Kruskal-Algorithmus – Prüfungssheet (Abgrenzung zu BST)

Datenstrukturen & Algorithmen – Spickzettel

1️⃣ Grundidee des Kruskal-Algorithmus

Der **Kruskal-Algorithmus** berechnet einen **minimalen Spannbaum (MST)** eines **ungerichteten, gewichteten Graphen**.

Ziel:

- alle Knoten verbinden
- keine Zyklen
- **minimales Gesamtgewicht**

Kruskal arbeitet **kantenorientiert**: Er wählt immer die aktuell **günstigste Kante**, solange kein Zyklus entsteht.

2️⃣ Voraussetzungen

- ungerichteter Graph
- gewichtete Kanten
- Graph darf **nicht zwingend zusammenhängend** sein → Ergebnis ist dann ein **minimaler Spannwald**

3️⃣ Wichtige Abgrenzung: Kruskal vs. BST

Kruskal-Algorithmus	Binary Search Tree (BST)
Graph-Algorithmus	Baum-Datenstruktur
arbeitet mit Kanten	arbeitet mit Knoten
sortiert nach Kantengewicht	sortiert nach Schlüsselwert
Ziel: minimale Gesamtkosten	Ziel: effiziente Suche
kein Suchbaum	Suchbaum

👉 **Ein MST ist kein BST.** 👉 Kruskal benutzt **keinen BST**, sondern Sortierung + Mengenstruktur.

4️⃣ Algorithmus in Worten (prüfungsreif)

1. Betrachte alle Kanten des Graphen
2. Sortiere die Kanten **aufsteigend nach Gewicht**
3. Beginne mit einem leeren Spannbaum
4. Füge die nächste Kante hinzu, **wenn sie keinen Zyklus erzeugt**
5. Wiederhole, bis:
   - alle Knoten verbunden sind oder
   - genau **V − 1 Kanten** gewählt wurden

5️⃣ Zyklen vermeiden – Kerngedanke

Beim Hinzufügen einer Kante:

- verbindet sie zwei **verschiedene Komponenten** → erlaubt
- verbindet sie Knoten derselben Komponente → Zyklus → **verwerfen**

Typisch gelöst mit:

- **Union-Find / Disjoint Set**

6️⃣ Wichtige Eigenschaften

- Greedy-Algorithmus
- arbeitet global über alle Kanten
- MST enthält genau **V − 1 Kanten**
- eindeutig bei verschiedenen Kantengewichten

7️⃣ Typische Prüfungsfalle

❌ Falsch: „Kruskal berechnet kürzeste Wege."

✅ Richtig: Kruskal minimiert die **Gesamtkosten des Netzes**, nicht einzelne Pfade.

Kürzeste Wege:

- ungewichtet → BFS
- gewichtet → Dijkstra

8️⃣ Datenstrukturen bei Kruskal

Typisch verwendet:

- sortierte Kantenliste
- **Union-Find (Disjoint Set)** zur Zyklenerkennung

❗ Kein BST notwendig

9️⃣ Python-Beispiel (Kruskal mit Union-Find)

```python
class UnionFind:
    def __init__(self, n):
        self.parent = list(range(n))

    def find(self, x):
        if self.parent[x] != x:
            self.parent[x] = self.find(self.parent[x])
        return self.parent[x]

    def union(self, a, b):
        ra, rb = self.find(a), self.find(b)
        if ra != rb:
            self.parent[rb] = ra
            return True
        return False


def kruskal(edges, n):
    uf = UnionFind(n)
    mst = []

    edges.sort(key=lambda e: e[2])  # (u, v, gewicht)

    for u, v, w in edges:
        if uf.union(u, v):
            mst.append((u, v, w))

    return mst
```

🔟 Mini-Beispiel (gedanklich)

Kanten (sortiert): A–B (1) B–C (2) B–D (3) A–C (4) C–D (5)

Auswahl:

- A–B ✓
- B–C ✓
- B–D ✓
- A–C ✗ (Zyklus)

Gesamtgewicht minimal.

🔟 Merksätze für die Prüfung

> Kruskal = sortiere Kanten, prüfe Zyklen Kruskal ≠ kürzester Weg Kruskal ≠ BST

> Prim wächst vom Knoten, Kruskal wächst von den Kanten.

1️⃣ 1️⃣ Ultra-Kurz-Zusammenfassung

- Kruskal arbeitet kantenbasiert
- Greedy-Verfahren
- nutzt Sortierung + Union-Find
- minimiert Gesamtkosten
- kein Bezug zu BST