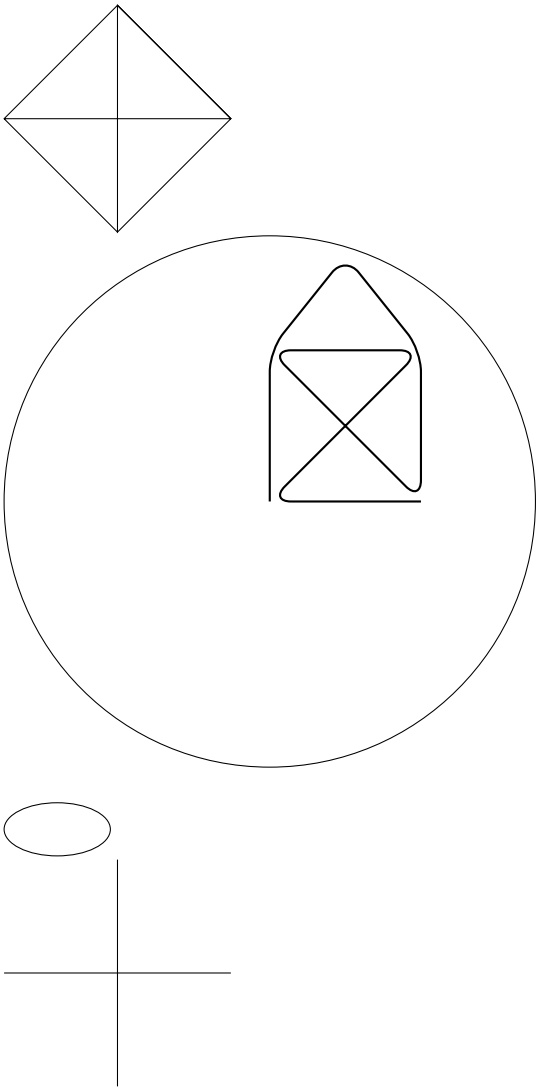
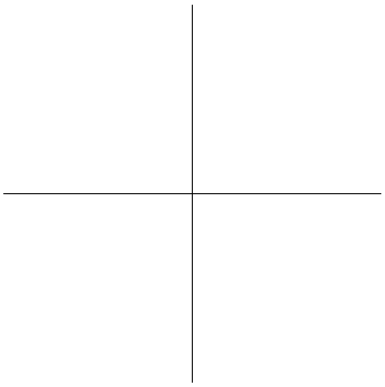
M*aking* G*raphs*

Sofiullah Iqbal Kiron

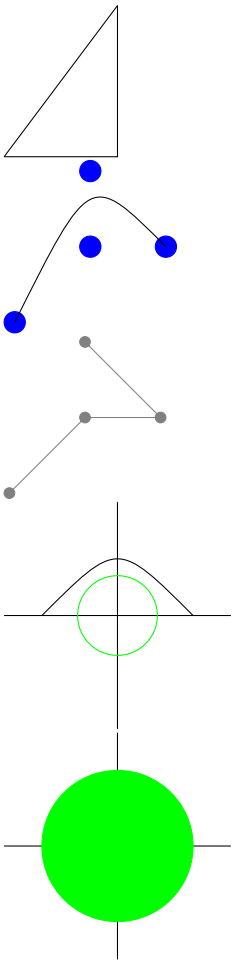
October 10, 2021

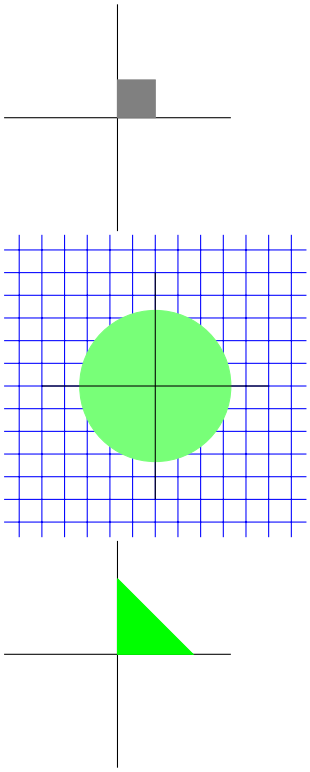


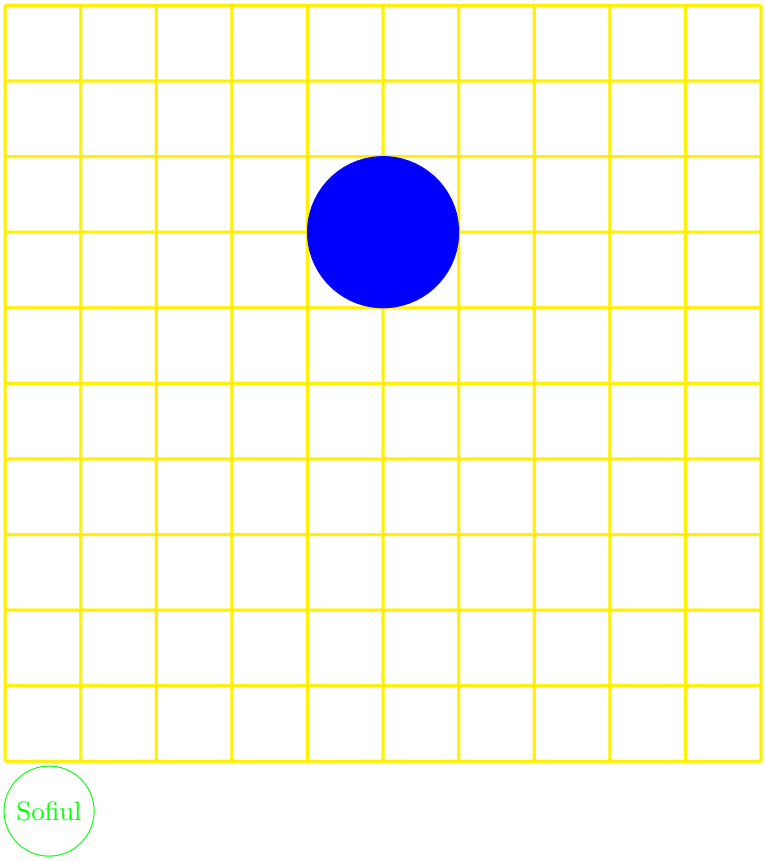
We are working on blah:

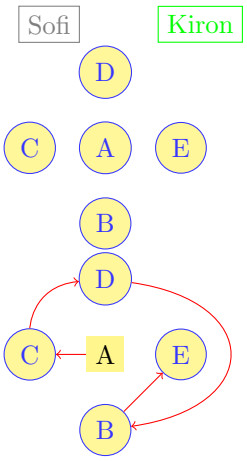


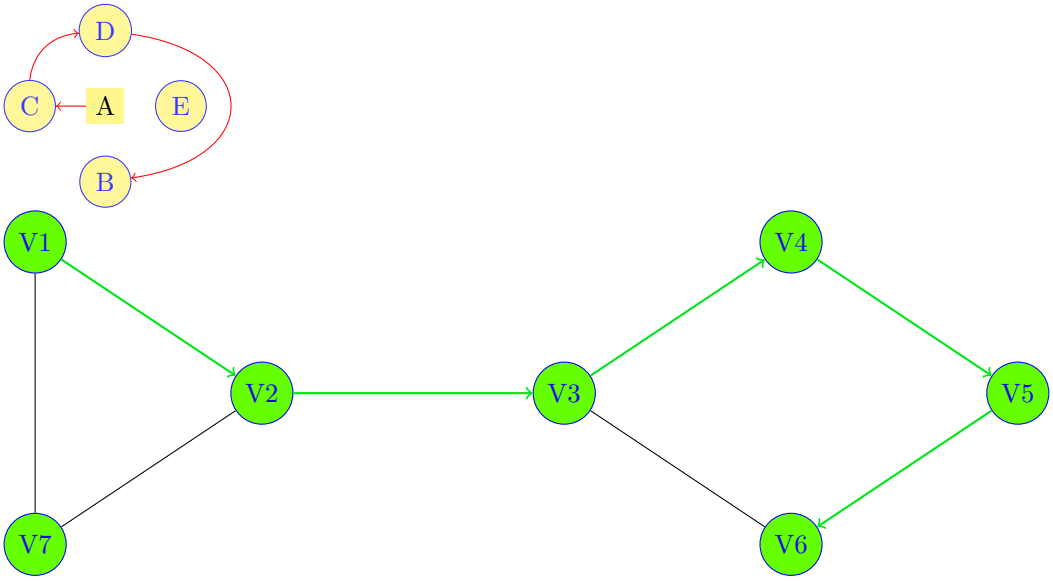
One unit is 1 cm initially.






Sofiul


Sofi
Kiron
D
C
A
E
B
D
C
A
E
B



1 Graph

1.1 Representation

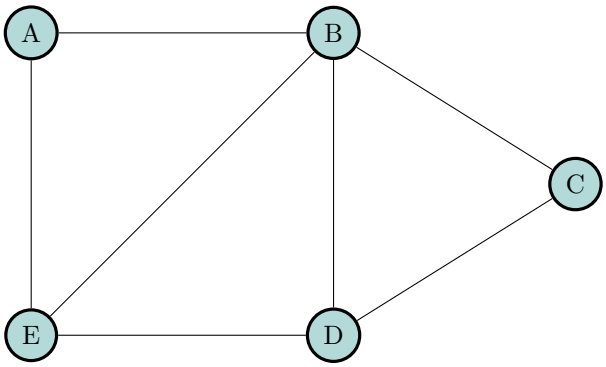
What is graph: Graph is a collection of nodes / vertices(**V**) and edges(**E**) placed between them. Represented as:

$$G = (V, E)$$

We can traverse vertices through edges. These edges might be weighted of non-weighted.

There are two kinds of graphs:

Un-directed: Not direction specified. We can traverse either direction between two vertices.



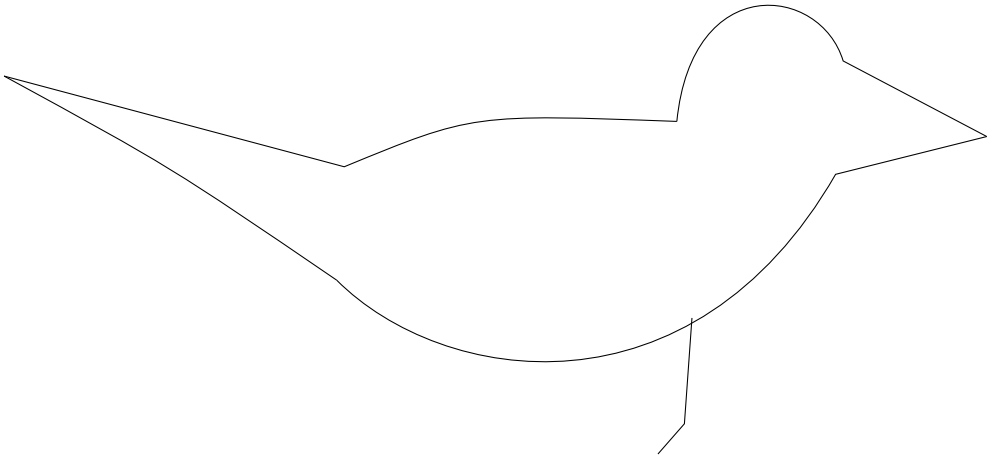
Directed: Can traverse only in the specified direction between that two vertices.

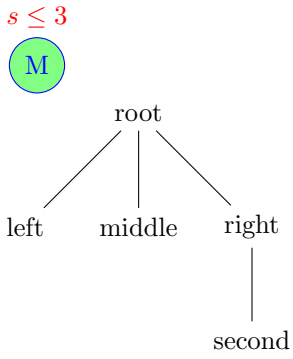
Two common ways to represent a graph:

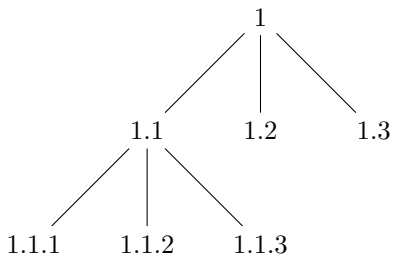
- Adjacency Matrix
- Adjacency List

Adjacency Matrix: Adjacency matrix is a 2D array which had the size of $(V * V)$, where V are the number of vertices in the graph. Adjacency matrix is symmetric if the graph is un-directed. Here is the adjacency matrix for above graph:

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

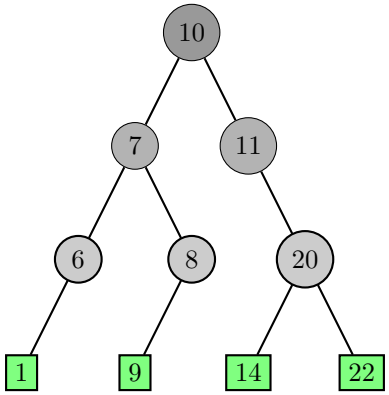






2 Binary Tree Illustration

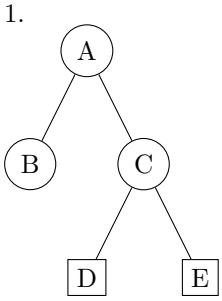
Here is the illustration of some Binary Trees. Trees are a common way of visualizing hierarchical data structures. A computer scientist's trees grow downward while a mathematician's tree will grow upward. Two binary tree will said to be similar if they both have same structure and said to be copy if they have same structure and same content at corresponding node.



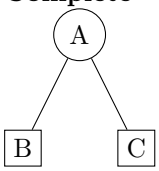
Internal nodes are represented by circle and external nodes or leafs are represented by rectangle / square.

2.1 Kinds of binary tree

1. **Full binary tree:** In every node for a tree that has either 0 or 2 child nor 1.



2. **Complete binary tree:** All the leafs are in same level.



3. **Perfect binary tree:**

2.2 Binary tree traverse

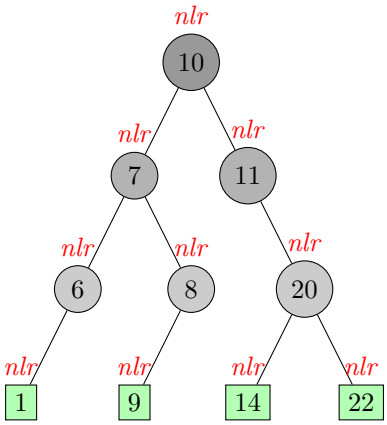
There are three (3) standard algorithm to traverse the binary tree. Those are:-

1. **Preorder** (*nlr*)
2. **Inorder** (*lnr*)
3. **Postorder** (*lrn*)

Okay, now this is the time for describing this three...

1. **Preorder**
   (a) Process the root.
   (b) Process the left subtree.
   (c) Process the right subtree.

   So we can easily say that, the form is *nlr*. Let's traverse previous binary tree by this *nlr* form.



Now we obtain that list by **Preorder Traverse:** $[10, 7, 6, 1, 8, 9, 11, 20, 14, 22]$. If we look out deeply in the traversed list, we gather that, in Preorder traverse:-

→ First element is head node.

→ and rightmost element are placed in last, that's mean, last element is the rightmost leaf of considered binary tree.

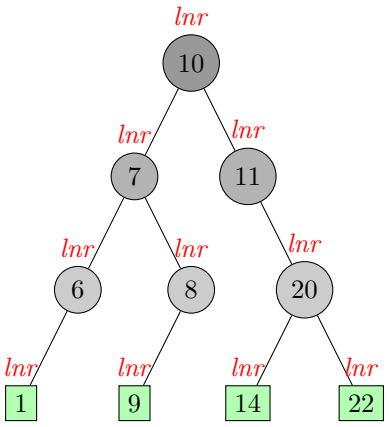
2. **Inorder:**

   (a) Process the left subtree.

   (b) Process the root.

   (c) Process the right subtree.

   The form for the **Inorder** traversal is *lnr*.



When we traverse the tree in *lnr* form, we obtain that list: $[1, 6, 7, 9, 8, 10, 11, 14, 20, 22]$. If you look at the core of the list, then you simply see that:-

→ First element in the list is leftmost node of that tree.

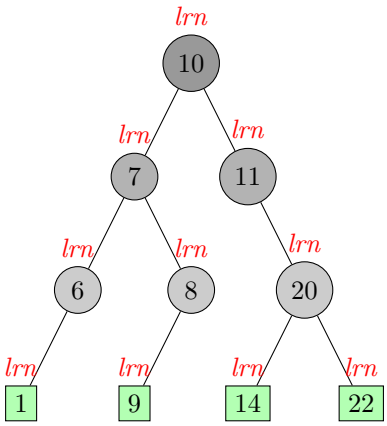
→ Last element in the list is rightmost node of that tree.

3. **Postorder:** traversal form is *lrn*.

   (a) Process the left subtree.

   (b) Process the right subtree.

   (c) at last, process the node.



By traversing in postorder algo., we obtain: $[1, 6, 9, 8, 7, 14, 22, 20, 11, 10]$.

Now, look at the list deeply:-

→ First element of the list is leftmost leaf of the considered binary tree.

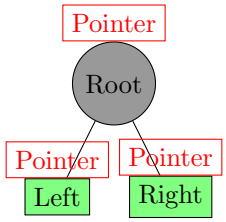
→ Last element of the list is head-node of the considered binary tree.

2.3 Binary tree implementation

A binary tree contains 3 parts at a node:-

1. Data
2. Pointer to the left subtree.
3. Pointer to the right subtree.

As we assume:-



We need a struct for implement this. For the illustration purpose in C++ language:-

```
1        struct node
2        {
3                int data;
4                node *left, *right;
5        };
```

Listing 1: struct for node

Then we need a function for make node, namely make_node.

```
node *make_node(int value)
{
    node *new_node = new node;
    new_node->data = value;
    new_node->left = NULL;
    new_node->right = NULL;
}
```

Listing 2: Function for make a new node

Three type of Traverse code:-

1. If we wanna traverse the tree in Preorder form, then use the following code:-

```
void preorder(node *root)
{
    if (root != NULL)
    {
        cout << root->data;
        preorder(root->left);
        preorder(root->right);
    }
}
```

Listing 3: Void function for preorder traverse

2. Else if we traverse the tree in Inoreder form by this code:-

```
void inorder(node *root)
{
    if (root != NULL)
    {
        inorder(root->left);
        cout << root->data;
        inorder(root->right);
    }
}
```

Listing 4: Void function for preorder traverse

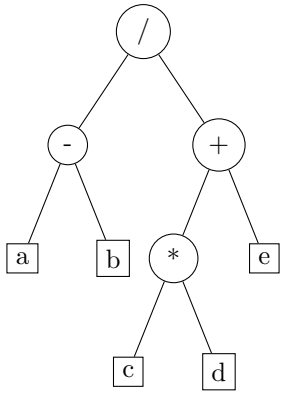
3. Or traverse Postorder form by this void code:-

```
void postorder(node *root)
{
    if (root != NULL)
    {
        postorder(root->left);
        postorder(root->right);
        cout << root->data;
    }
}
```

Listing 5: Void function for preorder traverse

2.4 Properties and application

There is an equation such as:- $E = (a - b)/(c * d) + e$. This equation can be represented by a binary tree as:



When we labeling a binary tree in level-order process, then we gather, if parent is k then:

- Left child is $2k$.
- Right child is $2k + 1$, so we can say, child of k is $\{2k, 2k + 1\}$.
- Parent of k is $\lfloor \frac{k}{2} \rfloor$.
- If number of nodes is n then depth, $d_n = \lfloor \log_2 n + 1 \rfloor$.
- If height of tree is h then total number of nodes: $n(Tn) = 2^h - 1$ where Tn is Total node.