Date 21/01/18

* What is linked list?

A linked list is a set of dynamically allocated nodes, arranged in such a way that each node contains one value ~~and~~ or more value and one or more pointer. pointer points to the next member. pointer NULL means it is the last node.

* Advantages

1. Dynamically allocates so when required it allocates.
2. insertion, Deletion easy.
3. stacks and Queues implementation are easy.
4. Item ~~can be~~ be added or removed from the middle of the list.
5. There ~~are~~ is no need to define an initial size for a linked list.
6.

Disadvantage

* linked list use more memory than arrays because of the storage used by their pointers.
* Element access is not so easy if we compare it with array. we can easily access any of elements in array.
* Reverse Traversing is difficult in linked list.

Application of linked list:

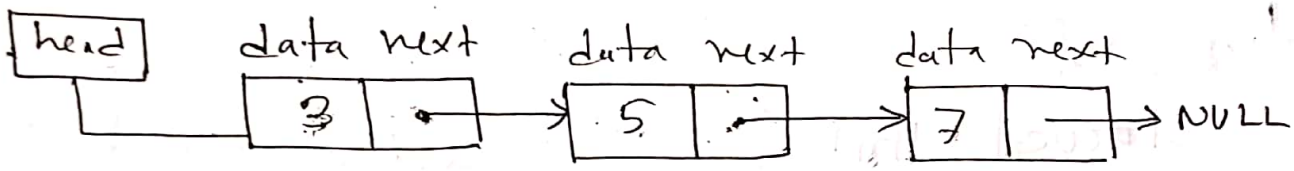
* implement stacks, queues, graphs, etc.
* undo functionality.

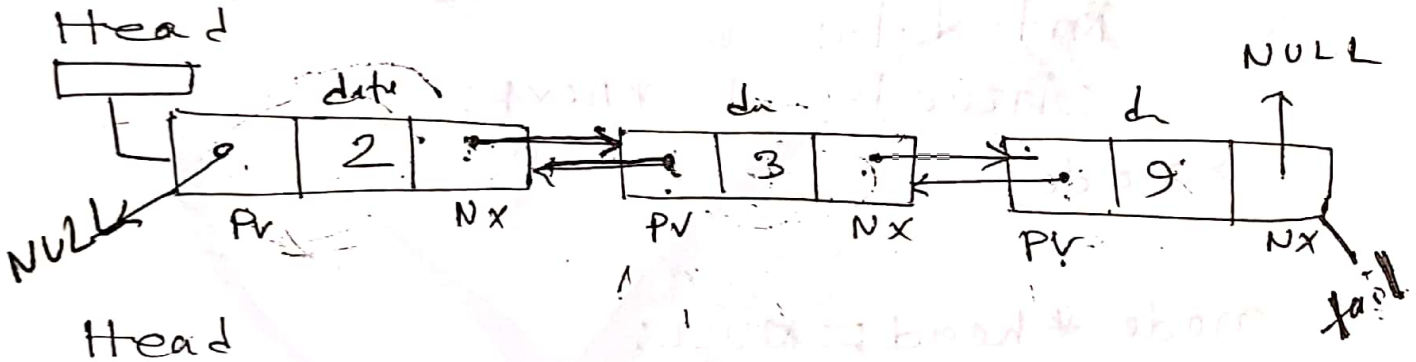
*** link list is a linear data structure.

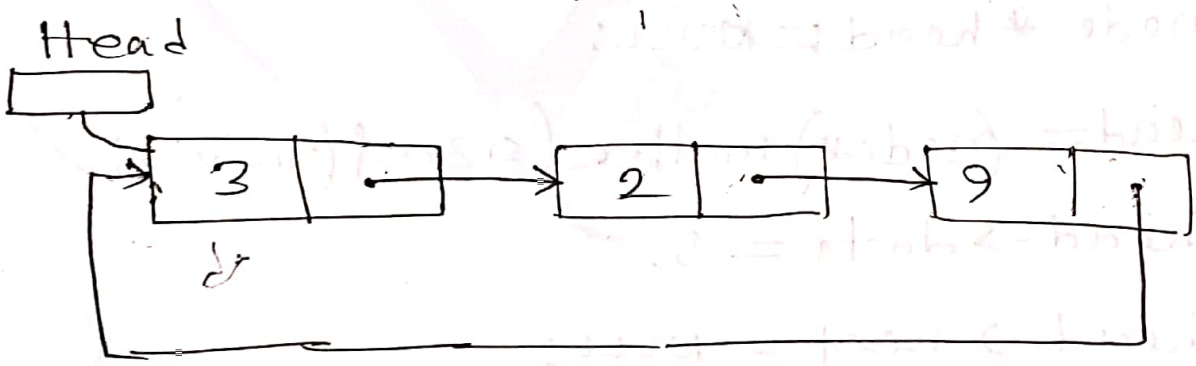
Date

Types of linked list:

1) Singly Linked list ✓
2) Doubly Linked list ✓
3) Circular linked list ✓







* linked list

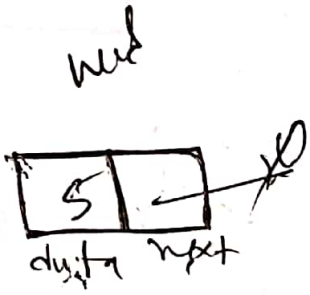
- insert *
- search *
- count/sum
- Delete *

```
typedef
  struct link {
      int data;
      struct link *next;
  } node;

node *head;
head = (node*) malloc(sizeof(node));
head -> data = 5;
head -> next = NULL;
```

self referential structure

when we write malloc it give blank node. then we gives it value and address.



Date

Data insert

* insert end
* insert beginning
* insert any position

```
* void insert_at_end (int data){
    node *newnode;
    newnode = (node*) malloc (sizeof (node));
    newnode -> data = data;
    newnode -> next = NULL;
    if (head == NULL){
        head = newnode;
    }
    else {
        node *current = head;
        While (current -> next != NULL){
            current = current -> next;
        }
        current -> next = newnode;
    }
```

```
* void insert_at_beginning(int data){
    node *newnode;
    newnode= (node*)malloc(sizeof(node));
    newnode->data = data;
    newnode->next = head;
    head = newnode;
}
```

```
*void insert_at_any_position (int data){
    int count= 1, position= 3;
    node *newnode;
    newnode = (node*)malloc (sizeof(node));
    newnode-> data = data;
    newnode->next = NULL;
    if (position == 1){
        newnode->next = head;
        head = newnode;
    }
```

Date

```
else{
    node *current = head;
    while(count < position-1){
        current = current->next;
        count++;
    }
    newnode->next = current->next;
    current->next = newnode;
}
```

```
void print(){
    node *current = head;
    while(current != NULL){
        // Printf(current->data);
        current = current->next;
    }
}

    Printf("%d", current->data);
```

```
* void deleteData( ) {
    int key = 5; // scanf();
    node *current = head;
    node *previous;
    if (head -> data == key) {
        head = head -> next;
        free(current);
    }
    else {
        while (current != NULL &&
                current -> data != key) {
            Previous = current;
            current = current -> next;
        }
        if (current != NULL ~~current -> data == key~~) {
            Previous -> next = current -> next;
            free (current);
        }
        else { Print(data not found); }
    }
}
```

Date.....

```
* void delete_position (){
    node *current = head;
    node *previous;
    int count=1, position = 3;
  if (position == 1){
      head = head->next;
      free(current);
  }
  else{
      node *current = head;
      while (count < position){
          Previous = current;
          current = current->next;
          count++;
      }
    Previous->next = current->next;
    free(current);
  }
}
```

Abetis Abecab

```
*void update(int data){
    node *current = head;
    int count=1, pos= 4;
    While (count < pos) {
        current= current ->next;
        count++;
    }
    ~~current = current -> data;~~
    current -> data = data;
}
```

(20)

```
*void search (int data) {
    node *current = head; int f=0;
    While (current != NULL) {
        if (current -> data == data){
            // found // break// f=1;
        }   f=1; break;
        else {
            current = current->next;
        }
    }
    (f==0) ? Print (Not found);
}
```

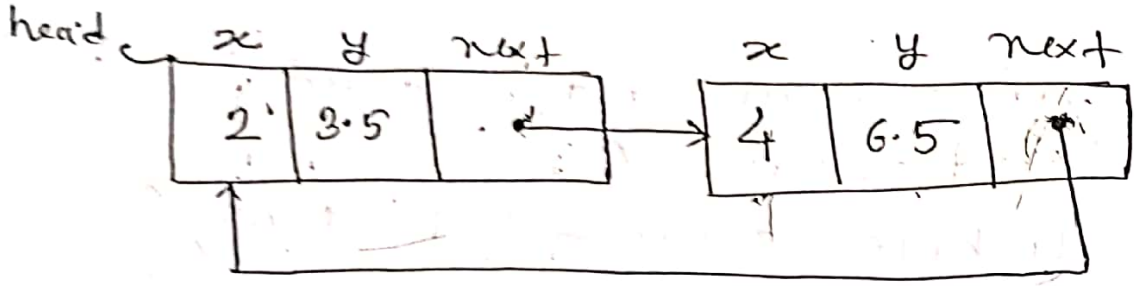
Date

```
* void sum(){
    node* current = head;
    int sum = 0;
    while(current != NULL){
        sum += current->data;
        current = current->next;
    }
    Print (sum);
}

* int/void count(){
    ----------
    int count = 0;
    ----------{
        count ++;
    ----------
    }
    // Print (count);
    return count;
}
```

linked list to code

(1)

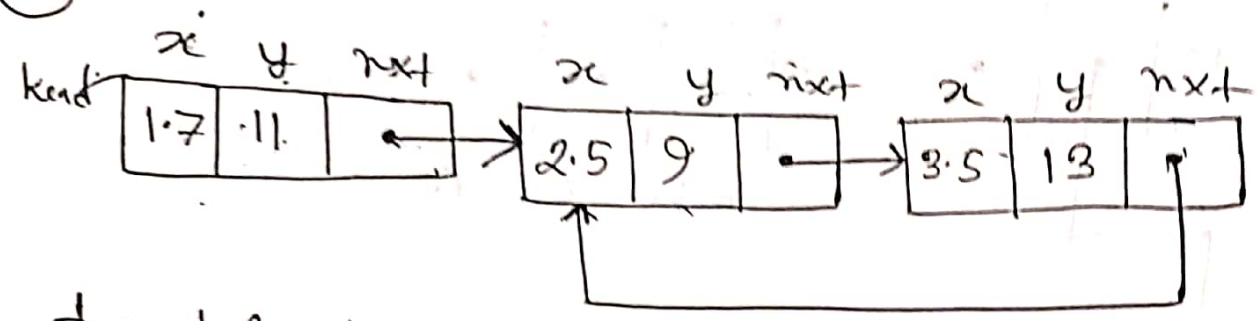



```
~~Stru~~ typedf struct Node{
        int ~~data1~~ x;
        float ~~data2~~ y;
        Struct Node *next;
}node;

node *head;
head=(node*)malloc(sizeof(node));
head->x=2; head->y=3.5;
head->next=(node*)malloc(sizeof(node));
head->next->x=4;
head->next->y=6.5;
head->next->next=head;
```

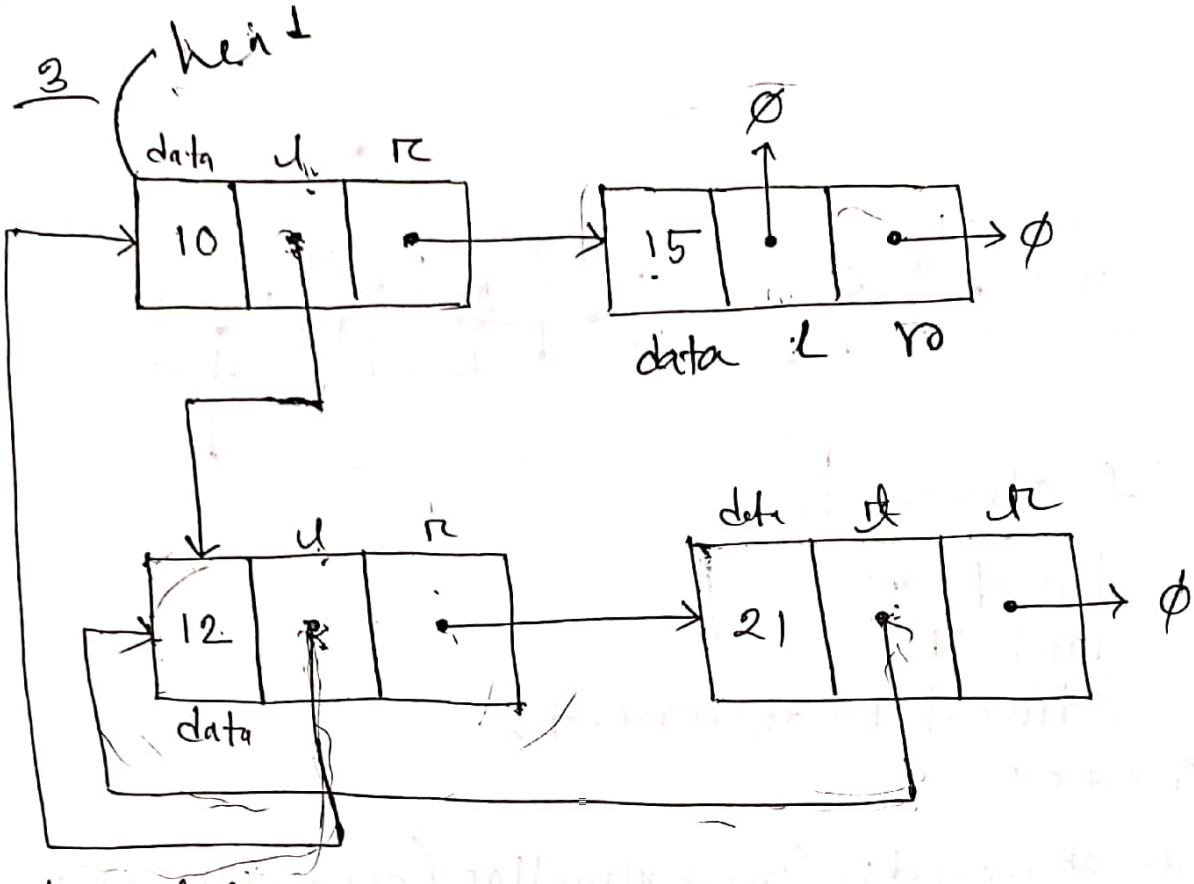
Date

(2)




```
typedef struct Node{
    float x;
    int y;
    struct Node *next;
}node;
node *head=(node*)malloc(sizeof(node));
head->x=1.7; head->y=11;
head->next=(node*)malloc(sizeof(node));
head->next->x=2.5; head->next->y=9;
head->next->next=(malloc);
head->next->next->x=3.5;
head->next->next->y=13;
head->next->next->next=
    head->next;
```

3



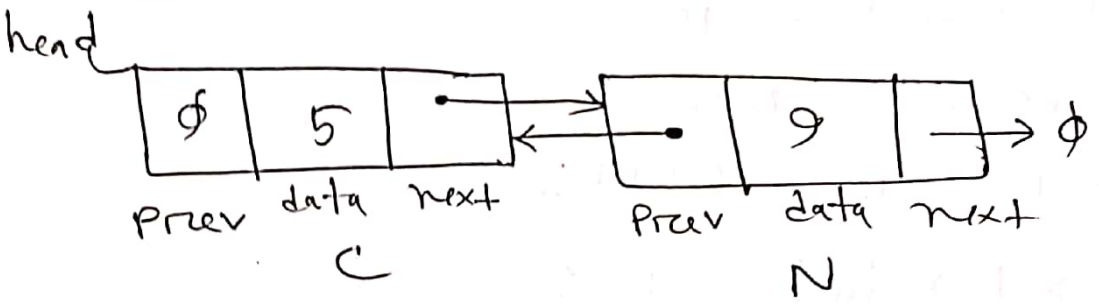

```
typedef
struct Node {
    int data;
    struct Node *l, *r;
} node;

node *head;
head = (malloc);
head -> data = 10;
head -> l = (malloc);
head -> r = (malloc);
```

Date

```
head->r->data = 15;
head->r->l = NULL;
head->r->r = NULL;
head->l->data = 12;
head->l->l = head;
head->l->r = (malloc);
head->l->r->data = 21;
head->l->r->l = head->l;
head->l->r->r = NULL;
```

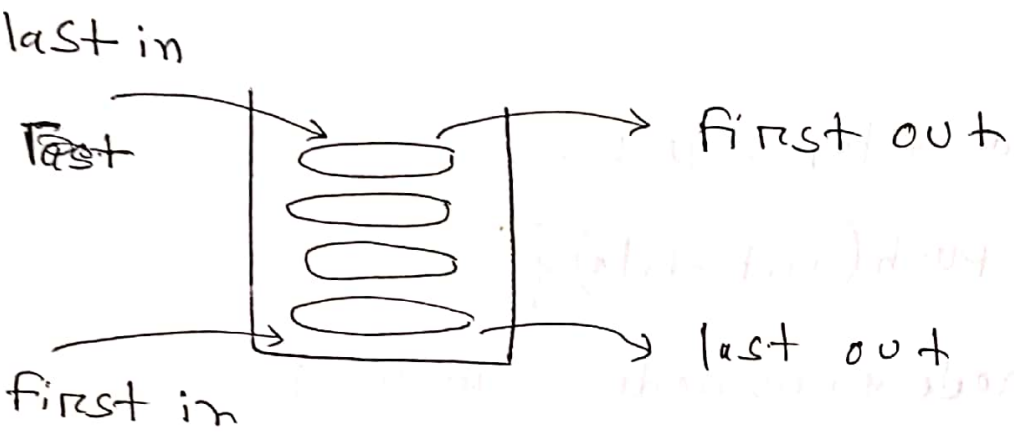
* Doubly linked list




```
node *head = NULL;
node *newnode = (malloc);
newnode -> data = 10;
newnode -> next = NULL;
newnode -> prev = NULL;
if (head == NULL) {
    head = newnode;
}
else{
    node *current = head;
  While (current->next != NULL){
      current = current->next;
  }
  current->next = newnode;
  newnode -> prev = current;
}
```

Date

Stack



~~FILO~~

LIFO —— last in first out

push — insert data in stack

pop — Take out data from the stack.

Abetis Abecab

Push / Pop

Another way
my github link

```
node *top = NULL;
void push(int data){
    node *newnode = (malloc)
    newnode -> data = data;
    temp newnode -> next = top;
    top = newnode;
}

void pop(){
    node * ~~newno~~ temp;
    if(top == NULL){
        // Print stack is empty
        return;
    }
    temp = top;
    top = top -> next;
    free(temp);
}
```

Date

Stack application

* infix to prefix

6 * 2 + 5 − 3 + 8 / 4

(((6 * 2) + 5 − 3 + (8 / 4)))

Prefix → + − + * 6 2 5 3 / 8 4

* infix to postfix

6 * 2 + 5 − 3 + 8 / 4

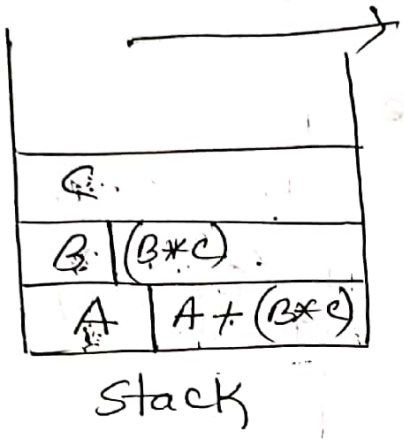
(((6 * 2) + 5 − 3 + (8 / 4)))

Postfix → 6 2 * 5 3 8 4 / + − +

Postfix to infix

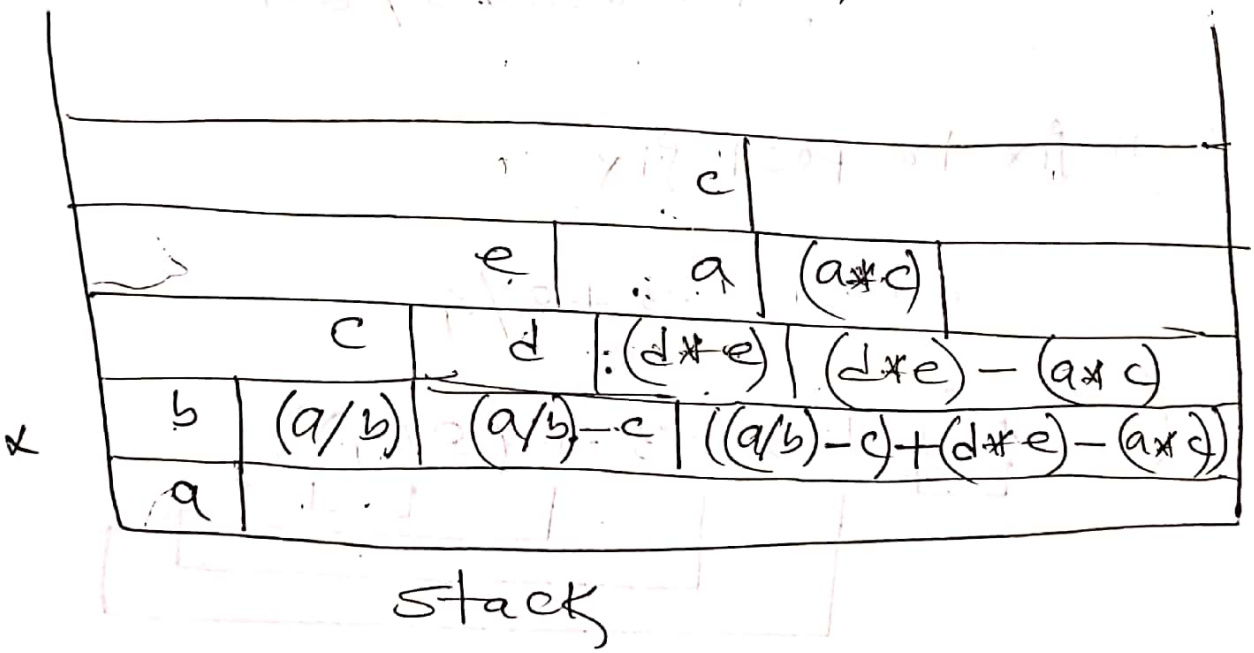
down → up

Postfix: ABC * +



infix: A + (B * C)

Post fix: ab/c − de * ac * − +



~~Post~~ infix: ((a/b) − c) + (d * e) − (a * c))

1.

Date

Prefix to infix

up – down

Prefix → + – ABC,

A	
B	(A–B)
C	(A–B)+C

stack

infix : (A–B)+C

Prefix : *+a–bc/–de+–fgh

f		d		b	a
g	(f–g)	e	(d–e)	c	(b–c)
h	(f–g)+h	((d–e)/((f–g)+h))			

stack

→ a+(b–c)

(a+(b–c)) * (d–e)/((f–g)+h)

betis let's do code Abecab

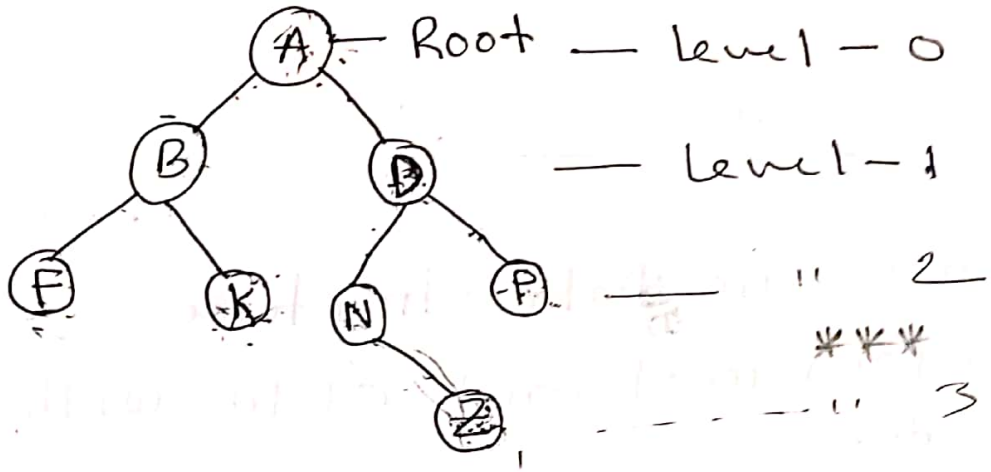
Date

Tree

10/18.9

* Non-linear Data structure.
* Contain root and child with leaf node.
* Node are connected throught edges.
* Various type (Binary, Ternary)
* Collection of nodes (or vertices) and edges without having any cycle.
* used to store information in heirarchical way.
* heirarchical Data structure.

*



1. size of the tree (No of nodes) = 8
2. Edges of the tree (No of connection) = 7
3. Root - A
4. leaf - {F, K, Z, P}
5. parent of K is - B
6. Child of B is - {F, K}
7. Siblings - F, K (would be same parent)
8. Height of tree (longest path from leaf to root) = 3
9. Height of the node (K) = 2
10. Depth of the tree (root to leaf) = 3
11. Depth of the node (K) = ϕ

Date.....

12. Ancestor of Nodes (Parent nodes including the node)

Ancestor of F = {F, B, A}

13. Proper Ancestor of F = {B, A}

14. Descendant of the Node (Child Node including the node)

Descendant of F = {F}

15. Proper descendant of D = {N, P, Z}

16. order / degree of the tree (No of subtree) = 2

Degree / order of the Node (N) = 1

17. level of tree (2^n) {root = 0}

Root = $2^0 = 1$

level 4 = $2^4 = 16$ | Binary tree.

$(2^{height+1})-1$

$2^{level} =$

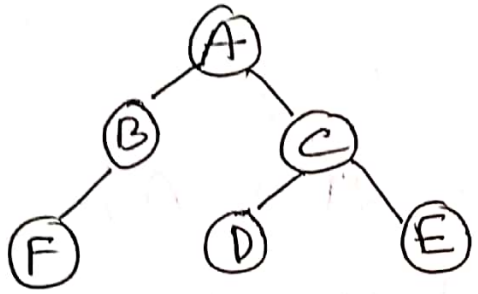
$2^0 = 1$ $2^2 = 4$

$2^1 = 2$ $2^3 = 8$

2 2 2 2

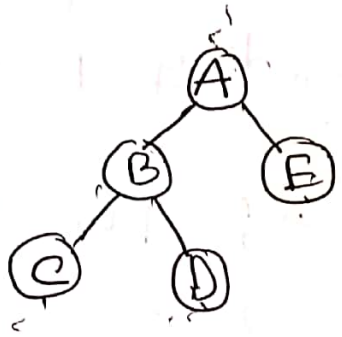
Abetis Abecab

1. Binary tree (max two children)



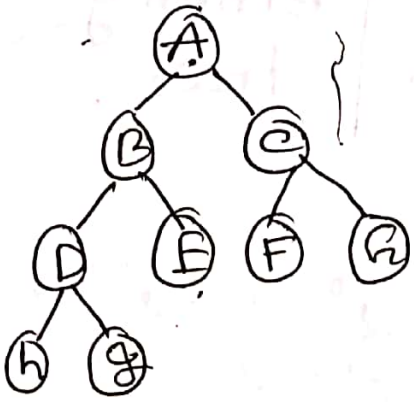
2. Full Binary tree / strictly / Proper / two Binary tree :

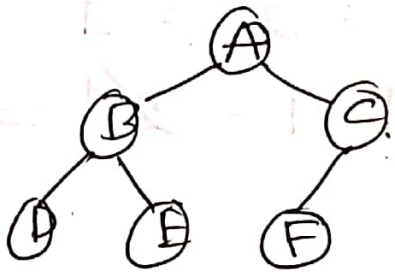
can have two or no child



3. complete Binary tree :

filled each level left to right as far as possiple.

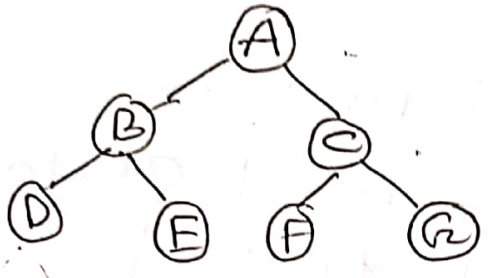




Date

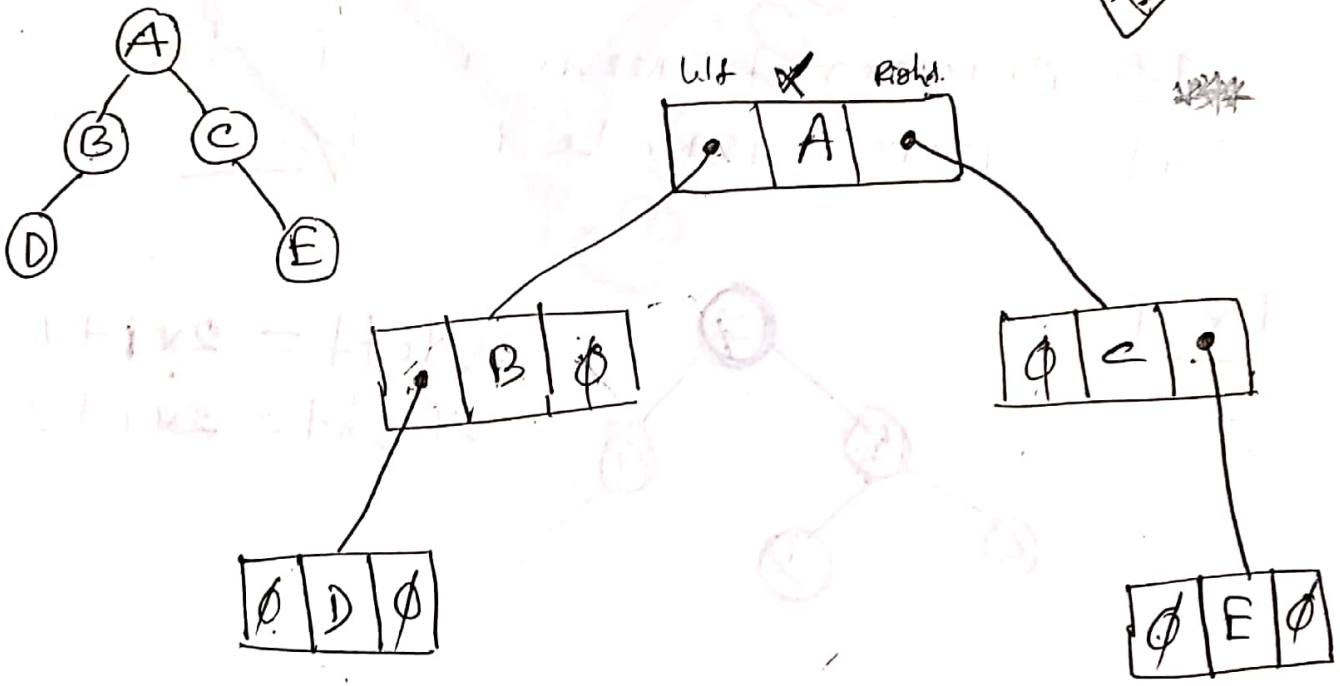
4. Perfect Binary tree.

All level must be filled.



Representation Binary tree in memory.

① linked Representation.



Abetis Abecab

(11) sequential Representation

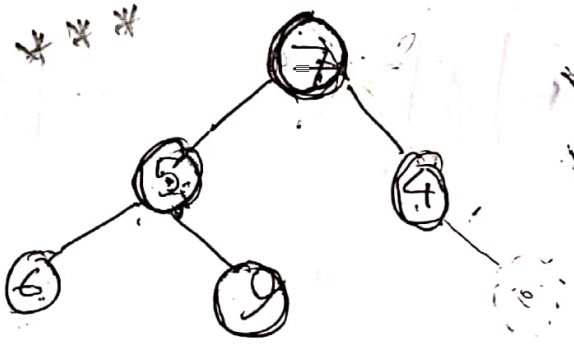
- need Array (tree[ ])
- Root tree[0]

If a node 'N' stored in tree[i] then it's left child will be in tree[2*i+1]
it's Right will be tree[2*i+2]

→ $2^n - 1$ for memory for N level tree.

If current node is i then it's parent node is i/2

Ex-1 ***



left = 2*i+1
Right = 2*i+2

7	5	4	6	9			
0	1	2	3	4	5	6	

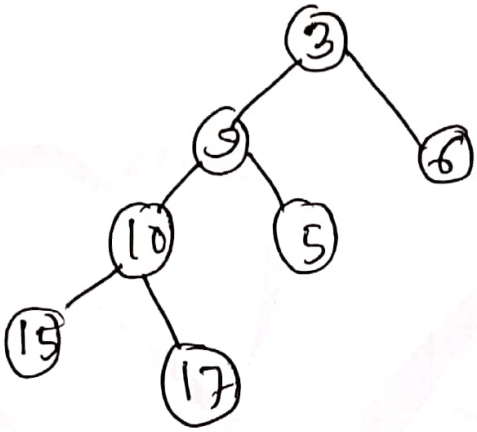
Tree to Array

12-1

Date

Ex-2

0	1	2	3	4	5	6	7	8	9
3	9	6	10	5			15	17	



Array to tree

$2*i+1$ — 2

$2*i+2$ — 6

$1+2+2$

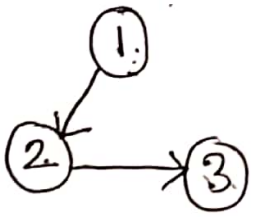
Abetis Abecab

Traversing Binary Tree ***

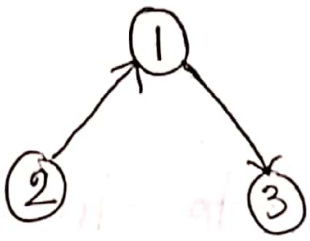
(I) Pre-order (NLR)
(II) in-order (LNR)
(III) Post-order (LRN)

N = Node
R = Right
L = Left

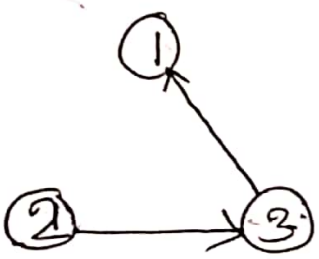
(I)



(II)

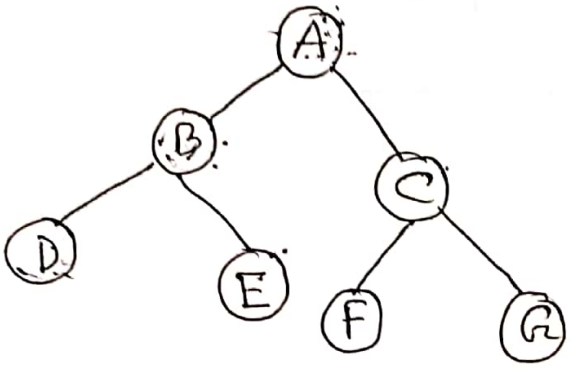


(III)



Date.....

Ex-1

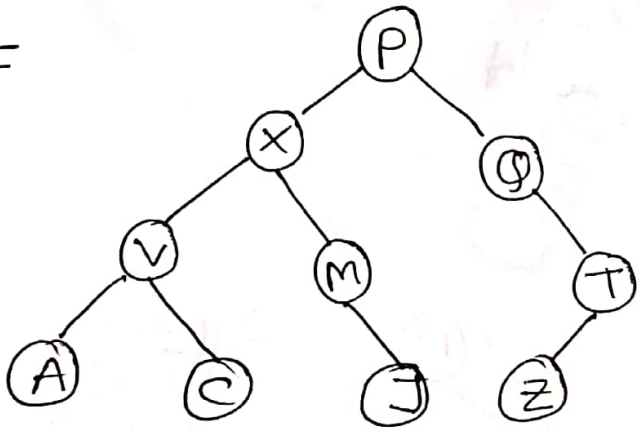


Pre-order: (NLR) → { ABDECFG }

In-order: (LNR) → { DBEAFCG }

Post-order: (LRN) → { DEBFGCA }

Ex-2



Pre-order (NLR) – { PXVACMJQTZ }

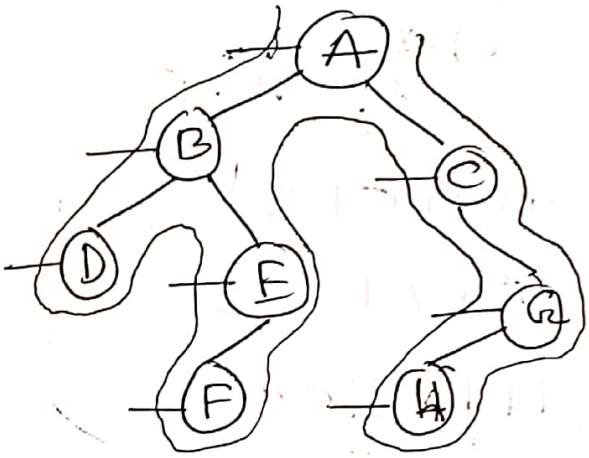
In-order (LNR) – { AVCXMJPQZT }

Post-order (LRN) – { ACVJMXZTQP }

Pre-order → —O

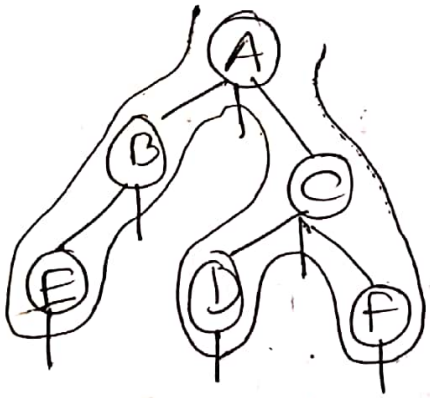
in-order → O

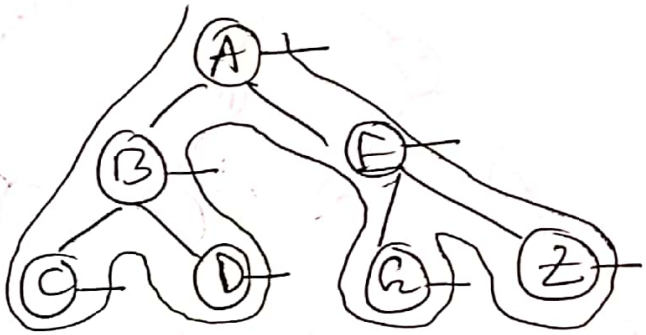
Post-order → O—



নিজে চেষ্টা করি

Pre— A B D E F C G H





in— E B A D C F

Post - C D B G Z E A

Date

*Binary Search Tree

* Data would be sorted.

```
int Bsearch(node *temp, int data){
    if (temp == NULL) return 0;
    if (temp->data == data) return 1;
    if (data < temp->data)
        return Bsearch(temp->left, data);
    else return Bsearch(temp->right, data);
}
```

* Delete node from BST

github

Data structure

Abetis Abecab

BST operation ***

① Insert operation

compare the value with the root node.

if value > node = go right

if value < node = go left

② Deletion

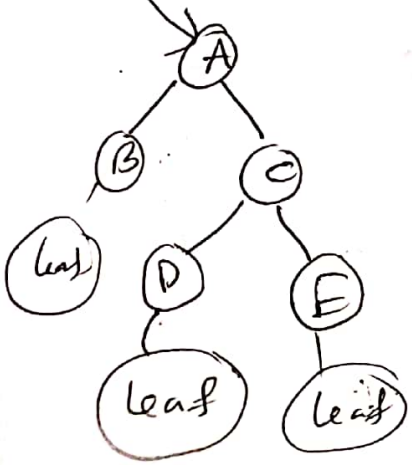
Find the value 1

if leaf node just delete.

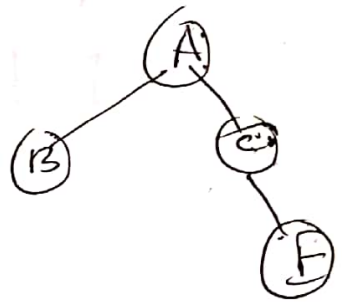
→ if not leaf

* swap with higher child

* Delete node



delete D

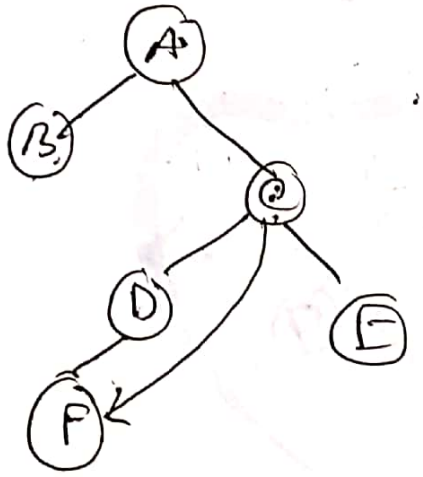


BST → Duplicate value insert [illegible]

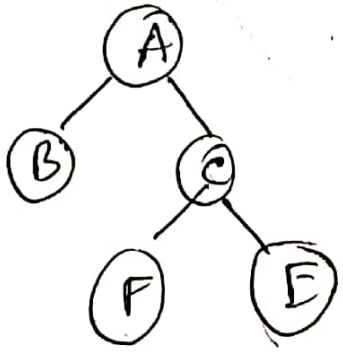
Date

If not leaf

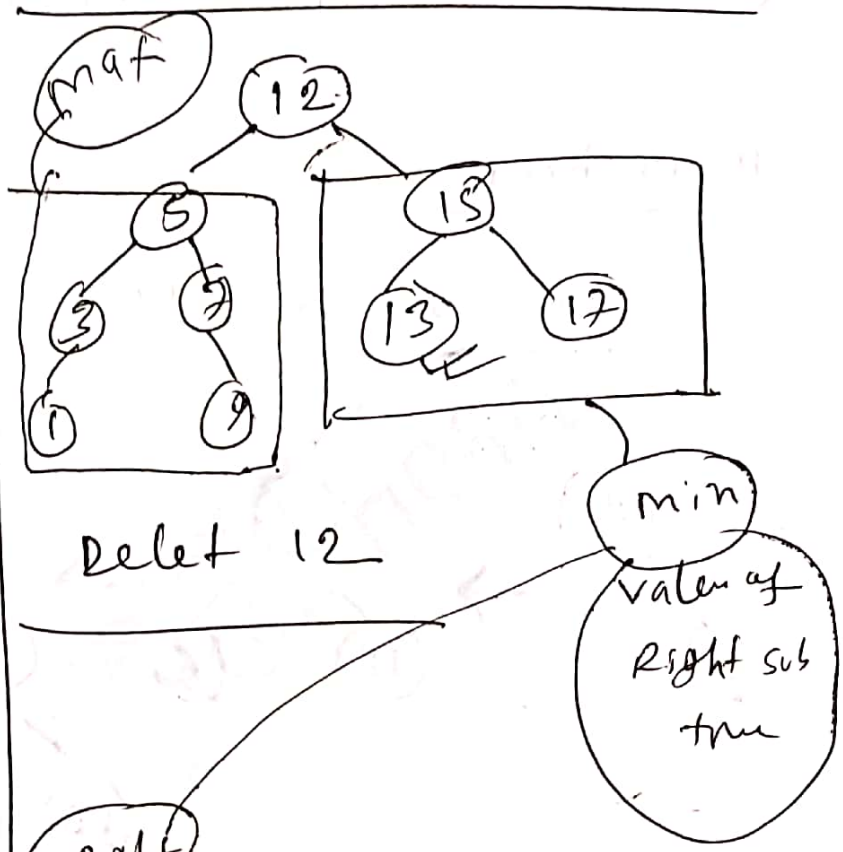
if one child have the node which we want to delete.



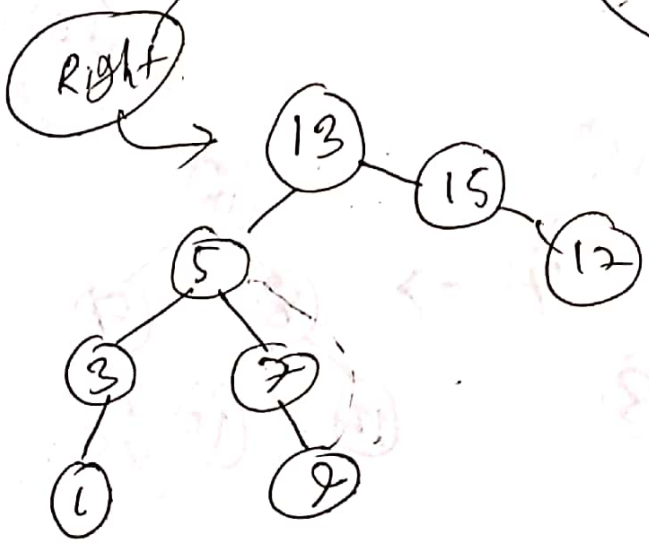
delete D



if two child



Delet 12



OR we can pick max value from Left subtree

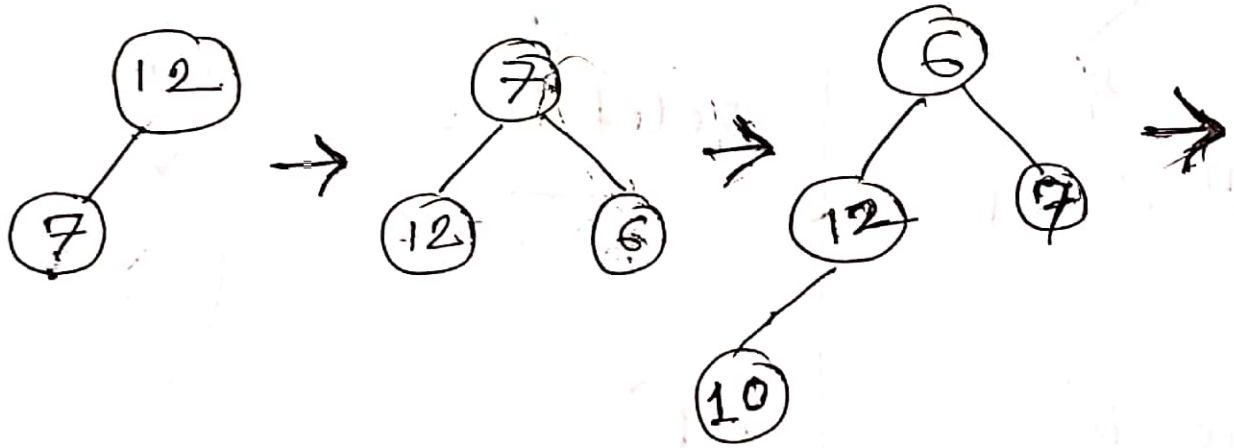
Abetis Abecab

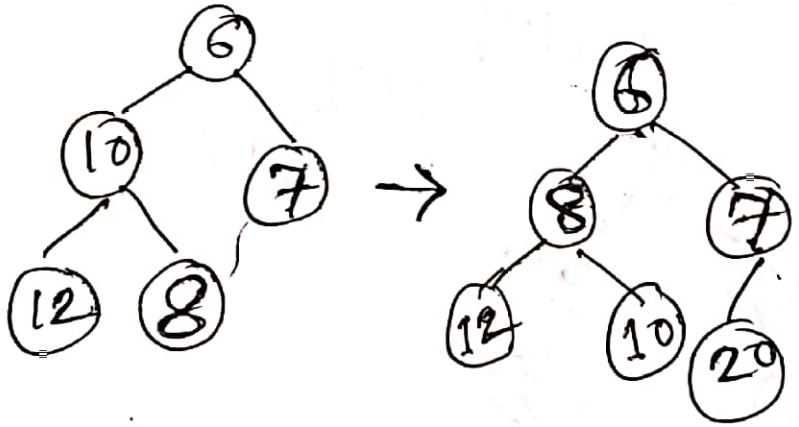
Binary Heap ***

(I) min heap — child >= Parents
(II) max heap — child <= Parents

min heap

12 7 6 10 8 20



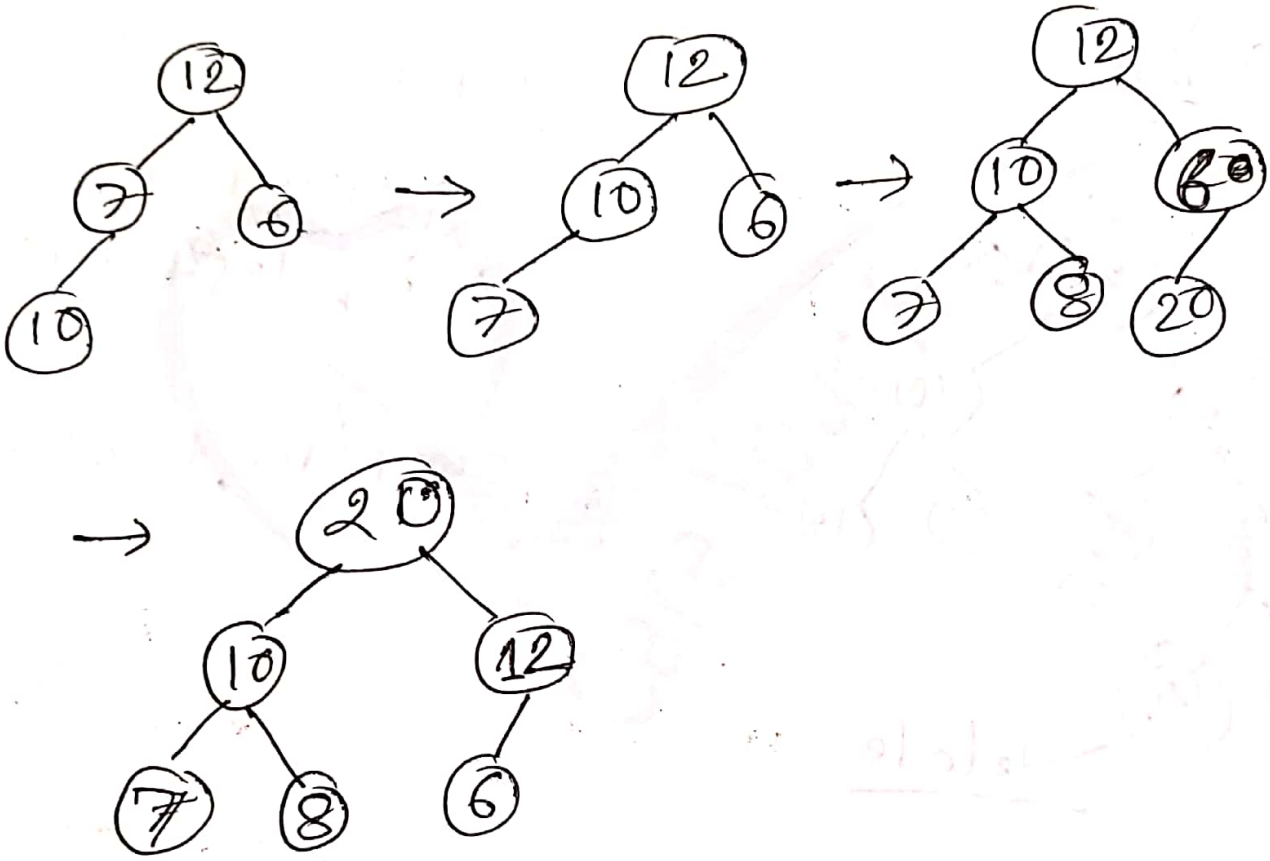


* must be complete Binary tree.

Date

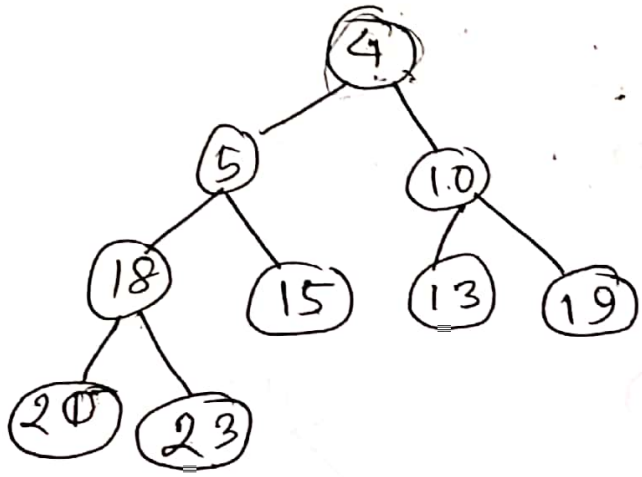
max heap

12 7 6 10 8 20



* must be complete Binary tree.

Deletion min heap

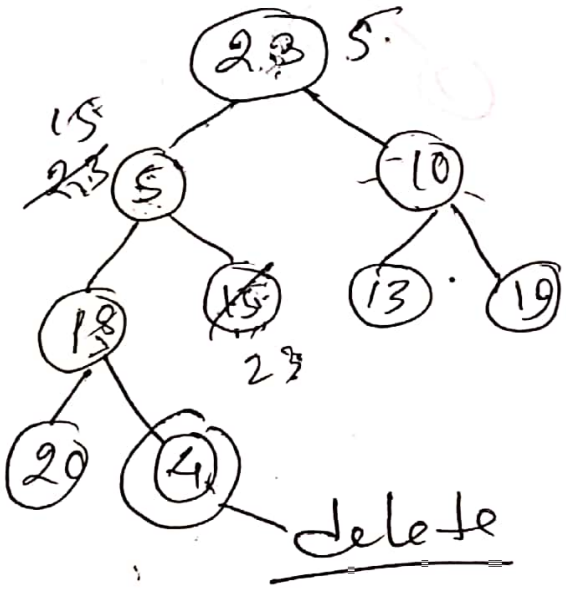


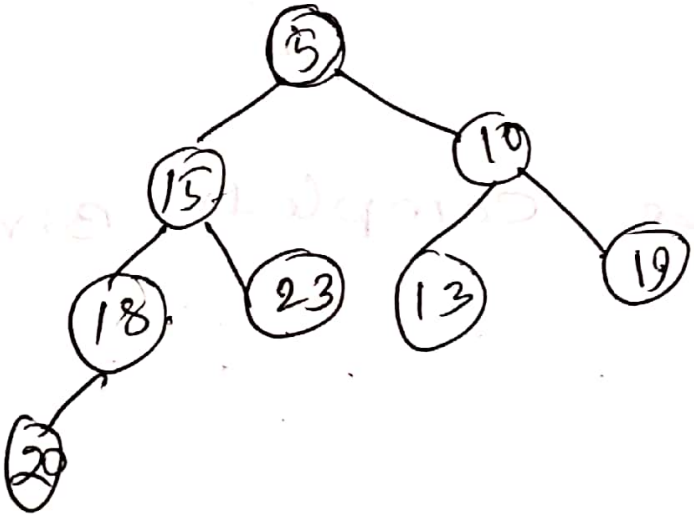
*** Always root node delete করা হয় max and min heap এ দিয়ে করতে।

step 1

~~swap root and last leaf~~

swap(root, last leaf)

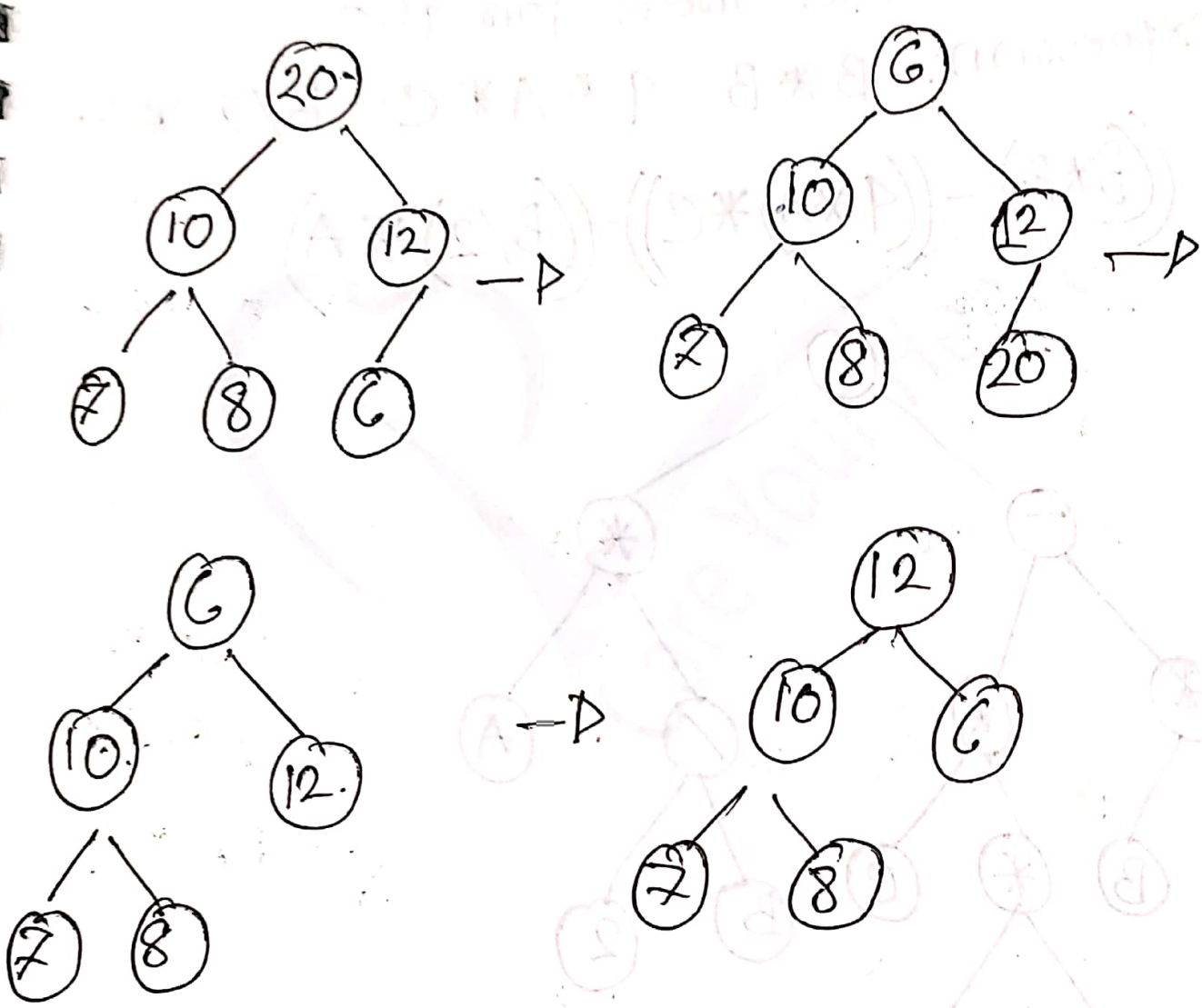




Date

max heap deletion

same as min heap (format same)



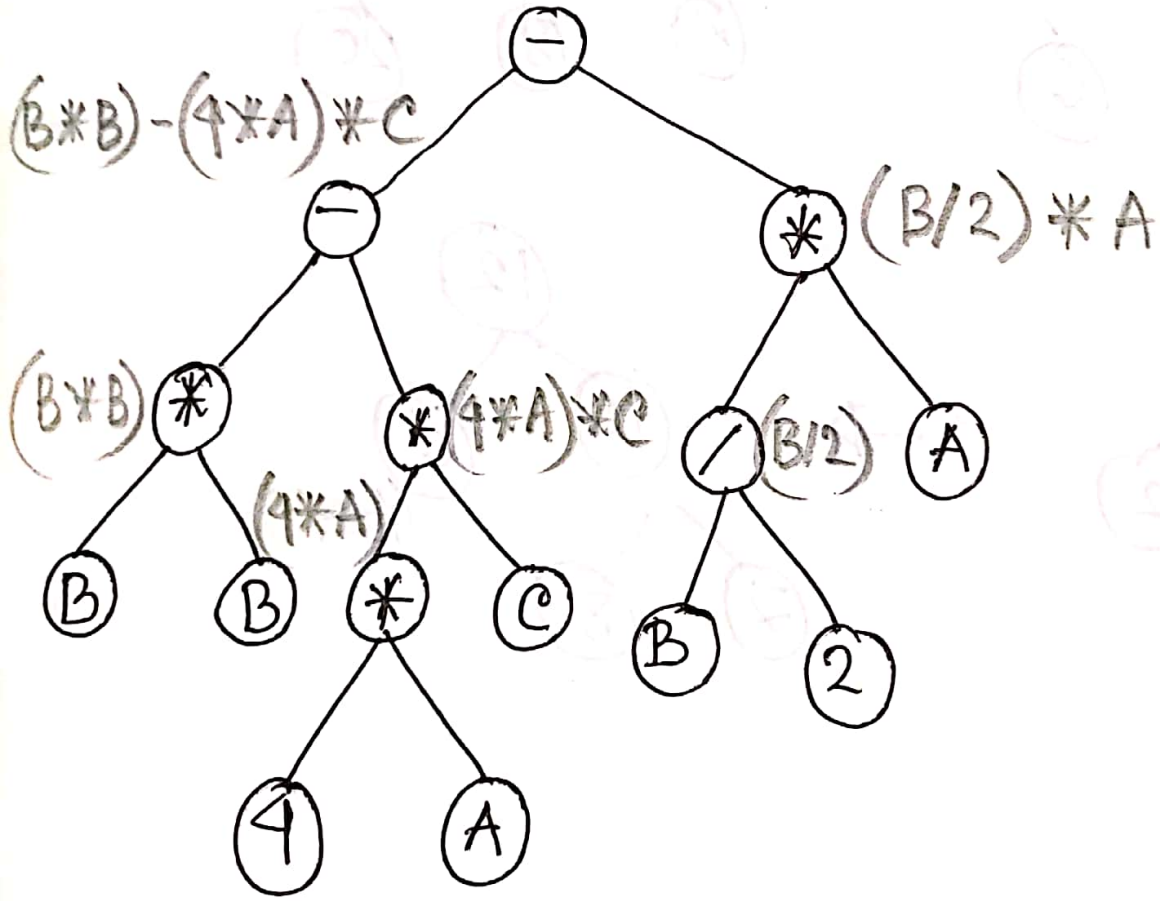
Abetis Abecab

***

For the expression processing answer the following question:

(a) Draw the tree for the expression: B * B - 4 * A * C - B/2 * A

((B*B) - ((4 *A) *C)) - ((B/2) * A)



Date

Graph

* Non-linear Data structure.
* A graph G is an ordered pair of a set V of vertices and a set E of edges.

ordered pair: *( )*

$(a, b) \neq (b, a)$ if $a \neq b$

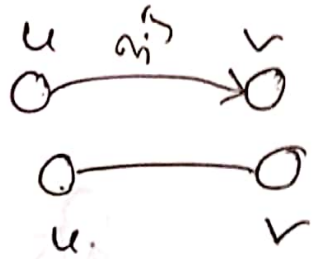
unordered pair *{ }*

$\{a, b\} = \{b, a\}$ order is not important

Edges

Can be directed.

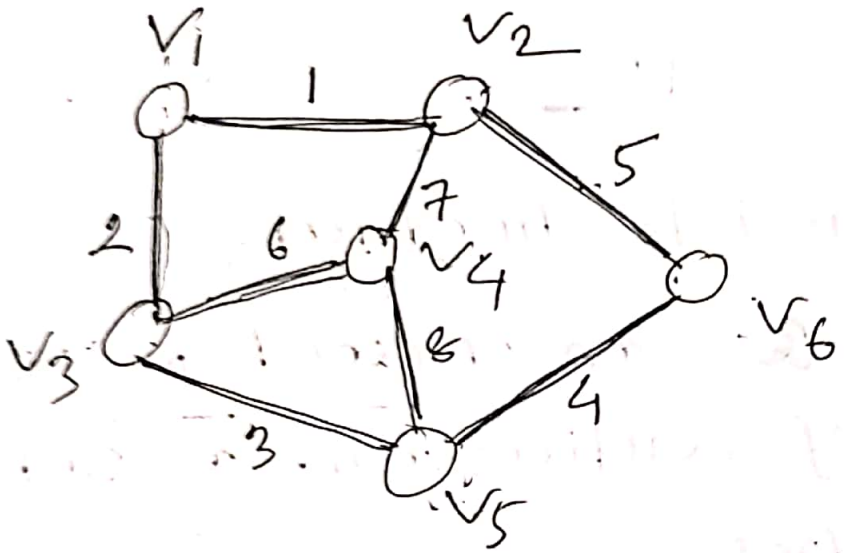
Can be undirected



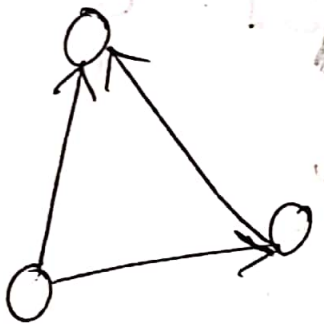
undirected is unordered.

{u, v) — Bi-directed.

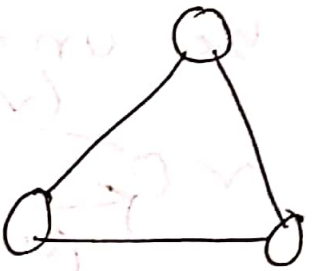
Abetis Abecab



$$E = \{ \{v_1, v_2\}, \{v_1, v_3\}, \{v_3, v_4\}, \{v_3, v_5\}, \{v_2, v_4\}, \{v_4, v_5\}, \{v_2, v_6\}, \{v_5, v_6\} \}$$



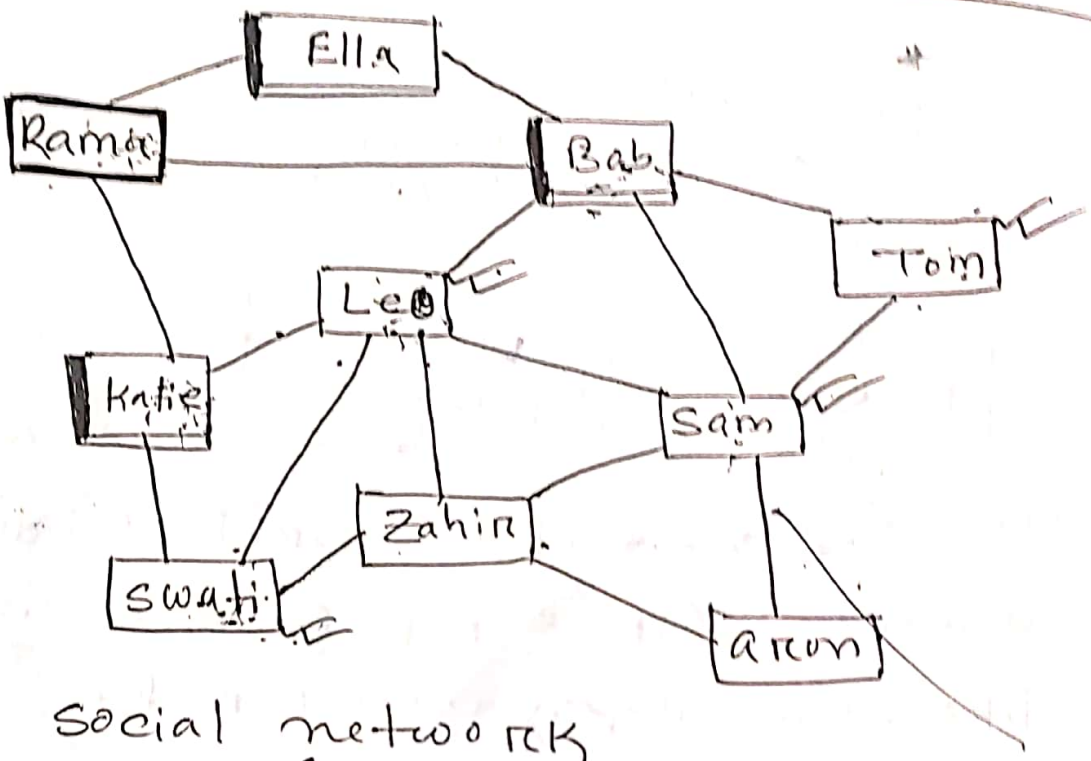
directed graph
or Digraph



undirected
graph

Date

(Standard Problem) *** (undirected)



Social network
(facebook)

Real Example: undirected graph.

Rama friend: {Ella, Bob, Katie}

Rama's suggest: {Lee, swati, sam, Tom}

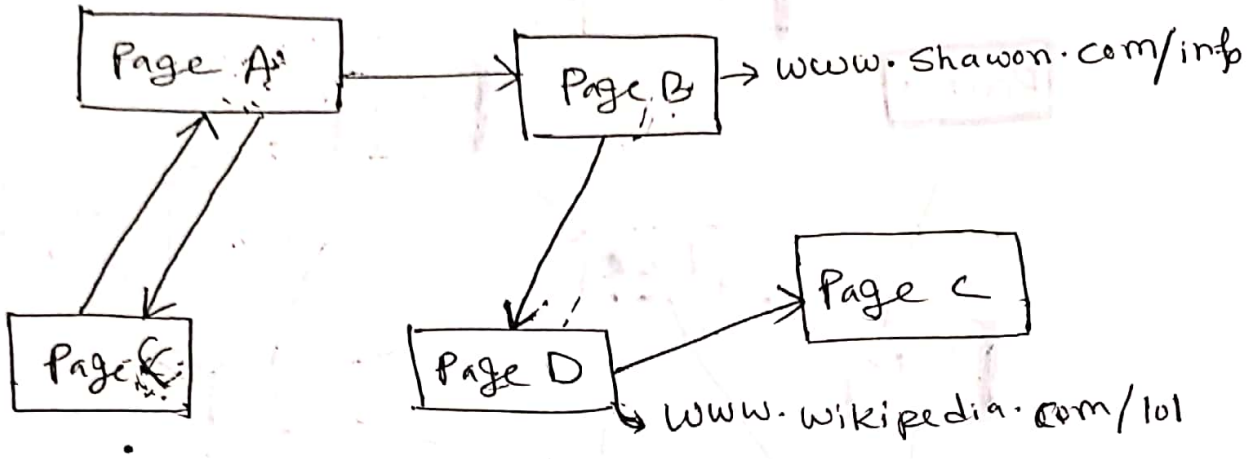
Q: Can suggest some friends to Rama?

Q: find all nodes having length of shortest path from Rama equal to 2.

Same

Abetis Abecab

Real Example (directed graph)

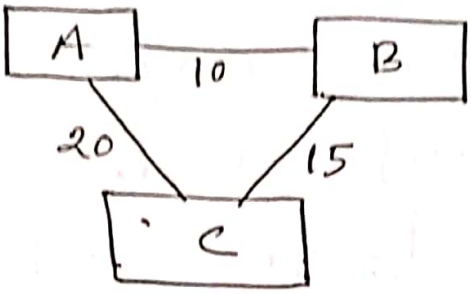


We considered it ~~on~~ World Wide web. ~~We can~~ suppose page B have link like www.wikipedia.com/101. We can go Page B to Page D ~~But we can't~~ By clicking link. But ~~we~~ wikipedia has no link like www.shawon.com/info. So I can't go here by clicking. So connection is one way. That's why it is directed.

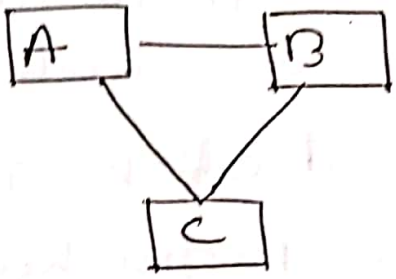
Date

Weighet vs unweighet graph

A, B, C is Three cities.



So length between A and B cities are 10 (km). This is the example of weighed.



This is unweighed graph. So, we can assume that all edges having weigh = 1 unit

- Social network — unweighted graph and also undirected.
- World wide web — unweighted and directed graph.
- Intercity road network — weighted and undirected graph.

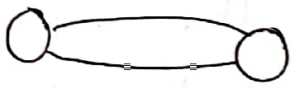
*** some road can be directed and weighted graph.

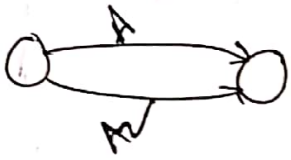
Abetis Abecab

Self loop:

If we think about world wide web. ~~To have a web~~ www.shawon.com/info. If I click again in info it would be refreshed. so thats the self loop works.

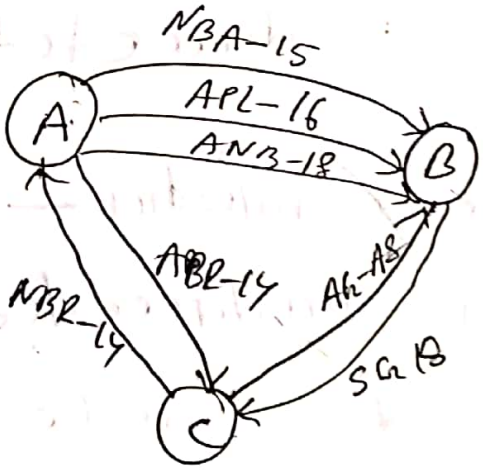
multi edge





Real Example

We considered three cities. One city to another city can have different flight and different cost.



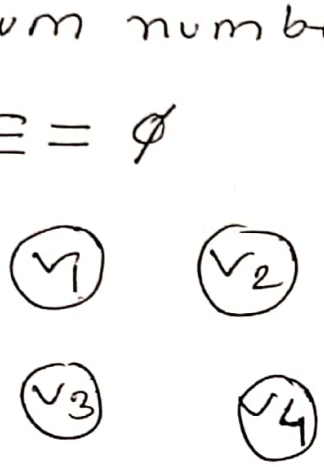
If there are no self-loops or multiedges, it's called simple graph.

Date

Number of edges (No self loop, No multiedges)

minimum number of ~~[illegible]~~ edges

is, $E = \phi$



$V = \{v_1, v_2, v_3, v_4\}$

$|V| = 4$

$E = \phi$

dis-connected.

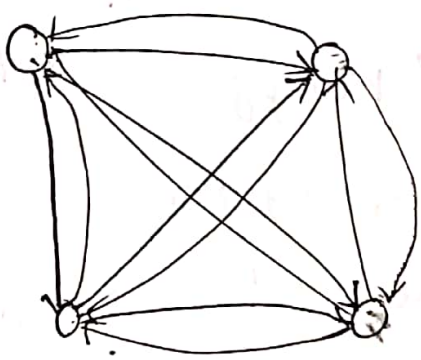
maximum number of edges,

$E = n * (n-1)$

if $|V| = n$

then, $0 \leq |E| \leq n(n-1)$

$\boxed{\text{for directed graph}}$



1 vertex is connected with 3 vertex.

total 4 vertex. $4 \times 3 = 12$

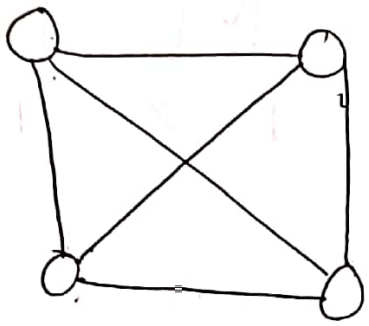
$n * (n-1) = 12$

Abetis Abecab

Undirected graph (simple graph)

$$\frac{n * (n-1)}{2}$$

if $|V| = n$, then $0 \leq |E| \leq \frac{n(n-1)}{2}$



Bi-directional

so, $\frac{n(n-1)}{2} = 6$

$\boxed{\text{no loop / no multi-edge}}$

Called simple graph

if $|V| = 10$, $|E| \leq 90$

if $|V| = 100$, $|E| \leq 9900$

maximum number of edges would be close to square of number of vertices.

so it would be Dense

close to square of number of vertices.

Date

✓ Dense → too many edges ✓

✓ sparse → to few edges ✓

↓

typically close to a number of vertices and not more than that.

There is no defined boundary for what can be called dense and what can be called sparse. It all depends on context.

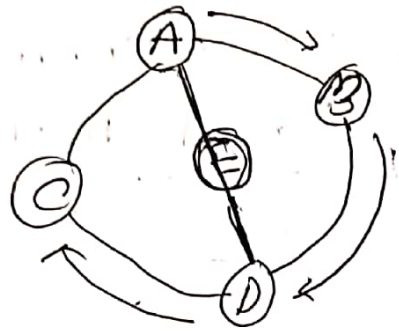
We typically store a dense graphs and something called adjacency matrix, and for sparse graph we typically use something called adjacency list

Abetis Abecab

(walk is perfect) / ~~we could also say path.~~

~~Path~~ — A sequence of vertices where each adjancent pair is connected by an edge

walk

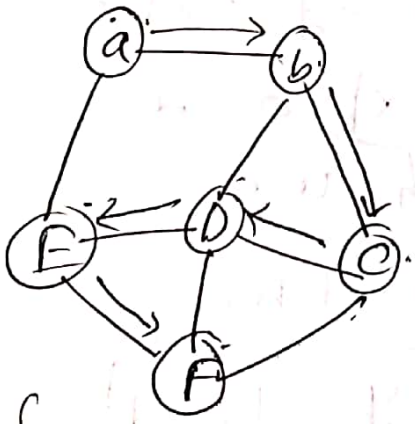


{A, B, D, C} — path.

simple path : no vertices (and thus no edges) are repeated.

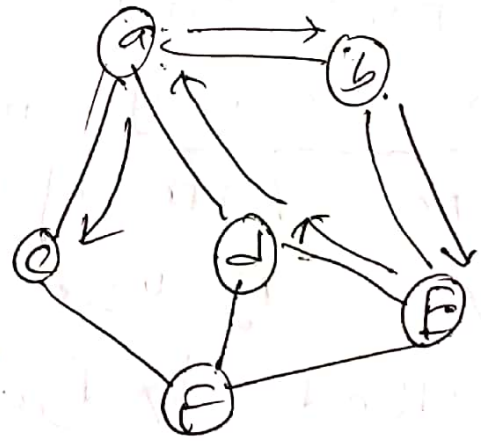
or (path)

Trail — a walk in which no edges are repeated. but vertex can be repeated.



{A, b, c, d, E, F}

simple (Path)

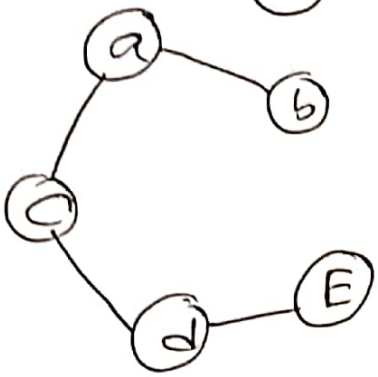


{a, b, E, d, a, c}

(Trail)

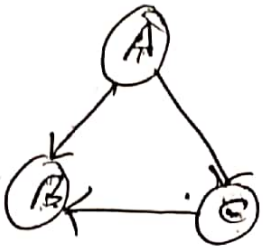
Date.....

(1)



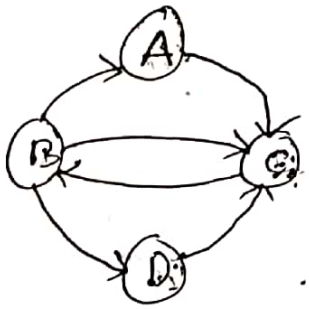
Connected graph

(2)



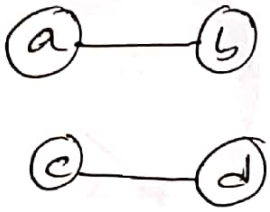
weakly connected graph

(4)



strongly connected graph

(3)



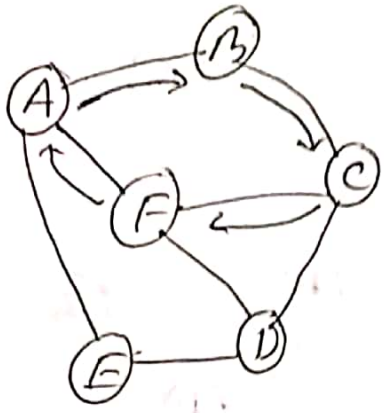
disconnected graph

(2) weakly because we can't go A vertex from c vertex. ~~OR~~ or etc.

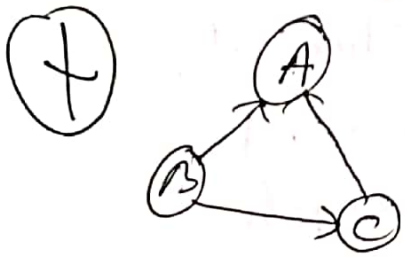
(4) we can go any vertex from any vertex that's why it is strongly connected.

** intercity road is strongly connected graph.

Abetis Abecab



{A, B, C, F, A}
is cycle.
start and end
vertex are same.



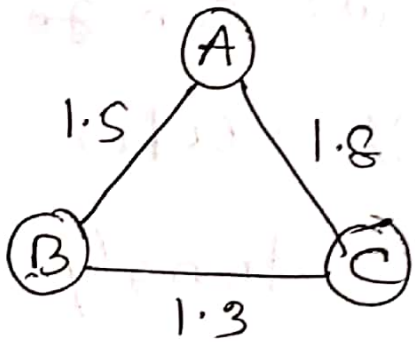
This is not a cycle.
because we can't go
A vertex to C vertex.
This is called acycle.

both are for directed graph

Cyclic and acyclic

✓ undirected graph alway symmetric

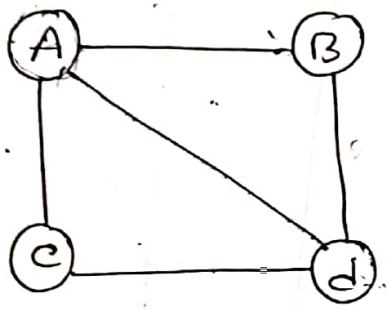
	A	B	C
A		1.5	1.8
B	1.5		1.3
C	1.8	1.3	



Date

Graph Representation

① Adjucency matrix



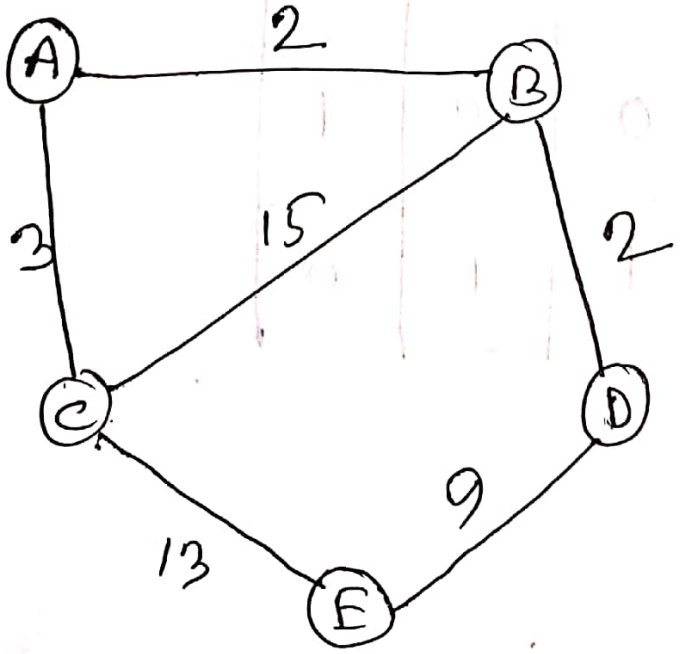
graph to matrix (unweighted graph)

	A	B	C	d
A	0	1	1	1
B	1	0	0	1
C	1	0	0	1
d	1	1	1	0

Abetis Abecab

matrix to graph. (weighted)

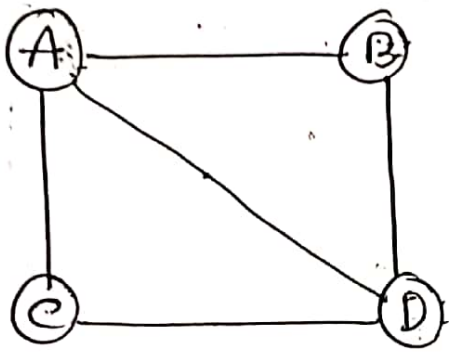
	A	B	C	D	E
A	0	2	3	0	0
B	2	0	15	2	0
C	3	15	0	0	13
D	0	2	0	0	9
E	0	0	13	9	0

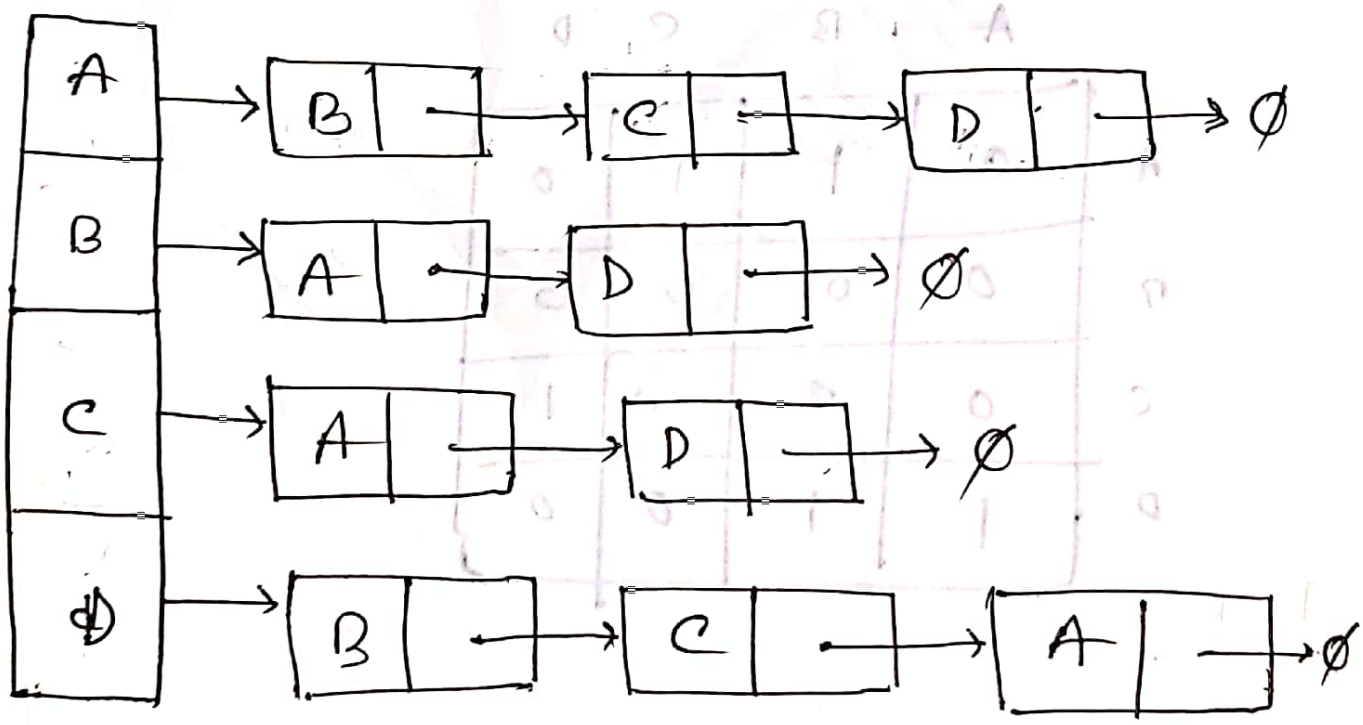


Date

(11) Adjucency list

adj(1).size()

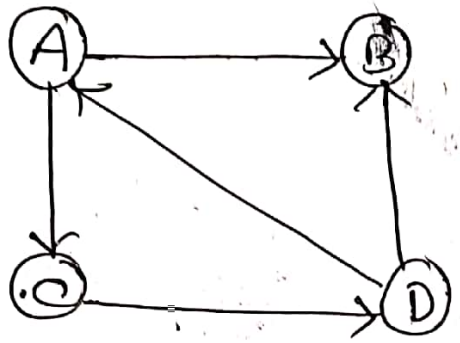




** Can be Directed graph. ***

Abetis Abecab

Directed graph



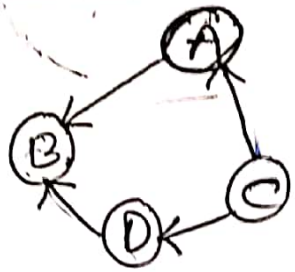
matrix

	A	B	C	D
A	0	1	1	0
B	0	0	0	0
C	0	0	0	1
D	1	1	0	0

list

Date

Topological order



* In-degree =
* out-degree =

A = In(1), out(1)

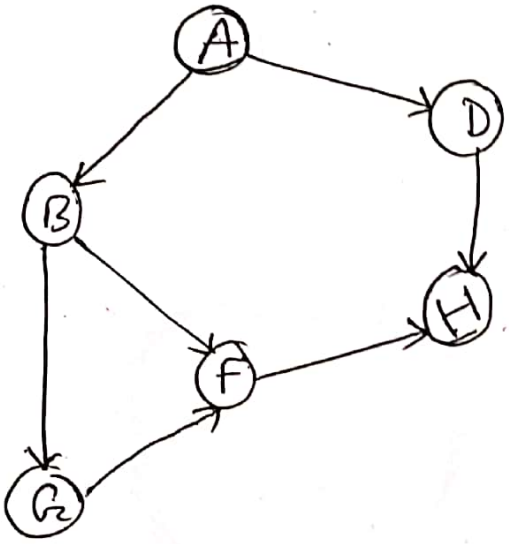
B = In(2), out(0)

C = In(0), out(2)

D = In(1), out(1)

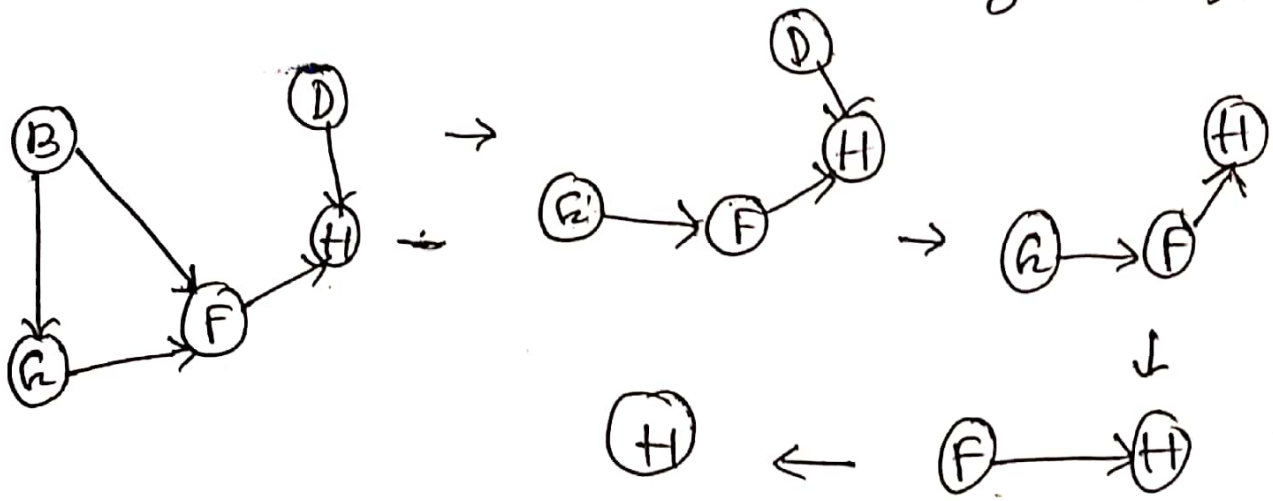
Total In degree = 4

Total outdegree = 4



order based ~~index~~ and indegree – ∅

* Start with a vertex having indegree = ∅



Traversing = A B D G F H