DAY-09

* height of a node = max(ht. of left subtree, ht. of its right subtree) + 1
* height of every leaf node = 1
* height of a tree = max height of any node in a given tree
* "right skewed binary search tree": it is a bst in which only right link is used to stored an addr of child nodes, i.e. left link of each node is NULL.
* "left skewed binary search tree": it is a bst in which only left link is used to stored an addr of child nodes, i.e. right link of each node is NULL.
* max ht. of a binary search = "n", whereas n = no. of ele's in a bst.
* a bst with max height for given input size is referred as "imbalanced bst", because
as the ht. of bst is max, operations like addition, deletion & searching gets inefficient.
* if we want to achieve operations like addition, deletion & searching, ht. of binary search tree must be as min as possible i.e. log n
* min ht. of a binary search tree = log n, n = input size.
* bst having min ht. for given input size is also called as "balanced binary search tree".
* bst can be said balanced bst only if all nodes in it are balanced.
* we can say node is balanced node only if its balance factor is either -1 OR 0 OR +1.
* balance factor of a node = (ht. of left subtree - ht. of its right subtree)
* if balance factor of a node < -1 --> left imbalanced
* if balance factor of a node > +1 --> right imbalanced

* "self-balanced binary search tree": in this type of bst, while addition as well deletion of node it is making sure that bst is remains balanced, this concept/type of bst has been designed by two mathematecians:
    1. adelson velsinki
    2. lendis
and hence self balanced binary search tree is also called as "AVL" tree.
===

+ "graph": it is a non-linear, advanced data structure which is a collection of logically related similar and dissimilar type of elements contains:
- finite no. of elements referred as "vertices", also called as "nodes", and
- finite no. of ordered/unordered pairs of vertices referred as an "edges", also called as an "arcs", whereas an edges may carries weight/cost/value and it may -ve.

G(V, E)
V = set of vertices/nodes = { 0, 1, 2, 3, 4 }
E = set of an edges/arcs = { (0,1), (0,2), (0,3), (1,2), (1,4), (2,3), (3,4) }

- if there is a direct edge between two vertices then those vertices are referred as "adjancent vertices", otherwise they are referred as non-adjancent vertices.

(u, v) == (v, u) -> unordered pair -> undirected edge
(u,v) != (v,u) -> ordered pair -> directed edge

- there are two types of graph:
1. "undirected graph": graph which contains unordered pairs of vertices i.e. undirected edges.

2. "directed graph (di-graph)": graph which contains ordered pairs of vertices i.e. directed edges.

- graph can be catagorised into two more catagories:
1. "weighted graph": if edges in a graph carries weight/cost/value
2. "unweighted graph: if edges in a graph do not carries weight/cost/value

- there are two ways by which graph data structure can be presented:
1. adjancency matrix: by using 2-d array
2. adjancency list: by using array of linked lists.

* "loop": if there is an edge from any vertex to that vertex itself, such edges is referred as a loop.

* "connected vertices": if path exists between two vertices then those two vertices are referred as connected vertices.
- adjacent vertices are always connected, but vice-versa is not true.

* "connected graph": if any vertex in a graph is connected to remaining all vertices then such a graph is referred as a connected graph.

* "isolated vertex": if any vertex is not adjancent/connected with any other vertex in a given graph, then such a vertex is referred as an isolated vertex.

* "complete graph": if all vertices are adjacent to remaning all vertices in a given a graph then such a graph is referred as a complete graph.

* "cycle": if in a path in given graph, starting vertex and end vertex are same, then such a path is referred as a cycle.

* graph can be a tree but tree cannot be a graph.

* if graph is connected graph which do not contains a cycle then it is refered as a tree (specifically it is referred as spanning tree of a graph).

* "spanning tree": it is a subgraph of a graph can be formed by removing one or more edges from it in such a way that it should remains connected and do not contains a cycle.

* spanning tree must contains min (V-1) no. of edges, whereas V = no. of vertices
* one graph may has multiple spanning trees.
* "minimum spanning tree"/MST: spanning tree of a graph having min weight.

* weight of a graph: sum of weights of all its edges

- to find MST:
    - Prim's
    - Kruskal's

- Dijsktra's algo is used to find shortest distance of all vertices from given source vertex.