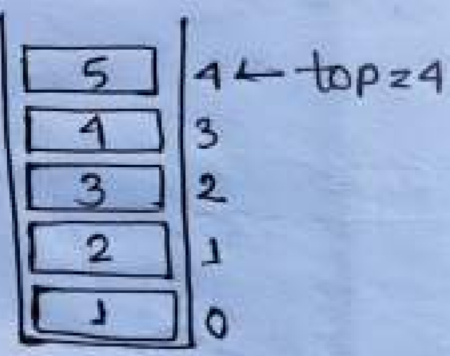
# Stack (Using Array)

A stack is an abstract data structure serving as a collection of elements that are inserted and removed according to the Last in First Out (LIFO) approach. It is a linear data structure. It performs some basic operations such as: push, pop, peek and traverse. Insertion and deletion can happen on the same end of a stack. The top of the stack is returned using the peek operation. It is a sequential data type unlike an array. In an array, we can access any of its elements using indexing, but we can only access the topmost element in a stack.




At stack, when the stack is ~~full~~ not full means when stack is empty its top is -1.




top is defined using index.

• There are two types of stack data structure : Static and Dynamic

1. Static Stack:

A Static stack has a bounded capacity. It can contain a limited number of elements. If a stack is full and does not have any space remaining for another element to be pushed to it, it is then called to be in an Overflow state.

In C static stack is implemented using an array, as arrays are static.

2. Dynamic Stack:

A Dynamic stack is a stack data structure whose capacity increase or decreases in runtime, based on the operations performed on it. In C a dynamic stack is implemented using a Linked List, as linked lists are dynamic data structures.

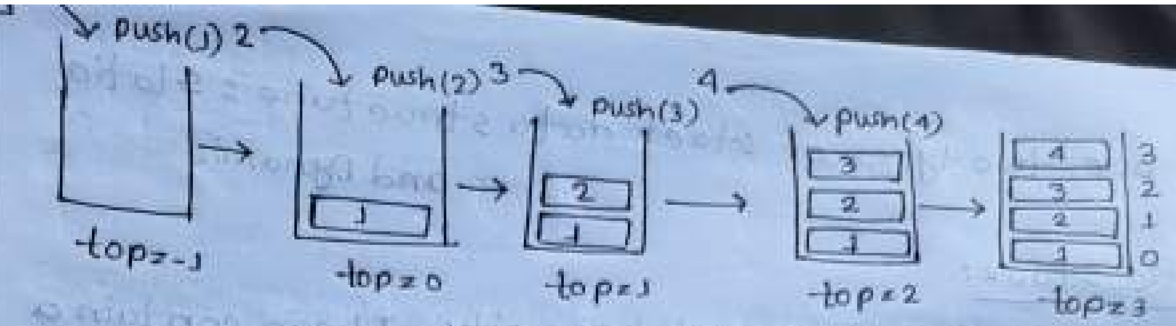
Stack Operations:

1. push (): [Scaler topics → DSA Tutorial → Stack in Data struc

• In stacks, if we try to add or insert elements if the stack is full, it results in a condition known as stack overflow.

In pop operation first check whether the stack is full if full print stack overflow else add data into the stack.

• The process of inserting new data in a stack is known as push.

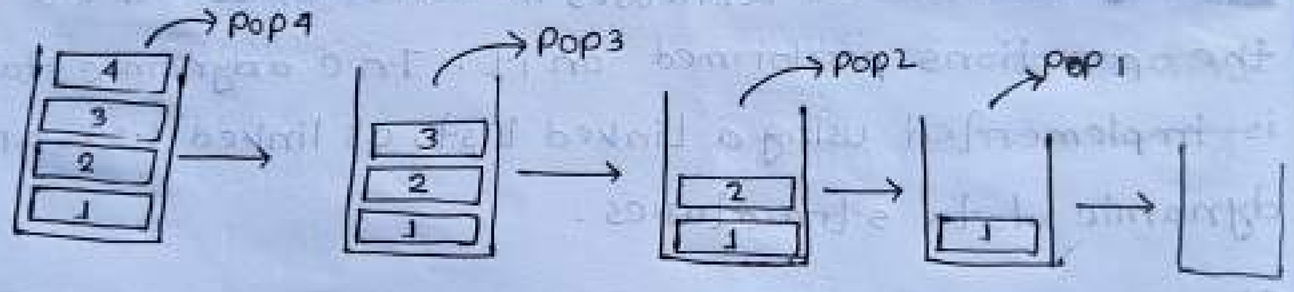



## 2. pop():

The process of deleting the topmost element from the stack is known as pop.

- In stacks if we try to pop or remove element if the stack is empty, it results in a condition known as underflow.

- Before removing the topmost element check whether the stack is empty, if empty then print stack underflow else access the topmost element.




## 3. Peek():

- We use the peek operation to display the topmost element of the stack.

## 4. IsEmpty ():

We use this operation to check whether the stack is empty or not.

5. Is-Full():

We use this operation to check whether the stack is full or not.

Time Complexity: Time complexity for stack operations is O(1).

stack implementation using array

```
#include <stdio.h>
#include <stdlib.h>
#define n 4
int stack[n];
int top = -1;
void push (){
 int value;
 if (top==n-1)
 printf("stack Overflow\n");
 else {
 printf(" Enter the value:");
 scanf("%d", &value);
 top++;
 stack [top]=value;}
 }
void pop (){
if (top==-1)
printf("stack Underflow\n");
else
 top--;
}
void peek (){
 printf("%d\n",stack [top]);}
void size (){
 printf ("%d\n",top+1);
 }
```

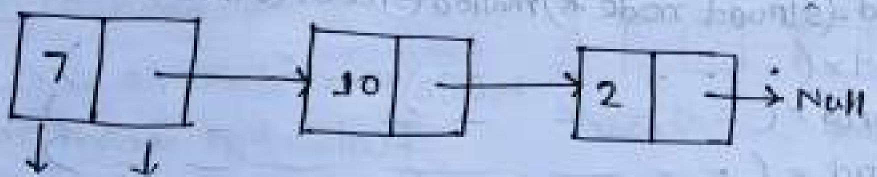
```c
Void isEmpty(){
 if(top==-1)
 printf("stack is empty\n");
 else
 printf("Stack isn't empty\n");
}
Void isFull(){
 if (top==n-1)
 printf("stack is Full\n");
 else
 printf("stack is not Full\n");
}
int main(){
 int choice;
 while(1){
 printf("1.Push\n");
 printf("2.Pop\n");
 printf("3.Peek\n");
 printf("4.Size\n");
 printf("5.Is Empty\n");
 printf("6.Is Full\n");
 printf("7. Display\n");
 printf("Make your choice :");
 scanf("%d", &choice);
 switch(choice){
 case 1:
  push();
  break;
 Case 2:
  pop();
  break;
 Case 3:
  peek();
  break;
 Case 4:
  size();
  break;
 case 5:
  isEmpty();
  break;
 case 6:
  isFull();
  break;
 case 7:
  display(); break;
 case 8:
  exit(1);
 default:
 printf("Insert right key\n");
 }
 }
}
Void display(){
for(int i=top; i>-1; i--){
 printf("%d ", stack[i]);
}
printf("\n");
}
```

# Linked List

- A linked list is a linear data structure, in which the elements are not stored at contiguous memory locations. The elements in a linked list are linked using pointers. In simple words, a linked list consists of nodes where each node contains a data field and a reference (address) to the next node in the list. The first node of a linked list is called the Head and it acts as an access point. On the other hand, the last node is Tail and it marks the end of a linked list by pointing to a NULL value.

## Linked Lists Vs Arrays:


→ In array, elements are stored in contiguous memory locations.




Data

Pointer to the next element

In linked lists elements are stored in non-contiguous memory location.

- Advantage of linked list over array:

→ Memory and capacity of an array remain fixed while in linked lists we can keep adding and removing elements without any capacity constraint.

## • Disadvantages of linked lists:

- Extra memory space for pointers is required (for every node).
- Random acess is not allowed as elements are not sto in contagious memory location.

**Example:** We cann't randomly set the value of an element such as: a[4] = 30

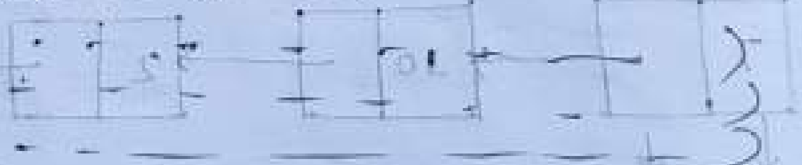
it is not possible in linked list.

## Basic structure:

```
•struct Node {
    int data;
    struct Node *next;
 };
```

## main function:

```
• int main () {
  struct node *head=(struct node *)malloc (sizeof (struct node));
  struct node *first = ( .
  struct node *second = (
  struct node *third = ( -
```



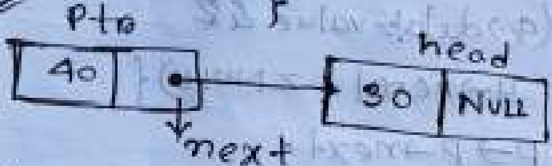
```
  head -> data = 12;          first -> data = 24;
  head -> next = first;       first -> next = second;
  second -> data = 32;        third -> data = 40;
  second -> next = third;     third -> next = NULL;
```

1. head = Insertfirst (head, 62);
2. head = insert atend (head, 62);
3. head = insertafternode (head, second, 62);
4. head = insert between (head, 62, 2);

1. head = deletefirst (head);
2. head = deletelast (head);
3. head = deleteatindex (head, 2);
4. head = deletebyvalue (head, 32);

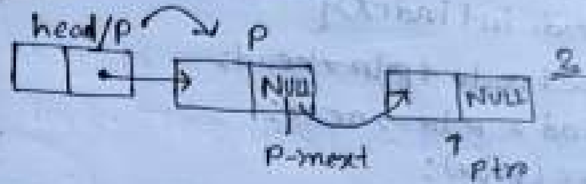
```
• Void traversal (struct node* ptr){
  while (ptr != NULL){
    printf("%d \n", ptr->data);
    ptr = ptr->next;
  }
}
```

1

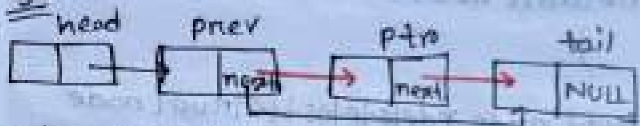



```
• struct node *Insertfirst(struct
      node *head, int data){
  struct node* ptr = malloc(sizeof( ));
  ptr->data = data [ ptr is the
  ptr->next = head;        new node]
  head = ptr;
  return head;}
```

```
• struct node *insertatend(struct
      node *head, int data){
  struct node* ptr = malloc ;
  struct node *p = head;
  while(p->next != NULL){
      p = p->next; }
  ptr->data = data;
  ptr->next = NULL;
  p->next = ptr;
  return head;
}
```

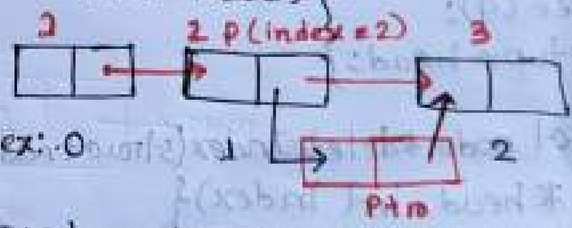



3




```
• Struct node* insertafternode {
    struct node * head, struct node*
        prev, int data){
  struct node *ptr = malloc ...
  ptr->data = data;
  ptr->next = prev->next;
  prev->next = ptr;
  return head; }
```

4




```
• struct node *insertbetween(
  struct node* head, int data, int
                         index){
  struct node *ptr = malloc ...
  struct node *p = head ;
  for(int i=0; i!=index-1; i++){
      p = p->next;
  }
```

```
ptr->data = data;
ptr->next = p->next;
p->next = ptr;
return head;
}
```

- struct node *deletefirst(struct node *head, int index){

```
struct node *ptr = head;
head = ptr->next;
free(ptr);
return head;
}
```

- struct node *deletelast(struct node *head){

```
struct node *p = head;
struct node *q = head->next;
while (q->next != NULL){
    p = p->next;
    q = q->next;}
p->next = NULL;
free(q);
return head;}
```

- struct node *deleteIndex(struct node *head, int index){

```
struct node *p = head;
struct node *q = head->next;
for (int i=0; i<index-1; i++){
    p = p->next; q = q->next;}
p->next = q->next;
free(q);
return head;
}
```

- struct node *deletebyvalue(struct node *head, int value){

```
struct node *p = head;
struct node *q = head->next;
while (q->data != value &&
       q->next != NULL){
    p = p->next;
    q = q->next;
}
if (q->data == value){
    p->next = q->next;
    free(q);
}
return head;
}
}
```

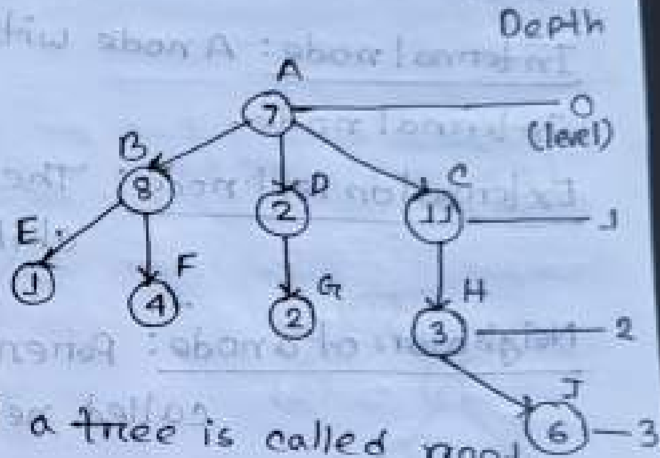
# Tree

What is tree?

⇒ A tree is a non-linear data structure because it does not store in a sequential manner. It is a hierarchical structure and a collection of nodes that are connected by edges and has a hierarchical relationship between the nodes.

## Basic Terms of tree:

for N node edges will be (N-1).




**Root:** The topmost node of a tree is called root. It doesn't have any parent node. Here node A is the root node.

**Child node:** A node which has an edge pointing to it from some other node. H is the child of C and J is the child of H.

**Parent Node:** A node which an edge pointing to some other node. C is the parent of H.

**Siblings:** Nodes belonging to the same parents are called siblings to each other. Node B, C, D are siblings of each other and they have same parent A.

Ancestor of a node: Any predecessor nodes on the path of the root to that node are called Ancestors of that Node. A, C, H are ancestors of node I.

Descendant: Any successor node on the path of the leaf node to that node. H and I are descendant of C.

Internal node: A node with at least one child is called Internal node.

External or leaf node: The node which does not have any child. E, F, G, H, I

Neighbour of a node: Parent or child nodes of a node are called neighbours of that node.

Subtree: Any node of the along with its descendant.

Depth: Depth of a node is the number of edges from root to that node. The depth of node A, C, H and I are 0, 1, 2, 3 respectively.

Height: Height of a node is the number of edges from that node to the leaf node. Height of node C is 2, and A is 3.

Height of a tree: Number of edges from root to the leaf node of longest distance.

Degree of a node: The total count of subtrees attached to that node.

The degree of root node is 3 because it has at most 3 children.

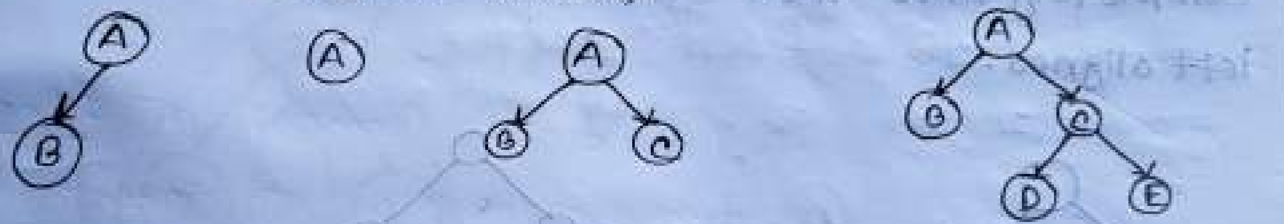
Degree of Tree: degree হচ্ছে ঐ node এর no of children.

degree of tree means maximum degree of a node.

level of node: The count of edges on the path from the root node to that node.

## Binary Tree

It is a type of tree where each node should have at most two children. (0,1,2)




Maximum no of node for height h:

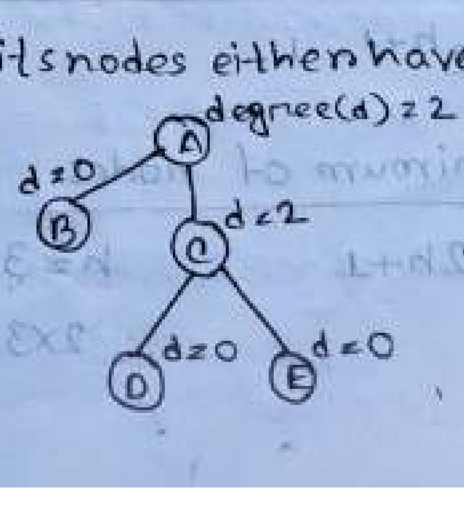
$(2^{h+1} - 1) = n \rightarrow$ node

Minimum no of node for height h:

$h+1 \rightarrow n$ (node)

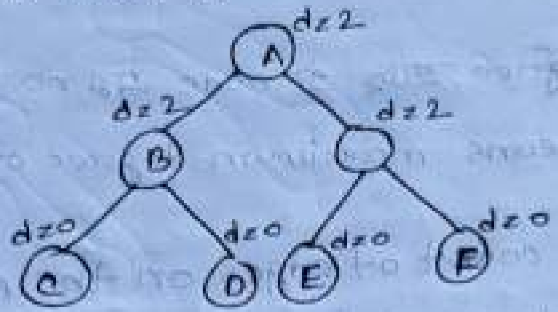
Full binary tree: (Extended binary tree)

It is a type of tree where each of its nodes either have 2 children or is a leaf node.



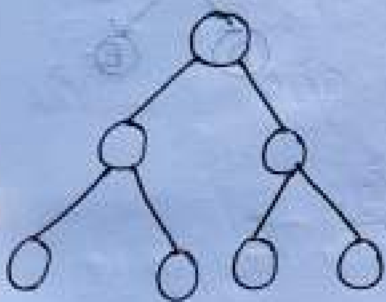
Perfect binary tree:

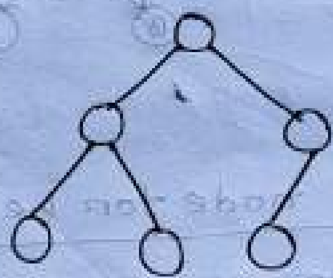
It is a type of tree when each node have exactly 2 nodes.




Complete binary tree:

A complete binary tree has all its levels completely filled except the last level. And if the last level is not completely filled then the last levels keys must be left aligned.





Binary

Maximum node of a full binary tree

$2^{h+1}-1$

Minimum of nodes

$2h+1$ $h=3$

$\therefore 2\times3+1=7$ nodes

Complete Binary tree (maximum nodes) : $2^{h+1} - 1$

(minimum nodes) : $2^h$

## Binary tree travels

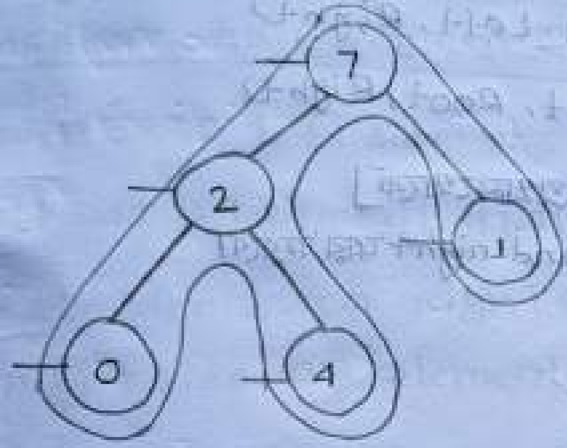
Preorder ( Root, Left, right)

postorder ( left, right, root)

Inorder ( Left, Root, right)

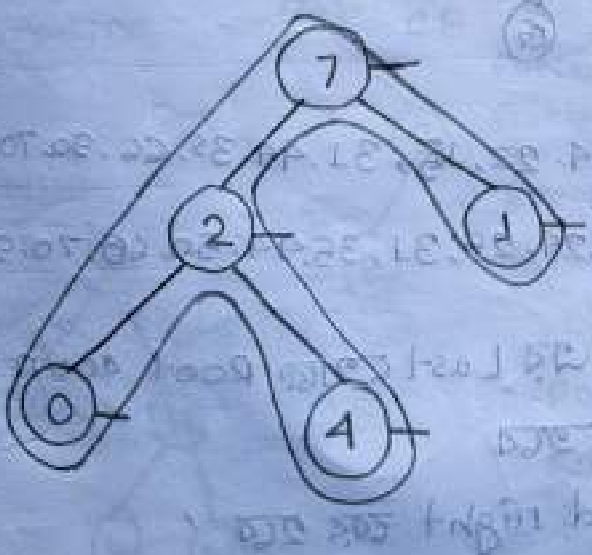
**PreOrder:**

7 2 0 4 1

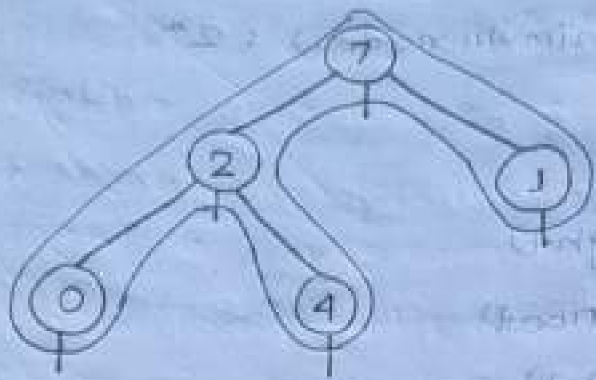



**Postorder:**

0 4 2 1 7



Inorder:



0 2 4 7 1

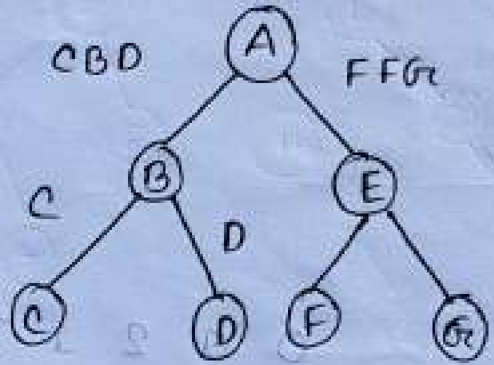
Construct binary tree from preorder and inorder

Preorder: ~~ABC~~DEFG (Root, Left, Right)

Inorder: $\underbrace{CBD}_{L}A\underbrace{FEG}_{R}$ (Left, Root, Right)

Preorder থেকে Root নিবো [প্রথম থেকে]

Inorder থেকে root এর Left and right বের করবো

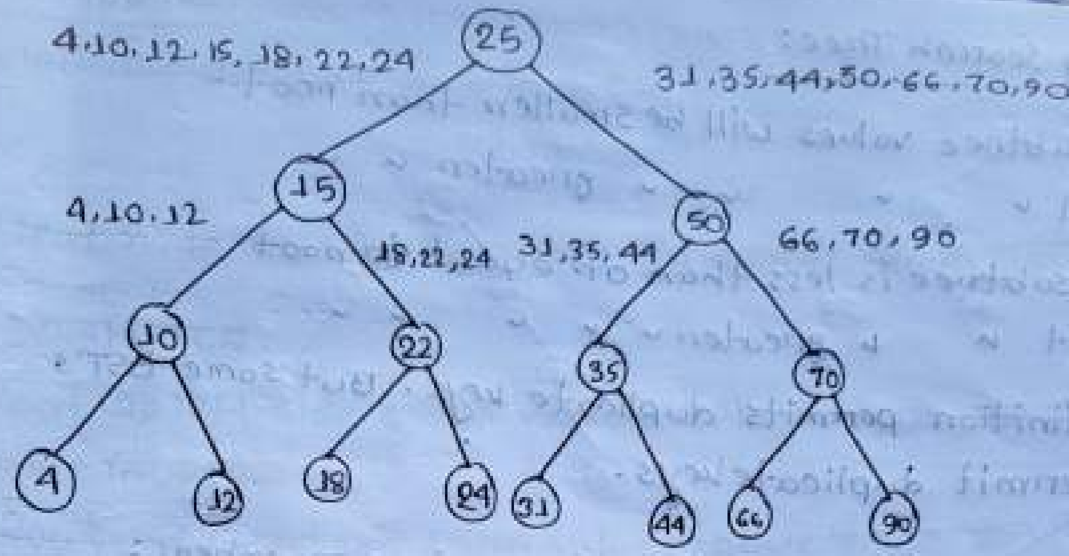


Postorder : 4, 12, 10, 18, 24, 22, 15, 31, 44, 35, 66, 90, 70, 50, 2

Inorder: 4, 10, 12, 15, 18, 22, 24, 25, 31, 35, 44, 50, 66, 70, 90

Postorder (L, R, Root) : এর Last থেকে Root count শুরু হবে

Inorder থেকে Left and right বের হবে।




## Preorder and Postorder থেকে

i) Preorder থেকে left side বের করবো

⇒ Left side এর element গুলো post order থেকে পাবো।

ii) postorder থেকে right side বের করবো

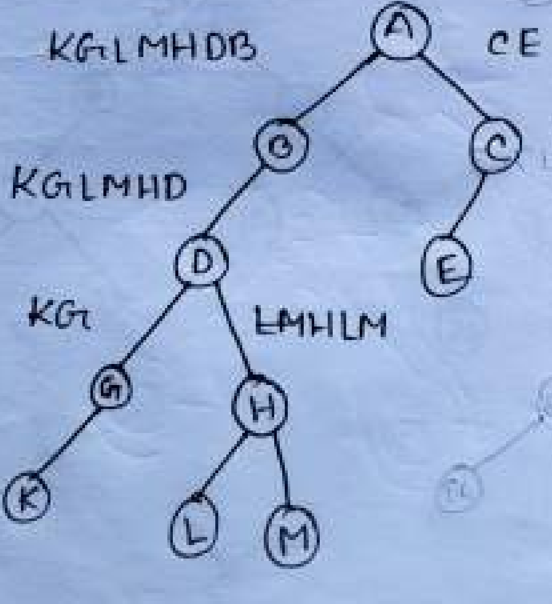
⇒ right side এর element গুলো preorder থেকে পাবো

Pre: ABDGKHLMCE (Root, L, R)

Post: KGLMHDBECA (L, R, Root)

Pre এর First and Post এর last same হলে তা root

Pre থেকে Left B post থেকে B এর পূর্ববর্তী KGLMHD সব B এর left element. এবং post থেকে C right এবং pre থেকে E হচ্ছে right এর element




Pre: BDGKHLM
Post: KGLMHDB

Pre: DGKHLM,
Post: KGLMHD

pre: GK
Post: KG

pre: HLM
Post: LMH

pre: CE
post: EC

# Binary Search Tree:

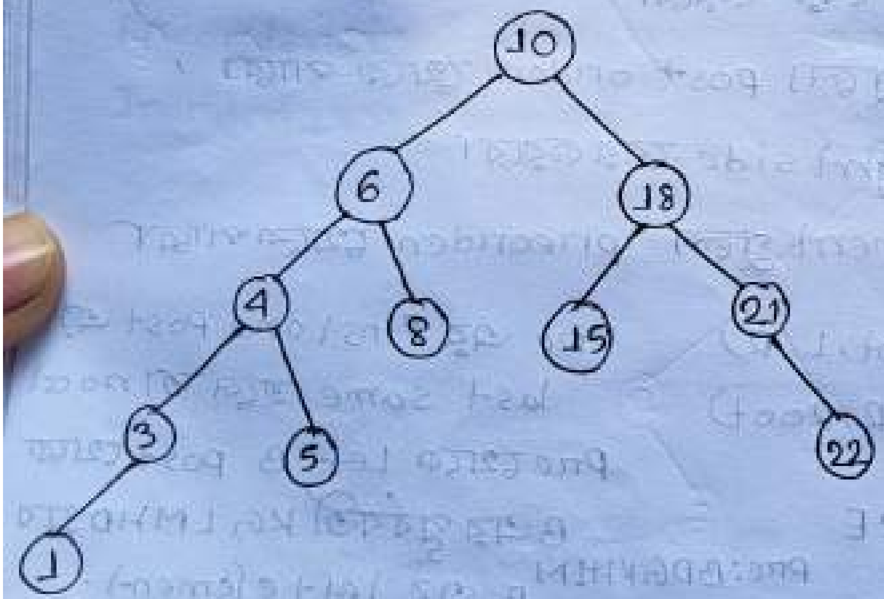
i) Left subtree values will be smaller than root.
ii) Right 〃 〃 〃 〃 greater 〃 〃
iii) left subtree is less than or equal to root
right 〃 〃 greater 〃 〃 〃 〃 〃

This definition permits duplicate keys. But some BST's don't permit duplicate keys.

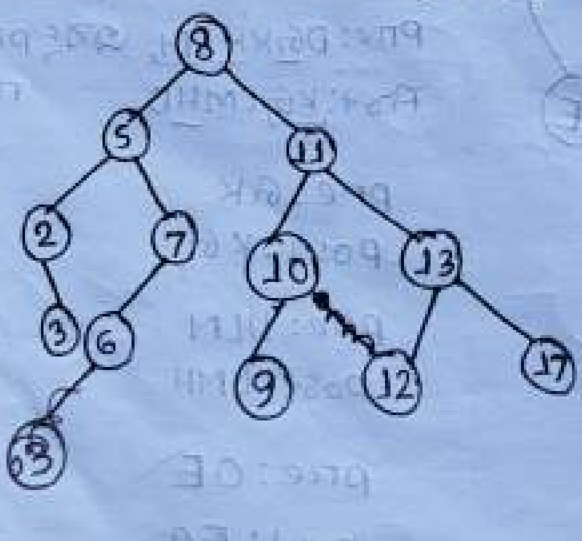
## Binary search tree insertion from given numbers:

Example: 10, 18, 6, 4, 21, 8, 15, 22, 3, 1, 5

⟶ Left to right



Ex2: 8, 5, 2, 7, 6, 11, 13, 10, 9, 12, 17, 3



## BST Deletion:

Deletion 3 প্রকার। যথা: i) leaf node
ii) non-leaf node
iii) root node

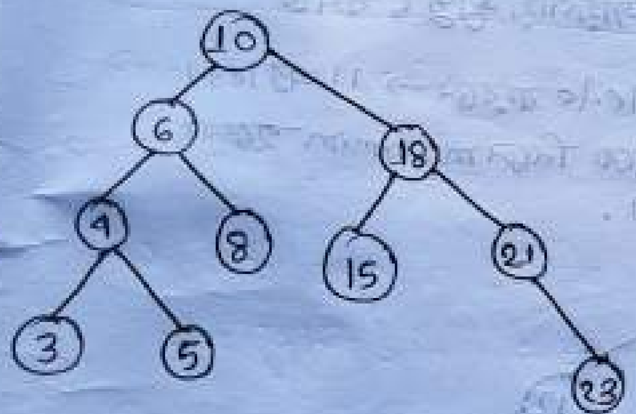
1) leaf node সহজেই delete করা যায় এরজন্য tree পরিবর্তিত হয় না।

2) অন্য ক্ষেত্রে.

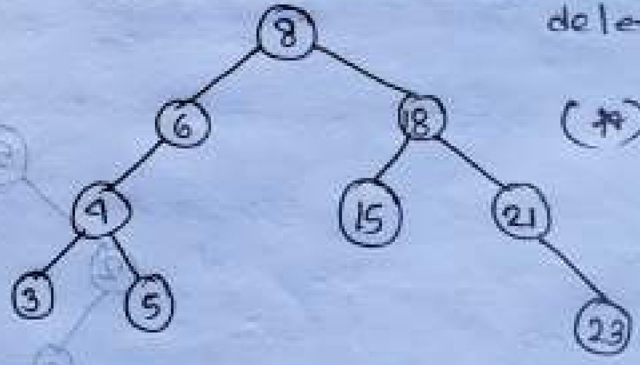
⇒ Root এর left side থেকে value replace করতে হলে maximum value নিতে হবে।

⇒ Root এর right side থেকে value replace করতে হলে minimum value নিতে হবে।

i) 10 delete করলে ii) 6 delete করলে
iii) 23 delete করলে

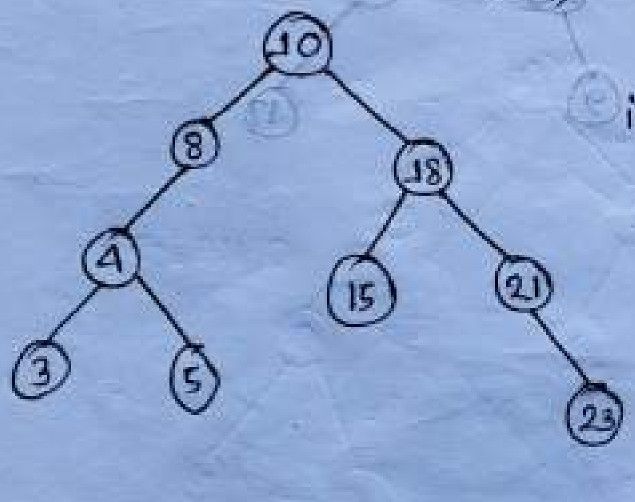


i) 10 root তাই left side থেকে নিলে 8 নিতে হবে (root node deletion)

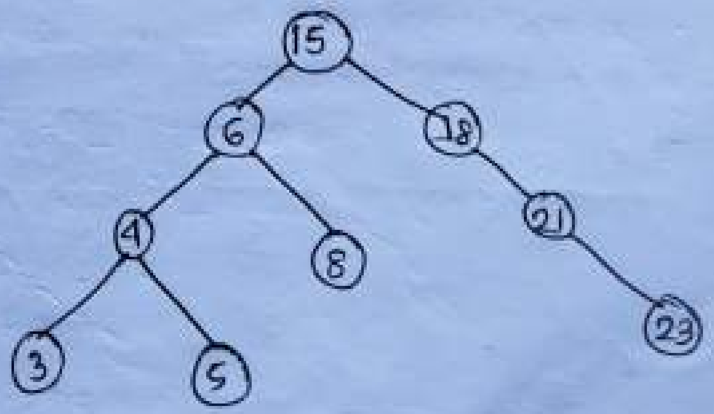


(*)

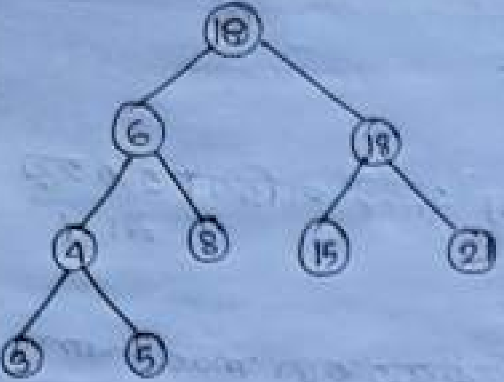
ii) 6 delete করলে (non-leaf)



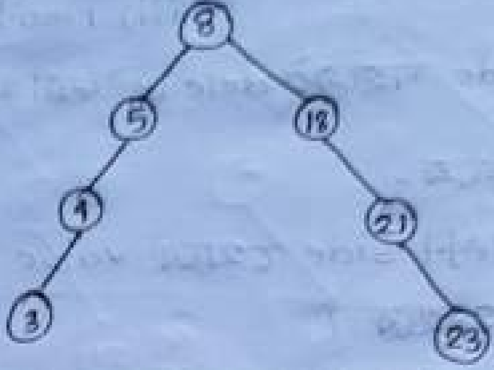
ii) আর right side থেকে নিলে 15 নিতে হবে



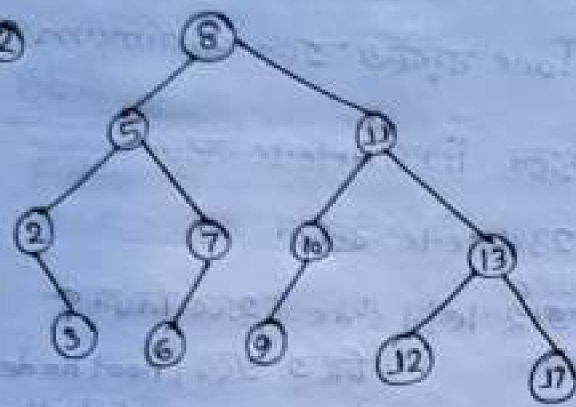
iii) 23 delete (leaf node deletion)



* চিহ্নিত tree থেকে 6 delete করলে



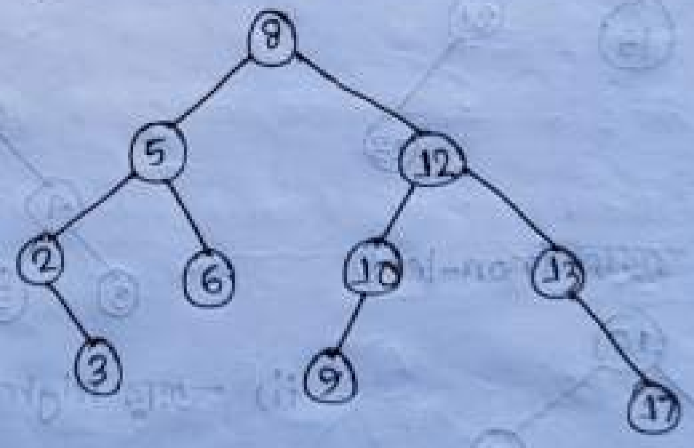
②



i) 7 delete করলে →

7 এর জায়গায় শুধু 6 বসবে

ii) 11 delete করলে → 11 এর left side থেকে নিলে minimum হলো 12.



## Huffman's coding

Haffman coding is a compression technique. It is used to reduce the size of data on message.

Example: Message: BCCABBDDAECCBBAEDDCC

Length: 20 → Message will be send ASCII code instead of characters.

ASCII codes → 8 by bits

∴ Total Number of bits → 20×8 = 160 bits.

Fix size of code:

| Character | Count | Frequency | Code |
|---|---|---|---|
| A | 3 | 3/20 | 000 |
| B | 5 | 5/20 | 001 |
| C | 6 | 6/20 | 010 |
| D | 4 | 4/20 | 011 |
| E | 2 | 2/20 | 100 |
| | 20 | | |

Table: 1

There are 5 char. And They are used in the message as their count times (A is used 3 times). Now we need to write code (Binary). If there were 4 charac. ($2^2 = 4$) codes will be: 00, 01, 10, 11. When there are 5 char. ($2^3 = 8$). We will use three bits code.

According to the Table: 1, There are 20 characters and each take 3 bits (code) = 20×3 = 60 bits → Size of the message

$5 \times 8 = 40$ bit
↑
size of characters

$5 \times 3 = 15$ bit
↑
size of total codes (There are 5 codes and they are each 3 bit)

∴ Table or chart of codes $= 40 + 15$
$= 55$ bits

Message $= 60$ bits
Table $= 55$ bits
115 bits

Variable size code:

Haffman ~~en~~coding:

The mess: BCCABBDDAECCBBAEDDCC

There are 20 characters and each char occupy 8 bits.

∴ The total size of the string is $= 20 \times 8 = 160$ bits.

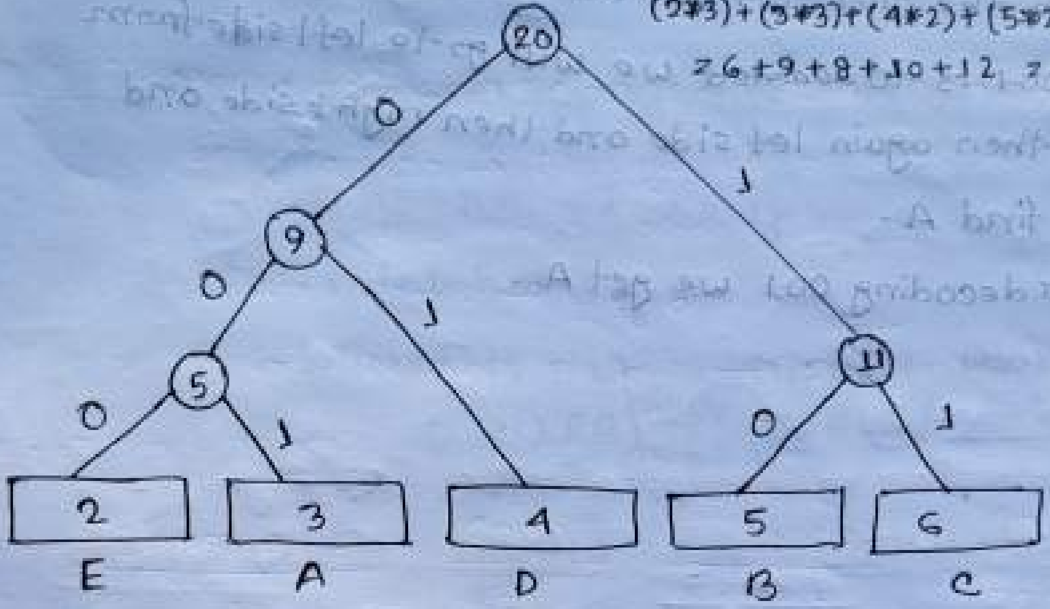
Huffman coding first creates a tree using the frequencies of the char. and generates code for each char.

1. First calculate the frequency of each char. of string.
2. Sort the characters according to the frequencies.

| Character | Count | Code | Size |
|---|---|---|---|
| A | 3 | 001 | → $(3 \times 3) = 9$ |
| B | 5 | 10 | → $(5 \times 2) = 10$ |
| C | 6 | 11 | → $(6 \times 2) = 12$ |
| D | 4 | 01 | → $(4 \times 2) = 8$ |
| E | 2 | 000 | → $(2 \times 3) = 6$ |
| $(5 \times 8) = 40$ bits | 20 bits | 12 bits | 45 bits |

3. take two minimum frequency nodes and add a new internal node with the sum of the frequency.

Weighted path length from tree =
$(2*3)+(3*3)+(4*2)+(5*2)+(6*2)$
$= 6+9+8+10+12 = 45$




Take two smallest frequencies and add them to make an internal node. 2+3=5 here left frequencies are 5, 4, 5, 6. we take 5+4=9. Then left 9, 5, 6, two smallest are 5+6=11. Then add 11+9=20. Use 0 for left side edges and 1 for right side edges. And then find the code of ~~freq~~ the characters.

Now the encoded message is:

B C C A B B D D A E C C B B A E D D C C
10 11 11 001 10 10 01 01 001 000 11 11 1010001000 0101 1111

Now count the total size after encoding:

Size = (40 + 12 + 45) = 97 bits

$\therefore$ compression ration = $\frac{97}{160}$ =

Now decoding

For decoding the code, we can take the code and traverse through the tree to find the character.

when 001 is to decoded we will go to left side from root, then again lef side and then right side and we will find A.

So, after decoding 001 we get A.

# AVL Tree

AVL Tree is a height balanced binary search tree.
(Adelson - Velsky - Landis)

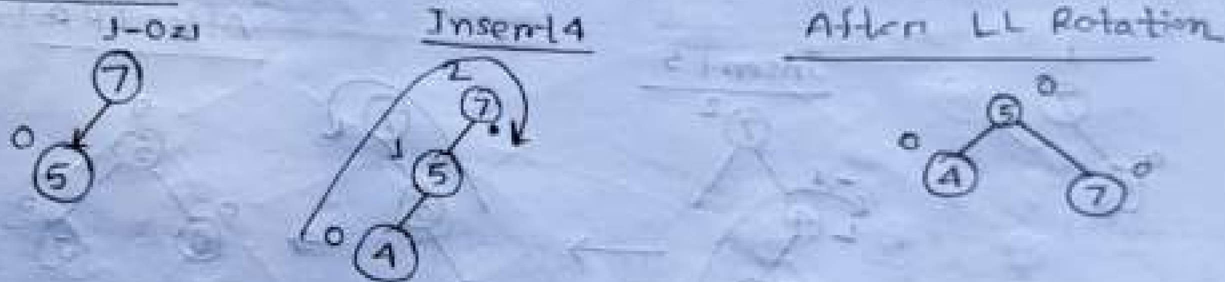
Balance factor = height of left subtree - height of right subtree

$= hl - hr = \{-1, 0, 1\}$

$\therefore |bf| = |hl - hr| \leq 1$

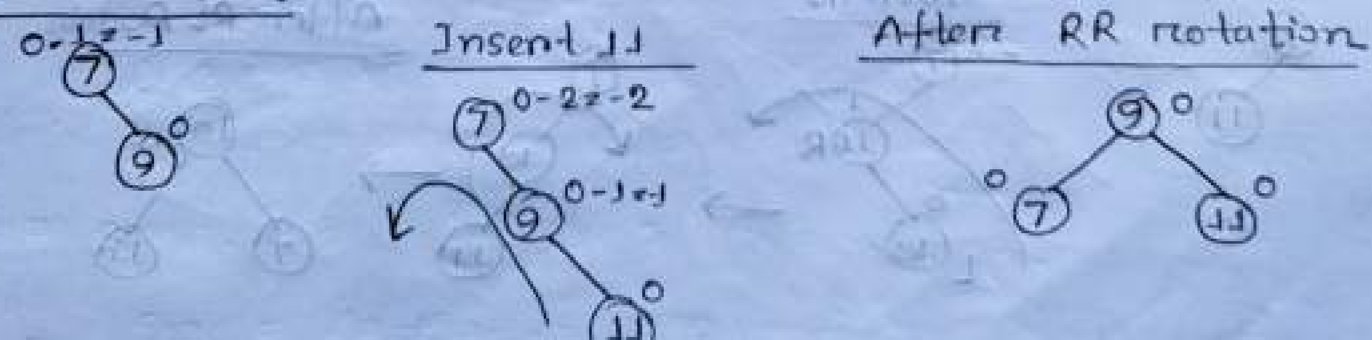
There are 4 rotations: i) LL, ii) RR (single rotation); iii) LR, iv) RL (Double Rotation)

## LL Rotation:




As the balance factor in 7 is 2. So it is imbalanced. And It is imbalanced for Left-Left or LL type. So its rotation will be LL rotation.

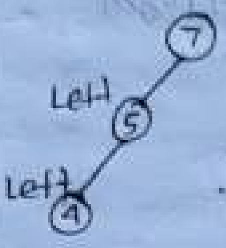
## RR rotation




After inserting 11 the balance factor of 7 becomes (-2) that is imbalance for right to right because hight in right side is too large. So we need RR rotation.

## Why LL Rotation

Because new element 4 was inserted in the left of left

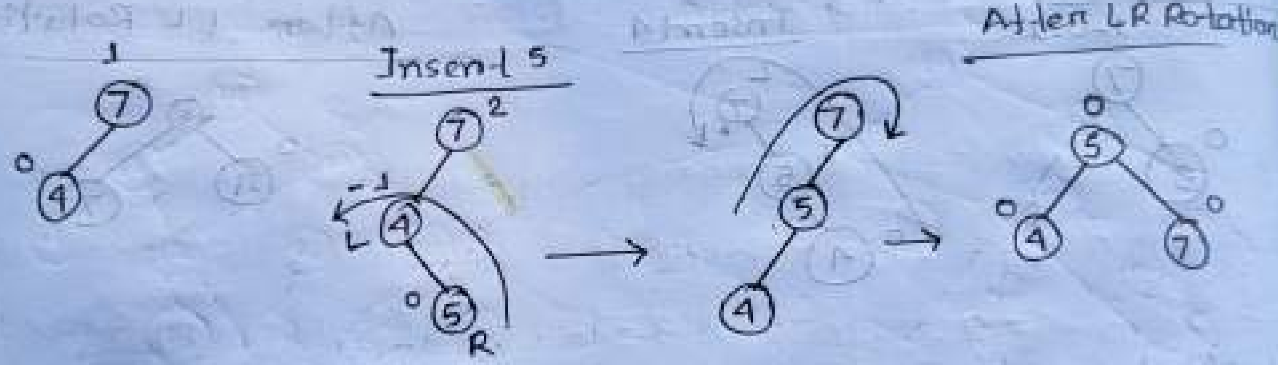



## Why RR Rotation

New element 11 was inserted in the right of right

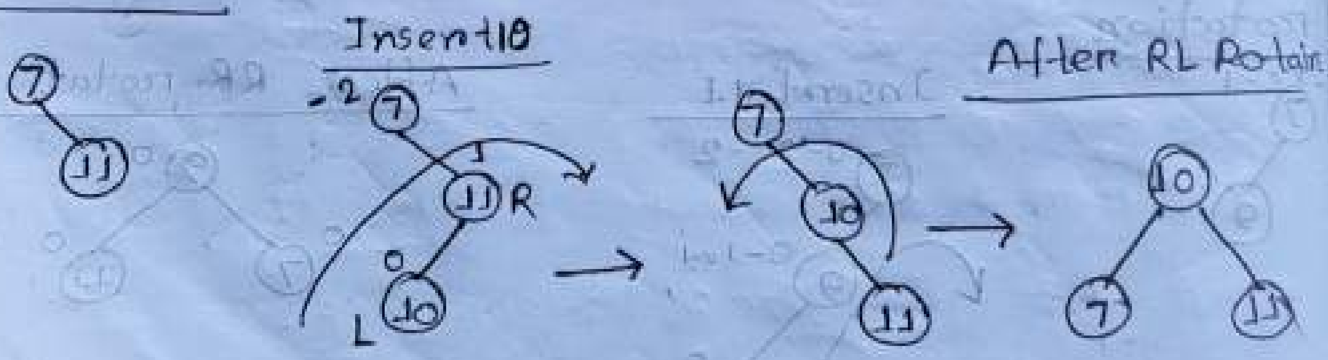
## LR Rotation:

When insertion will happen in(Left का Right).




LR Rotain ( First anticlockwise rotation then clockwise rotation)

## RL Rotain:




RL Rotation ( First clockwise then anticlockwise).

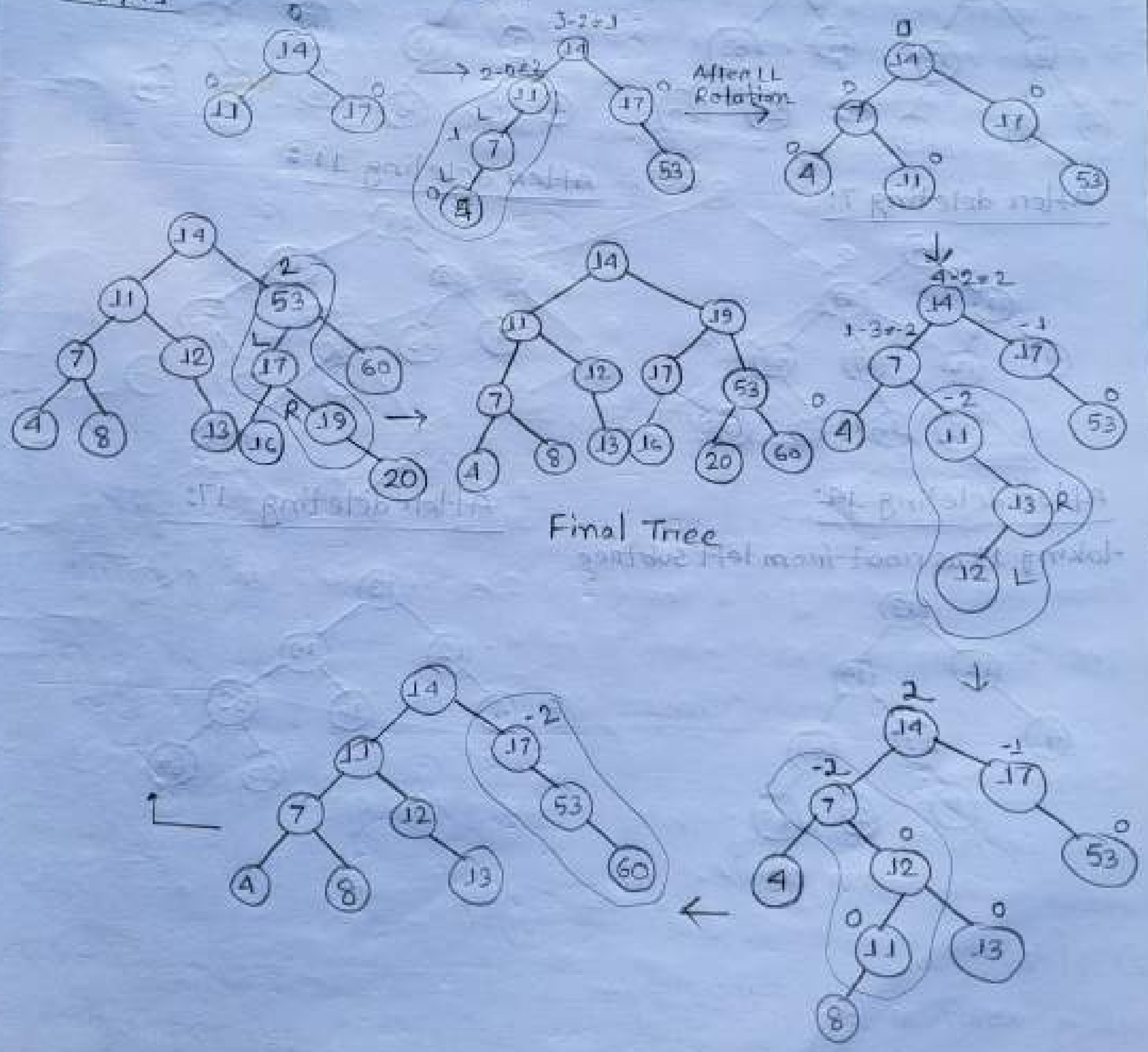
# AVL Insertion

14, 17, 11, 7, 53, 4, 13, 12, 8, 60, 19, 16, 20

⟶ Start from here

- After inserting each node check the balance factor if the tree is not balanced then rotate it according to its imbalance. (Tips : imbalance থাকলে balance করার সময় সর্বদা 3 টি value এর মধ্যে median value root এ বসবে biggest value right এ আর smallest value left এ).

Step: 1

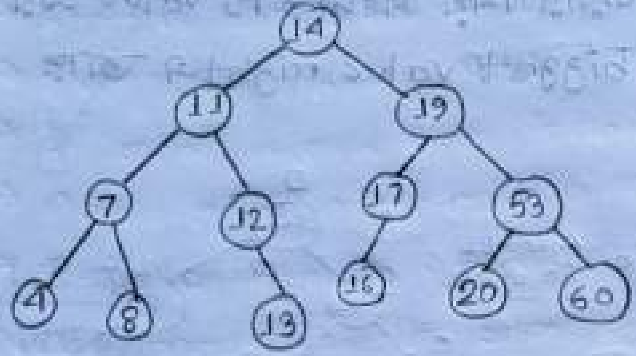



Final Tree

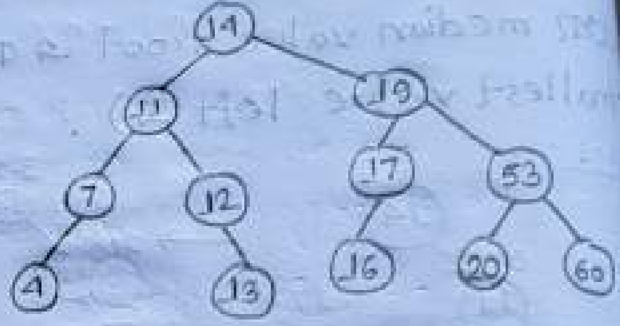
# AVL Deletion

- Its deletion is alike BST deletion but after deleting one have to check the balance factor and if the tree is imba then we have to balance it.

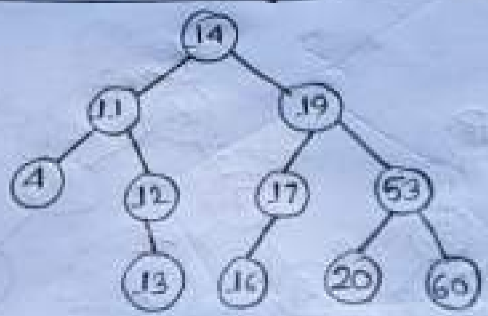
- 8, 7, 11, 14, 17 → delete these.



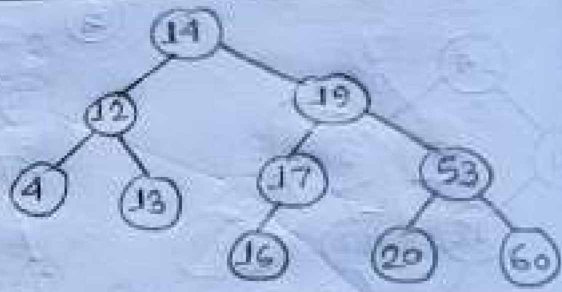
After deleting 8:



After deleting 7:

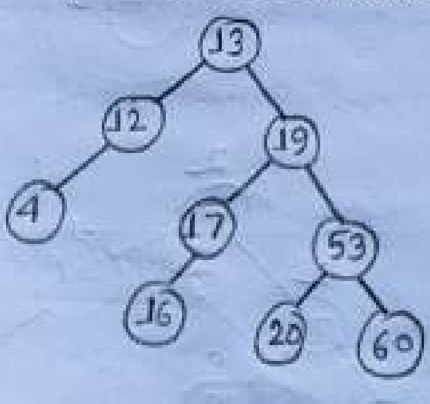


After deleting 11:

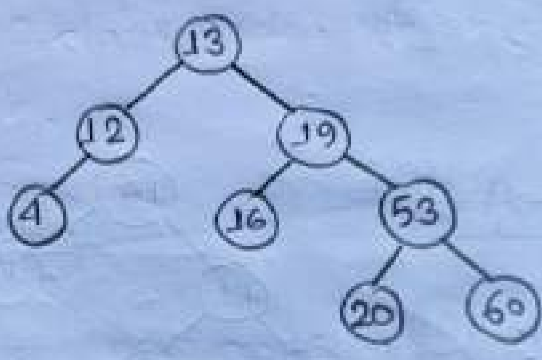


After deleting 14:

taking 13 as root from left subtree



After deleting 17:



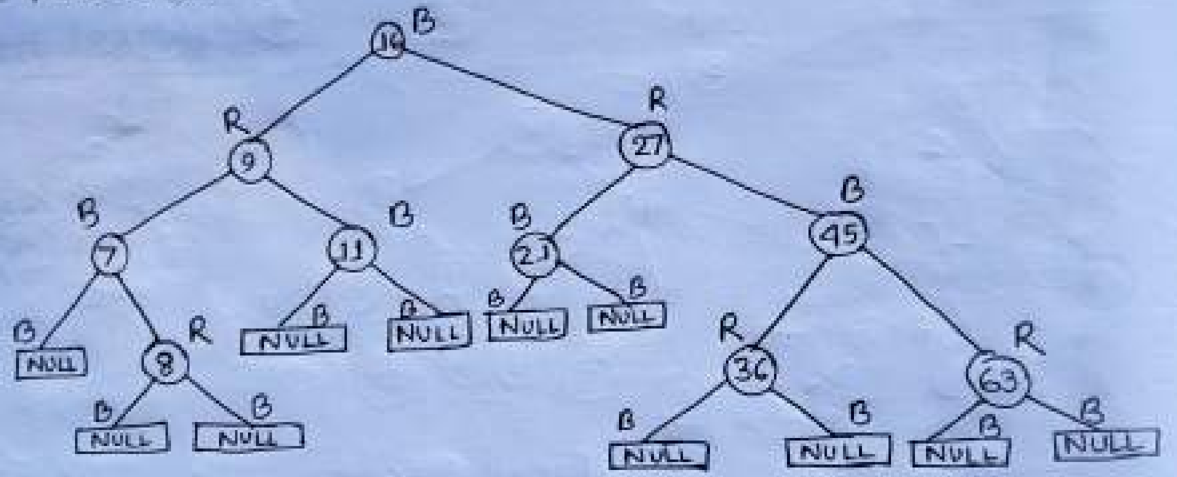
# Threaded Binary Tree

A threaded binary tree is a type of binary tree data struc- where the empty left and right child pointers in a binary tree are replaced with threads that link nodes directly to their in order predecessor or in order successor.

# Red Black Tree

Red Black tree is a self balancing binary search tree in which every node is colored with either red or black. It is a self-balancing BST.

→ Every node is either red or black.
→ Root is always black.
→ Every leaf which is NULL is Black
→ If node is red then its children are black.
→ A red node can't have a red parent or red child.
→ A red node can't have a red (including root) to any of its descendants
→ Every path from a node (including root) to any of its descendants NULL nodes has the same number of black nodes.




From root to every null there are 2 black nodes which justify the last condition related to path.

## Insertion

Conditions:

1) If the tree is empty then create a new node as root node with color black.
2) If tree is not empty create new node as leaf node with color red.

3) If parent of new node is black then exit.

4) If parent of new node is red then check the color of newnodes uncle.

⇒① If uncle is black or null then do suitable rotation and recolor
[If rotation LL and RR, then Grand Parent and Parent node will be recolored, if rotation is RL or LR then Grand Parent and child node is recolored]

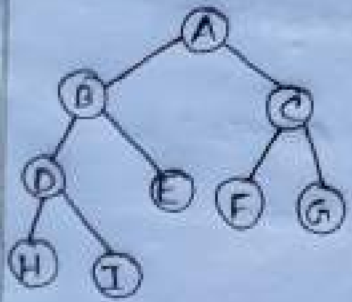
⇒② If uncle is red then recolor and also check it that if grandparent of new node is not root then recolor it and recheck.

* 10, 18, 7, 15, 16, 30, 25, 40, 60, 2, 1, 70

## Array Representation of Binary Tree (Sequential Representation

- If node is at $i^{th}$ index:
  left child at = $(2*i)$ index
  right child at = $[(2*i)+1]$ index
  parent at = $\lfloor i/2 \rfloor$ index

- In this case zero index is skipped we start from index 1.




array representation for this tree:

| A | B | C | D | E | F | G | H | I |
|---|---|---|---|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |

(left to right representation)

- for $i = 5^{th}$ index parent is at $\lfloor i/2 \rfloor = \lfloor 5/2 \rfloor = 2^{nd}$ index
  - Parent of E is B.

# Stack

## Infix, prefix, postfix

| Operator | Precedence | Associativity |
|---|---|---|
| $ or ↑ or ^ | highest | Right to left |
| *, / | Next highest | Left to right |
| +, - | lowest | Left to right |

### Infix

<operand><operator><operand>

- A+B
- 2*3+9/3-5

= 6+9/3-5
= 6+3-5
= 9-5
= 4

### Prefix

<operator><operand><operand>

- +AB
- A*B+C/D

= *AB+C/D
= *AB+/CD
= +*AB/CD

### postfix

<operand><operand><operator>

- A*B+C/D

= AB*+C/D
= AB*+CD/
= AB*CD/+

## Infix to postfix (Without stack)

- $a+b*c+(d*e+f)*g$

$= a+b*c+(\underbrace{de*}_{x}+\underbrace{f}_{y})*g$

$= a+b*c+(de*f+)*g$

$= a+b*c+\underbrace{(de*f+)}_{x}*\underbrace{g}_{y}$

$= a+b*c+(de*f+g)*)$

$= ab$

- $a+b*c+(d*e+f)*g$

$= a+b*c+(de*+f)*g$

$= a+b*c+(de*f+)*g$

$= a+bc*+\underbrace{(de*f+)}_{x}*\underbrace{g}_{y}$

$= \underbrace{a}_{x}+\underbrace{(bc*)}_{y}+(de*f+g*)$

$= \underbrace{(abc*+)}_{x}+\underbrace{(de*f+g*)}_{y}$

$= (abc*+de*f+g*+$

- $A+B*C-D/E*H$

$= A+(BC*)-D/E*H$

$= A+(BC*)-\underbrace{(DE/)}_{x}*\underbrace{H}_{y}$

$= \underbrace{A}_{x}+\underbrace{(BC*)}_{y}-(DE/H*)$

$= \underbrace{(ABC*+)}_{x}-\underbrace{(DE/H*)}_{y}$

$= (ABC*+DE/H*-$

- Bracket থাকলে প্রথমে Bracket এর কাজ করতে হবে
- Precedence অনুসারে চিহ্নের ব্যবহার করতে হবে যার precedence বেশি তার কাজ আগে করতে হবে এবং যার precedence কম তার কাজ পরে করতে হবে।
- একই predence এর চিহ্নের ক্ষেত্রে Associativity follow করতে হবে। অর্থাৎ একটি eqn এ একাধিক * এবং / চিহ্ন থাকলে বামপাশ থেকে কাজ শুরু করতে হবে যার চিহ্ন আগে আসবে তার কাজ হবে।
- Postfix এর জন্য দুটি operand এর মাঝে অবস্থিত operator কে last এ নিতে হবে।
- $\underbrace{ab*c}_{x}+\underbrace{de/-}_{y}$ এক্ষেত্রে শুধু '+' operator এর কাজ বাকি। তাই এর ডান

ও বন্ধনীর সবকিছুকে operand হিসেবে ধরে ($x+y \rightarrow xy+$) এই আকারে প্রকাশ করতে হবে। [ab*cde/-+]

## Infix to prefix (without stack)

• prefix করার ক্ষেত্রে operator কে প্রথমে আনতে হবে। [$x+y \rightarrow +xy$] এরূপ হতে হবে।

• (A*B)/D+C*F

$= \underbrace{(*AB)}_{x}/\underbrace{D}_{y}+C*F$

$= (/*ABD)+C*F$

$= \underbrace{(/*ABD)}_{x}+\underbrace{(*CF)}_{y}$

$= +/*ABD*CF$

• (A+B)*(C-D) = (+AB)*(C-D)

$= \underbrace{(+AB)}_{x}*\underbrace{(-CD)}_{y}$

$= *+AB-CD$

• যদি একাধিক Bracket থাকে তবে Left থেকে Bracket এর কাজ শুরু করতে হবে।

## postfix to Infix (without stack)

• Left to right scan করতে হবে।

• পরপর দুটি operand এরপর operator থাকলে তাকে operand দুয়ের মাঝে বসাতে হবে। দুইয়ের অধিক operand এর পর operator থাকলে সেক্ষেত্রে operator কে সবচেয়ে নিকটবর্তী operand দুটির মাঝে বসাতে হবে। ($abcd* \rightarrow ab(c*d)$)

• $A\underbrace{B}_{x}\underbrace{C}_{y}*+DE/H*-$

$= \underbrace{A}_{x}\underbrace{(B*C)}_{y}+DE/H*-$

$= A+(B*C)\underbrace{D}_{x}\underbrace{E}_{y}/H*-$

$= A+(B*C)\underbrace{(D/E)}_{x}\underbrace{H}_{y}*-$

$= A+\underbrace{(B*C)}_{x}\underbrace{((D/E)*H)}_{y}-$

$= A+(B*C)-((D\backslash E)*H)$

• এক্ষেত্রে operator এর precedence এর ব্যবহার হবে না।

## Prefix to infix

- Right to left এ যেতে হবে।
- আর বাকি সব postfix to infix এর মতোই

- $+/*ABD*\underset{x}{C}\underset{y}{F}$

$= +/*\underset{x}{A}\underset{y}{B}D(C*F) \rightarrow$ '*' operator সবচেয়ে কাছে আছে A ও B

$= +/\underbrace{(A*B)}_{x}\underbrace{D}_{y}(C*F)$

$= +\underbrace{((A*B)/D)}_{x}\underbrace{(C*F)}_{y}$

$= ((A*B)/D)+(C*F)$

$= (A*B/D)+(C*F)$

## Infix to Postfix (using stack)

Procedure:

i) Two operators of the same priority cannot stay together in a stack. (previous operator will pop)

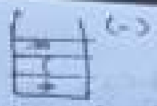
ii) Highest priority operation will not stay in the stack when lowest priority operation will be inserted.
(Highest priority operator will pop)

iii) (+, *) → when inside parenthesis pop all the two operators of the stack and place them in the postfix

iii) 


$(+,*) \rightarrow AB*+$

• A + (B * C − (D / E ^ F) * G) * H



(-) lowest priority so * will pop in postfix.

| Symbol | Stack | Postfix |
|---|---|---|
| A | | A |
| + | + | A |
| ( | +( | A |
| B | +( | AB |
| * | +(* | AB |
| C | +(* | ABC |
| - | +(- | ABC* |
| ( | +(-( | ABC* |
| D | +(-( | ABC*D |
| / | +(-(/ | ABC*D |
| E | +(-(/ | ABC*DE |
| ^ | +(-(/^ | ABC*DE |
| F | +(-(/^ | ABC*DEF |
| ) | +(-(/^) | ABC*DEF^/ |
| * | +(-* | ABC*DEF^/ |
| G | +(-* | ABC*DEF^/G |
| ) | +(-*) | ABC*DEF^/G*- |
| * | +* | ABC*DEF^/G*- |
| H | +* | ABC*DEF^/G*-H |
| | | ABC*DEF^/G*-H*+ |

# Infix to Prefix (Using stack)

i) Highest priority operator will not stay in the stack when lower priority operator is inserted.

ii) (+, *) → pop all the operation of the stack and place them in the prefix.

- A * B ^ C - D + E / F / (G + H)
- Reverse : (H + G) / F / E + D - C ^ B * A

| Symbol | Stack | Prefix |
|---|---|---|
| ( | ( | |
| H | ( | H |
| + | (+ | H |
| G | (+ | HG |
| ) | (+) | HG+ |
| / | / | HG+ |
| F | / | HG+F |
| / | // | HG+F |
| E | // | HG+FE |
| + | ~~//~~+ | HG+FE// |
| D | + | HG+FE//D |
| - | +- | HG+FE//D |
| C | +- | HG+FE//DC |
| ^ | +-^ | HG+FE//DC |
| B | +-^ | HG+FE//DCB |
| * | +-* | HG+FE//DCB^ |
| A | +-* | HG+FE//DCB^A |

HG+FE//DCB^A*-+

∴ prefix : +-*A^BCD//EF+GH

## Evaluation of Postfix Expression

- Left to right scan.
- operand পেলে তা stack এ push হবে আর operator পেলে তা Top two element এর মাঝে বসিয়ে calculation করে যে value পাওয়া যাবে তা stack এ push হবে।

- 6 2 3 + - 3 8 2 / + * 2 ^ 3 +

| Symbol | Stack |
| --- | --- |
| 6 | 6 |
| 2 | 6, 2 |
| 3 | 6, 2, 3 |
| + | 6, 5  $2+3=5$ |
| - | 1  $6-5=1$ |
| 3 | 1, 3 |
| 8 | 1, 3, 8 |
| 2 | 1, 3, 8, 2 |
| / | 1, 3, 4  $8/2=4$ |
| + | 1, 7  $3+4=7$ |
| * | 7  $1*7=7$ |
| 2 | 7, 2 |
| ^ | 49  $7^2=49$ |
| 3 | 49, 3 |
| + | 52  $49+3=52$ |

## Evaluation to Prefix Expression

i) • Right to left scan হবে

• − + 2 * 9 ^ 2 − 8 / * 3 4 2 1 ←

ii)

| Symbol | Stack | |
|---|---|---|
| 1 | 1 | |
| 2 | 1 2 | |
| 4 | 1 2 4 | |
| 3 | 1 2 4 3 | |
| * | 1 2 12 | 4*3=12 |
| / | 1 6 | 12/2=6 |
| 8 | 1 6 8 | |
| − | 1 2 | 8−6=2 |
| 2 | 1 2 2 | |
| ^ | 1 4 | 2^2=4 |
| 9 | 1 4 9 | |
| * | 1 36 | 4*9=36 |
| 2 | 1 36 2 | |
| + | 1 38 | 36+2=38 |
| − | 37 | 38−1=37 |

# Graph

Graph: A graph is an abstract data-type that consists of a set of objects that are connected to each other via links. These objects are called vertices and the links are called edges. A graph is represented as : G = {V, E} G = Graph space, V = set of vertices, E = set of Edges. If E is empty, the graph is known as Forest.

Vertex: Each node of a graph is represented as a vortex.

Edge: Edge represents a path or line between two vertices.

Adjacency: Two nodes are adjacent if they are connected to each other through vertices.

Edge from node A to node B [A is the initial node and B is the terminal node].

Path: Path can be defined as the sequence of nodes that are followed in order to reach some terminal node V from the initial node U.

Directed and Undirected Graph: A directed graph (digraph) is a graph in which the edges have a direction.

Undirected graph have edges that do not have a direction.

Closed Path: A path will be called closed path if the initial node is same as terminal node.

Graph Representation:

1. Adjacency matrix :

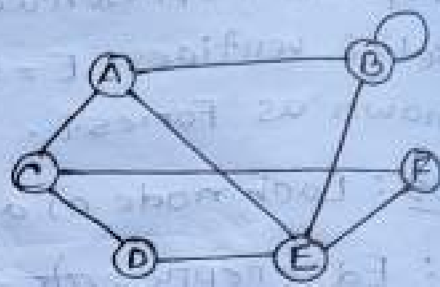
## Undirected Graph

i) no. of vertices বের করতে হবে।

ii) square matrix represent করে।

iii) adj[i,j] = 1 (loop / i,j adjacent হয়)

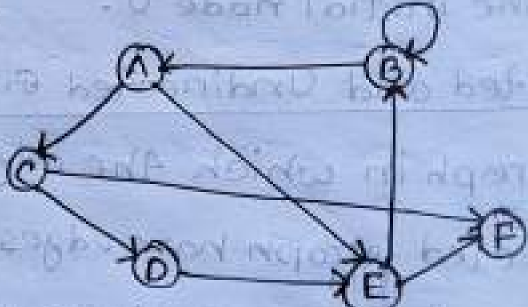
iv) adj[i,j] = 0 (i,j adjacent না হলে)




### Adjacency matrix representation:

$$
\begin{array}{c c}
 & \begin{array}{cccccc} A & B & C & D & E & F \end{array} \\
\begin{array}{c} A \\ B \\ C \\ D \\ E \\ F \end{array} &
\left[\begin{array}{cccccc}
0 & 1 & 1 & 0 & 1 & 0 \\
1 & 1 & 0 & 0 & 1 & 0 \\
1 & 0 & 0 & 1 & 0 & 1 \\
0 & 0 & 1 & 0 & 1 & 0 \\
1 & 1 & 0 & 1 & 0 & 1 \\
0 & 0 & 1 & 0 & 1 & 0
\end{array}\right]
\end{array}
$$

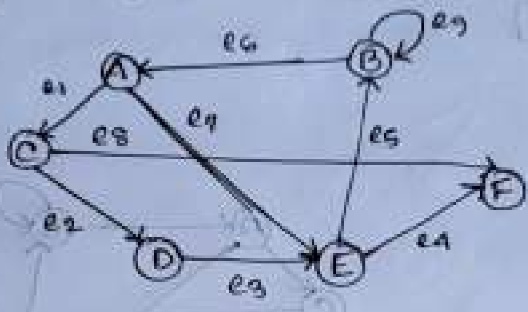
## Directed Graph: [Rules are same but now observe the direction of edges]




$$
\begin{array}{c c}
 & \begin{array}{cccccc} A & B & C & D & E & F \end{array} \\
\begin{array}{c} A \\ B \\ C \\ D \\ E \\ F \end{array} &
\left[\begin{array}{cccccc}
0 & 0 & 1 & 0 & 1 & 0 \\
1 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 1 \\
0 & 0 & 0 & 0 & 1 & 0 \\
0 & 1 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0
\end{array}\right]
\end{array}
$$

ii) Incidence Matrix:

1. No. of vertices and edges বের করতে হবে,
2. adj [i, j] = 1 (i → j outgoing)
3. adj. [i, j] = 0 (no connection)
4. adj. [i, j] = -1 (i ← j incoming)

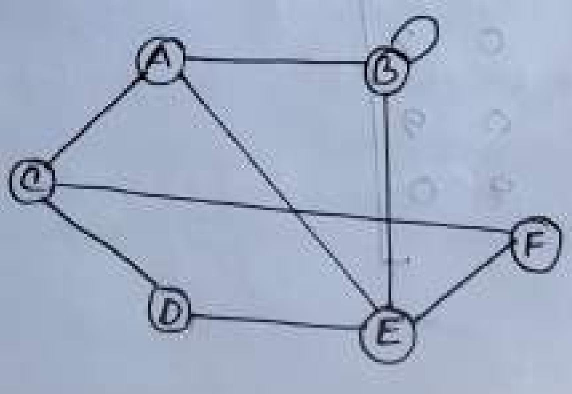



$$
\begin{array}{c|c}
 & \begin{matrix} e_1 & e_2 & e_3 & e_4 & e_5 & e_6 & e_7 & e_8 & e_9 \end{matrix} \\
\begin{matrix} A \\ B \\ C \\ D \\ E \\ F \end{matrix} &
\begin{bmatrix}
1 & 0 & 0 & 0 & 0 & -1 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & -1 & 1 & 0 & 0 & 1 \\
-1 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & -1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & -1 & 1 & 1 & 0 & -1 & 0 & 0 \\
0 & 0 & 0 & -1 & 0 & 0 & 0 & -1 & 0
\end{bmatrix}
\end{array}
$$

iii) Adjacency List:

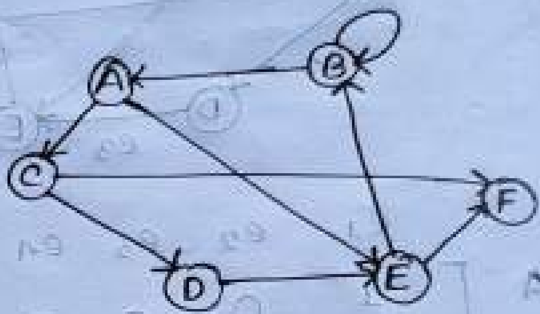
i) Find the nodes
ii) List down the adjacent nodes to each nodes.

Undirected graph:



A → B → C → E ×
B → A → B → E ×
C → A → D → F ×
D → C → E ×
E → A → B → F ×
F → C → E

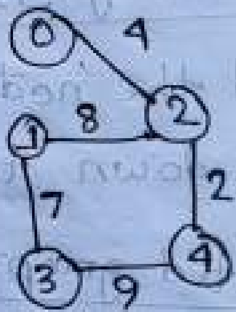
## Directed Graph:




A → C → E ×
B → A → B ×
C → D → F ×
D → E ×
E → B → F ×
F ×

iv) cost adjacency matrix:

1) Aij = cost for an edge between i and j, 0 otherwise.
2) If the cost can be 0:
Aij = cost for an edge between i and j, -1, otherwise.




$$\begin{array}{c|c} & \begin{matrix} 0 & 1 & 2 & 3 & 4 \end{matrix} \\ \begin{matrix} 0 \\ 1 \\ 2 \\ 3 \\ 4 \end{matrix} & \begin{bmatrix} 0 & 0 & 4 & 0 & 0 \\ 0 & 0 & 8 & 7 & 0 \\ 4 & 8 & 0 & 0 & 2 \\ 0 & 7 & 0 & 0 & 9 \\ 0 & 0 & 2 & 9 & 0 \end{bmatrix} \end{array}$$

## BFS (Breadth First Search)

i) Queue ব্যবহৃত হয়।

ii) যেই vertex নিয়ে কাজ করবো তার adjacent vertices queue তে insert করবো।

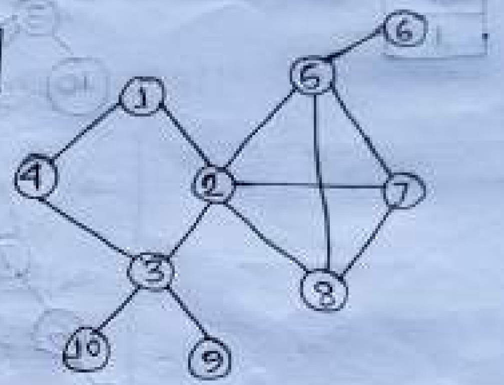
iii) Visited vertices নতুন করে insert হবে না। (Queue তে)

iv) Inserted vertices repeat হবে না / insert হবে না

Queue:

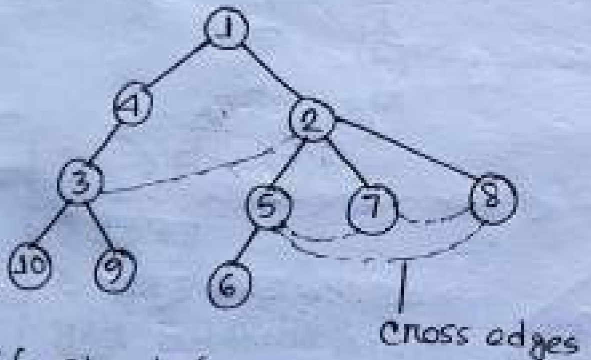
| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| ~~1~~ | ~~4~~ | ~~2~~ | ~~3~~ | ~~5~~ | ~~8~~ | ~~7~~ | ~~10~~ | ~~9~~ | ~~6~~ |

Results: 1, 4, 2, 3, 5, 8, 7, 10, 9, 6




BFS spanning tree:

Starting from 1




- We can start from any vertex and can visit the adjacent vertex in any order.
- In queue after completely visited a vertex cut down it.

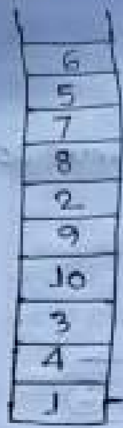
If start from 5:

5, 2, 7, 8, 6, 1, 3, 4, 10, 9

## DFS (Depth First Search)

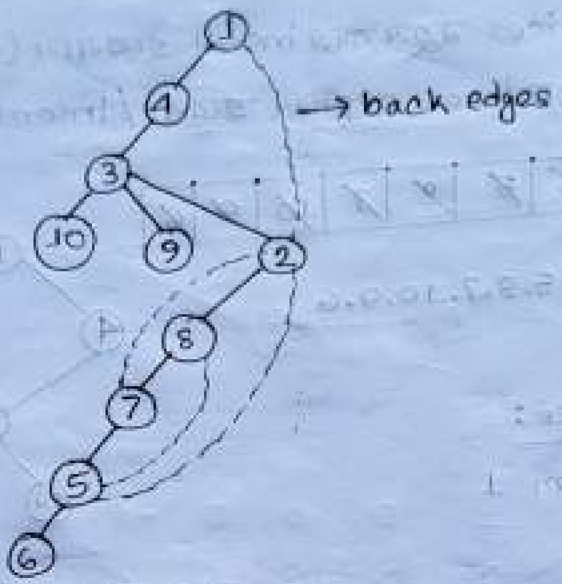
i) Stack use হবে।

ii) প্রথমে একটি node থেকে শুরু করতে হবে। যদি তার একাধিক adjacent vertex থাকে তবে যে কোন একটি নিতে হবে এবং তার adjacent দেখতে হবে এভাবে শেষ প্রান্তে পৌঁছালে আবার আগের node এ ফেরত আসতে হবে এবং যেসব adjacent nodes visit করা হয় নি তা একই উপায়ে visit করতে হবে।




Result: 1, 4, 3, 10, 9, 2, 8, 7, 5, 6

Spanning Tree:




## Classification of edge is DFS

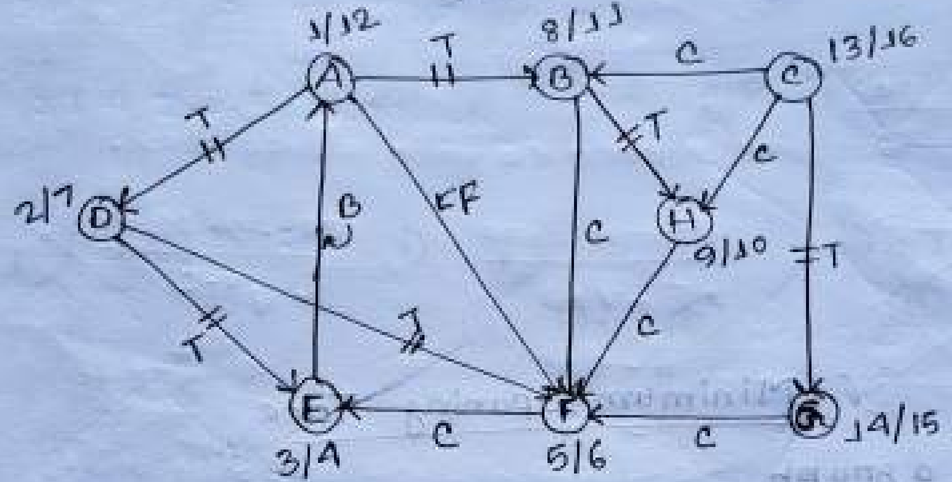
**Tree edge:** DFS apply করার পর যে edge আসবে।

**Back edge:** E(x,y) [x node to y node] যেখানে starting time y < x হবে (x node এর time বেশি হবে) এবং y-to x path থাকবে but path অবশ্যই tree edge এ থাকবে।

**Forward ege:** E(x,y) যেখানে y > x এবং x to y path থাকবে।

Cross edge: $E(x,y)$ কোন path থাকবে না। অর্থাৎ বাকি ৩ টি condition fulfill না হলে তা cross edge.

Time: First node এ start time 1 হবে পরবর্তীতে 1 করে বাড়বে এবং যে কোন node যদি আর অন্য কোনো node এ না যেতে পারে তবে তার end-time লিখতে হবে। end-time হবে তার start time এর সাথে 1 যোগ করে




B → F

B = x = 8

F = y = 5

y < x but there is no (tree edge) path from F to B.

So cross edge (C).

A → F

A = x = 1

F = y = 5

y > x

∴ A → F

∴ Forward Edge (F)

E → A

E = x = 3

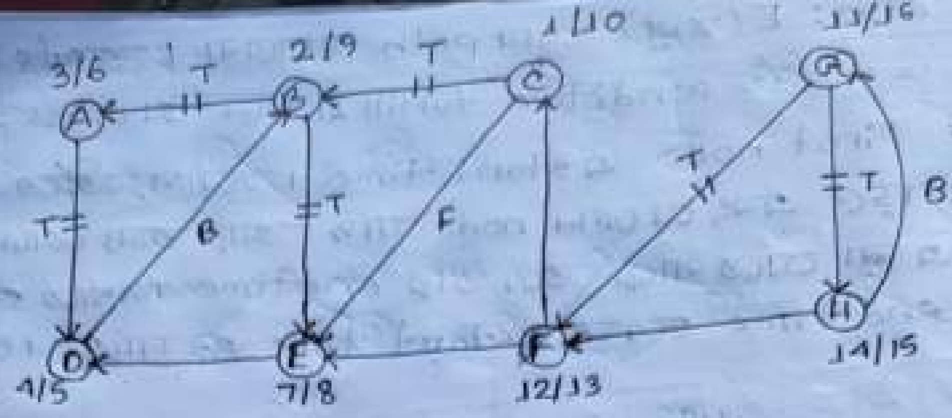
A = y = 1

y < x

and there is a tree edge path from A → E (A → D → E)

and then (E → A)

So Backward Edge (B)




## Minimum Spaning Tree

⇒ i) subset of a graph

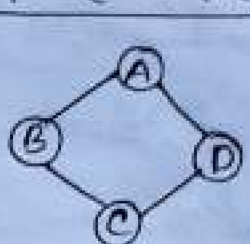
ii) cycle হবে না (অর্থাৎ edges দ্বারা graph divided হবে না)

iii) vertices disconnected হবে না


→ এখানে 2 টি cycle.

iv) নতুন কোন edge add/delete হবে না অর্থাৎ spanning tree তৈরির সময় যে edge যেখানে ছিল সেখানেই থাকবে।

### Graph (without cycle)




$G(V, E) \rightarrow$ Main set

$S(V', E') \rightarrow$ Subset

$V' = V \rightarrow$ no. of vertices

$E' = |V| - 1 \rightarrow$ no of edges.

In this graph $\rightarrow V' = 4$

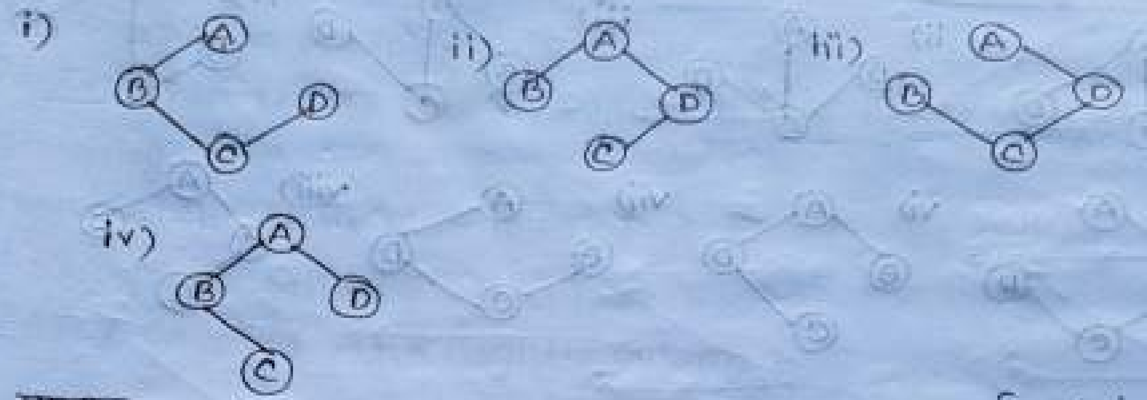
$E' = 4 - 1 = 3$

### No of spanning tree

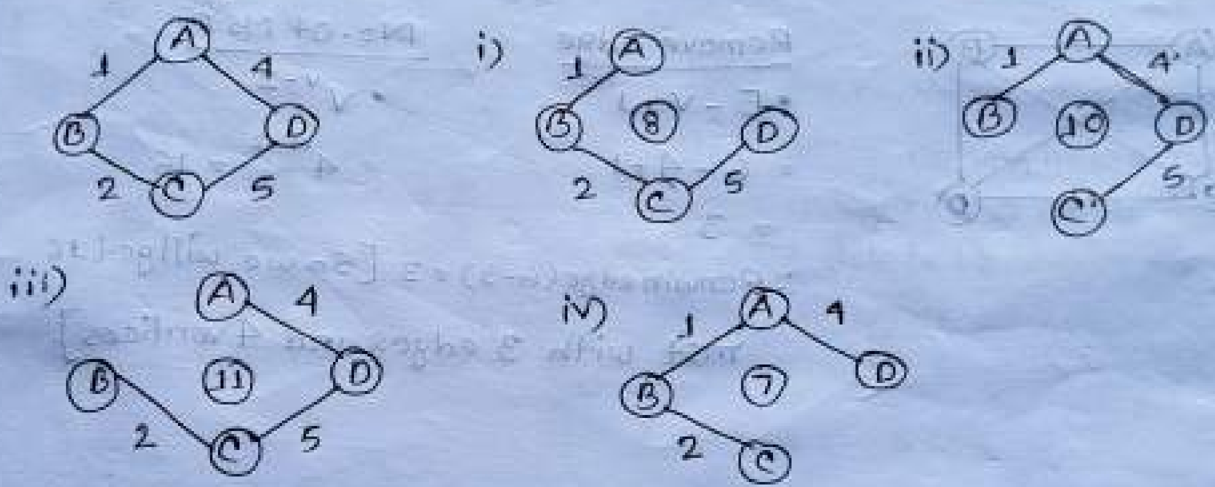
$^{E}C_{E'}$

$\Rightarrow {}^{4}C_{3} \Rightarrow 4$

• 4 spanning trees with 4 vertices and 3 edges:

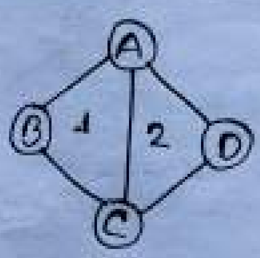



যেহেতু graph weighted নয় তাই উপরোক্ত সবগুলোই minimum spanning tree.




In weighted graph graph with minimum cost is the minimum spanning tree.

Graph (With cycle)




$E' = |V| - 1$

$= 3$

$E = 5$

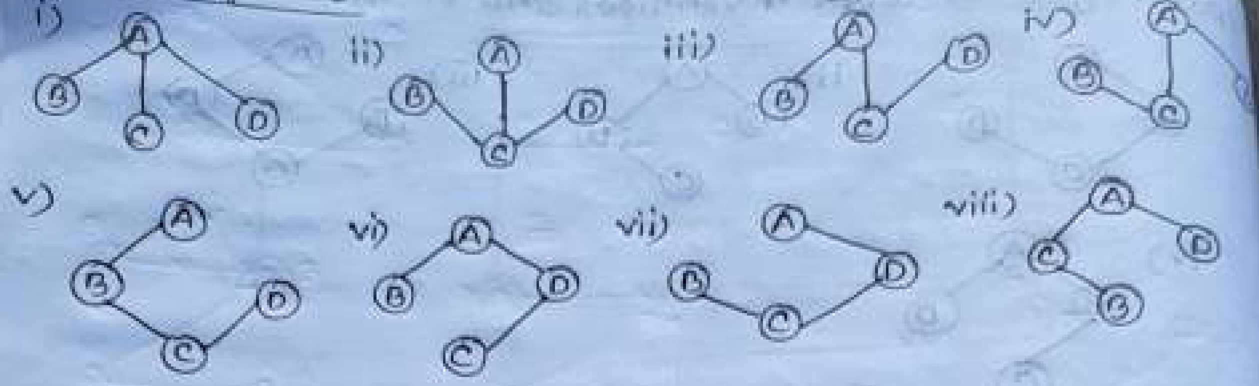
No of spanning tree

$= ({}^{E}C_{E'}$ - No of cycle$)$

$= ({}^{5}C_{3} - 2) = (10 - 2)$

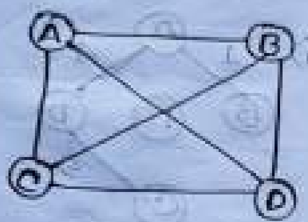
$= 8$

8 spanning trees:




Complete Graph:

Where all vertices are connected with all vertices.




Remove edge

• $E - V + 1$

$= 6 - 4 + 1$

$= 3$

No. of MST

• $V^{V-2}$

$= 4^{4-2} = 16$

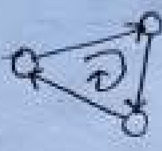
$\therefore$ Remain edge $(6-3) = 3$ [So we will get 16 mst with 3 edges and 4 vertices]

## Topological sorting

It is a linear ordering of its vertices such that for every ordered edge UV for vertex u to v, u comes before vertex v in the ordering.

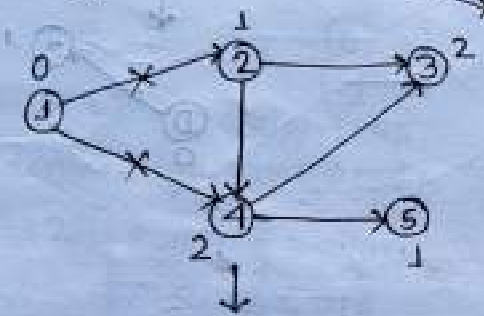
- Graph should be Directed and Acyclic (means without cycle).
- Every DAG will have atleast one topological sorting.

DAG = Directed And Acyclic Graph.



→ cycle

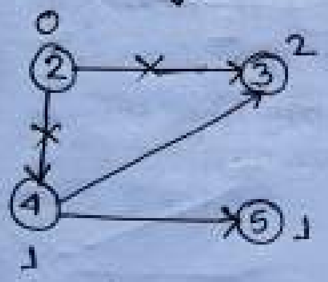
→ no cycle [cycle will be formed according to direction]

- 

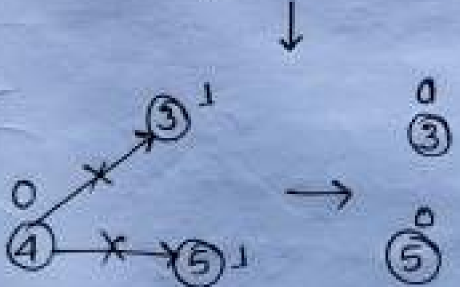
- find the number of indegree [num of edges coming in to a nodes]
- whose indegree is 0 write down it and delete its edges and again count indegree and repeat the same process.

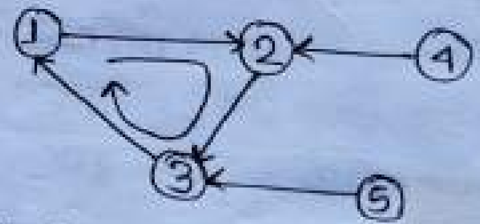
Sorting: (i) 1 → 2 → 4 → 3 → 5

(ii) 1 → 2 → 4 → 5 → 3

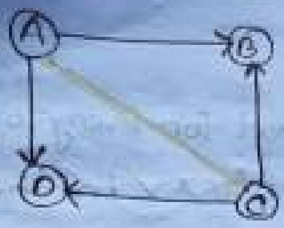


- For this graph there are two topological sorting available.



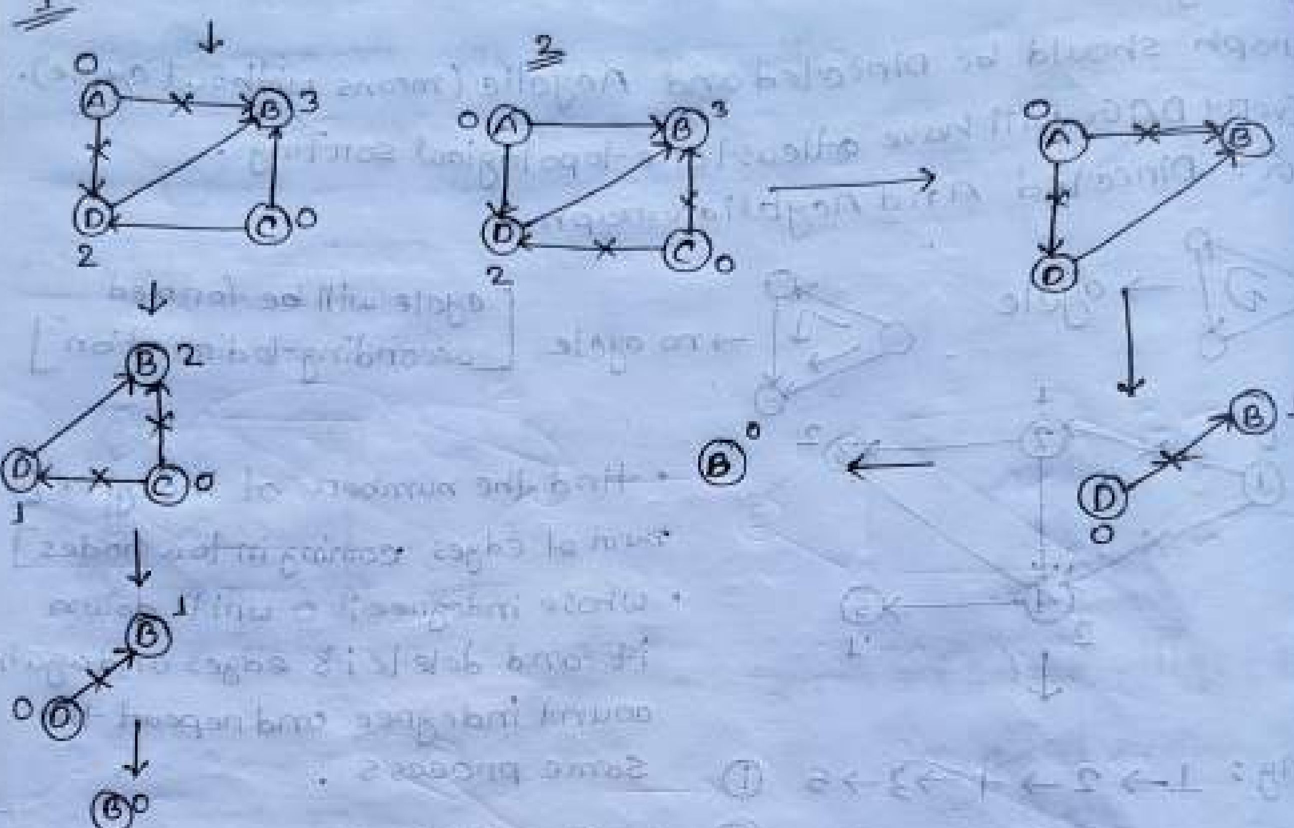
- 

As there is a cycle in this graph so topological sorting is not possible.



Case 1: A C D B

Case 2: C A D B

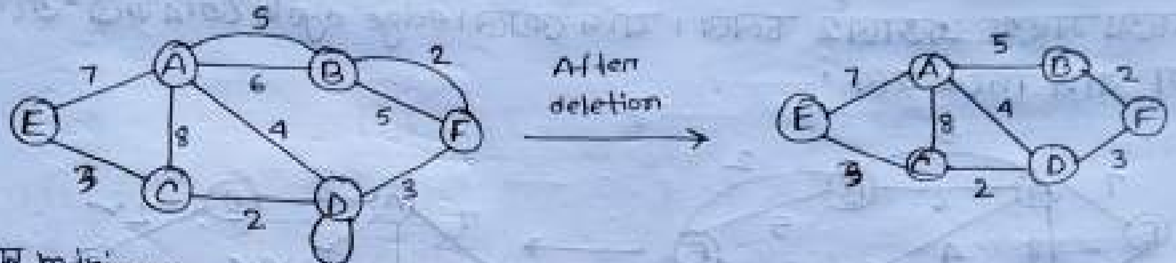


## Kruskal's Alogorithm

i) Graph থেকে loop delete করতে হবে।

ii) Parallel edge delete করতে হবে অর্থাৎ যে edge এর weight বেশি তা বাদ দিতে হবে। (  )

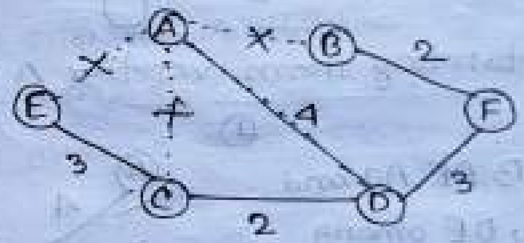

iii) নতুন MST তৈরির সময় কোনো cycle হওয়া যাবে না। যেসকল edge cycle তৈরি করবে তাদের বাদ দিতে হবে। MST এ (V', E') এর সংখ্যা ($V' = V$, $E' = |V|-1$).




এখন minimum weight এর order এ edge সাজাতে হবে:

BF → 2
CD → 2
DF → 3
CE → 3
AD → 4
AB → 5
AE → 7
AC → 8

Sorting weight অনুসারে একটি Graph বানাবো যেখানে কোনো cycle হবে না। [Graph টি সুবিধার জন্য পূর্বের Graph এর মতো করেই বানানো হবে]

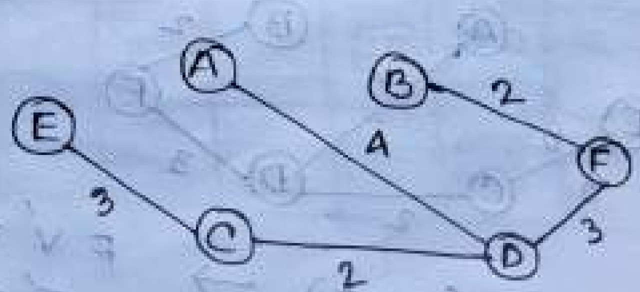



Graph টিতে AE, AC, AB এই তিনটি edge বসানো হয়নি কারণ এরা cycle তৈরি করে।

Graph এ → $V' = 6$

→ $E' = 5$ [V-1]

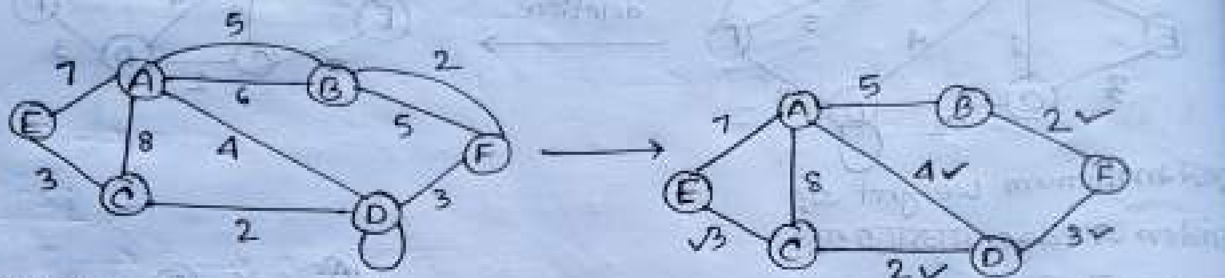
যা কিনা MST এর condition Fulfill করে। তাই এটি হলো MST. এর weight minimum.




Weight → 14

# Prim's Algorithm

যেকোনো একটি node থেকে শুরু করতে হবে। ঐ node এর adjacent edge গুলোর মধ্যে যার weight minimum সেই edge এবং তার সাথের node print করতে হবে। এবার ২টি node এর ই adjacent edge দেখতে হবে এবং দুটির মধ্যে যেটির weight minimum তা draw করতে হবে। আর যদি দুটি minimum weight পাওয়া যায় তবে যে কোন একটি draw করতে হবে। প্রত্যেকবার node সংখ্যা বাড়ার সাথে সাথে এভাবেই চলবে। যদি কোনো edge cycle তৈরি করে তবে তা বাদ দিতে হবে।



Starting from vertige A (Among AB, AE, AC, AD; AD is minimum)

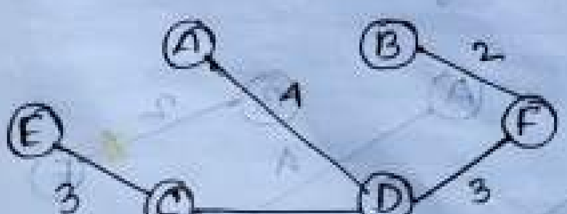
ii) (AB, AE, AC and DC, DE among them DC is minimum)

iii) (Among the edges at A, D, C → CE and DF minimum so we can choose anyone of them)

① ②

④ ③

⑤



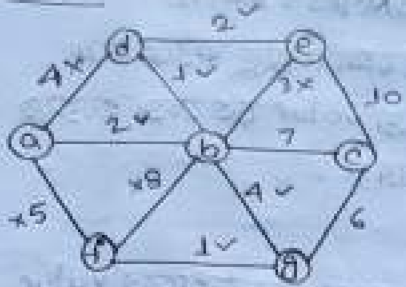
[ But after that we can't any edge among AB, AE or AC because each of them make cycle ]

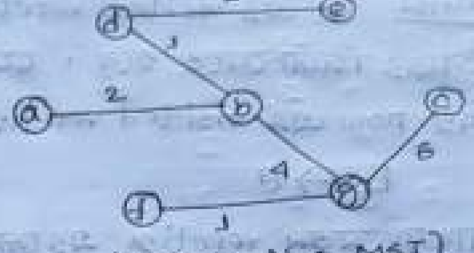
(MST) → $V' = 6 = V$

$E' = 5$ [$|V|-1$]

• Weight = 14

Problem 1: (start from a)





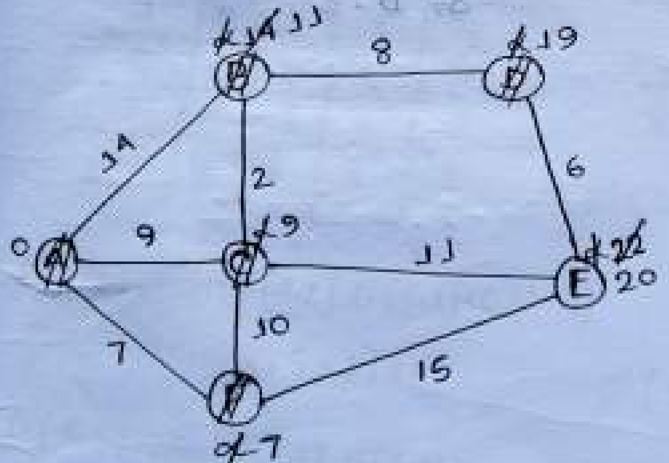

## Dijkstra's Algorithm (single source shortest path)

- u (node) to v (node)
- If $(d(u) + c(u,v) < d(v))$
  then $d(v) = d(u) + c(u,v)$

$d(u)$ = distance of u

$c(u,v)$ = cost of u to v

- At first starting node কে 0 এবং বাকি সবাই α করে দিতে হবে। এরপর এক এক করে সব node এর জন্য কাজ করতে হবে এবং যদি উপরের শর্ত fulfill করে তবে value change হবে নাহলে same থাকবে।



visited vertices (যার value সবচেয়ে ছোট সে হবে)

List of vertices

| | A | B | C | D | E | F |
|---|---|---|---|---|---|---|
| A | $\boxed{0}$ | α | α | α | α | α |
| F | | 14 | 9 | α | α | $\boxed{7}$ |
| C | | 14 | $\boxed{9}$ | α | 22 | |
| B | | $\boxed{11}$ | | α | 20 | |
| D | | | | $\boxed{19}$ | 20 | |
| E | | | | | $\boxed{20}$ | |

প্রথমে A এর সব adjacent এ দেখতে হবে এবং value chang হলে change করতে হবে। প্রাপ্ত value গুলোর মধ্যে যার value সবচেয়ে কম সেই vertix কে visited vertix এ নিতে হবে এবং তার adjacent এ দেখতে হবে। এভাবেই প্রক্রিয়াটি চলবে।

• Question : যদি বলা হয় A to E shortest path.

Ans: E এর visited vertices এর value থেকে শুরু করতে হবে উপরের দিকে যেতে হবে। যে row তে গিয়ে value টে চেঞ্জ হয়েছে সেই row এর visited vertix কে নিতে হবে।

$E \rightarrow C$

আবার নতুন vertix এর ক্ষেত্রেও আগের মতো করে উপরের value check করতে হবে যতক্ষণ না নতুন vertex আসবে।

$E \rightarrow C \rightarrow A$

$\therefore$ Now $A \rightarrow C \rightarrow E$

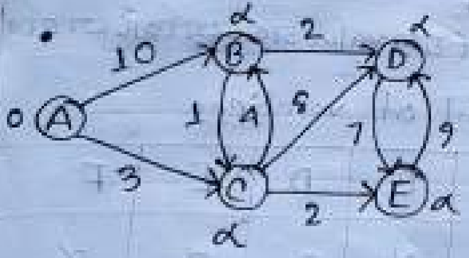
$= 9 + 11 = 20 \rightarrow$ Equals to the value visited vertix of E.

• $A \rightarrow D$

Ans:

$D \rightarrow B \rightarrow C \rightarrow A$

$\therefore A \rightarrow C \rightarrow B \rightarrow D = 9+2+8 = 19 \rightarrow$ Equals to the value of visited vertix of D.

• 


| | A | B | C | D | E |
|---|---|---|---|---|---|
| A | 0 | $\alpha$ | $\alpha$ | $\alpha$ | $\alpha$ |
| C | | 10 | 3 | $\alpha$ | $\alpha$ |
| E | | 7 | | 11 | 5 |
| B | | 7 | | 11 | |
| D | | | | 9 | |

# Floyd Warshall Algorithm
## (All pairs shortest path)

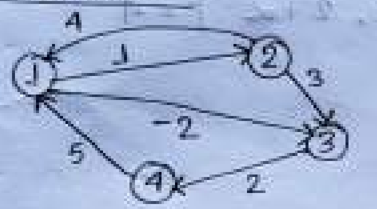
- Works with positive and negative edges (but not with negative cycle)


value of cycle = -2 - 2 + 1 = -3 [so algorithm is not appliable here]

**Formula:**

$$D^{k}[i,j] = \min\{D^{k-1}[i,j],\ D^{k-1}[i,k] + D^{k-1}[k,j]\}$$

**Problem:**




$D^0$

| | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| 1 | 0 | 1 | -2 | $\alpha$ |
| 2 | 4 | 0 | 3 | $\alpha$ |
| 3 | $\alpha$ | $\alpha$ | 0 | 2 |
| 4 | 5 | $\alpha$ | $\alpha$ | 0 |

$D^1$

| | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| 1 | 0 | 1 | -2 | $\alpha$ |
| 2 | 4 | 0 | 2 | $\alpha$ |
| 3 | $\alpha$ | $\alpha$ | 0 | 2 |
| 4 | 5 | 6 | 3 | 0 |

যেহেতু node ① নিয়ে কাজ করছি তাই 1 এর row এবং coloumn $D^0$ এর অনুরূপ হবে।

$k = 1 \rightarrow$ for first node

$$D^1[2,3] = \min\{D^0[2,3],\ D^0[2,1] + D^0[1,3]\}$$
$$= \min\{3,\ 4+(-2)\}$$
$$= \min\{3, 2\} = 2$$

$$D^1[2,4] = \min\{D^0[2,4],\ D^0[2,1] + D^0[1,4]\}$$
$$= \min\{\alpha,\ 4+\alpha\} = \alpha$$

$$D^1[3,2] = \min\{D^0[3,2],\ D^0[3,1] + D^0[1,2]\}$$
$$= \min\{\alpha,\ \alpha+1\} = \alpha$$

$$D^1[3,4] = \min\{D^0[3,4],\ D^0[3,1] + D^0[1,4]\}$$
$$= \min\{2,\ \alpha+\alpha\} = 2$$

$$D^1[4,2] = \min\{D^0[4,2],\ D^0[4,1] + D^0[1,2]\}$$
$$= \min\{\alpha,\ 5+1\} = 6$$

$$D^1[4,3] = \min\{D^0[4,3],\ D^0[4,1] + D^0[1,3]\}$$
$$= \min\{\alpha,\ 5+(-2)\}$$
$$= 3$$

$\underline{D^2}$ When $k=2$ for 2nd node [row and column 2 from $D^1$]

| | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| 1 | 0 | 1 | -2 | $\alpha$ |
| 2 | 4 | 0 | 2 | $\alpha$ |
| 3 | $\alpha$ | $\alpha$ | 0 | 2 |
| 4 | 5 | 6 | 3 | 0 |

$D^2[1,3] = \min\{D^1[1,3], D^1[1,2] + D^1[2,3]\}$
$= \min\{-2, 1+3\} = -2$

$D^2[1,4] = \min\{D^1[1,4], D^1[1,2] + D^1[2,4]\}$
$= \min\{\alpha, 1+\alpha\} = \alpha$

$\underline{D^3}$

| | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| 1 | 0 | 1 | -2 | 0 |
| 2 | 4 | 0 | 2 | 4 |
| 3 | $\alpha$ | $\alpha$ | 0 | 2 |
| 4 | 5 | 6 | 3 | 0 |

$D^3[2,1] = \min\{D^2[2,1], D^2[2,3]+D^2[3,1]\}$
$= \min\{4, 2+\alpha\} = 4$

$D^3[2,4] = \min\{D^2[2,4]+D^2[2,3]+D^2[3,4]\}$
$= \min\{\alpha, 2+2\} = 4$

$\underline{D^4}$

| | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| 1 | 0 | 1 | -2 | 0 |
| 2 | 4 | 0 | 2 | 4 |
| 3 | 7 | 8 | 0 | 2 |
| 4 | 5 | 6 | 3 | 0 |

$D^4[1,2] = \min\{D^3[1,2], D^3[1,4] + D^3[4,2]\}$
$= \min\{1, 0+6\} = 1$

$D^4[1,3] = \min\{D^3[1,3], D^3[1,4] + D^3[4,3]\}$
$= \min\{-2, 0+3\} = -2$