

# Minimaler Spannbaum (Minimum Spanning Tree, MST)

## 1 Grundidee

Ein **minimaler Spannbaum (MST)** ist ein Teilgraph eines **ungerichteten, gewichteten Graphen**, der:

- alle Knoten verbindet
- keine Zyklen enthält
- die **Summe der Kantengewichte minimiert**

→ Ein Spannbaum mit minimalen Gesamtkosten.

---

## 2 Voraussetzungen

- **Ungerichteter Graph**
  - **Gewichtete Kanten**
  - Der Graph muss **zusammenhängend** sein (sonst erhält man einen minimalen Spannwald)
- 

## 3 Eigenschaften & Laufzeiten (Übersicht)

Eigenschaft	Wert
Anzahl Kanten	$V - 1$
Zyklen	keine
Gewicht	minimal
Existenz	eindeutig, wenn alle Gewichte verschieden sind

---

## 4 Typische Algorithmen für MST

### ◆ Prim-Algorithmus

- Greedy-Algorithmus
- Start bei einem Knoten
- wächst den Baum schrittweise

## ◆ Kruskal-Algorithmus

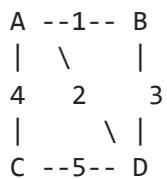
- Greedy-Algorithmus
- sortiert alle Kanten
- fügt Kanten hinzu, wenn kein Zyklus entsteht

→ Beide liefern **denselben minimalen Spannbaum** (bei gleichen Gewichten evtl. andere Struktur).

---

## 5 Beispiel

Graph:



Minimaler Spannbaum:

- A–B (1)
- B–D (3)
- A–C (4)

Gesamtgewicht = **8**

## Minimaler Spannbaum – Algorithmus in Worten (Grundidee)

Ein minimaler Spannbaum ist ein Teilgraph eines ungerichteten, gewichteten Graphen.

Er verbindet alle Knoten des Graphen.

Dabei enthält er keine Zyklen.

Die Summe der Kantengewichte ist minimal.

Ein minimaler Spannbaum enthält bei  $V$  Knoten genau  $V - 1$  Kanten.

## Prim-Algorithmus – Algorithmus in Worten

Der Algorithmus beginnt bei einem beliebigen Startknoten.

Dieser Knoten wird in den Spannbaum aufgenommen.

Anschließend werden alle Kanten betrachtet,

die von bereits aufgenommenen Knoten zu noch nicht aufgenommenen Knoten führen.

Die Kante mit dem kleinsten Gewicht wird ausgewählt.

Der zugehörige neue Knoten wird in den Spannbaum aufgenommen.

Dieser Vorgang wird wiederholt,

bis alle Knoten im Spannbaum enthalten sind.

## Kruskal-Algorithmus – Algorithmus in Worten

Der Algorithmus betrachtet zunächst alle Kanten des Graphen.  
Diese Kanten werden aufsteigend nach ihrem Gewicht sortiert.  
Danach werden die Kanten der Reihe nach betrachtet.  
Eine Kante wird nur dann in den Spannbaum aufgenommen,  
wenn sie keinen Zyklus erzeugt.  
Dieser Vorgang wird fortgesetzt,  
bis der Spannbaum genau  $V - 1$  Kanten enthält.

---

## 6 Besonderheiten / Prüfungsrelevante Hinweise

- MST ist **nicht** dasselbe wie kürzester Weg
  - MST minimiert **Gesamtgewicht**, nicht einzelne Pfade
  - Nur für **ungerichtete Graphen**
  - Sehr häufige Prüfungsfrage:
    - Unterschied MST vs. Dijkstra?
- 

## 7 Vor- und Nachteile

### Vorteile

- optimale Verbindung aller Knoten
- nützlich für Netzwerke (Kabel, Straßen, Leitungen)

### Nachteile

- keine Pfadinformationen
  - nicht anwendbar auf gerichtete Graphen
- 

## Merksatz für die Prüfung

*Ein minimaler Spannbaum verbindet alle Knoten eines ungerichteten Graphen ohne Zyklen und mit minimalem Gesamtgewicht.*

---

## 8 Python-Implementierung (Prim-Algorithmus)

```
In [ ]:
```

```
import heapq

def prim(graph, start):
    visited = set()
    mst = []
    pq = [(0, start, None)] # (gewicht, knoten, vorgänger)

    while pq:
        weight, node, parent = heapq.heappop(pq)
        if node in visited:
            continue

        visited.add(node)
        if parent is not None:
            mst.append((parent, node, weight))

        for neighbor, w in graph[node]:
            if neighbor not in visited:
                heapq.heappush(pq, (w, neighbor, node))

    return mst

# Beispiel
graph = {
    "A": [("B", 1), ("C", 4)],
    "B": [("A", 1), ("D", 3), ("C", 2)],
    "C": [("A", 4), ("B", 2), ("D", 5)],
    "D": [("B", 3), ("C", 5)]
}

print(prim(graph, "A"))
```