🟦 Skip-Liste (Skip List)

1️⃣ Grundidee

Eine **Skip-Liste** ist eine probabilistische Datenstruktur, die eine **sortierte Liste** durch mehrere zusätzliche Ebenen („Abkürzungen“) beschleunigt.

- Unterste Ebene: normale **verkettete Liste**
- Obere Ebenen: überspringen mehrere Elemente
- Ziel: ähnliche Performance wie balancierte Bäume, aber einfacher aufgebaut

➡️ Skip-Listen erreichen **erwartete O(log n)** für Suche, Einfügen und Löschen.

2️⃣ Voraussetzungen

- Elemente müssen **vergleichbar** sein (Ordnung <, >)
- Zufallsmechanismus zur Bestimmung der Ebenenhöhe eines Knotens
- Verkettete Listen als Grundstruktur

3️⃣ Laufzeiten & Eigenschaften

Eigenschaft	Wert
Suche (Average)	O(log n)
Einfügen (Average)	O(log n)
Löschen (Average)	O(log n)
Worst Case	O(n)
Speicherbedarf	O(n)
In-place	nein
Stabil	ja

Hinweis: Die logarithmische Laufzeit gilt **im Erwartungswert**, nicht garantiert.

4️⃣ Aufbau & Funktionsweise

Beispiel (vereinfacht, 3 Ebenen):

Ebene 2:

```
-∞ --------> 10 --------> 30 --------> +∞
```

Ebene 1:

```
-∞ ----> 10 ----> 20 ----> 30 ----> +∞
```

Ebene 0:

```
-∞ -> 5 -> 10 -> 15 -> 20 -> 30 -> 40 -> +∞
```

Suche erfolgt:

- von oben nach unten
- von links nach rechts

5️⃣ Schritt-für-Schritt-Beispiel (Suche)

Suche nach **20**:

1. Starte oben links
2. Gehe nach rechts, solange der nächste Wert ≤ 20 ist
3. Gehe eine Ebene tiefer
4. Wiederhole bis Ebene 0
5. Element gefunden

6️⃣ Besonderheiten / Prüfungsrelevante Hinweise

- Alternative zu AVL- oder Rot-Schwarz-Bäumen
- Einfacher zu implementieren als balancierte Bäume
- Häufig in Datenbanken und Key-Value-Stores
- Benötigt Zufall → probabilistische Garantie

7️⃣ Vor- und Nachteile

Vorteile

- erwartete O(log n)
- einfacher als balancierte Bäume
- stabil

- gute Performance in der Praxis

Nachteile

- Worst Case O(n)
- zusätzlicher Speicher für Ebenen
- Zufallsabhängigkeit

🧠 Merksatz für die Prüfung

Skip-Listen beschleunigen sortierte Listen durch zusätzliche Ebenen und erreichen erwartete O(log n) durch Zufall.

8️⃣ Python-Implementierung (vereinfachtes Lehrbeispiel)

In [1]:
```python
import random

MAX_LEVEL = 4
P = 0.5


class SkipNode:
    def __init__(self, value, level):
        self.value = value
        self.forward = [None] * (level + 1)


class SkipList:
    def __init__(self):
        self.level = 0
        self.header = SkipNode(None, MAX_LEVEL)

    def random_level(self):
        lvl = 0
        while random.random() < P and lvl < MAX_LEVEL:
            lvl += 1
        return lvl

    def insert(self, value):
        update = [None] * (MAX_LEVEL + 1)
        current = self.header

        for i in range(self.level, -1, -1):
            while current.forward[i] and current.forward[i].value < value:
                current = current.forward[i]
            update[i] = current
```

```
        lvl = self.random_level()

        if lvl > self.level:
            for i in range(self.level + 1, lvl + 1):
                update[i] = self.header
            self.level = lvl

        new_node = SkipNode(value, lvl)

        for i in range(lvl + 1):
            new_node.forward[i] = update[i].forward[i]
            update[i].forward[i] = new_node

    def search(self, value):
        current = self.header
        for i in range(self.level, -1, -1):
            while current.forward[i] and current.forward[i].value < value:
                current = current.forward[i]

        current = current.forward[0]
        return current and current.value == value


# Beispiel
sl = SkipList()
for v in [5, 10, 15, 20, 30]:
    sl.insert(v)

print(sl.search(20))  # True
print(sl.search(25))  # False
```

```
True
False
```