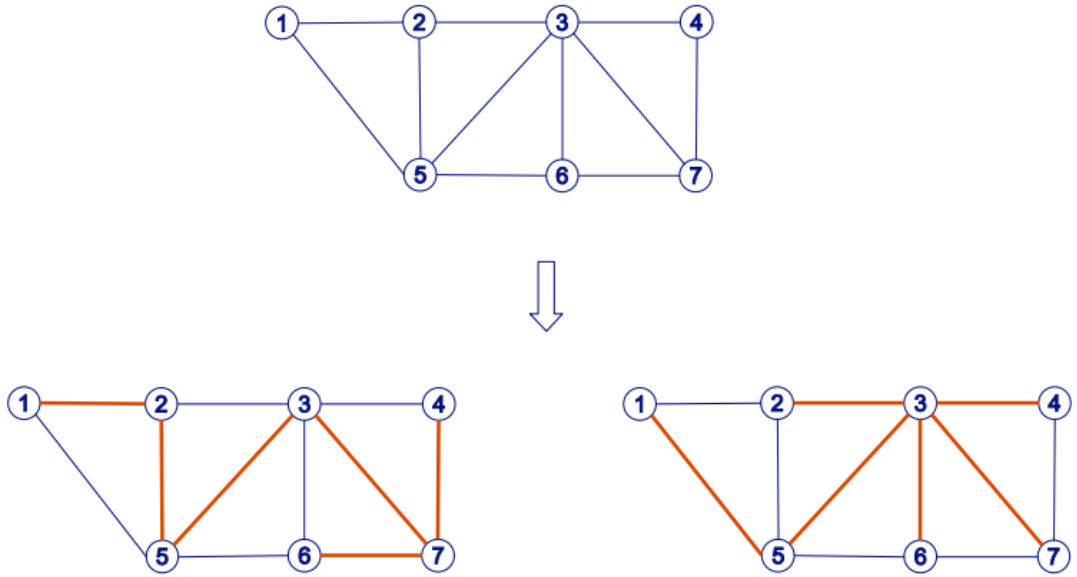
Graphentheorie II

Kantonsschule Sursee
Dr. Philipp Hurni

Agenda

- Repetition Graph und dessen Darstellung
- Einfache Graph-Algorithmen
- Der Minimum Spanning Tree Algorithmus (Algorithmus von Prim)
- Übungen mit Algorithmus von Prim
  - "Manuell"
  - Mit Python

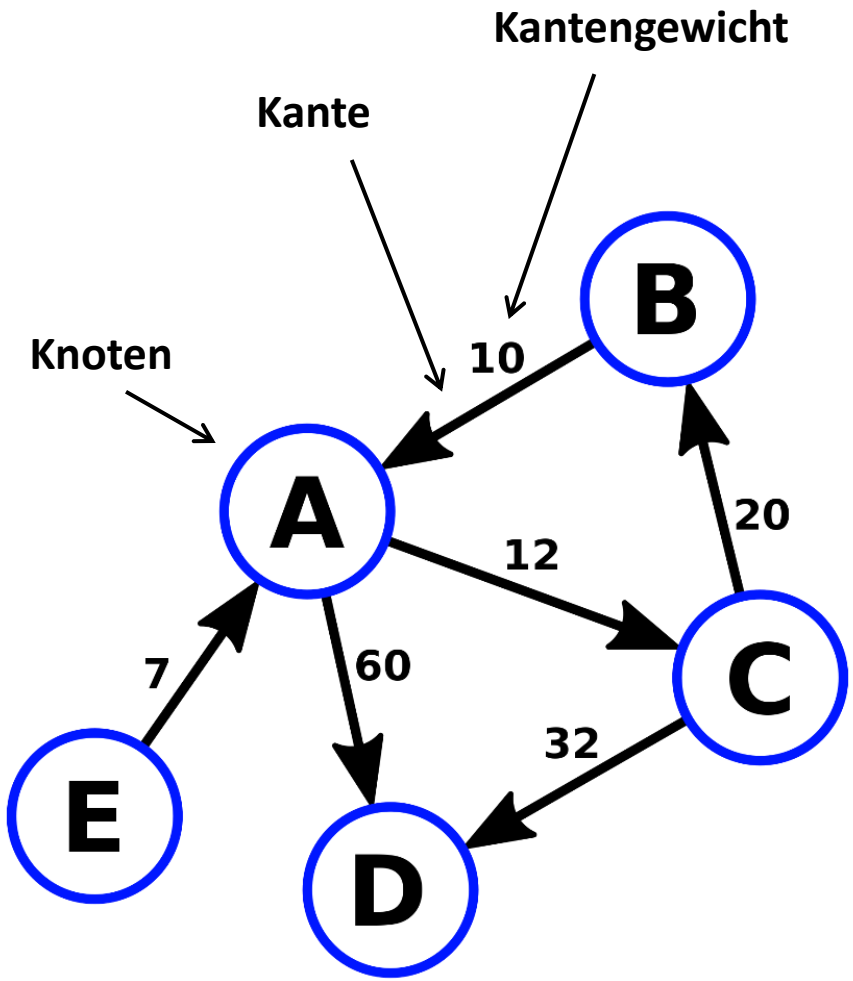


Was ist ein Graph?

Definition (Wikipedia):

Ein Graph ist eine abstrakte Struktur, die eine Menge von Objekten zusammen mit den zwischen diesen Objekten bestehenden Verbindungen repräsentiert.

- Die Objekte werden dabei **Knoten** des Graphen genannt.
- Die paarweisen Verbindungen zwischen Knoten heissen **Kanten**



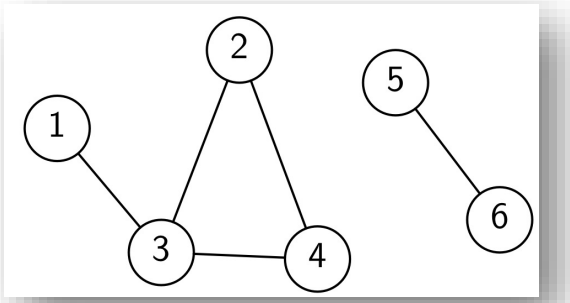
Darstellung eines Graphen – mit Mengen

Mathematisch formale Beschreibung:

Ein Graph besteht aus:

- einer Menge von Knoten **V** (engl. Vertices)
- einer Menge von Kanten **E** (engl. Edges)

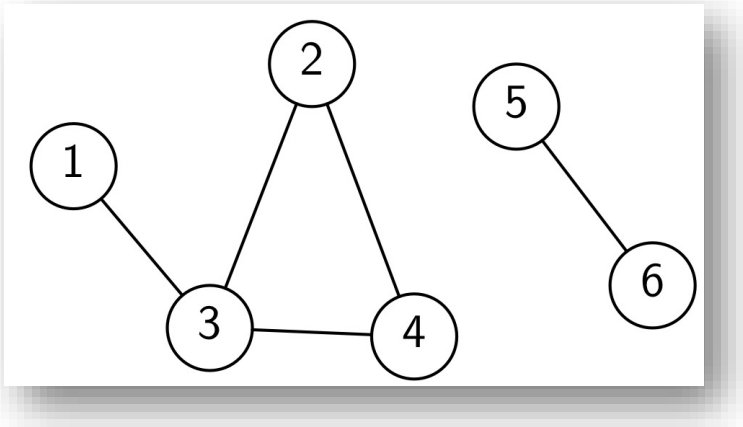
Datenstruktur für einen Graphen



Anmerkung: hier ist der Graph ungerichtet – das heisst, eine Verbindung von 1 nach 3 bedeutet auch eine Verbindung von 3 nach 1. Das kann man auch anders definieren.

$$V = \{1,2,3,4,5,6\}$$
$$E = \{(1,3); (2,3); (2,4); (3,4); (5,6)\}$$

Darstellung eines Graphen – mit Adjazenzliste/matrix



Adjazenzliste

1: 3
2: 3,4
3: 1,2,4
4: 2,3
5: 6
6: 5

2D-Liste. Für jeden Knoten ist gespeichert, zu welchem Knoten er eine Kante besitzt

Adjazenzmatrix

0	0	1	0	0	0
0	0	1	1	0	0
1	1	0	1	0	0
0	1	1	0	0	0
0	0	0	0	0	1
0	0	0	0	1	0

n x n Matrix mit Werten falls Werte

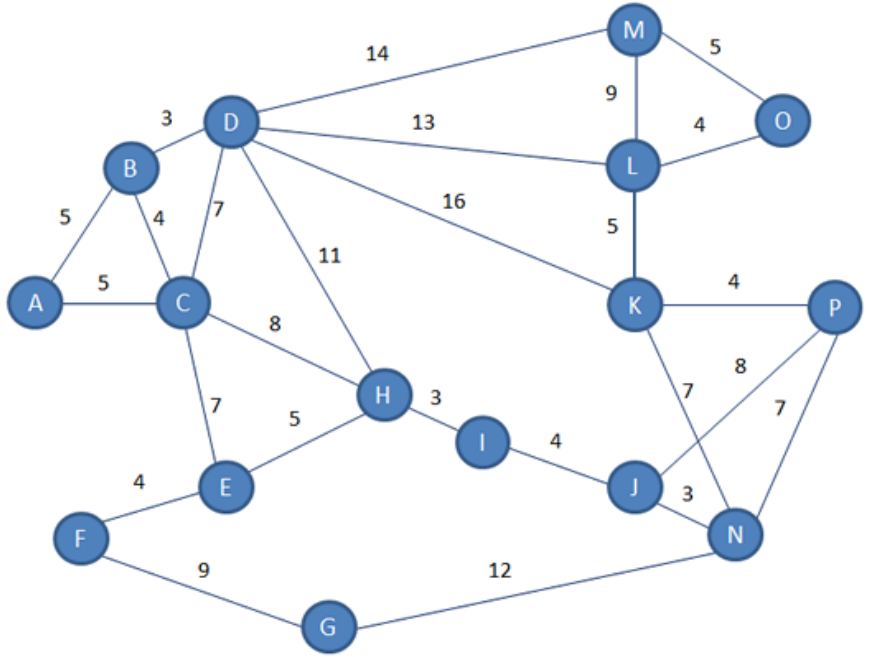
- >0 → es gibt eine Kante
- 0 → keine Kante)

Einfache Graph Algorithmen

Was ist ein Graph-Algorithmus?

- Ein Graph-Algorithmus ist eine Reihe von Anweisungen, die darauf abzielen, ein Problem zu lösen, das sich auf Graphen bezieht.

- Ein Graph-Algorithmus nimmt diesen Graphen als Eingabe und führt spezifische Operationen durch. Ein Graph-Algorithmus ist daher ein Rezept, um sinnvolle Informationen aus einem Graphen zu extrahieren oder Probleme innerhalb dieses Netzwerks zu lösen.



Übung D: Erkennung "Gerichteter Graph"

Ausgehend von einer Adjazenzmatrix, entwickle einen Algorithmus zur Erkennung eines gerichteten Graphes

1. Schritt – beschreibe den Algorithmus mittels "Pseudocode"

2. Schritt: vervollständige das Python Programm Aufgabe D1

Adjazenzmatrix

$$\begin{matrix} 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 \end{matrix}$$

$n \times n$ –Matrix mit Werten
(>=1 → Kante, 0 → keine Kante)

Lösung D

1. Algorithmus in Pseudocode

Sei $V = \{1,2,3,4,5,6\}$ die Menge der Knoten, A die Adjazenzmatrix mit 6 Zeilen und 6 Spalten.

Gehe durch i=0…6 und j=0…6. Falls alle Felder A[i][j] jeweils A[j][i] gleich sind, haben wir einen ungerichteten Graphen. Falls in 1 oder mehr Fällen diese Bedingung nicht stimmt, ist der Graph gerichtet.

Adjazenzmatrix

0	0	1	0	0	0
0	0	1	1	0	0
1	1	0	1	0	0
0	1	1	0	0	0
0	0	0	0	0	1
0	0	0	0	1	0

$n \times n$ −Matrix mit Werten
(>=1 → Kante, 0 → keine Kante)

Lösung D

siehe Graphentheorie_Aufgabe_D - Loesung.py

2. Schritt: Python Programm Aufgabe D

```
matrix = [[0,5,7,4,0,0,0],
          [5,0,3,0,9,0,0],
          [7,3,0,5,0,8,1],
          [4,0,5,0,0,3,0],
          [0,9,0,0,0,0,4],
          [0,0,8,3,0,0,1],
          [0,0,1,0,3,1,0]]

# wir brauchen sogenannte "Pandas"-Dataframes
AdjMatrix = pandas.DataFrame(matrix, index=labels, columns=labels)

asymmetrie_gefunden = False
for i in range(len(matrix)):
    for j in range(len(matrix[i])):
        if(matrix[i][j] != matrix[j][i]):
            asymmetrie_gefunden = True
            print("matrix[", i, "][", j, "] != matrix[", i, "][", j, "]")
```

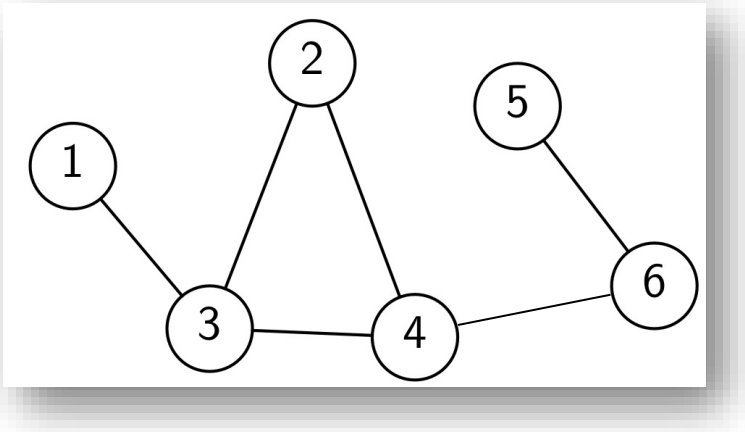
Adjazenzmatrix

0	0	1	0	0	0
0	0	1	1	0	0
1	1	0	1	0	0
0	1	1	0	0	0
0	0	0	0	0	1
0	0	0	0	1	0

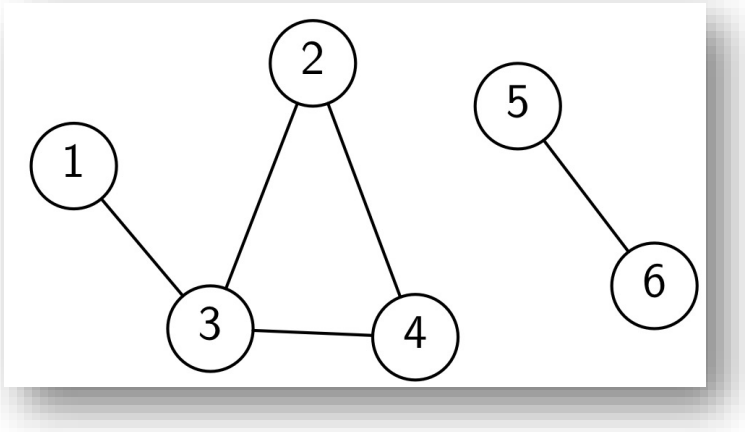
n x n - Matrix mit Werten
(>=1 → Kante, 0 → keine Kante)

Übungen E, F - Weitere einfache Graph Algorithmen

Aufgabe E: zusammenhängende Graphen erkennen
Aufgabe F: zyklische Graphen erkennen



Zyklischer Graph, zusammenhängend



Zyklischer Graph, nicht zusammenhängend

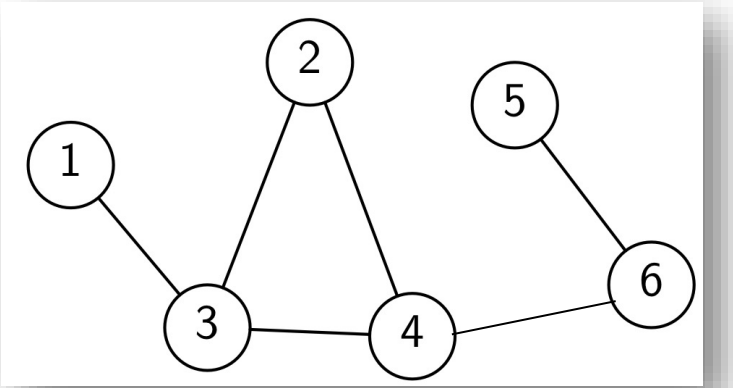
Lösung E

Algorithmus in Pseudocode

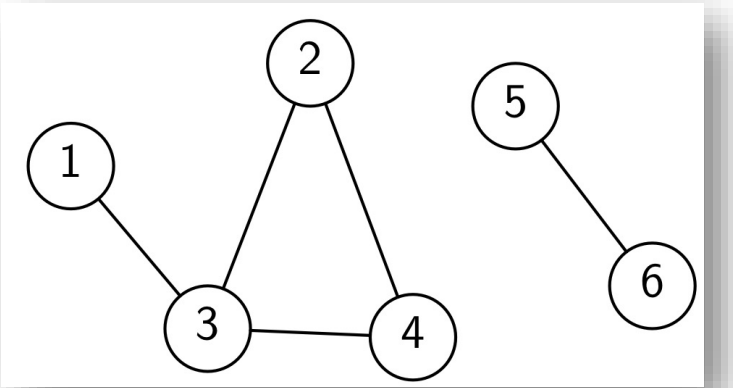
1. Startpunkt festlegen: Der Algorithmus beginnt bei einem beliebigen Knoten (hier ist es Knoten 1).

2. Besuchte Knoten verfolgen: Es wird eine Liste geführt, die alle Knoten enthält, die bereits "besucht" wurden, also von denen aus man andere Knoten erreicht hat. Am Anfang enthält diese Liste nur den Startknoten.
   - Erkundung der Nachbarn: Der Algorithmus geht nun schrittweise vor:
   - Er nimmt sich den nächsten Knoten aus der Liste der besuchten Knoten.
   - Er sucht alle direkten Nachbarn dieses Knotens (also alle Knoten, mit denen er direkt verbunden ist und deren Wert in der Matrix nicht Null ist).
   - Wenn ein Nachbar noch nicht in der Liste der besuchten Knoten ist, wird er der Liste hinzugefügt. Das zeigt an, dass der "erkundete" Bereich des Graphen wächst.

3. Wiederholen bis keine neuen Knoten gefunden werden: Dieser Prozess wird wiederholt, solange neue, unbesuchte Knoten gefunden werden können. Sobald keine neuen Knoten mehr erreicht werden können, stoppt der Prozess.



Zyklischer Graph, zusammenhängend



Zyklischer Graph, nicht zusammenhängend

Lösung F

Algorithmus in Pseudocode

1. Graph-Darstellung: Der Graph wird als Adjazenzmatrix definiert.

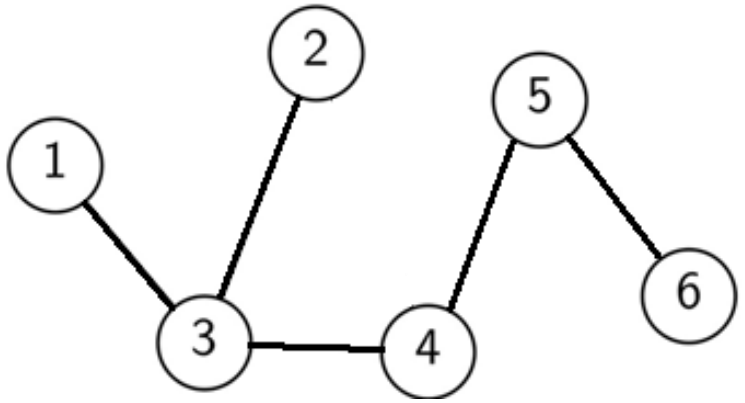
Startpunkt & Verfolgung: Der Algorithmus startet bei Knoten 0 und merkt sich die bereits besuchten Knoten.

Suche nach Zyklen: Er durchläuft die Nachbarn der besuchten Knoten. Wenn ein Nachbar bereits besucht wurde und dieser Nachbar nicht der direkt vorherige Knoten ist, wurde ein Zyklus gefunden.

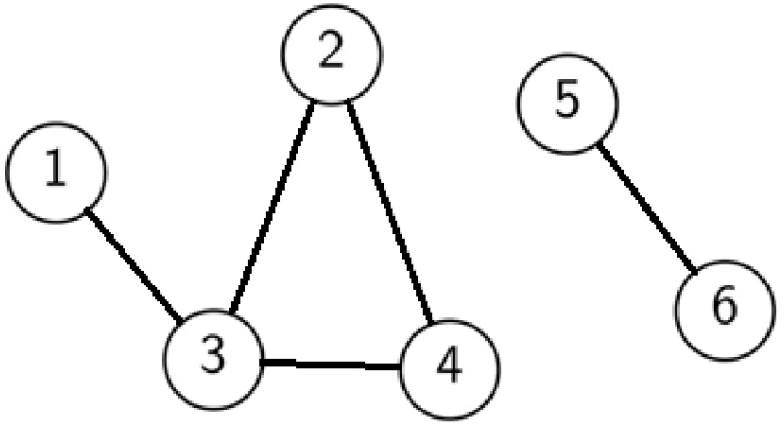
Ergebnis:

Findet er einen solchen wiederkehrenden Knoten, ist der Graph zyklisch.

Kann er alle erreichbaren Knoten besuchen, ohne auf einen solchen "Doppelgänger" zu stoßen, ist der Graph nicht zyklisch.



Nicht zyklischer Graph, zusammenhängend



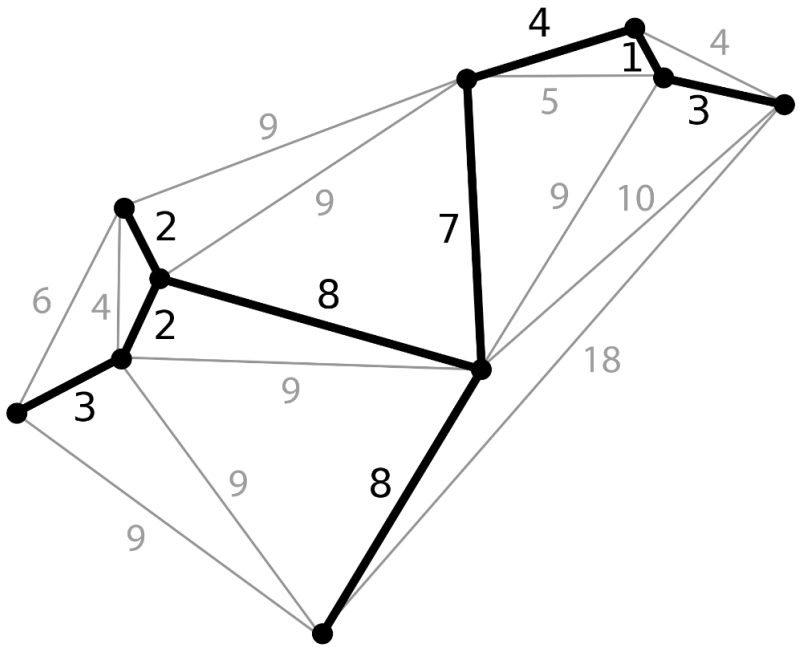
Zyklischer Graph, nicht zusammenhängend

Komplexere Graph Algorithmen

Minimum Spanning Tree / Minimaler Spannbaum

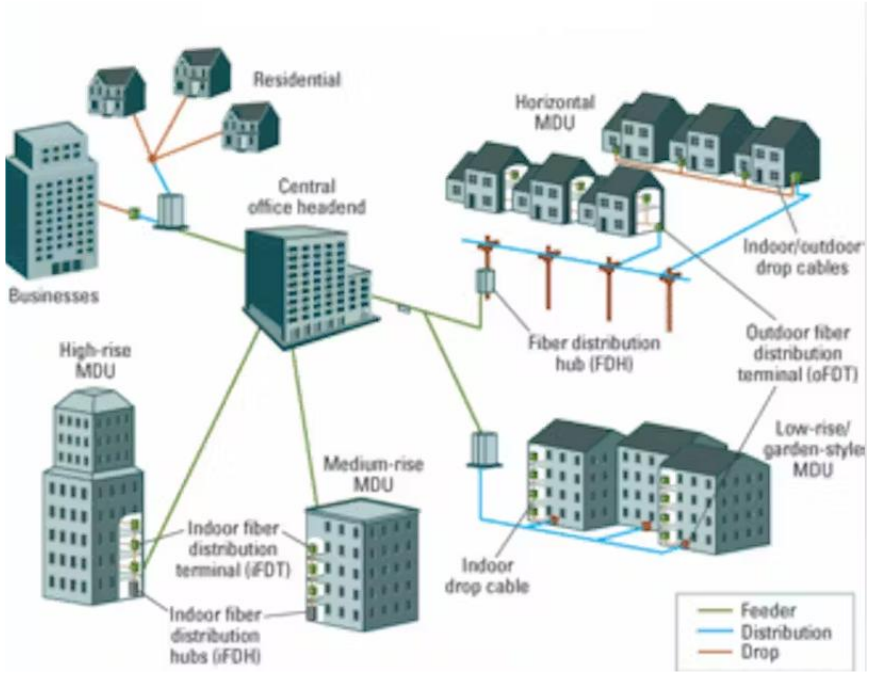
Ein Minimaler Spannbaum (auch aufspannender Baum oder Gerüst genannt – auf Englisch minimum spanning tree) ist in der Graphentheorie ein Teilgraph eines ungerichteten Graphen, der ebenfalls ein Baum ist und alle Knoten dieses Graphen enthält. Spannbäume existieren nur in zusammenhängenden Graphen. (Wikipedia)

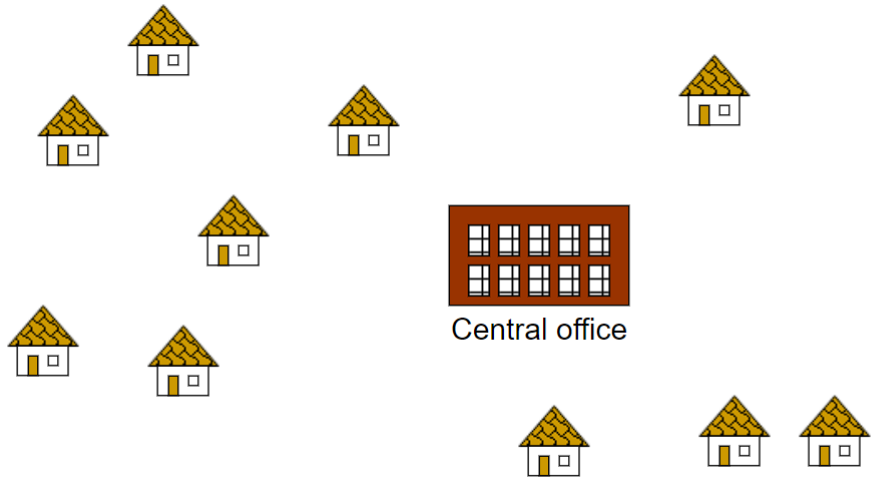
Baum: zusammenhängender, ungerichteter Graph ohne Zyklen.



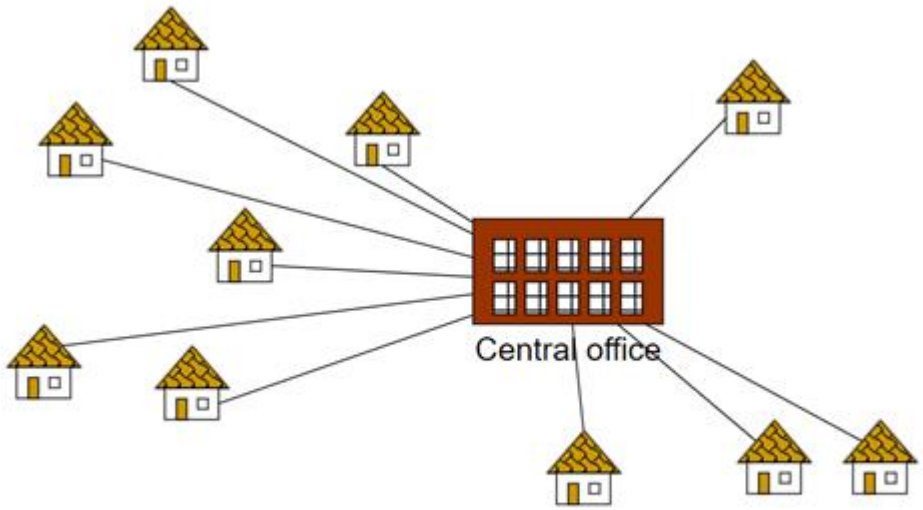
Minimum Spanning Tree / Minimaler Spannbaum

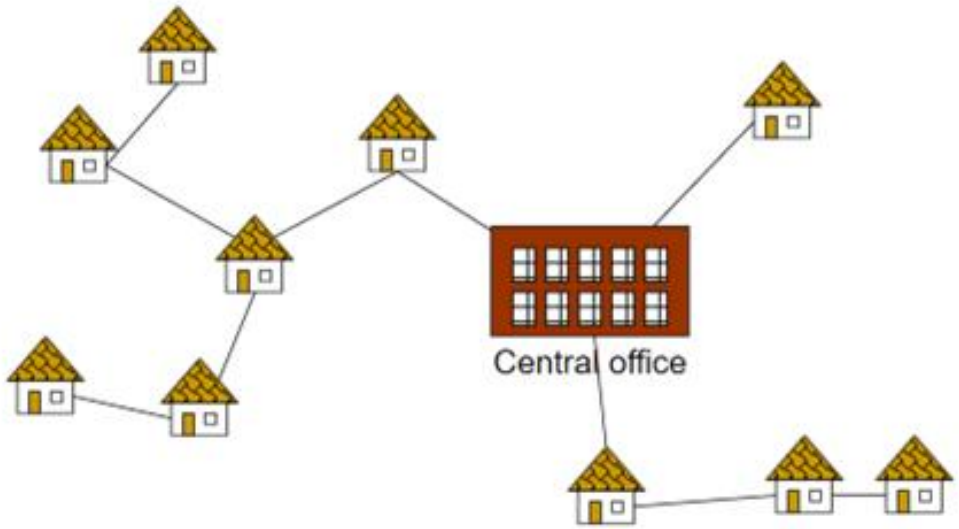
- Netzwerk-Probleme – Internet oder Strom





Minimum Spanning Tree / Minimaler Spannbaum





Prim's Algorithmus

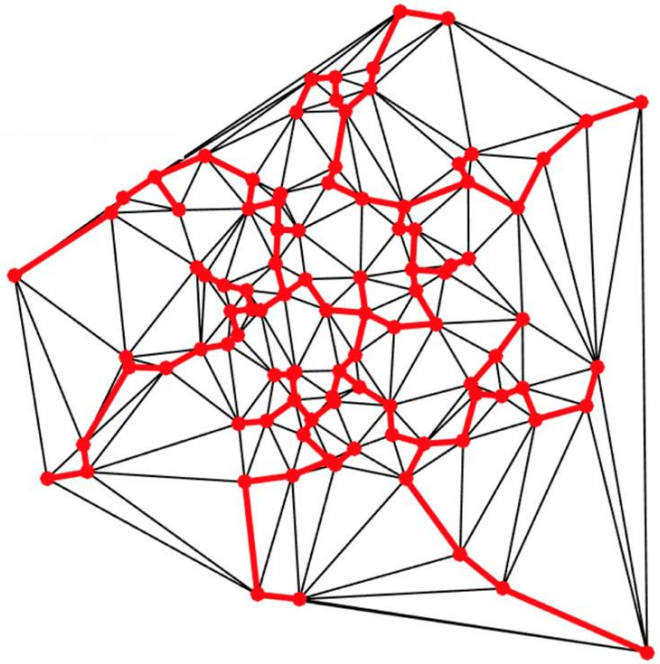
Shortest Connection Networks
And Some Generalizations

By R. C. PRIM

(Manuscript received May 8, 1957)

The basic problem considered is that of interconnecting a given set of terminals with a shortest possible network of direct links. Simple and practical procedures are given for solving this problem both graphically and computationally. It develops that these procedures also provide solutions for a much broader class of problems, containing other examples of practical interest.

- Entwickelt 1957 von Robert C. Prim – Princeton University
- Der Algorithmus beginnt mit einem beliebigen Knoten des gegebenen Graphen. In jedem Schritt wird die neue Kante mit minimalem Gewicht gesucht, die einen weiteren Knoten mit T verbindet. Diese und der entsprechende Knoten werden zu T hinzugefügt. Das Ganze wird solange wiederholt, bis alle Knoten in T vorhanden sind. Dann ist T ein minimaler Spannbaum



Prim's Algorithmus – in A Nutshell

- Wähle einen beliebigen Knoten als Start für den Graphen T.
- Solange T noch nicht alle Knoten enthält:
  - Wähle eine Kante **e** mit **minimalem** Gewicht aus, die einen noch nicht in T enthaltenen Knoten **v** mit T verbindet.
  - Füge e und v dem Graphen T hinzu.

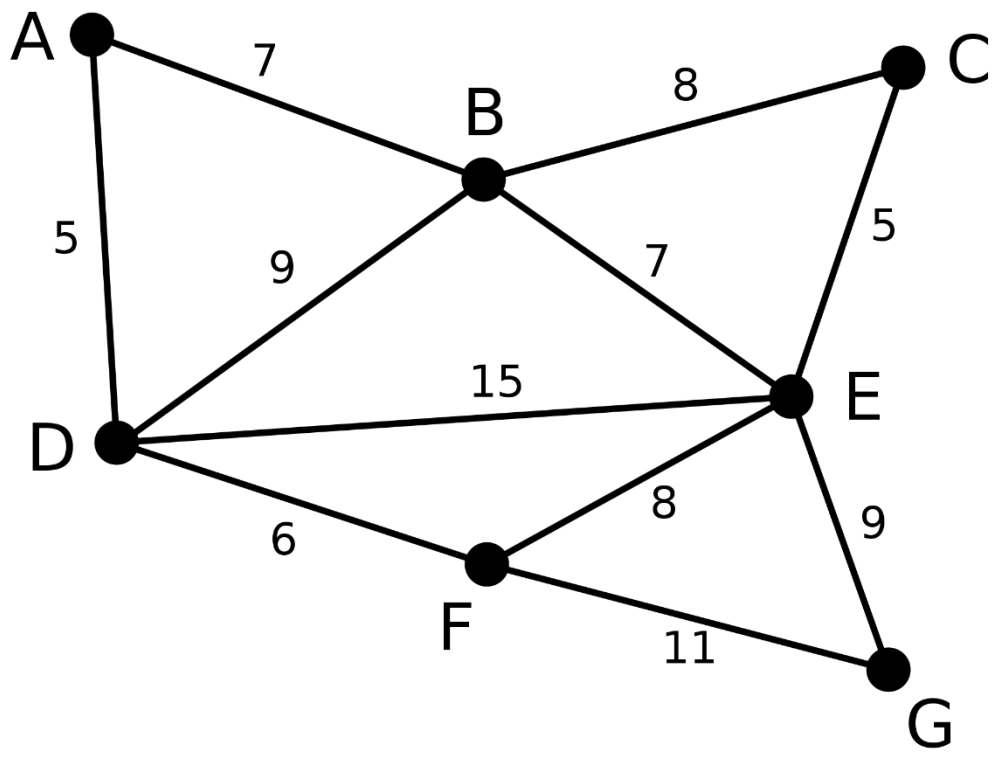
Ausgabe: minimaler Spannbaum

Prim's Algorithmus – Schritt 1

Dies ist der Graph, zu dem wir nun Schritt für Schritt mittels Algorithmus von Prim einen minimalen Spannbaum berechnen.

Merke: Es kann mehr als 1 minimalen Spannbaum geben!

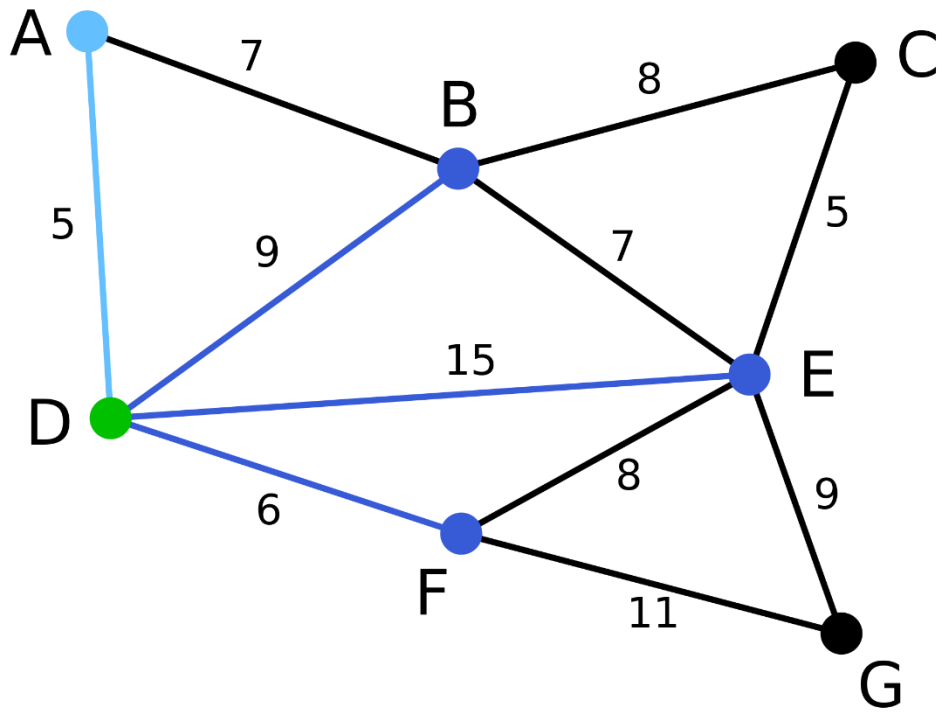
Die Zahlen bei den einzelnen Kanten geben das jeweilige Kantengewicht an.



Prim's Algorithmus – Schritt 2

Als Startknoten für den Graphen T wird der Knoten D gewählt. (Es könnte auch jeder andere Knoten ausgewählt werden.)

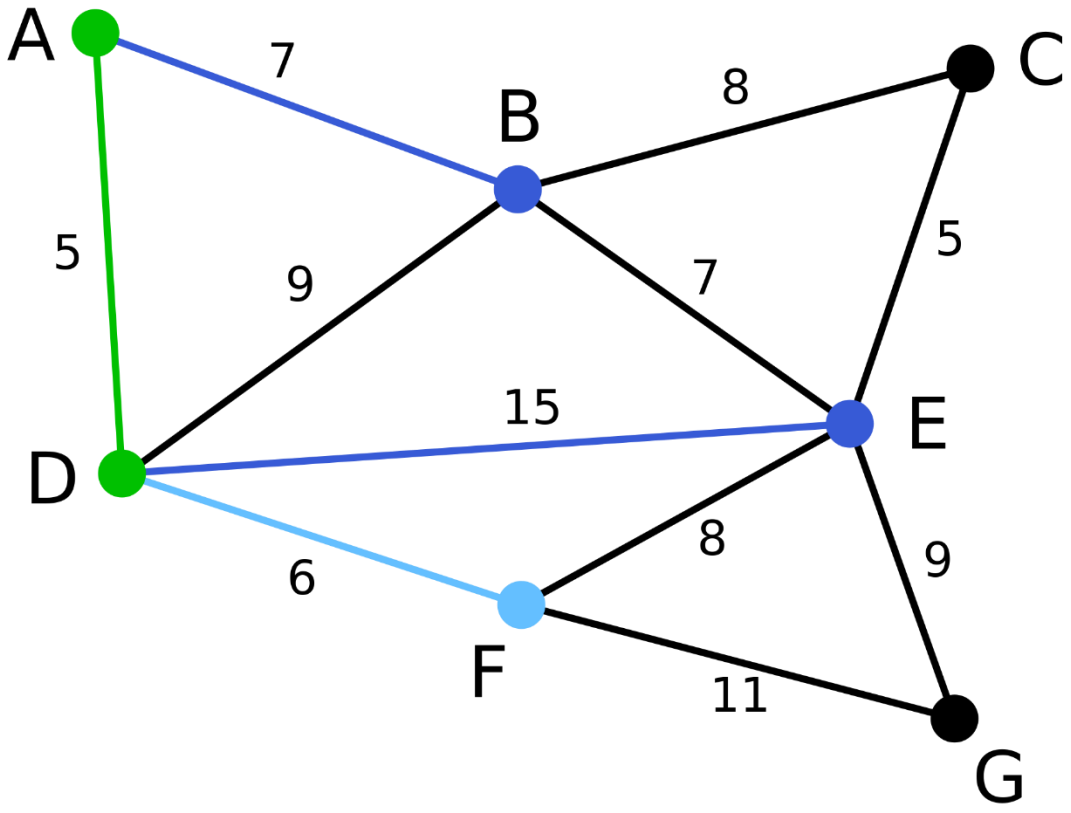
Mit dem Startknoten können die Knoten A, B, E und F jeweils unmittelbar durch die Kanten DA, DB, DE und DF verbunden werden. Unter diesen Kanten ist DA diejenige mit dem geringsten Gewicht und wird deshalb zusammen mit dem Knoten A zum Graphen T hinzugefügt.



Prim's Algorithmus – Schritt 3

Mit dem bestehenden Graphen T kann der Knoten B durch die Kanten AB oder DB, der Knoten E durch die Kante DE und der Knoten F durch die Kante DF verbunden werden.

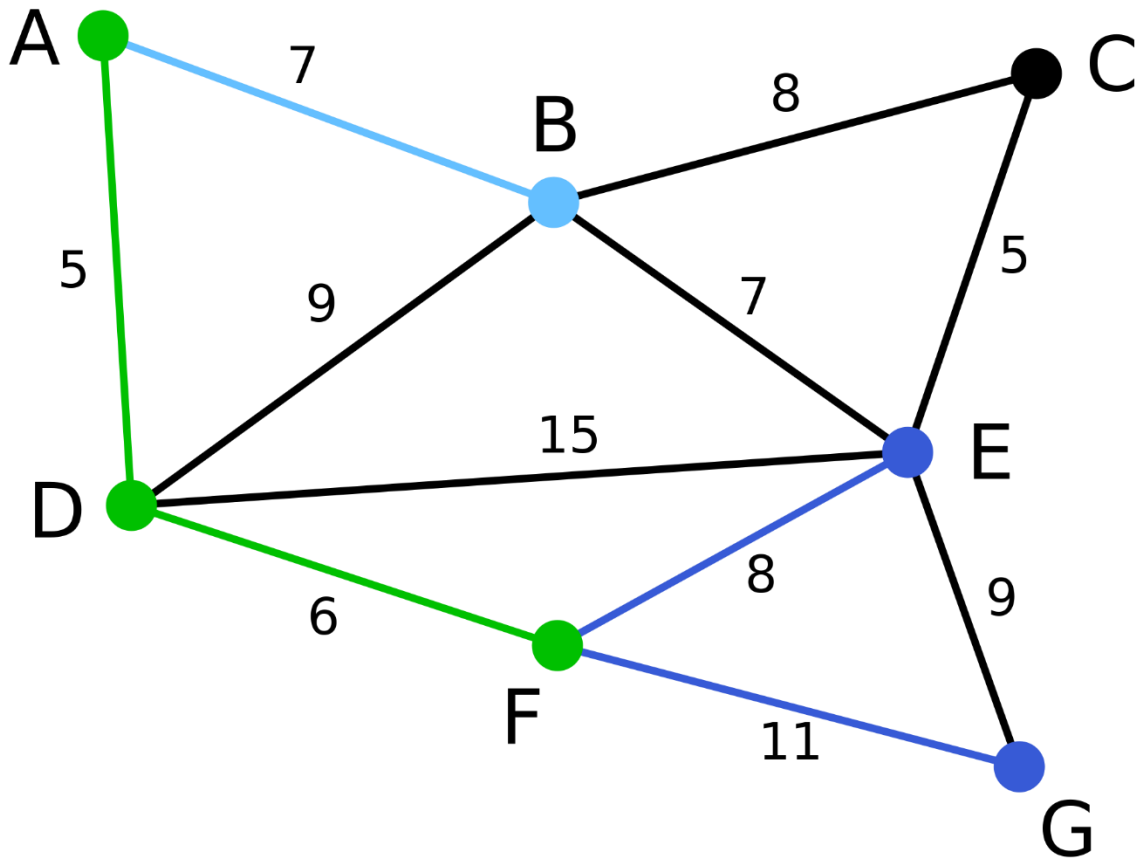
Unter diesen vier Kanten ist DF diejenige mit dem geringsten Gewicht und wird deshalb zusammen mit dem Knoten F zum Graphen T hinzugefügt.



Prim's Algorithmus – Schritt 4

Mit dem bestehenden Graphen T kann der Knoten B durch die Kanten AB oder DB, der Knoten E durch die Kanten DE oder FE und der Knoten G durch die Kante FG verbunden werden.

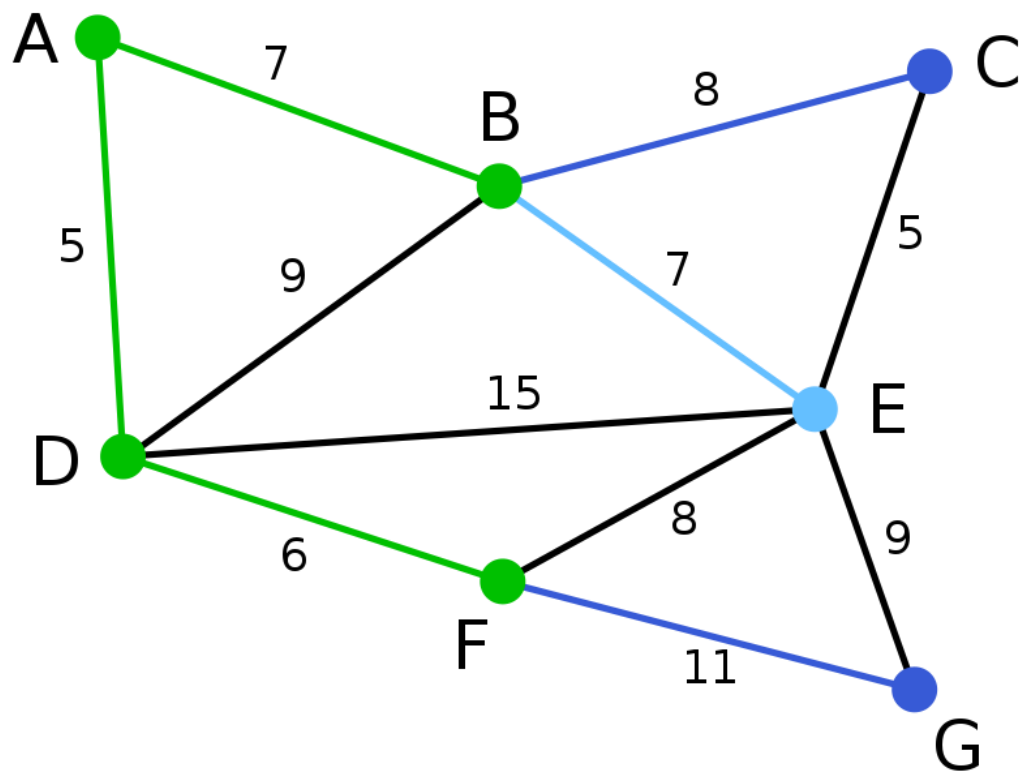
Unter diesen Kanten ist AB diejenige mit dem geringsten Gewicht und wird deshalb zusammen mit dem Knoten B zum Graphen T hinzugefügt.



Prim's Algorithmus – Schritt 5

Mit dem bestehenden Graphen T kann der Knoten C durch die Kante BC, der Knoten E durch die Kanten BE, DE oder FE und der Knoten G durch die Kante FG verbunden werden.

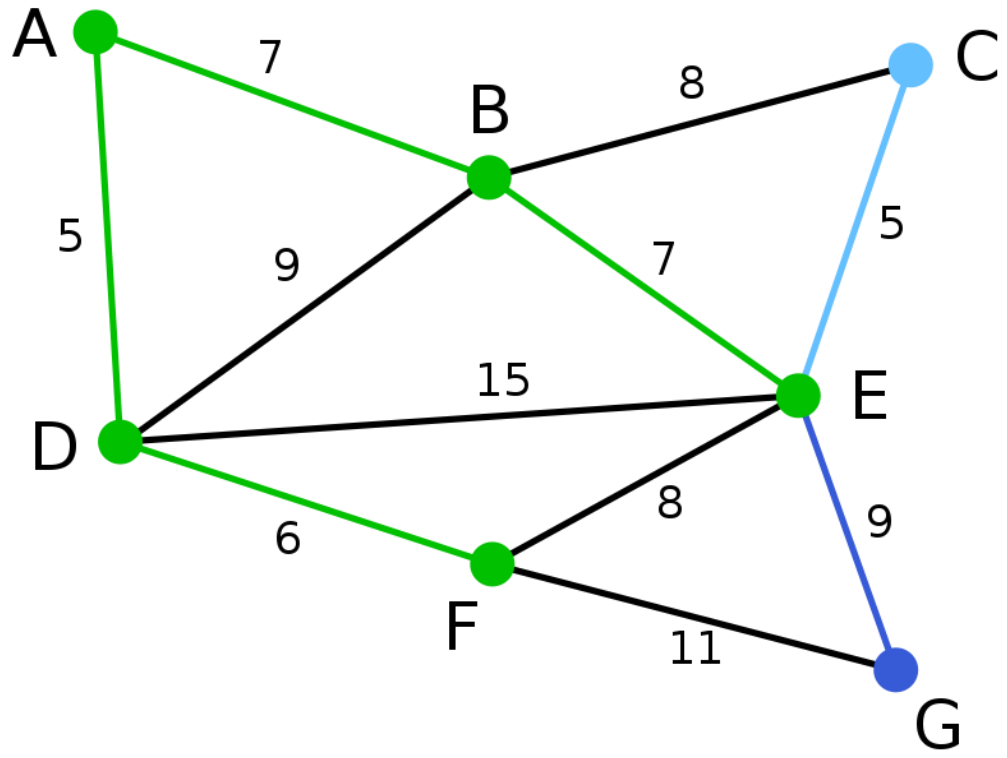
Unter diesen Kanten ist BE diejenige mit dem geringsten Gewicht und wird deshalb zusammen mit dem Knoten E zum Graphen T hinzugefügt.



Prim's Algorithmus – Schritt 6

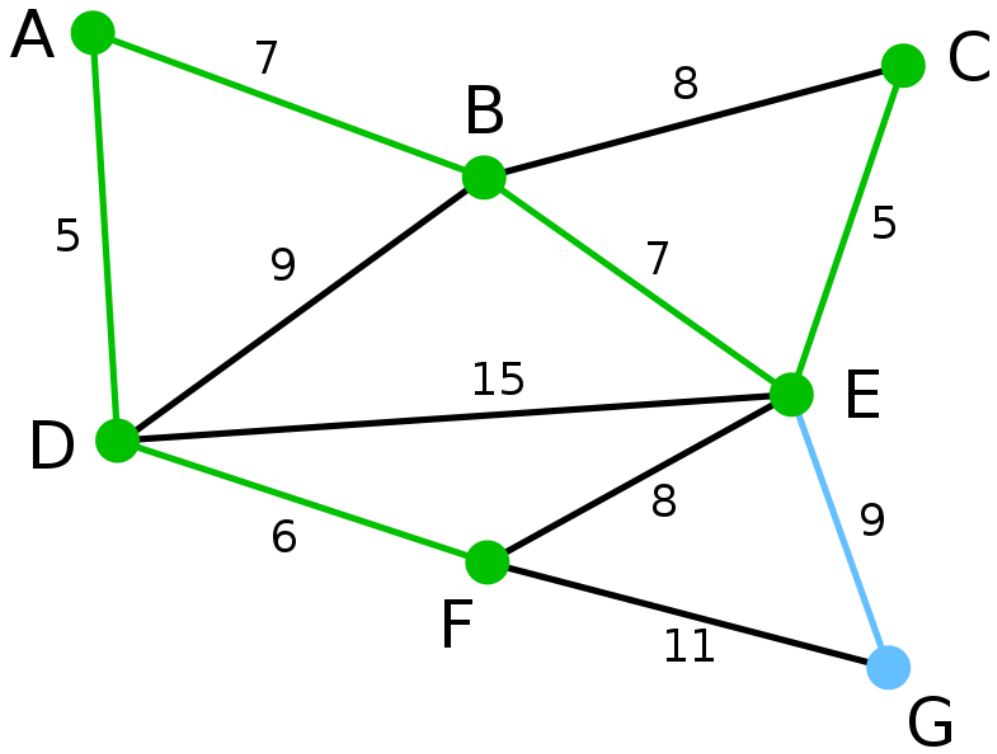
Mit dem bestehenden Graphen T kann der Knoten C durch die Kanten BC oder EC und der Knoten G durch die Kanten EG oder FG verbunden werden. Unter diesen Kanten ist

EC diejenige mit dem geringsten Gewicht und wird deshalb zusammen mit dem Knoten C zum Graphen T hinzugefügt.



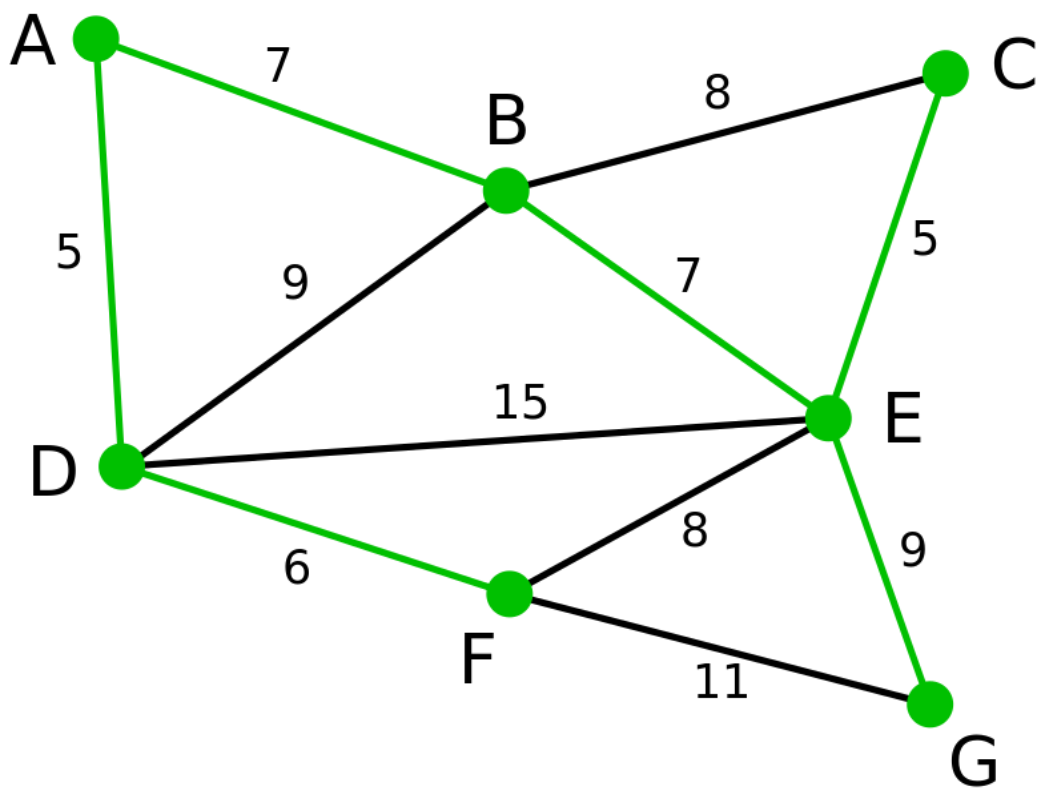
Prim's Algorithmus – Schritt 7

Der verbliebene Knoten G kann durch die Kanten EG oder FG mit dem Graphen T verbunden werden. Da EG unter diesen beiden das geringere Gewicht hat, wird sie zusammen mit dem Knoten G zum Graphen T hinzugefügt.



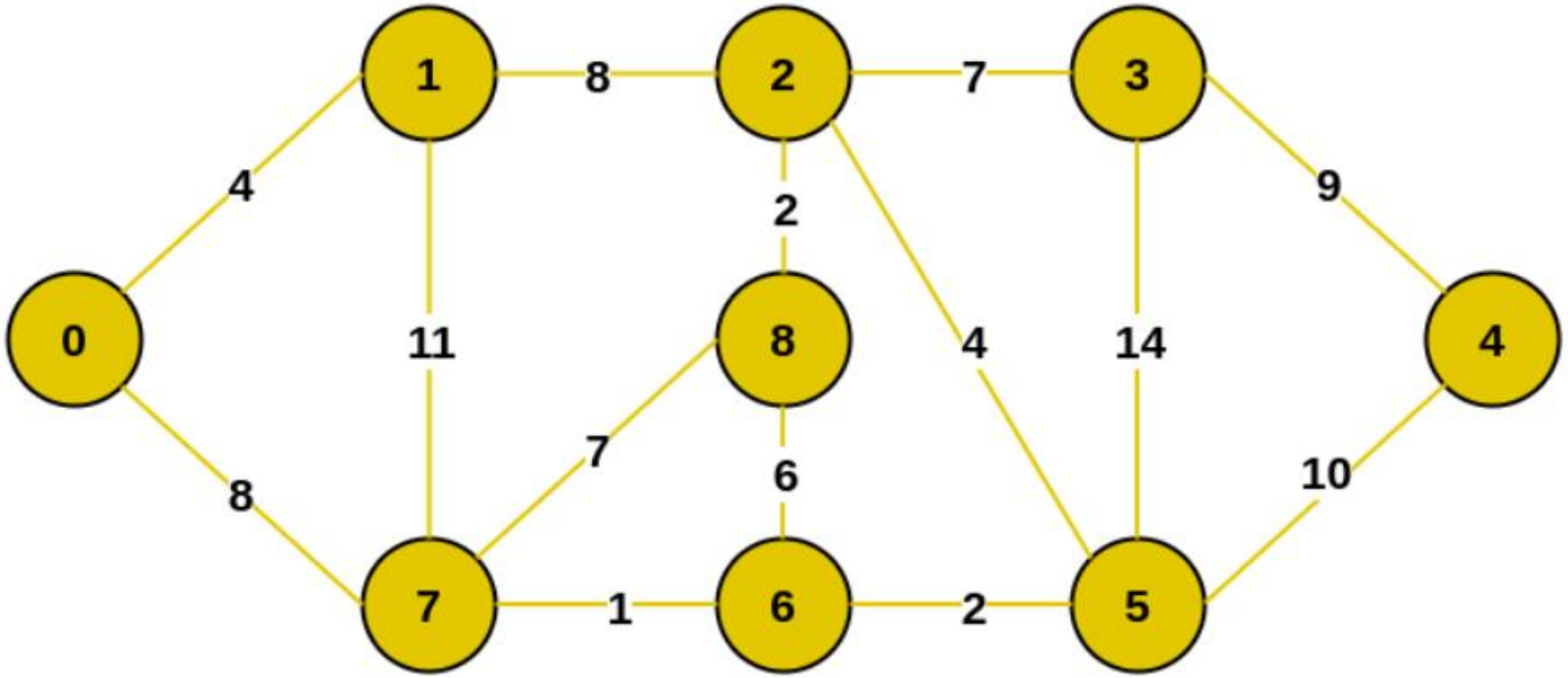
Prim's Algorithmus – Schritt 8

Der Graph T enthält jetzt alle Knoten des Ausgangsgraphen und ist ein minimaler Spannbaum dieses Ausgangsgraphen.



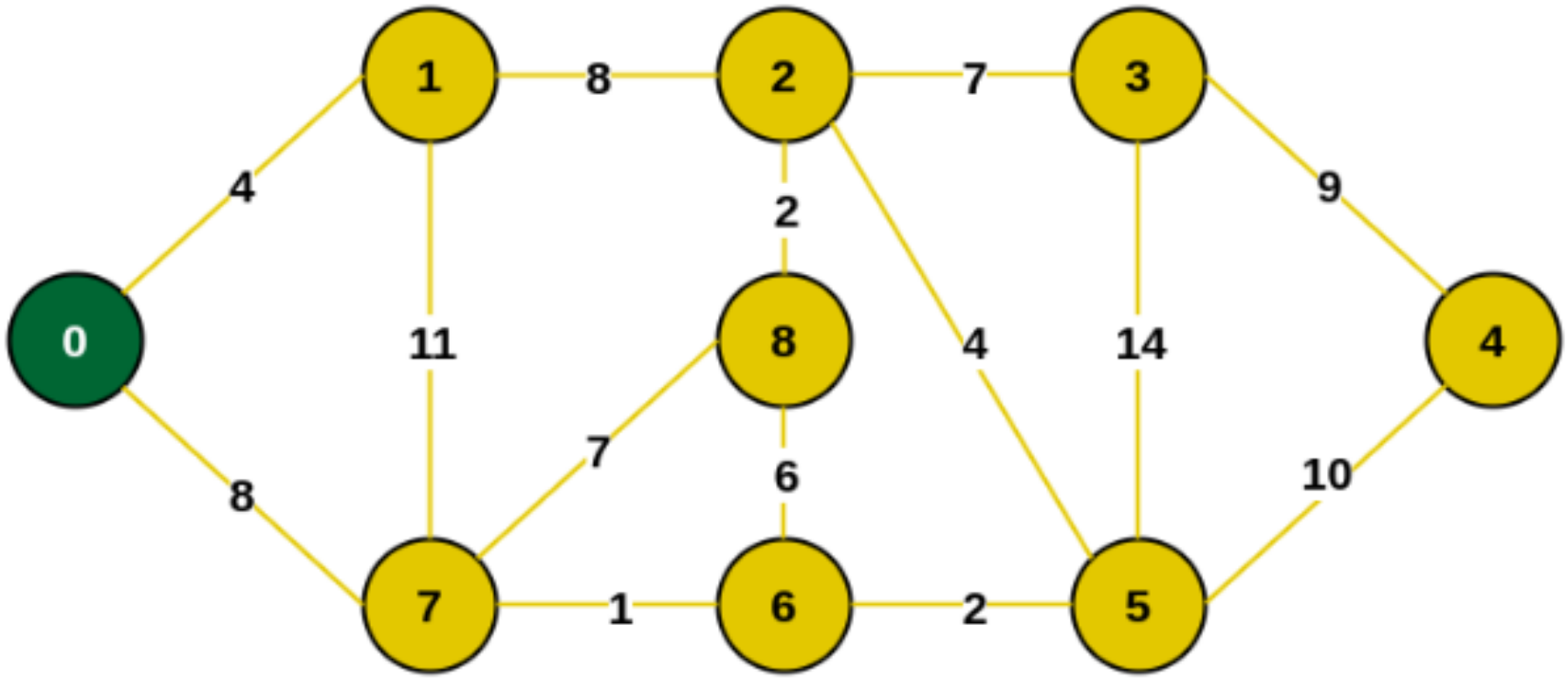
Übungen zu Minimum Spanning Tree

Übung G – leite den MST ab

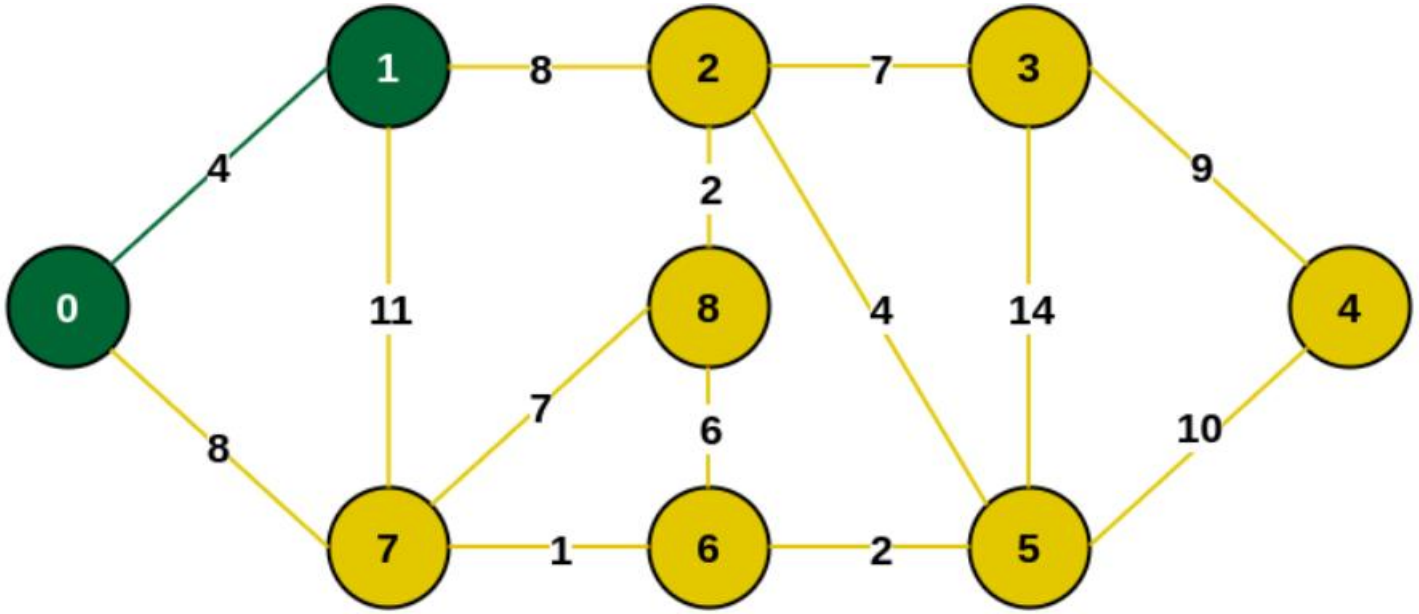


Hinweis: es gibt genau 2 Lösungen für den MST hier

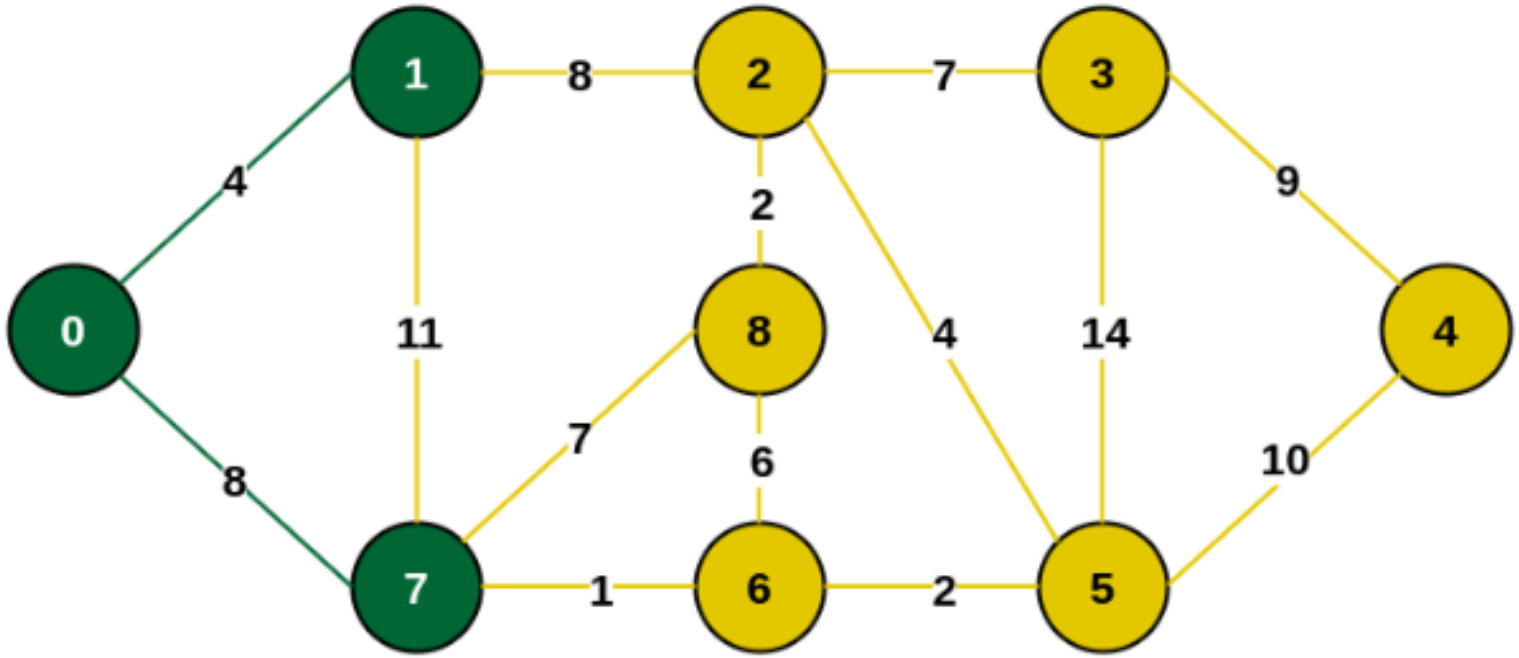
Lösung G – Schritt für Schritt A



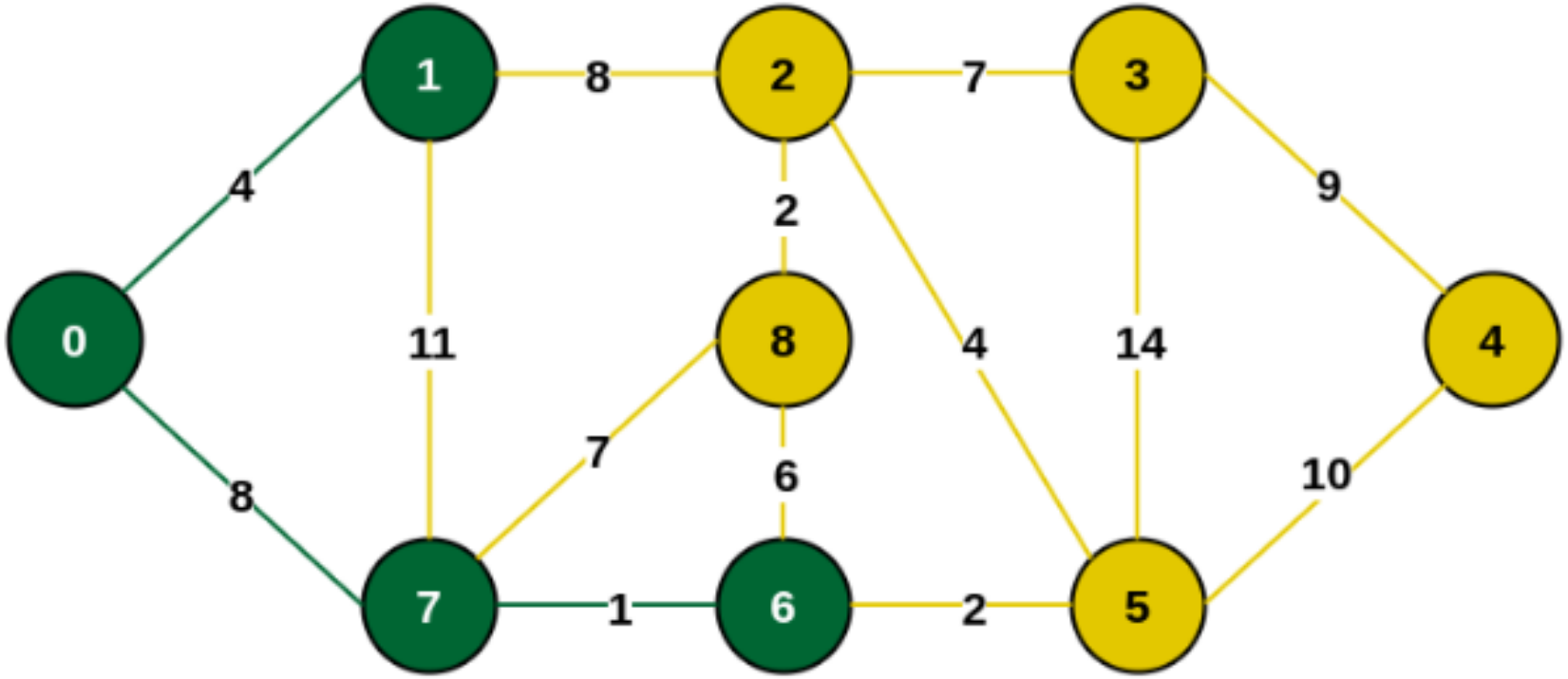
Lösung G – Schritt für Schritt B



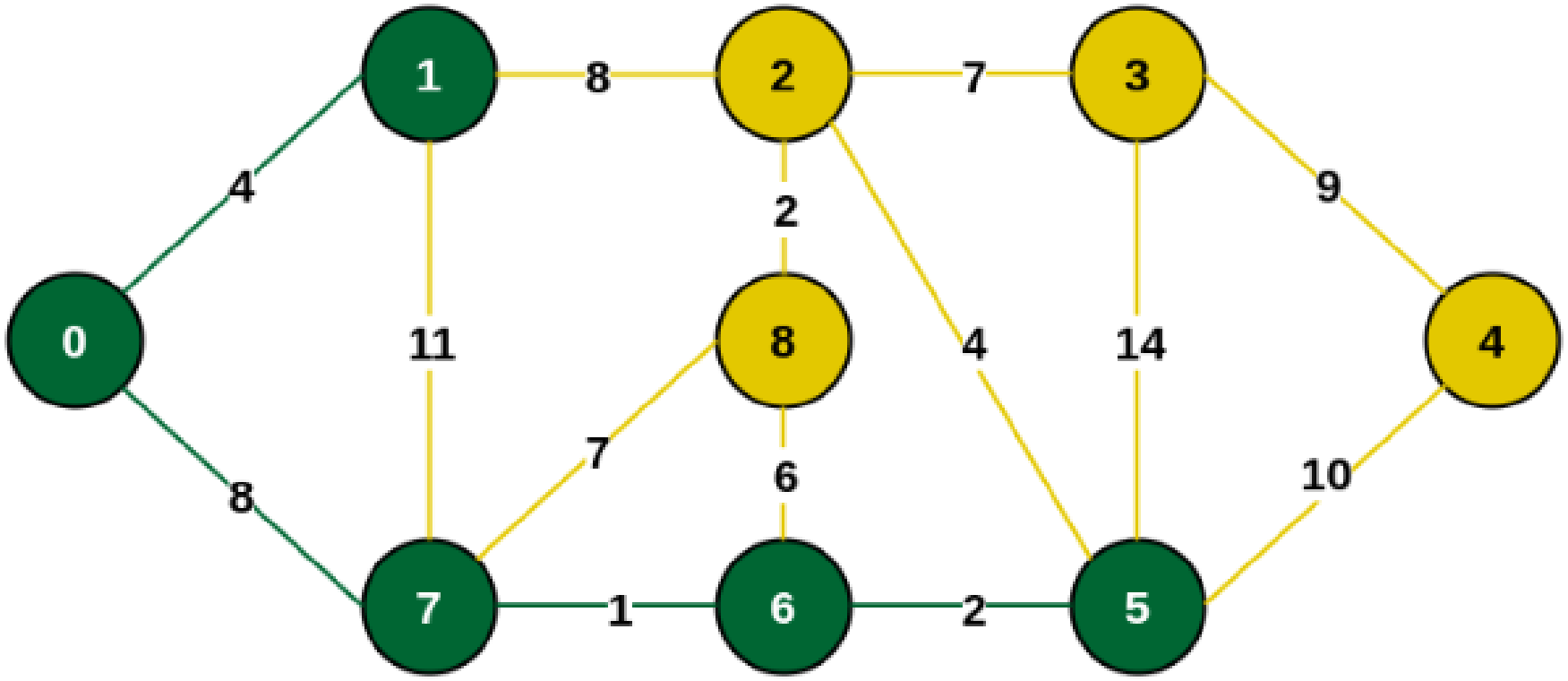
Lösung G – Schritt für Schritt C



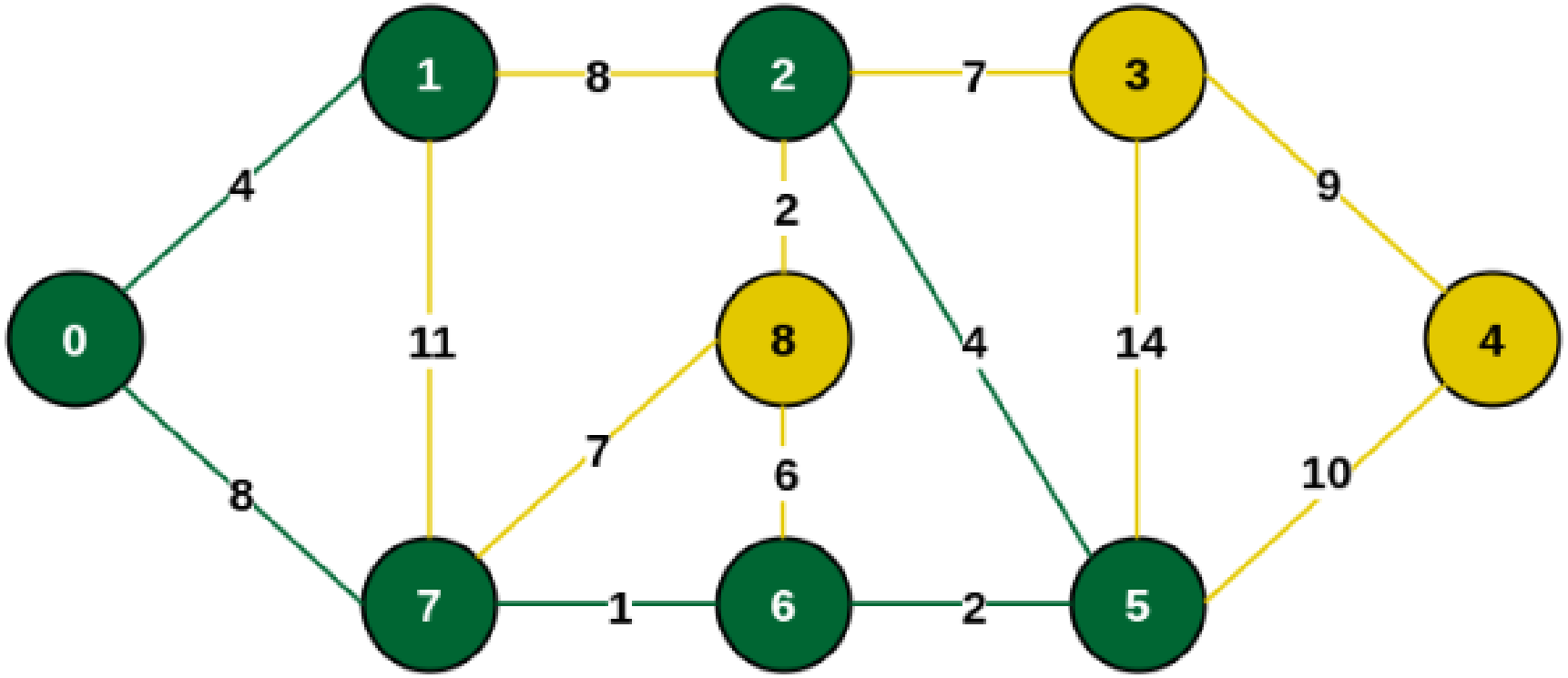
Lösung G – Schritt für Schritt D



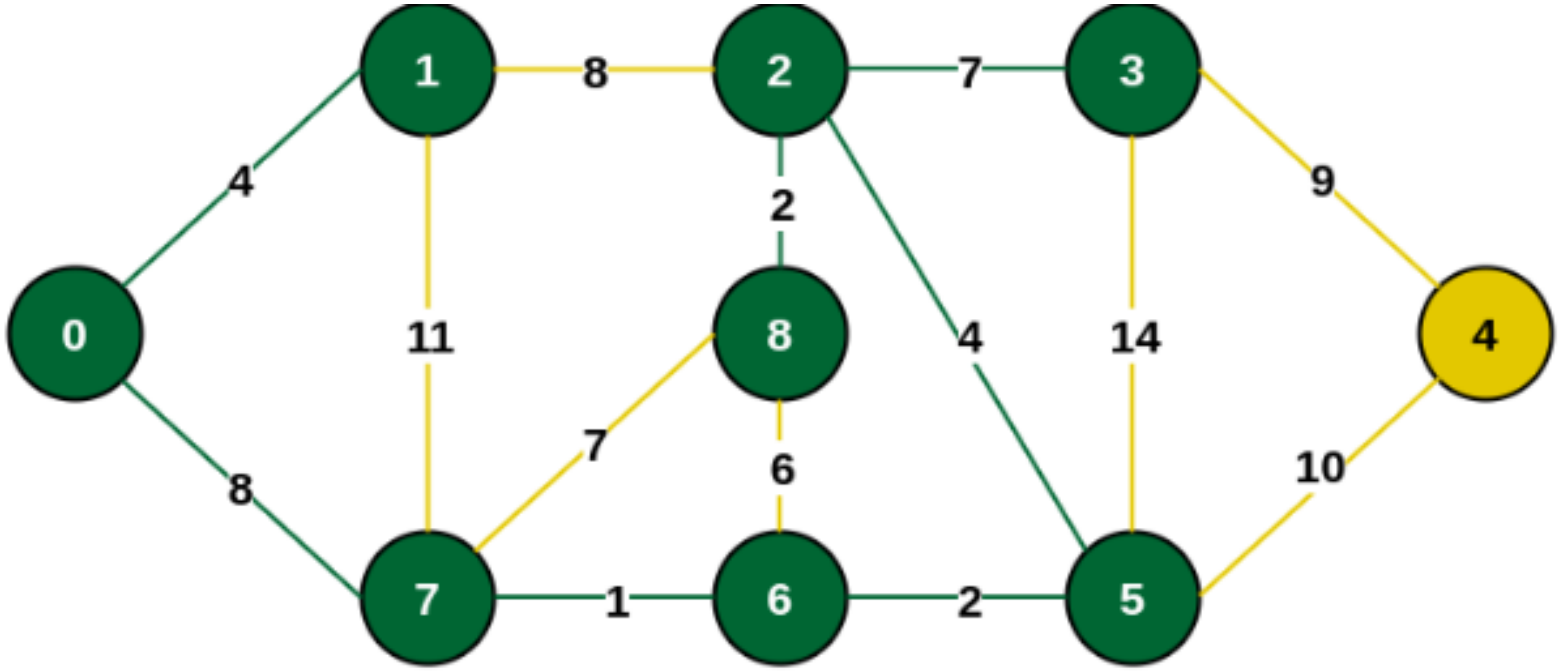
Lösung G – Schritt für Schritt E



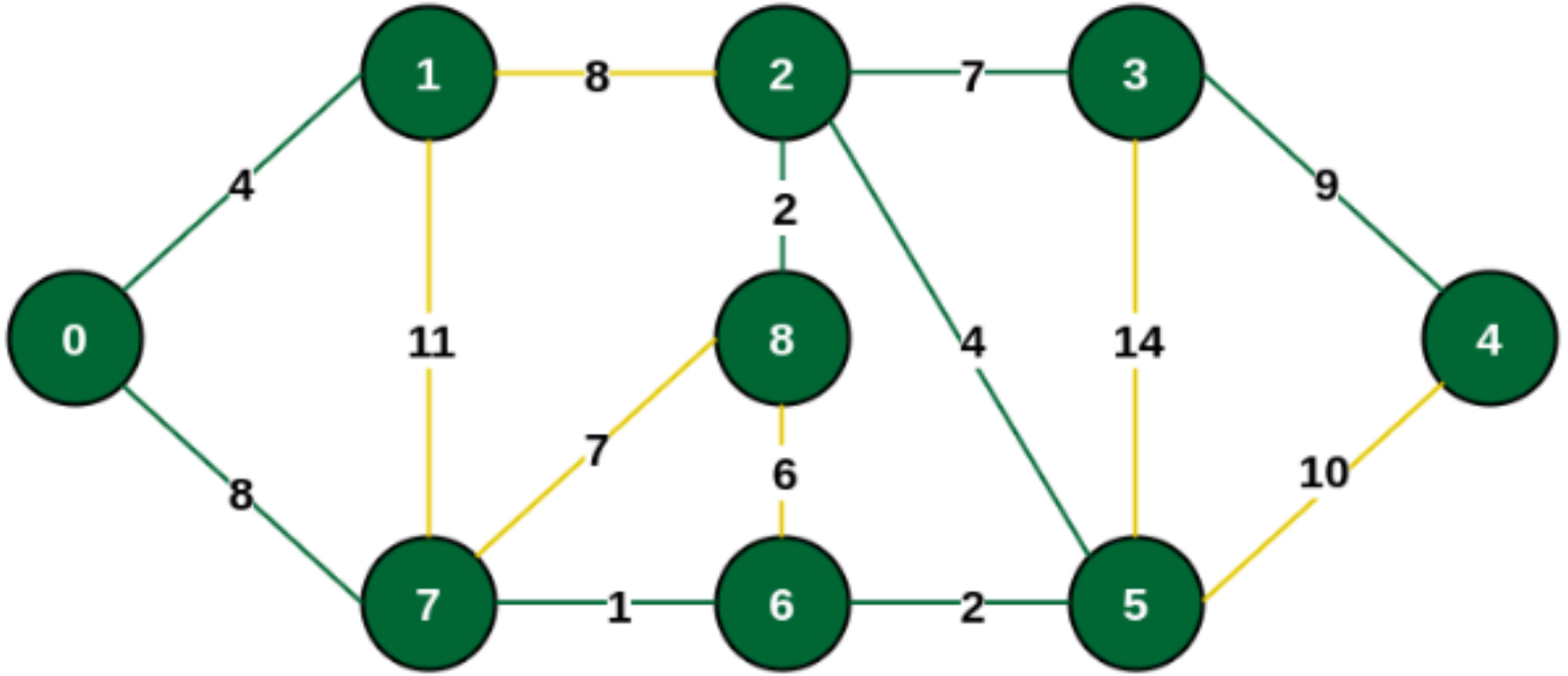
Lösung G – Schritt für Schritt F



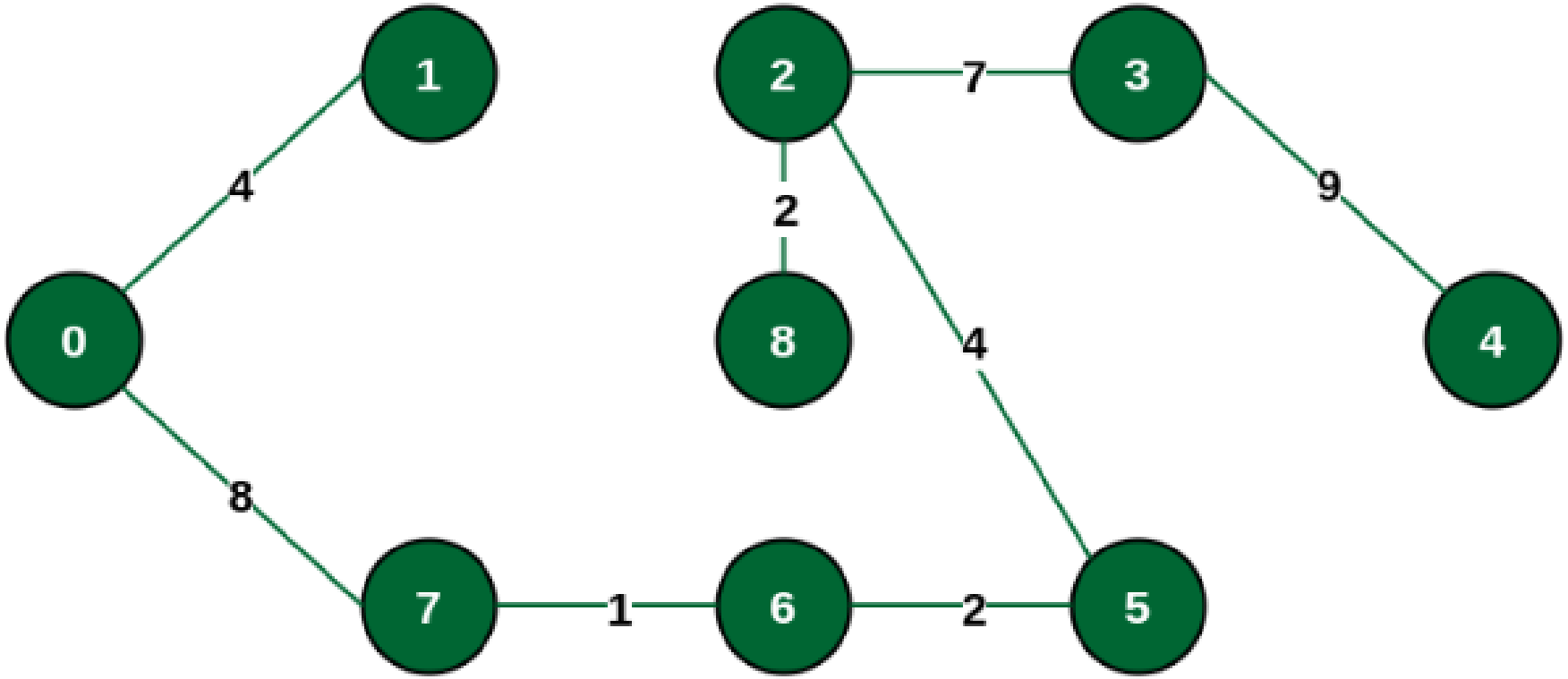
Lösung G – Schritt für Schritt G



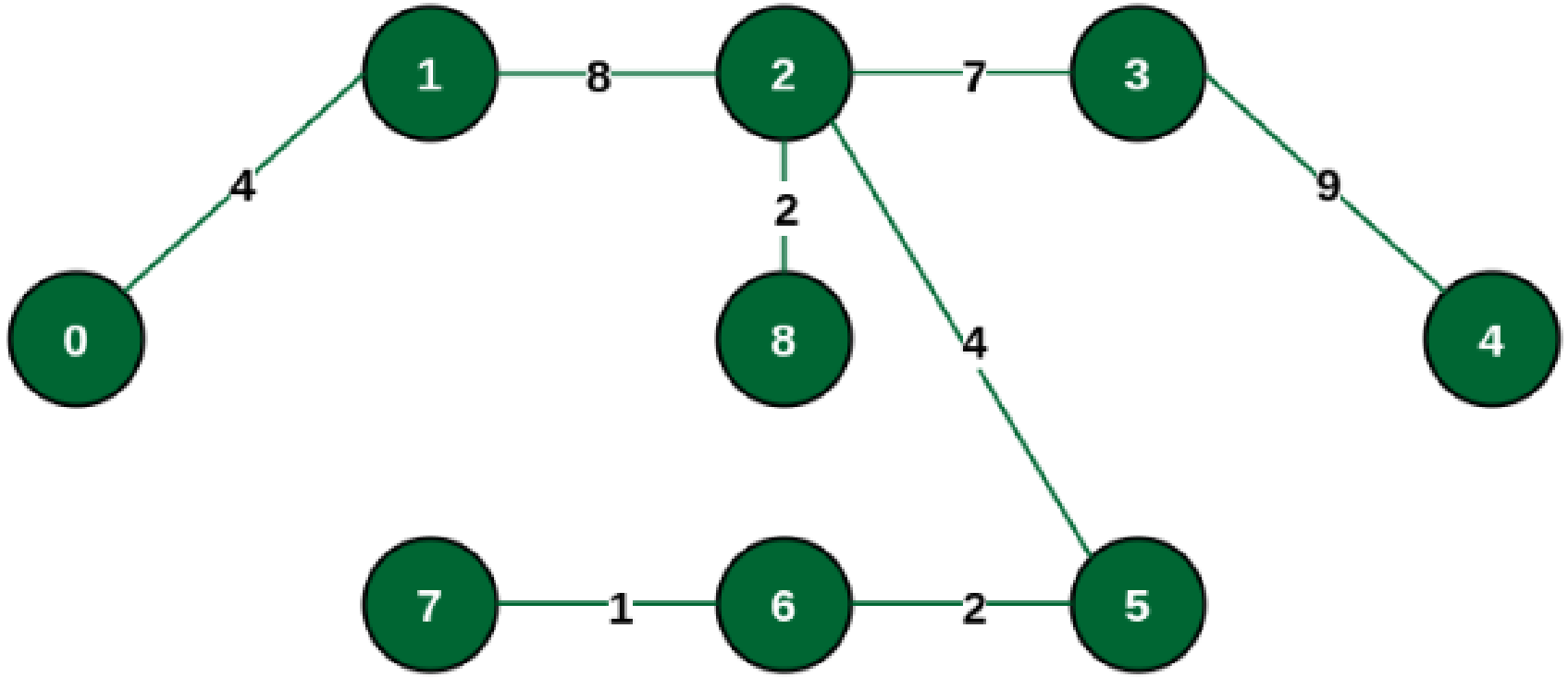
Lösung G – Schritt für Schritt H



Lösung G – Schritt für Schritt I



Lösung G – Schritt für Schritt J – Alternative Lösung

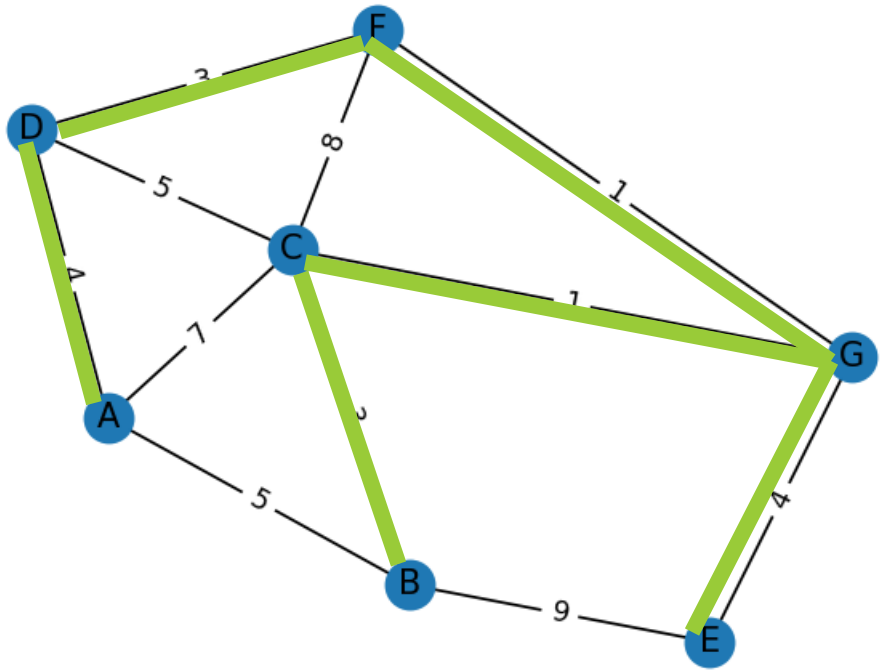


Aufgabe H

- Das Programm in Aufgabe F erstellt den abgebildeten Graphen
- Die folgenden 2 Zeilen berechnen den MST und drucken alle Kanten

```
T = nx.minimum_spanning_tree(G)
print(T.edges())
```

- [('A', 'D'), ('B', 'C'), ('C', 'G'), ('D', 'F'), ('E', 'G'), ('F', 'G')]
- Erste Aufgabe: verifiziere die Lösung



Zusatzaufgabe I

Schreibe ein Programm, welches den MST berechnet aus der Adjazenzmatrix