1. Define the following terminologies with examples

(i) Complete Graph

Definition:
A complete graph is a graph in which **every pair of distinct vertices is connected by an edge**.

Example:
A complete graph with 4 vertices is denoted as $\mathbf{K_4}$.

```
Vertices = {A, B, C, D}
Edges = AB, AC, AD, BC, BD, CD
```

(ii) Subgraph

Definition:
A subgraph is a graph formed from a **subset of vertices and edges** of an original graph.

Example:
If
$G = (V = \{1,2,3,4\}, E = \{(1,2),(2,3),(3,4),(4,1)\})$,
then a subgraph can be:
$V' = \{1,2,3\}, E' = \{(1,2),(2,3)\}$

(iii) Connected Graph

Definition:
A graph is said to be connected if **there exists at least one path between every pair of vertices**.

Example:
A linear graph:
1 — 2 — 3 — 4
All vertices are reachable from any other vertex, hence it is a connected graph.

(iv) Path and Cycle

Path

Definition:
A path is a sequence of vertices in which **each adjacent pair is connected by an edge** and no vertex is repeated.

Example:
$1 \rightarrow 2 \rightarrow 3 \rightarrow 4$ is a path of length 3.

Cycle

Definition:
A cycle is a path in which the **first and last vertices are the same**.

Example:
$1 \rightarrow 2 \rightarrow 3 \rightarrow 1$ forms a cycle.

(v) Component and Cut Vertex

Component

Definition:
A component is a **maximal connected subgraph** of a graph.

Example:
If a graph has two disconnected parts, each part is called a component.

Cut Vertex (Articulation Point)

Definition:
A cut vertex is a vertex whose **removal increases the number of connected components** of the graph.

Example:
In the graph:
1 — 2 — 3
Vertex **2** is a cut vertex because removing it disconnects the graph.

---

2. **Defiine hashing. explain different hashing function with example.Discuss the properties of a good hash function**

Ans: • Hashing is a technique to map keys of any size to **fixed-size indices** in a **hash table** using a **hash function**.

- It is mainly used for **fast access and retrieval** of data.

- Mathematically: h:K→{0,1,2,...,m−1}h: K \rightarrow \{0,1,2,...,m-1\}h:K→{0,1,2,...,m−1}

- KKK = set of keys, mmm = table size, h(k)h(k)h(k) = hash value

Hash Functions

a) Division Method

- Formula: h(k)=kmod mh(k) = k \mod mh(k)=kmodm
- Example: Key = 123, Table size = 10 → h(123)=123mod 10=3h(123) = 123 \mod 10 = 3h(123)=123mod10=3

b) Multiplication Method

- Formula: h(k)=⌊m(kAmod 1)⌋h(k) = \lfloor m (kA \mod 1) \rfloorh(k)=⌊m(kAmod1)⌋, where 0<A<10 < A < 10<A<1
- Example: Key = 123, m=10m = 10m=10, A=0.618A = 0.618A=0.618 → h(123)=9h(123) = 9h(123)=9

c) Mid-Square Method

- Square the key, extract **middle digits** as hash value.
- Example: Key = 123 → 1232=15129123^2 = 151291232=15129 → middle digits = 51

d) Folding Method

- Divide key into parts, add them, take modulo table size.
- Example: Key = 123456 → 12+34+56 = 102 → h(k)=102mod 10=2h(k) = 102 \mod 10 = 2h(k)=102mod10=2

e) Universal Hashing

- Randomly selects a hash function from a set to **reduce collisions**.
- Useful in **secure applications**.

Properties of a Good Hash Function

- **Uniformity:** Keys should be **evenly distributed**.
- **Deterministic:** Same key always maps to **same index**.
- **Efficient:** Computation should be **fast**.
- **Minimize Collisions:** Two keys rarely map to **same index**.
- **Defined Range:** Hash values must lie **within table size**.

---

3. **Explain two types of Leftis Tress**

Ans: A **Leftist Tree** is a **priority queue implemented using a binary tree**.

- It is **skewed to the left**, meaning the **shortest path to a null node is always on the right**.
- It is used for **efficient merge operations** of heaps.

1. Max Leftist Tree

Characteristics:

1. The **key of a node** is **greater than or equal to the keys of its children**.
2. **Right path is the shortest**, left path may be longer.
3. Useful for **quickly finding the maximum element**.

Operations:

- **Merge:** Combine two max leftist trees while maintaining the leftist property.
- **Insert:** Merge new node with existing tree.
- **Delete max:** Remove root (maximum) and merge its children.

Example:

```
      50
     /  \
   40    30
  /  \
20    10
```

- Root (50) is maximum.
- Right path (30 → null) is shortest.

2. Min Leftist Tree

Characteristics:

1. The **key of a node** is **less than or equal to the keys of its children**.
2. **Right path is shortest**, left path may be longer.
3. Useful for **quickly finding the minimum element**.

Operations:

- **Merge:** Combine two min leftist trees maintaining leftist property.
- **Insert:** Merge new node with existing tree.
- **Delete min:** Remove root (minimum) and merge its children.

Example:

```
    10
   /  \
  20   30
 / \
40  50
```

- Root (10) is minimum.
- Right path (30 → null) is shortest.