

# Greedy - Algorithmen

Seminarvortrag

17.06.2008

John Freytag

# Übersicht

- Einführung in das Thema
  - Das "Kassierer-Problem"
- Bestimmen eines minimalen Spannbaums
  - Kurze Wiederholung der Graphentheorie
  - Der Algorithmus von Kruskal
    - Union-Find – Problem
  - Der Algorithmus von Prim

# Einführung in das Thema

”Greedy”-Algorithmen: Lösungen von Optimierungsproblemen

- Erreichen des Optimums durch schrittweise Wahl von lokalen Optima
- Schritte sind irreversibel
- Verfahren kann auch für Approximations-Algorithmen verwendet werden



# Beispiel

- Kassierer-Problem:
  - Auszahlen des Wechselgeldes mit so wenig Geldeinheiten wie möglich



# Graphentheorie

- Ungerichteter, gewichteter Graph:
  - Menge von Knoten (Vertices)
  - Knoten verbunden durch gewichtete Kanten (Edges)
    - Kantengewicht => "Kosten der Kante"
    - Kanten gelten für beide Richtungen
- Baum:
  - Graph ohne Zyklen, d.h. schreitet man den Graphen in einer Richtung ab, kann man an einem Knoten nie zweimal vorbei kommen.
- Spannbaum:
  - Baum zu einem Graphen, der alle Knoten des Graphens enthält

# Algorithmus von Kruskal

- Finden eines minimalen Spannbaums
  - Spannbaum ist minimal, wenn Summe der Kantengewichte minimal ist
  - Idee:
    - Kanten nach Gewicht sortieren
    - Schrittweise alle Kanten in aufsteigender Reihenfolge betrachten:
      - Führt die Kante zu einem Zyklus → Verwerfen
      - Wenn kein Zyklus → Kante übernehmen
- Beispiel

# Algorithmus von Kruskal

- Problem:
  - Für einen Menschen ist es eindeutig, wenn eine zusätzliche Kante einen Zyklus bilden würde.
  - Aber wie lässt sich das algorithmisch effizient lösen?
  - → Union-Find - Problem

# Union-Find – Problem

- **Gedanke:**
  - Aus jedem Knoten des Original-Graphens einen Sub-Baum bilden
  - Sind die Knoten an einer Kante nun in verschiedenen Sub-Bäumen, kann es durch diese Kante keinen Zyklus geben  
→ Kante übernehmen und beide Sub-Bäume zu einem vereinigen
  - Sonst: Kante verwerfen
- **Zwei Teilprobleme:**
  - Find: Welcher Sub-Baum enthält den aktuell betrachteten Knoten?
  - Union: Wie soll das Zusammenlegen zweier Sub-Bäume realisiert werden?

# Union-Find – Problem

- Effiziente Implementierung:
  - Sub-Bäume werden durch "kanonischen Knoten" (Wurzel) repräsentiert und als (**Daten-**)Baum-Struktur gespeichert
  - Baum-Struktur wird durch ein einzelnes Array erzeugt, das jedem Knoten im Graphen dessen Vater-Knoten zuweist
  - Ist ein Knoten sein eigener Vaterknoten, so ist er das Wurzelement und somit der kanonische Knoten des Sub-Baums
- Beispiel

# Union-Find – Problem

- Union von Sub-Bäumen:
  - Der Wurzelknoten des einen Baumes wird an den Wurzelknoten des anderen angehängt
  - Wird der anzuhängende Baum willkürlich gewählt, können degenerierte Bäume auftreten
  - Optimierungsmöglichkeit:
    - Wahl nach Größe (Anzahl Knoten) oder Höhe (Anzahl Knoten des längsten Weges)
    - Kleineren Baum an größeren Baum anhängen
    - Somit haben Bäume mit Höhe  $h$  wenigstens  $2^h$  Knoten

# Pseudocode

Zwei Arrays, deren Größe der Anzahl an Knoten im Graphen entspricht:

Vertex[] **parentMap** ← Weist einem Vertex seinen Vater im **Daten-Baum** zu  
int[] **size** ← Hält zu jedem kanonischen Knoten (also zu jedem Sub-Baum) dessen Größe. **Initial sind alle Werte 1.**

```
Vertex find (Vertex v) {  
    Vertex currentVertex = v;  
    while (parentMap[currentVertex] != currentVertex) {  
        currentVertex = parentMap[currentVertex];  
    }  
    return currentVertex;  
}
```

# Pseudocode

// **a** und **b** sind hier kanonische Knoten,  
// repräsentieren also einen Sub-Baum

```
void union(Vertex a, Vertex b) {  
    if ( size[ a ] < size[ b ] )  
        parentMap[ a ] = b;  
    else  
        parentMap[ b ] = a;  
}
```

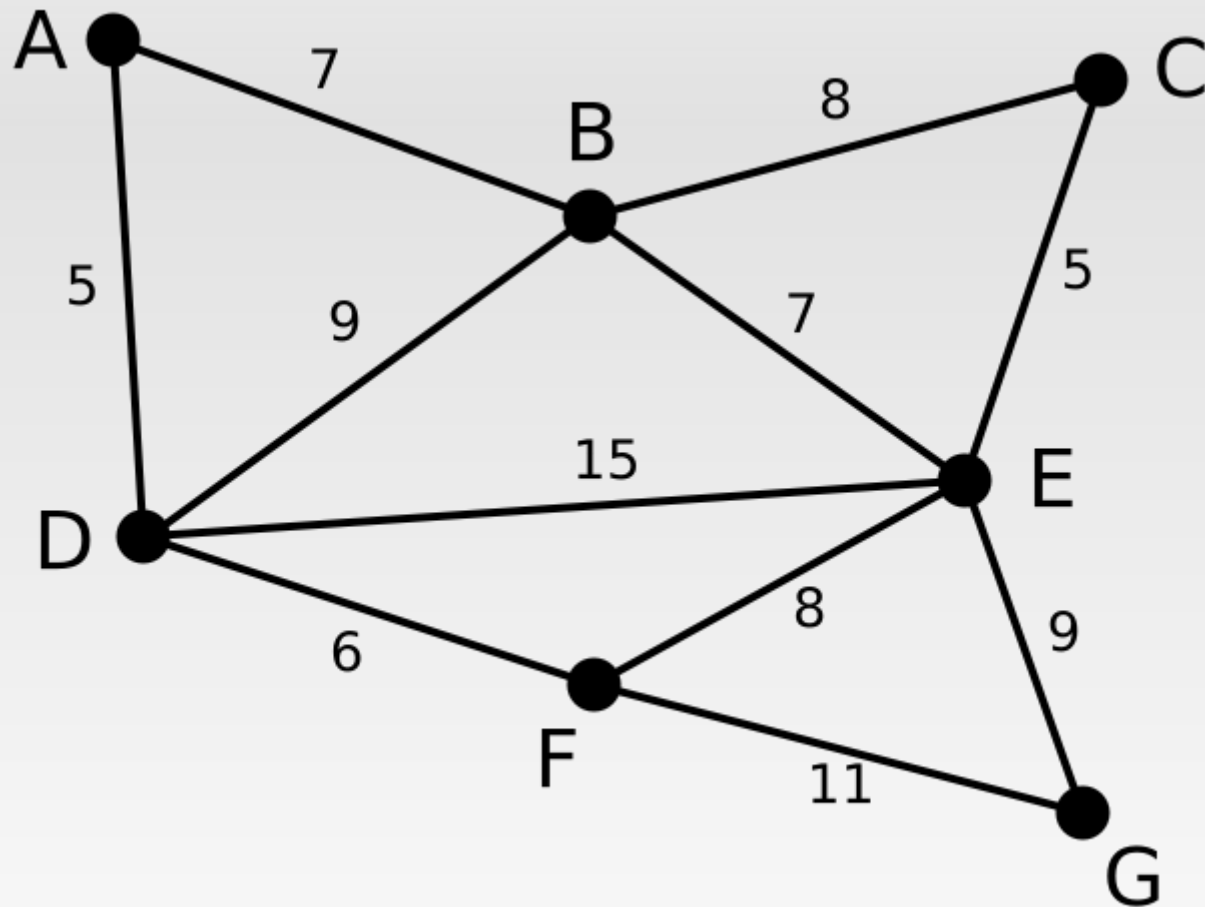
# Laufzeitanalyse

- Größen:
  - $e$ : Anzahl der Kanten des Originalgraphen (Edges)
  - $v$ : Anzahl der Knoten (Vertices)
  - $n$ : Anzahl der Knoten in einem Teilbaum
- Teilprobleme:
  - Sortieren der Kanten:  $O(\log e)$
  - Finden der Teilbäume:  $O(v \log n)$
  - Union-Operationen:  $O(v)$
- Ergibt zusammen:
  - $O(e \log v)$

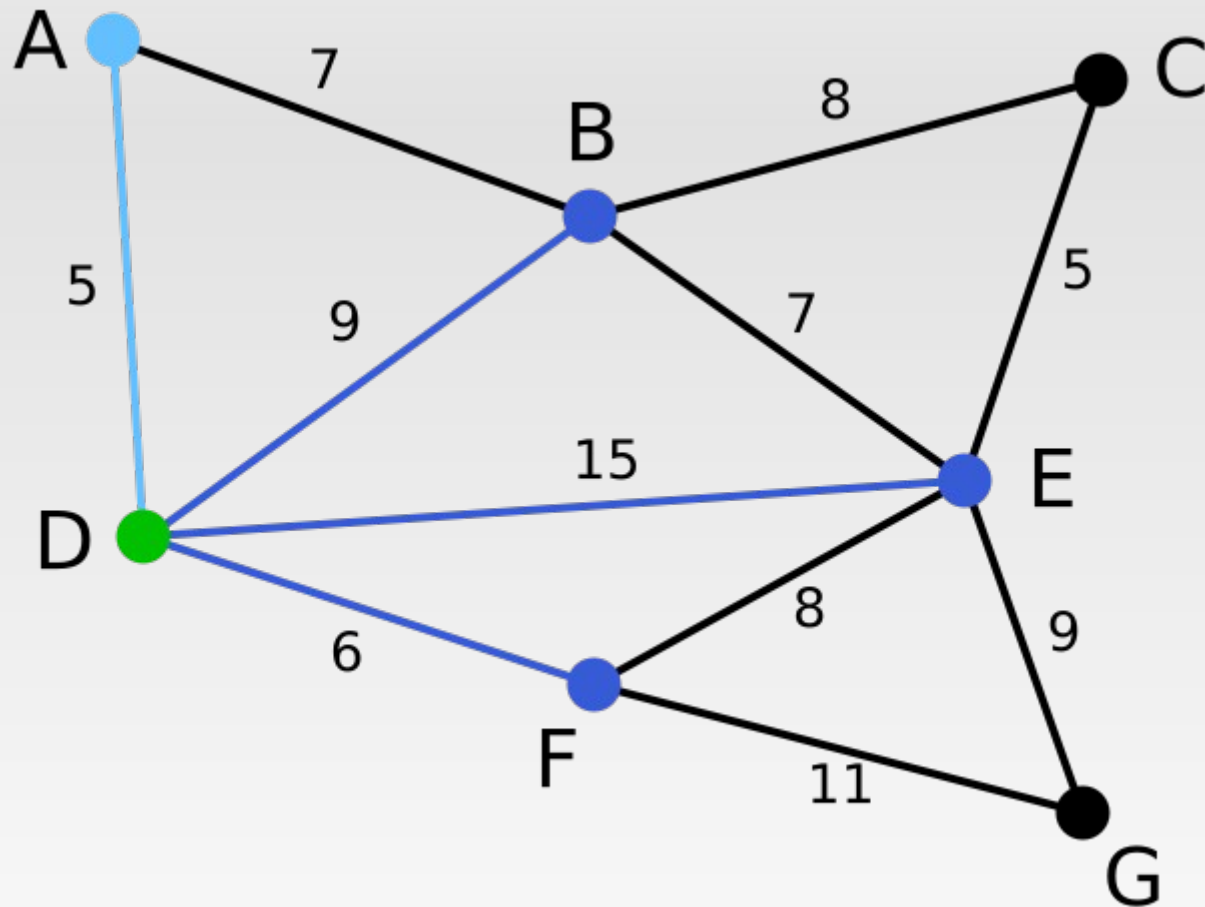
# Algorithmus von Prim

- Eigentlich von Vojtěch Jarník, 1957 von Robert C. Prim und 1959 von Edsger Dijkstra wiederentdeckt.
- Idee: Starte mit einem Knoten und lasse ihn zu min. Spannbaum wachsen
- Starte mit einem beliebigen Knoten als Start-Baum  $T$
- Solange in  $T$  noch nicht alle Knoten enthalten sind:
  - Wähle Kante  $e$  mit minimalen Gewicht, die  $T$  mit einem Knoten  $v$  verbindet, der noch nicht in  $T$  ist.
  - Füge  $e$  und  $v$  zu  $T$  hinzu

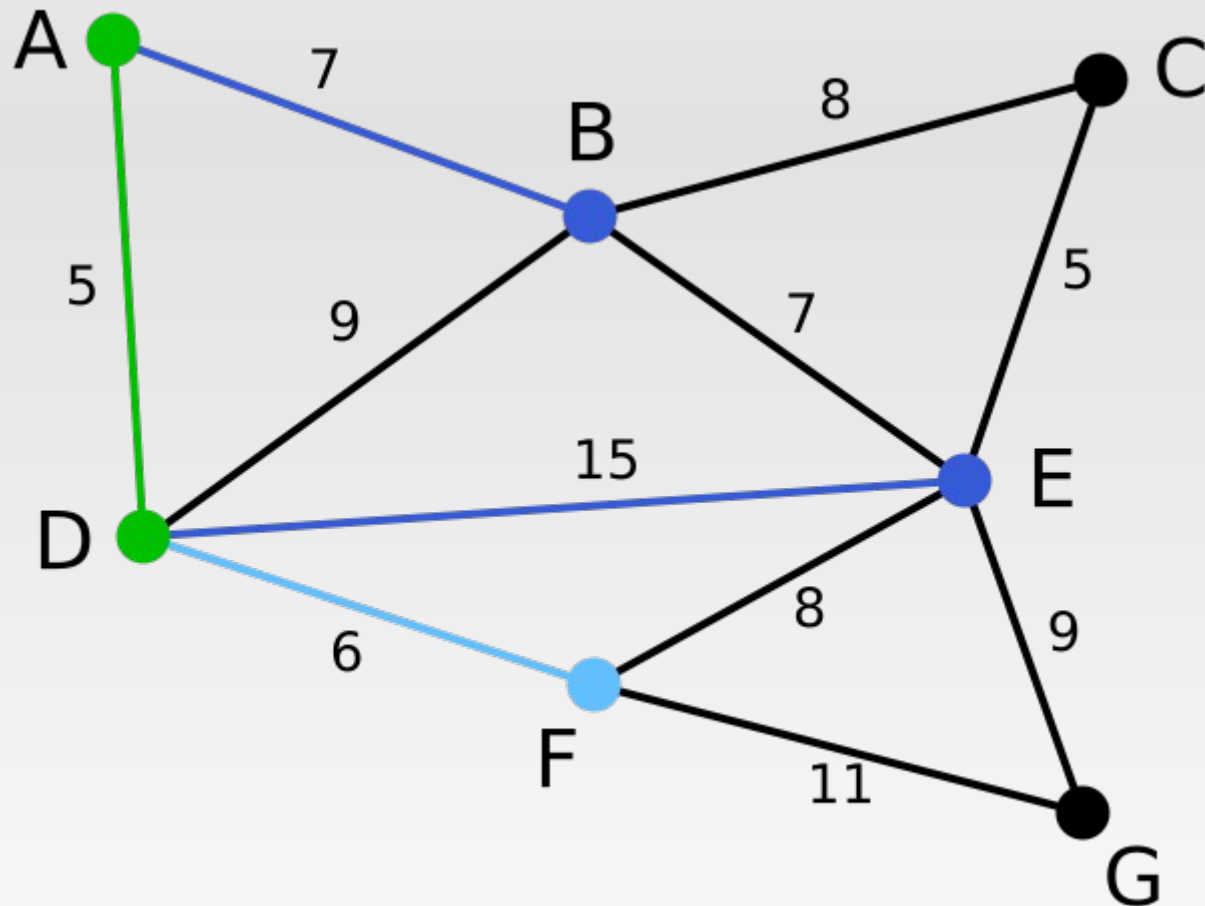
# Algorithmus von Prim: Beispiel



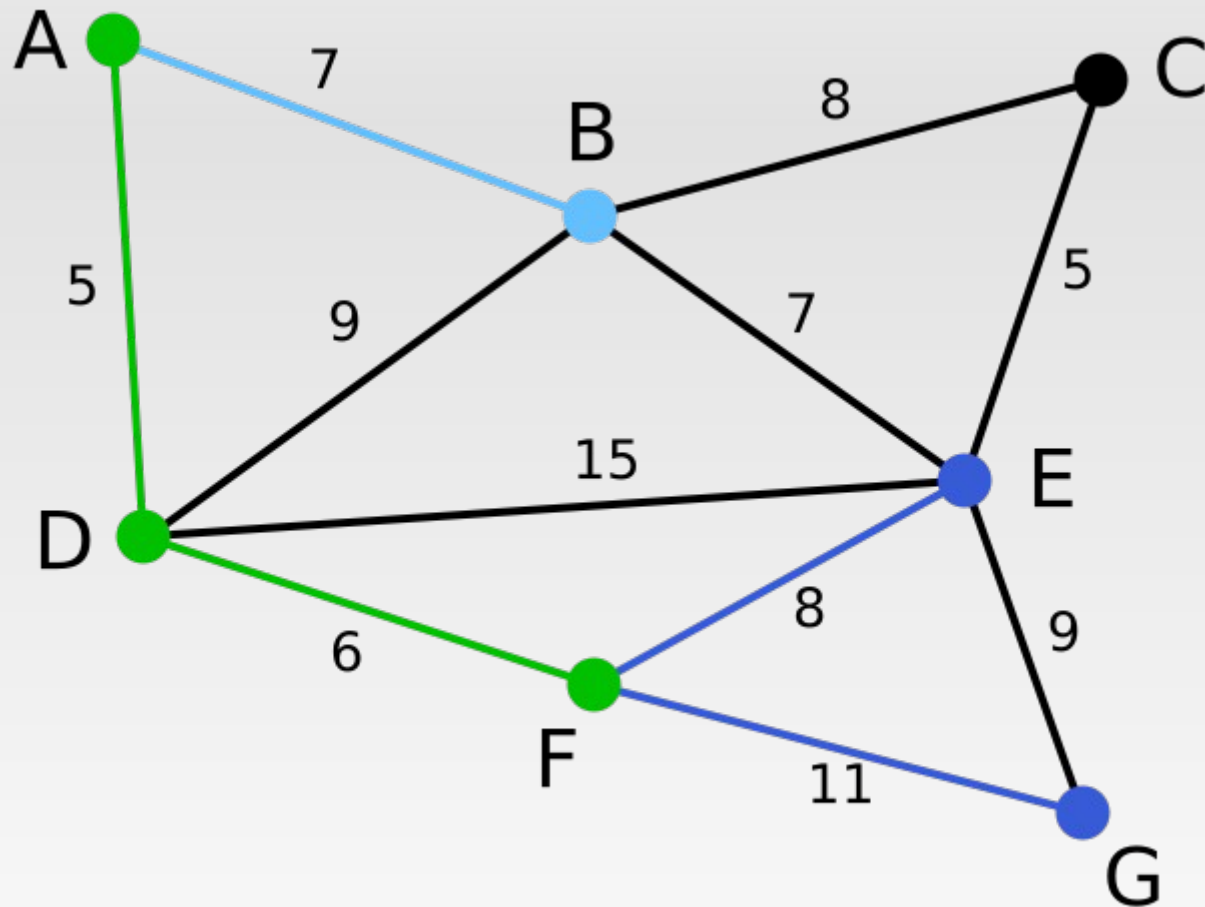
# Algorithmus von Prim: Beispiel



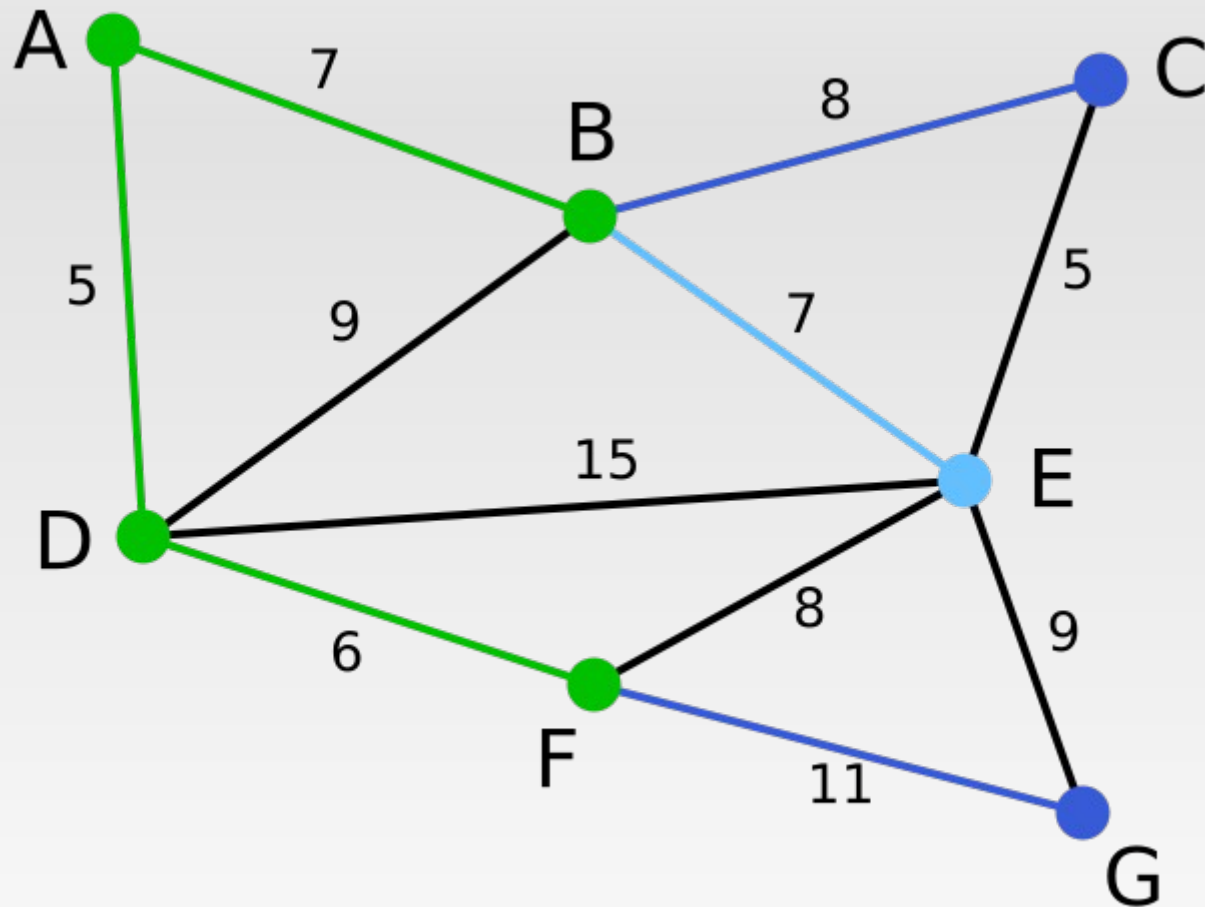
# Algorithmus von Prim: Beispiel



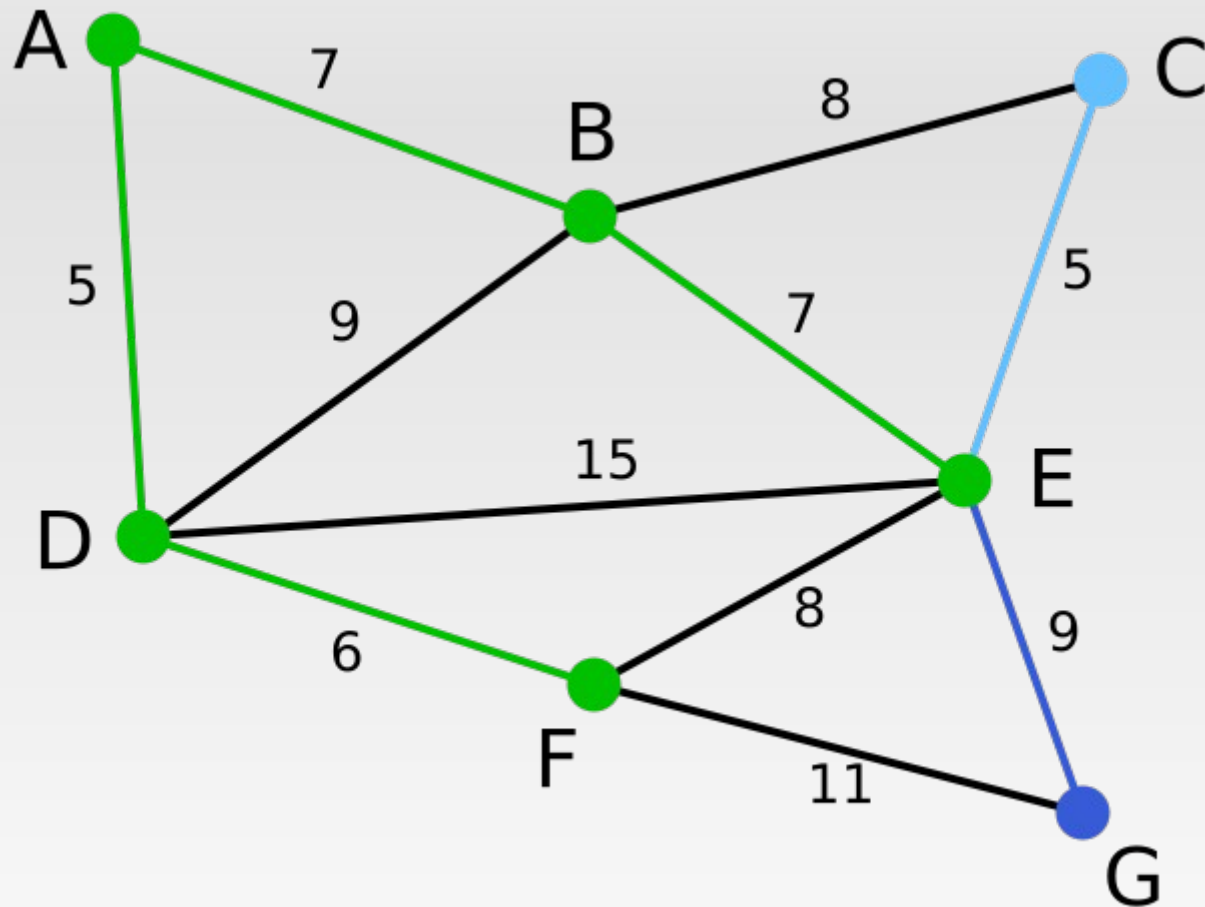
# Algorithmus von Prim: Beispiel



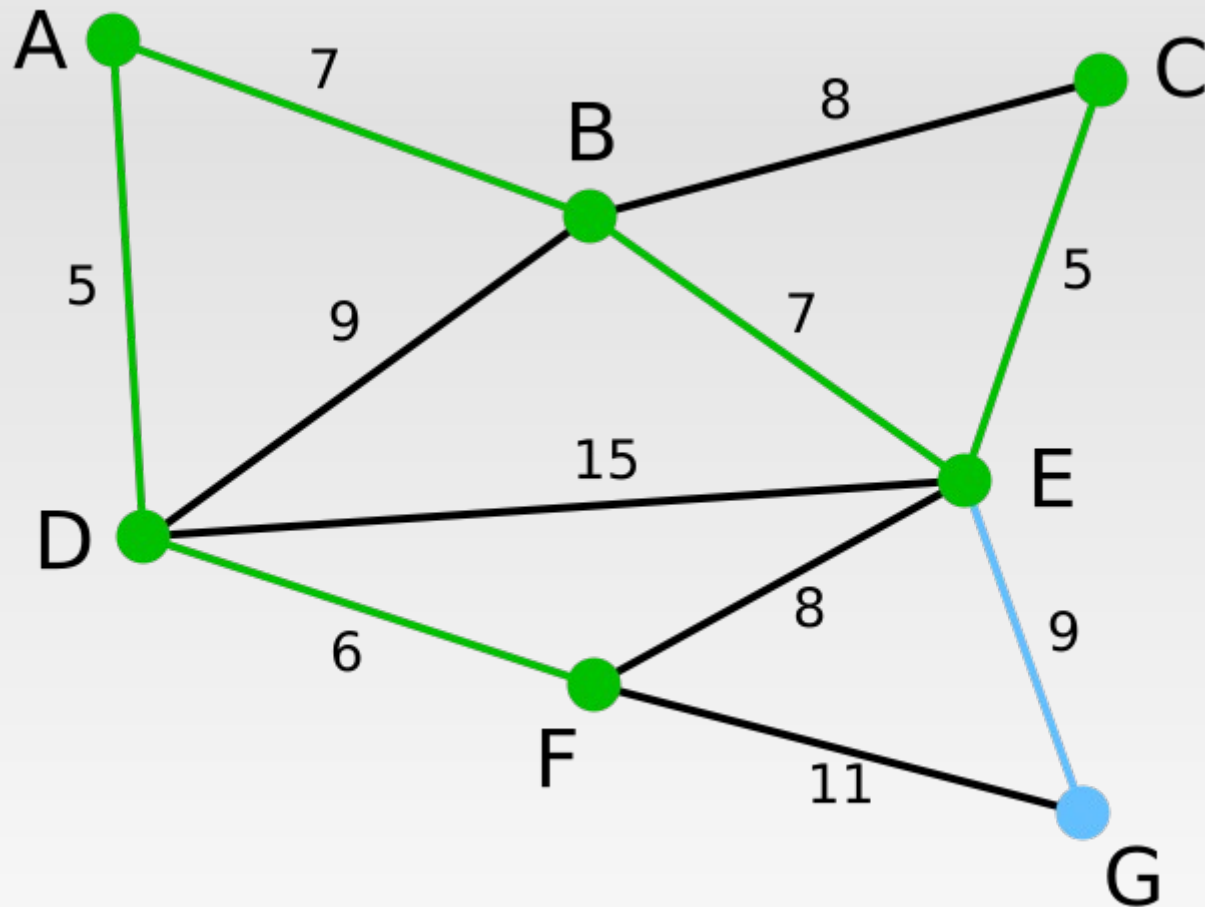
# Algorithmus von Prim: Beispiel



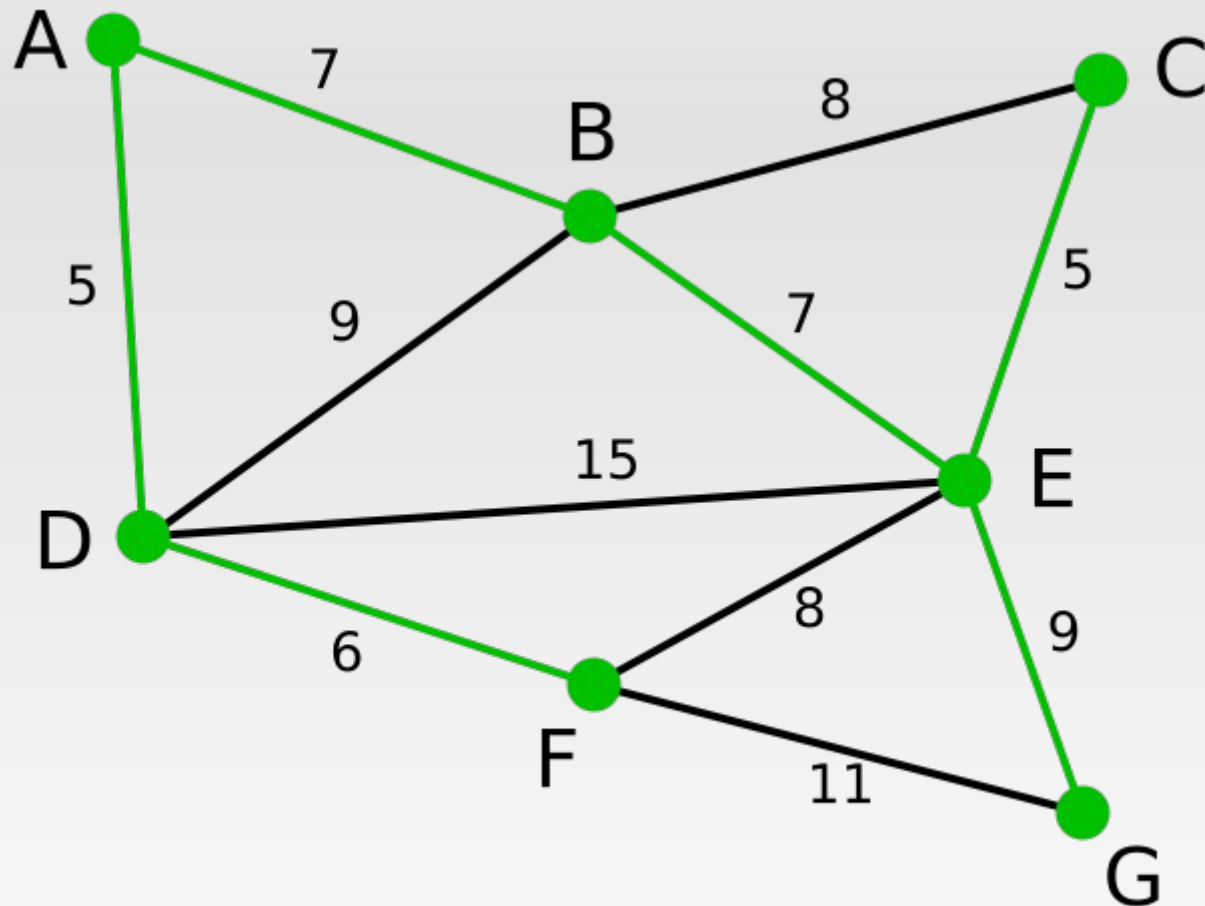
# Algorithmus von Prim: Beispiel



# Algorithmus von Prim: Beispiel



# Algorithmus von Prim: Beispiel



# Algorithmus von Prim

- Effiziente Implementierung:
  - Speicherung des Graphens als Angrenzungs-Matrix (Adjacency matrix), um effizient alle an einen Knoten angrenzenden Knoten abrufen zu können
  - Sortierung der noch nicht in  $T$  aufgenommenen Knoten in einer Priority Queue
    - Sortiergewicht ist das Gewicht der minimalen Kante, mit dem sich der Knoten  $v$  zu  $T$  verbinden lässt oder unendlich, falls keine solche Verbindung besteht. Zusätzlich wird die verbindende Kante gespeichert
    - Wird ein Knoten hinzugefügt, werden alle angrenzenden Knoten angepasst:
      - Ist ihr Sortiergewicht größer als das Gewicht  $w$  der Kante zu  $v$ , dann setze Sortiergewicht auf  $w$  und speichere neue Kante am Eintrag
    - So werden immer nur Knoten mit Informationen versorgt, die auch für  $T$  in Frage kommen!

# Laufzeitanalyse

- Operationen auf der Priority-Queue (Entfernung des Kopfes, Einfügen, Ändern der Priorität:  $O(\log V)$ )
- $V - 1$  Löschungen werden durchgeführt
- $E$  Überprüfungen und mögliche Änderungen der Prioritäten
  - $(V - 1 + E) O(\log V) = \mathbf{O(E \log V)}$

# Ende des Vortrages

Danke für Eure Aufmerksamkeit!

Noch Fragen?