Algorithmentheorie

Daniel Neuen (Universität Bremen)

WiSe 2023/24

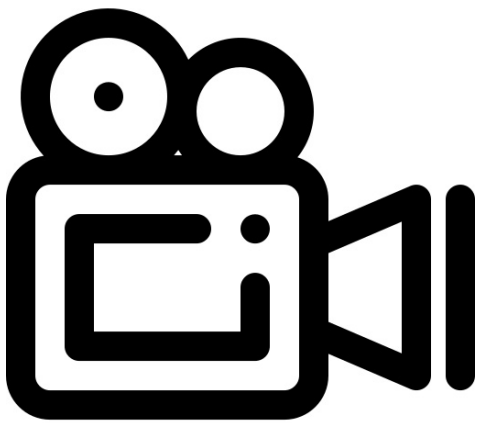
Grundlagen der Graphentheorie

5. Vorlesung

Aufzeichnung der Vorlesung

Diese Vorlesung wird aufgezeichnet und live gestreamt.

- Aufzeichnungen nur der Lehrenden durch sich selbst.
- Bei Rückfragen aus dem Auditorium und Diskussion bitte deutlich anzeigen, falls das Mikro stumm geschaltet werden soll.



Organisatorisches

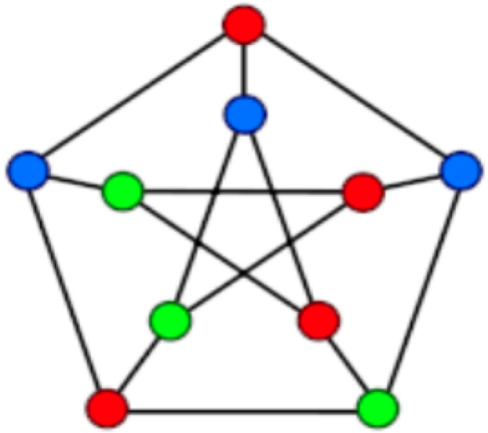
Übung:

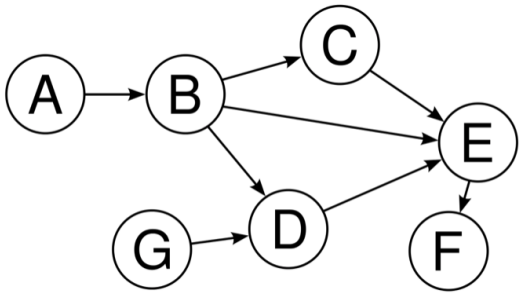
- Wechsel des Tutoriums erlaubt, wenn nicht anders möglich

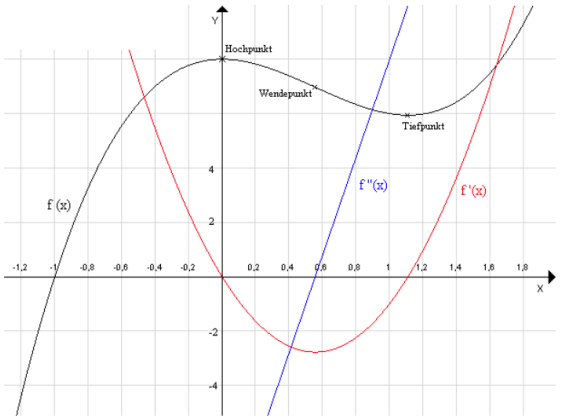
Klausur:

- Klausur findet voraussichtlich am 19./20. Februar statt

Graphen



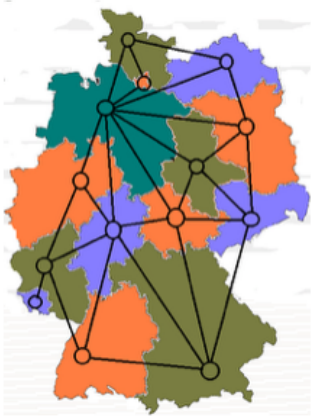
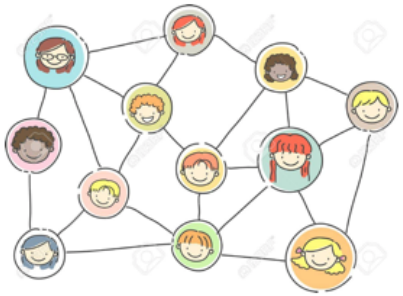
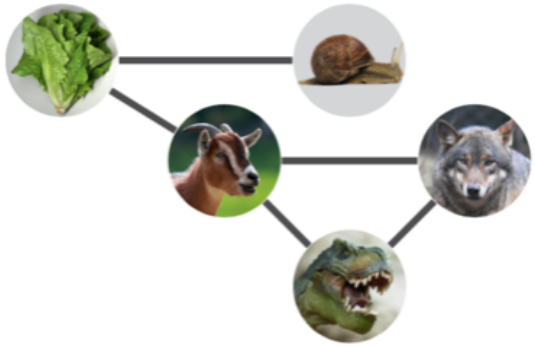




Graphen modellieren...

...physikalische Netzwerke, virtuelle Netzwerke, Zusammenhänge, Abhängigkeiten, Beziehungen und vieles mehr



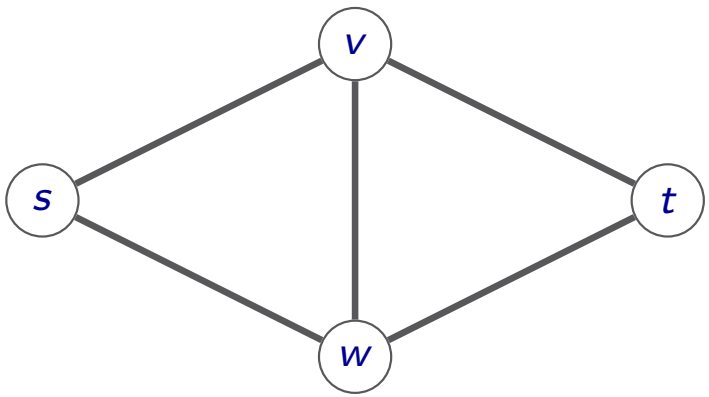





Ungerichtete Graphen

Definition

Ein ungerichteter Graph $G = (V, E)$ ist ein Paar bestehend aus

- einer endlichen Menge V von Knoten (vertices), auch $V(G)$
- und einer Menge E von ungeordneten Paaren von Knoten, genannt Kanten (edges), auch $E(G)$.

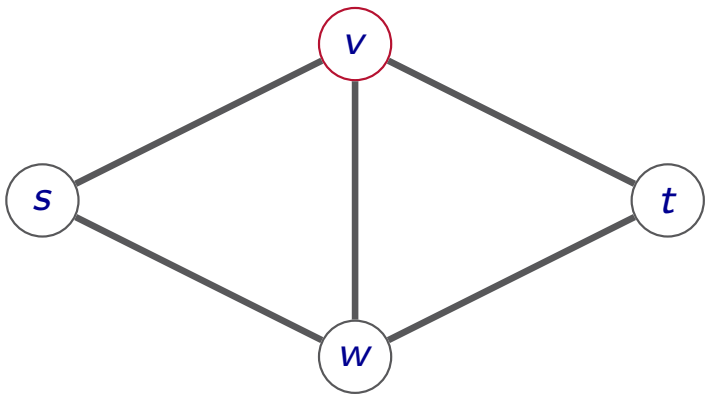


Ungerichtete Graphen

Definition

Ein ungerichteter Graph $G = (V, E)$ ist ein Paar bestehend aus

- einer endlichen Menge V von Knoten (vertices), auch $V(G)$
- und einer Menge E von ungeordneten Paaren von Knoten, genannt Kanten (edges), auch $E(G)$.



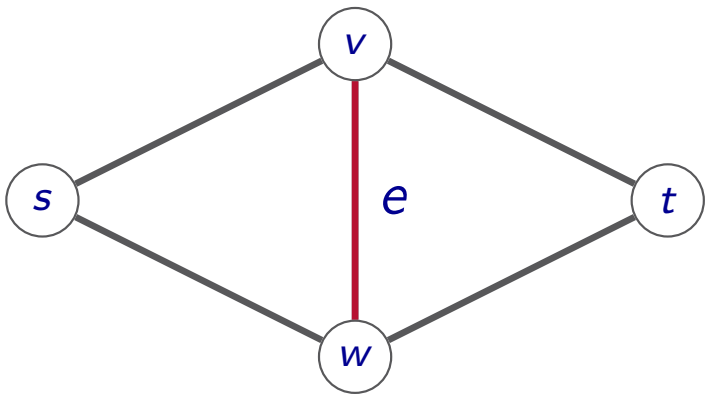
Knoten v

Ungerichtete Graphen

Definition

Ein ungerichteter Graph $G = (V, E)$ ist ein Paar bestehend aus

- einer endlichen Menge V von Knoten (vertices), auch $V(G)$
- und einer Menge E von ungeordneten Paaren von Knoten, genannt Kanten (edges), auch $E(G)$.



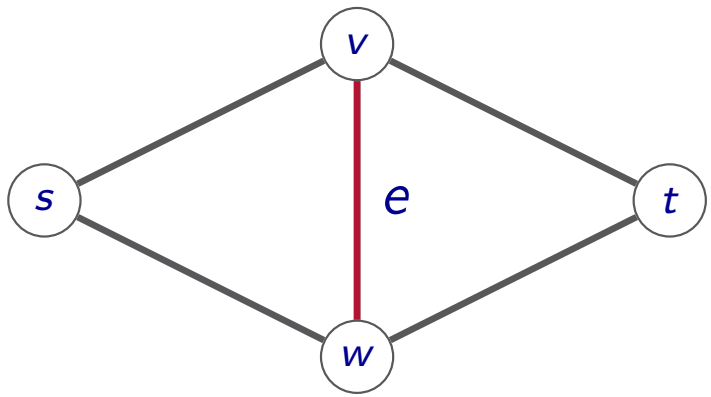
Knoten v und Kante $e = \{v, w\}$

Ungerichtete Graphen

Definition

Ein ungerichteter Graph $G = (V, E)$ ist ein Paar bestehend aus

- einer endlichen Menge V von Knoten (vertices), auch $V(G)$
- und einer Menge E von ungeordneten Paaren von Knoten, genannt Kanten (edges), auch $E(G)$.



Knoten v und Kante $e = \{v, w\}$

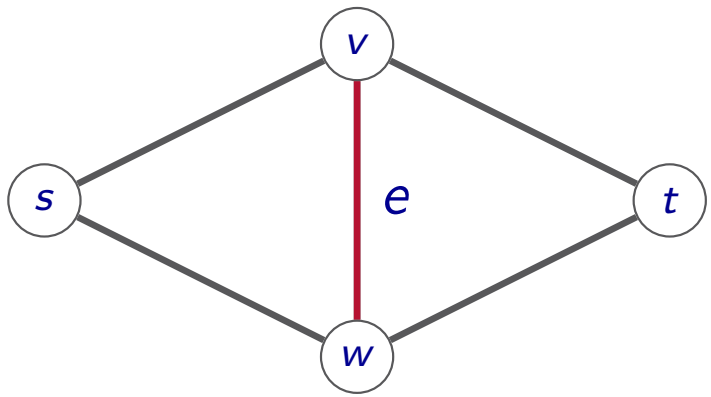
Notation: Kante e auch vw oder (v, w)

Ungerichtete Graphen

Definition

Ein ungerichteter Graph $G = (V, E)$ ist ein Paar bestehend aus

- einer endlichen Menge V von Knoten (vertices), auch $V(G)$
- und einer Menge E von ungeordneten Paaren von Knoten, genannt Kanten (edges), auch $E(G)$.



Knoten v und Kante $e = \{v, w\}$

Notation: Kante e auch vw oder (v, w)

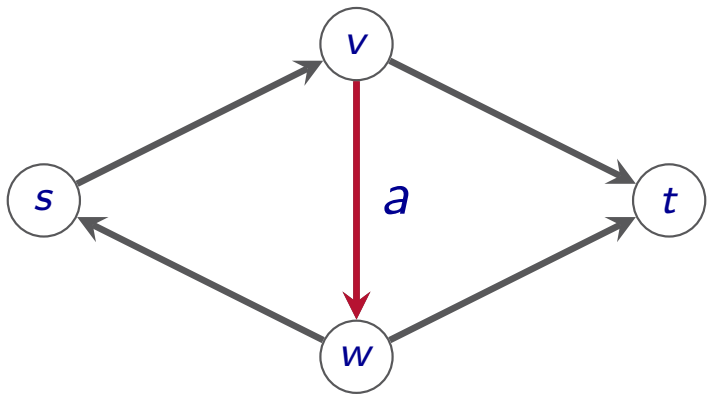
Sprechweise: v, w sind adjazent (benachbart); e ist inzident zu v, w

Gerichtete Graphen (Digraphen)

Definition

Ein gerichteter Graph (Digraph) $G = (V, A)$ ist ein Paar bestehend aus

- einer endlichen Menge V von Knoten (vertices), auch $V(G)$
- und einer Menge A von geordneten Paaren von Knoten, genannt (gerichtete) Kanten (Bögen, arcs), auch $A(G)$.



Kante $a = (v, w)$

Sprechweise: v, w sind adjazent (benachbart) und
a ist inzident zu v und w

Weitere Begriffe

Sei $G = (V, E)$ ein Graph. Typischerweise: $n = |V|$ und $m = |E|$.

Weitere Begriffe

Sei $G = (V, E)$ ein Graph. Typischerweise: $n = |V|$ und $m = |E|$.

parallele Kanten: Kanten, die dieselben Endpunkte haben

Weitere Begriffe

Sei $G = (V, E)$ ein Graph. Typischerweise: $n = |V|$ und $m = |E|$.

parallele Kanten: Kanten, die dieselben Endpunkte haben

Schleife, Schlinge: eine Kante, die nur zu einem Knoten inzident ist

Weitere Begriffe

Sei $G = (V, E)$ ein Graph. Typischerweise: $n = |V|$ und $m = |E|$.

parallele Kanten: Kanten, die dieselben Endpunkte haben

Schleife, Schlinge: eine Kante, die nur zu einem Knoten inzident ist

einfacher Graph: Graph ohne parallele Kanten und Schleifen

(Wir betrachten nur einfache Graphen.)

Weitere Begriffe

Sei $G = (V, E)$ ein Graph. Typischerweise: $n = |V|$ und $m = |E|$.

parallele Kanten: Kanten, die dieselben Endpunkte haben

Schleife, Schlinge: eine Kante, die nur zu einem Knoten inzident ist

einfacher Graph: Graph ohne parallele Kanten und Schleifen

(Wir betrachten nur einfache Graphen.)

isolierter Knoten: zu keiner Kante inzident

Weitere Begriffe

Sei $G = (V, E)$ ein Graph. Typischerweise: $n = |V|$ und $m = |E|$.

parallele Kanten: Kanten, die dieselben Endpunkte haben

Schleife, Schlinge: eine Kante, die nur zu einem Knoten inzident ist

einfacher Graph: Graph ohne parallele Kanten und Schleifen

(Wir betrachten nur einfache Graphen.)

isolierter Knoten: zu keiner Kante inzident

vollständiger Graph: enthält alle nur möglichen Kanten

K_n: ungerichteter vollständiger Graph auf n Knoten

Weitere Begriffe

Sei $G = (V, E)$ ein Graph. Typischerweise: $n = |V|$ und $m = |E|$.

parallele Kanten: Kanten, die dieselben Endpunkte haben

Schleife, Schlinge: eine Kante, die nur zu einem Knoten inzident ist

einfacher Graph: Graph ohne parallele Kanten und Schleifen
(Wir betrachten nur einfache Graphen.)

isolierter Knoten: zu keiner Kante inzident

vollständiger Graph: enthält alle nur möglichen Kanten
K_n: ungerichteter vollständiger Graph auf n Knoten

gewichteter (Di)graph $G = (V, E, w)$ ist ein (gerichteter) Graph (V, E) mit einer Gewichtsfunktion $w : E \to \mathbb{R}$.
$\to w(e)$ können Kosten oder Länge der Kante $e \in E$ sein.

Knotengrad

Definition

- Der Grad $\deg(v)$ eines Knoten v ist die Anzahl inzidenter Kanten.

Knotengrad

Definition

- Der Grad $\deg(v)$ eines Knoten v ist die Anzahl inzidenter Kanten.
- In einem gerichteten Graphen $G = (V, A)$ sei für Knoten $v \in V$

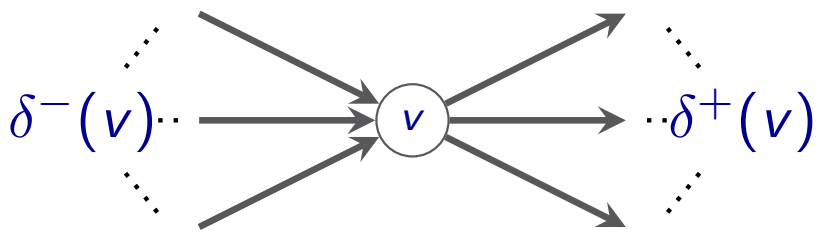
  $\delta^-(v) := \{(u, v) \in A\}$ die Menge eingehender Kanten aus v
  $\delta^+(v) := \{(v, u) \in A\}$ die Menge ausgehender Kanten aus v.

Knotengrad

Definition

- Der Grad $\deg(v)$ eines Knoten v ist die Anzahl inzidenter Kanten.
- In einem gerichteten Graphen $G = (V, A)$ sei für Knoten $v \in V$

  $\delta^-(v) := \{(u, v) \in A\}$ die Menge eingehender Kanten aus v
  $\delta^+(v) := \{(v, u) \in A\}$ die Menge ausgehender Kanten aus v.
- In einem gerichteten Graphen unterscheiden wir den Eingangsgrad $\deg^-(v) = |\delta^-(v)|$ und den Ausgangsgrad $\deg^+(v) = |\delta^+(v)|$.



Knotengrad - nützliche Sätze

Satz (Handshake Lemma)

In einem einfachen ungerichteten Graphen $G = (V, E)$ gilt

$$\sum_{v \in V} \deg(v) = 2|E|.$$

Knotengrad - nützliche Sätze

Satz (Handshake Lemma)

In einem einfachen ungerichteten Graphen $G = (V, E)$ gilt

$$\sum_{v \in V} \deg(v) = 2|E|.$$

Korollar

In einem einfachen ungerichteten Graphen ist die Anzahl der Knoten mit ungeradem Grad gerade.

Knotengrad - nützliche Sätze

Satz (Handshake Lemma)

In einem einfachen ungerichteten Graphen $G = (V, E)$ gilt

$$\sum_{v \in V} \deg(v) = 2|E|.$$

Korollar

In einem einfachen ungerichteten Graphen ist die Anzahl der Knoten mit ungeradem Grad gerade.

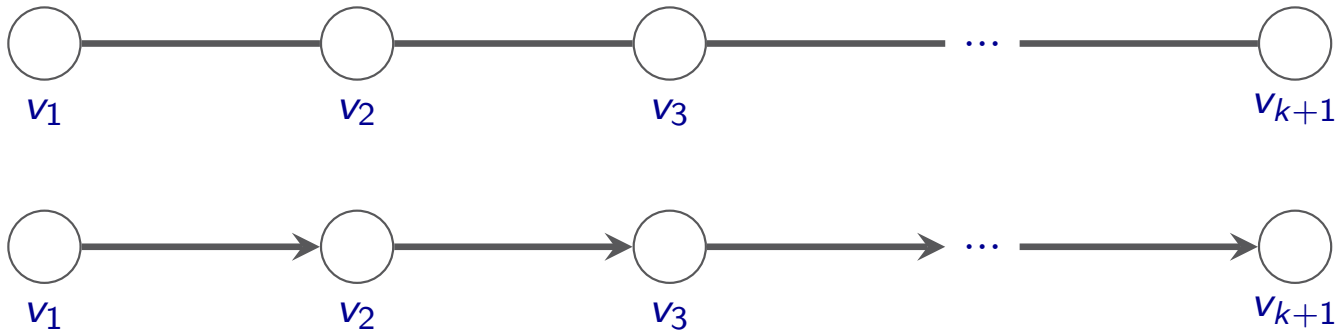
Beweise $\rightarrow$ Übung

Weg, Pfad, Kantenzug

Definition

Ein Weg (auch Kantenzug) der Länge k in einem (gerichteten) Graphen $G = (V, E)$ von einem Knoten v zu einem Knoten w ist eine endliche Folge von Knoten $v_1, v_2, ..., v_{k+1}$, sodass:

- $(v_i, v_{i+1}) \in E$ für $1 \leq i \leq k$ und
- $v = v_1$ und $w = v_{k+1}$.

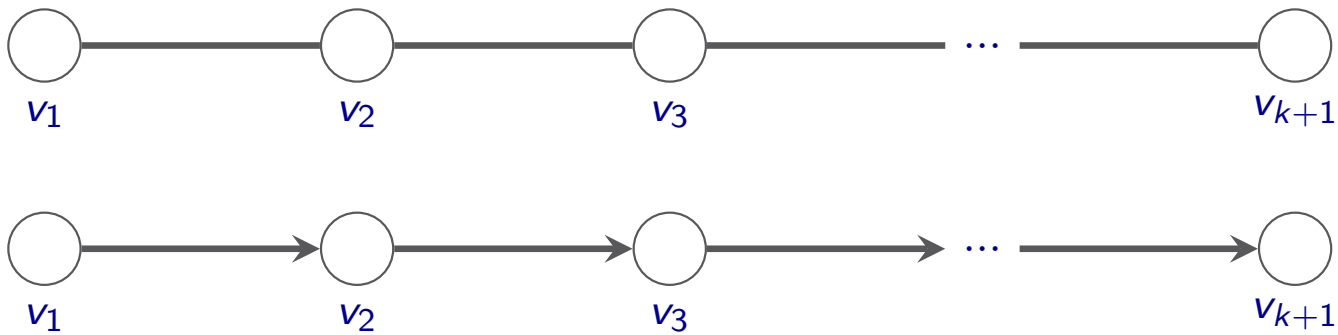


Weg, Pfad, Kantenzug

Definition

Ein Weg (auch Kantenzug) der Länge k in einem (gerichteten) Graphen $G = (V, E)$ von einem Knoten v zu einem Knoten w ist eine endliche Folge von Knoten $v_1, v_2, ..., v_{k+1}$, sodass:

- $(v_i, v_{i+1}) \in E$ für $1 \leq i \leq k$ und
- $v = v_1$ und $w = v_{k+1}$.

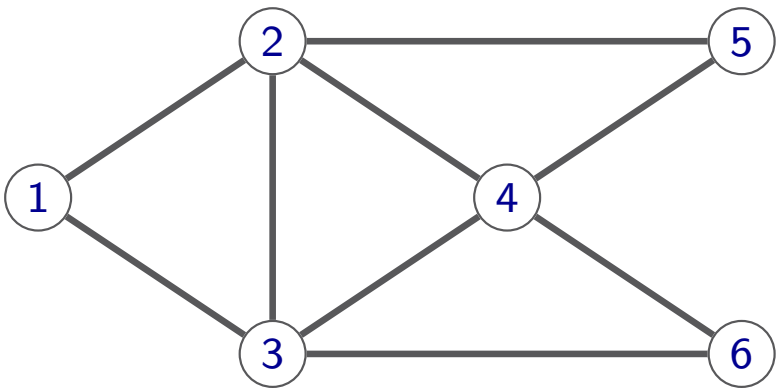


Ein Weg ist einfach (oder elementar oder ein Pfad), wenn kein Knoten (und damit keine Kante) mehrfach vorkommt.

Zusammenhang (ungerichtete Graphen)

Definition

Ein ungerichteter Graph $G = (V, E)$ ist zusammenhängend („connected“), wenn es in G zwischen allen Paaren von Knoten $u, v \in V$ einen Weg von u nach v gibt.

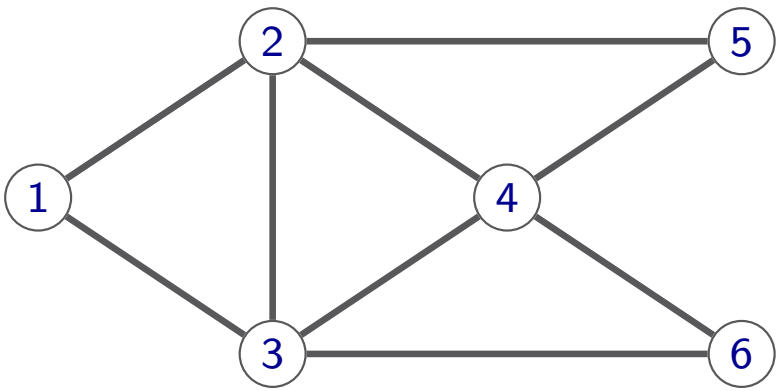


Zusammenhängender Graph

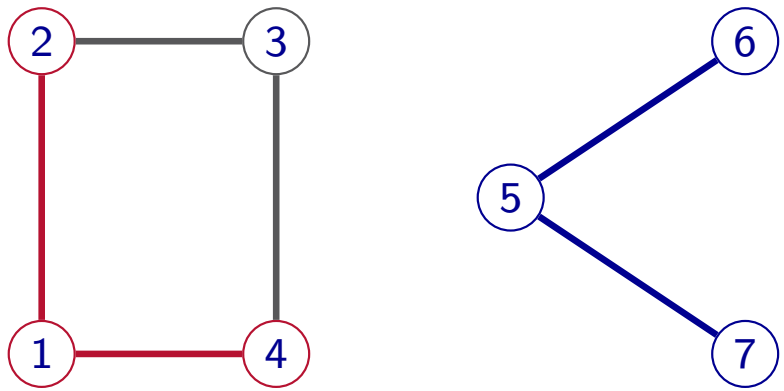
Zusammenhang (ungerichtete Graphen)

Definition

Ein ungerichteter Graph $G = (V, E)$ ist zusammenhängend („connected“), wenn es in G zwischen allen Paaren von Knoten $u, v \in V$ einen Weg von u nach v gibt.



Zusammenhängender Graph



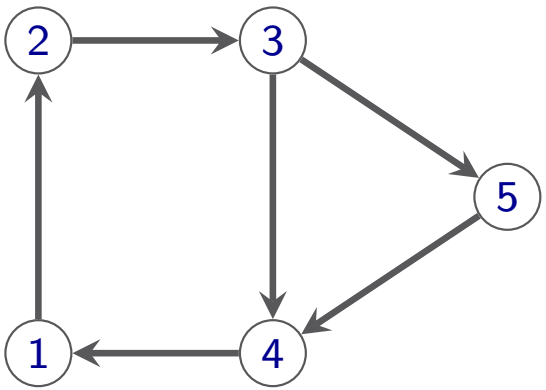
Der blaue Teilgraph ist eine Zusammenhangskomponente, der rote Teilgraph nicht

Eine Zusammenhangskomponente („connected component“) eines ungerichteten Graphen G ist ein maximal zusammenhängender Teilgraph.

Zusammenhang (gerichtete Graphen)

Definition

Ein gerichteter Graph $G = (V, A)$ ist stark zusammenhängend („strongly connected“), wenn es in G zwischen allen Paaren von Knoten $u, v \in V$ einen gerichteten Weg von u nach v gibt.

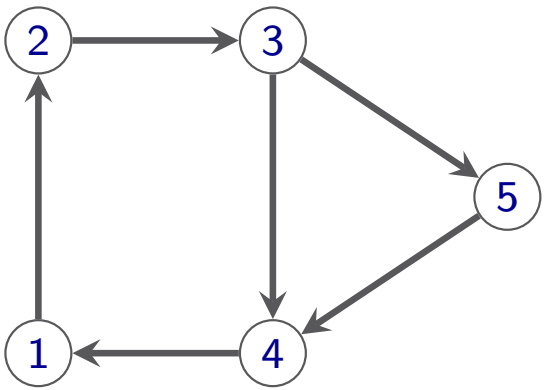


Stark zusammenhängender Graph

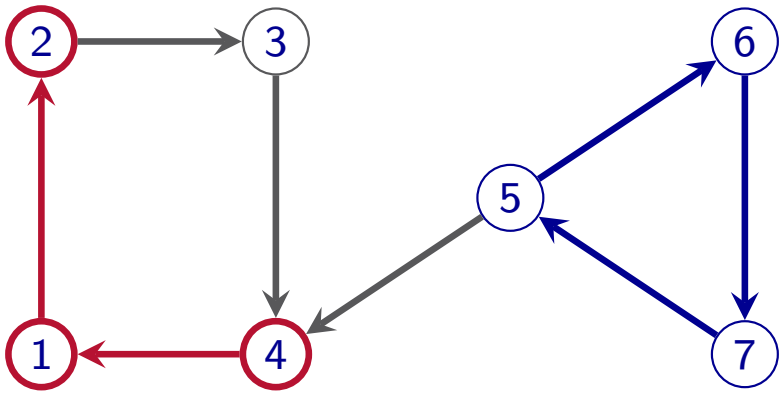
Zusammenhang (gerichtete Graphen)

Definition

Ein gerichteter Graph $G = (V, A)$ ist stark zusammenhängend („strongly connected“), wenn es in G zwischen allen Paaren von Knoten $u, v \in V$ einen gerichteten Weg von u nach v gibt.



Stark zusammenhängender Graph



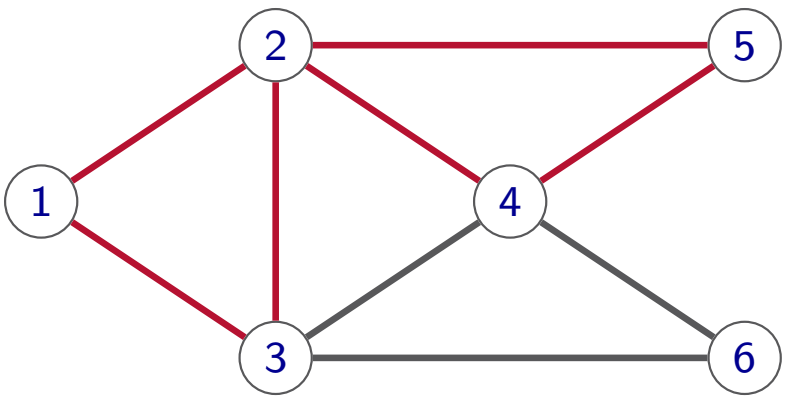
Der blaue Teilgraph ist eine starke Zusammenhangskomponente, der rote Teilgraph nicht

Eine starke Zusammenhangskomponente eines gerichteten Graphen G ist ein maximal stark zusammenhängender Teilgraph.

Kreise

Definition

Ein geschlossener Weg in einem (gerichteten) Graphen $G = (V, E)$ ist ein (gerichteter) Weg mit gleichem Start- und Endknoten. Ein Kreis ist ein Weg $v_1, \ldots, v_k$ mit $v_1 = v_k$ und $v_i \neq v_j$ für $1 \leq i < j < k$.

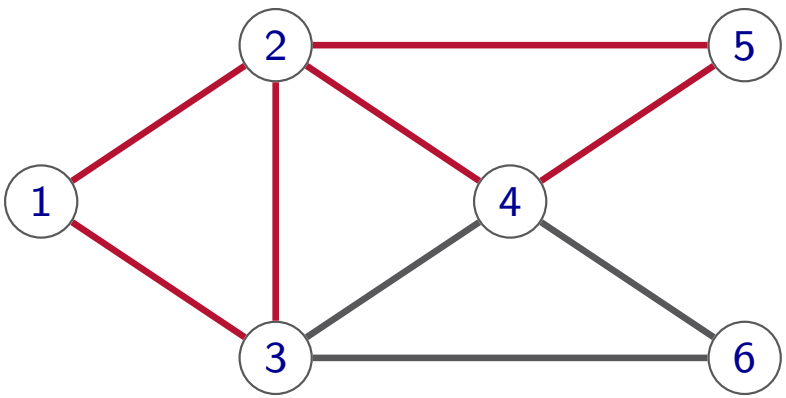


Der Weg (1,2,4,5,2,3,1) ist kein Kreis

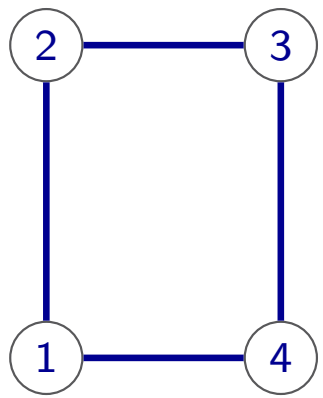
Kreise

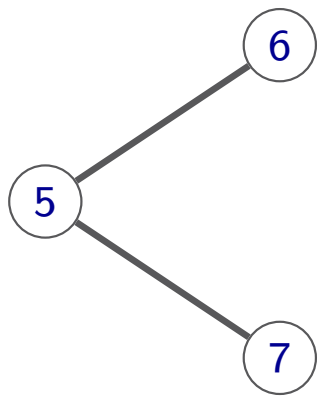
Definition

Ein geschlossener Weg in einem (gerichteten) Graphen $G = (V, E)$ ist ein (gerichteter) Weg mit gleichem Start- und Endknoten. Ein Kreis ist ein Weg $v_1, \ldots, v_k$ mit $v_1 = v_k$ und $v_i \neq v_j$ für $1 \leq i < j < k$.



Der Weg (1,2,4,5,2,3,1) ist kein Kreis





Der Weg (1,2,3,4,1) ist ein Kreis

Bipartite Graphen

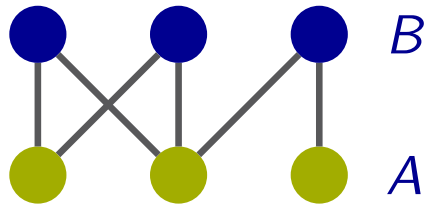
Definition

Ein Graph $G = (V, E)$ heißt bipartit, wenn es eine Partition von V in zwei disjunkte Teilmengen A und B gibt, so dass keine Kante $e \in E$ beide Endpunkte in derselben Teilmenge hat. $A \dot{\cup} B$ heißt Bipartition von G.

Bipartite Graphen

Definition

Ein Graph $G = (V, E)$ heißt bipartit, wenn es eine Partition von V in zwei disjunkte Teilmengen A und B gibt, so dass keine Kante $e \in E$ beide Endpunkte in derselben Teilmenge hat. $A \dot{\cup} B$ heißt Bipartition von G.

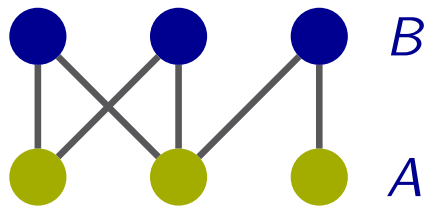


Ein Graph $G = (A \cup B, E)$ heißt vollständig bipartit, wenn das Hinzufügen einer Kante die Bipartitheit zerstören würde. ($\rightarrow K_{n,m}$)

Bipartite Graphen

Definition

Ein Graph $G = (V, E)$ heißt bipartit, wenn es eine Partition von V in zwei disjunkte Teilmengen A und B gibt, so dass keine Kante $e \in E$ beide Endpunkte in derselben Teilmenge hat. $A \dot{\cup} B$ heißt Bipartition von G.



Ein Graph $G = (A \cup B, E)$ heißt vollständig bipartit, wenn das Hinzufügen einer Kante die Bipartitheit zerstören würde. ($\rightarrow K_{n,m}$)

Proposition (König 1916)

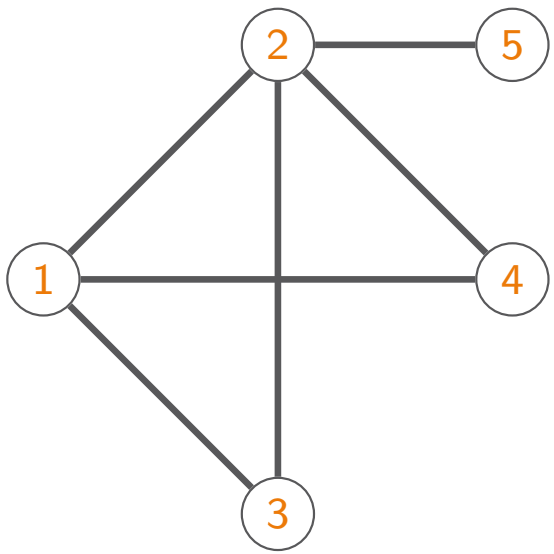
Sei $G = (V, E)$ ein ungerichteter Graph. G ist bipartit $\Leftrightarrow$ G hat keinen geschlossenen Weg ungerader Länge.

Kodierung von Graphen

Kodierung von Graphen

Kantenliste

Kantenliste eines Graphen $G = (V, E)$: Liste aller Kanten in E.



Kantenliste:
$\{1,2\}, \{1,3\}, \{1,4\}, \{2,3\}, \{2,4\}, \{2,5\}$

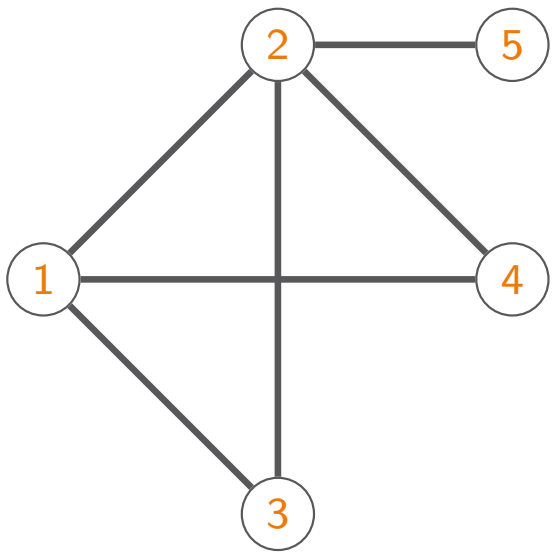
Kodierung von Graphen

Kantenliste

Kantenliste eines Graphen $G = (V, E)$: Liste aller Kanten in E.

Adjazenzlisten (Nachbarschaftslisten)

Sei $G = (V, E)$ ein Graph. Eine Adjazenzliste für einen Knoten $v \in V$ ist eine Liste aller Nachbarn von v, $N(v) := \{w \in V \mid (v, w) \in E\}$.



Kantenliste:
$\{1,2\}, \{1,3\}, \{1,4\}, \{2,3\}, \{2,4\}, \{2,5\}$

Array von Adjazenzlisten:

$1:$ $2, 3, 4$
$2:$ $1, 3, 4, 5$
$3:$ $1, 2$
$4:$ $1, 2$
$5:$ 2

Kodierung von Graphen

Adjazenzmatrix

Sei $G = (V, E, (w))$ ein (un)gerichteter (gewichteter) einfacher Graph. Die zugehörige Adjazenzmatrix $A \in \mathbb{N}^{n \times n}$ ist definiert mit

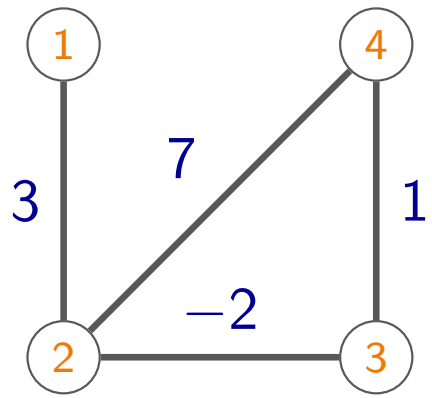
$$a_{ij} = \begin{cases} 1 & \text{falls } (i,j) \in E \\ 0 & \text{sonst} \end{cases} \quad \text{bzw.} \quad a_{ij} = \begin{cases} w_{ij} & \text{falls } (i,j) \in E \\ 0 & \text{sonst.} \end{cases}$$

Kodierung von Graphen

Adjazenzmatrix

Sei $G = (V, E, (w))$ ein (un)gerichteter (gewichteter) einfacher Graph. Die zugehörige Adjazenzmatrix $A \in \mathbb{N}^{n \times n}$ ist definiert mit

$$a_{ij} = \begin{cases} 1 & \text{falls } (i,j) \in E \\ 0 & \text{sonst} \end{cases} \quad \text{bzw.} \quad a_{ij} = \begin{cases} w_{ij} & \text{falls } (i,j) \in E \\ 0 & \text{sonst.} \end{cases}$$



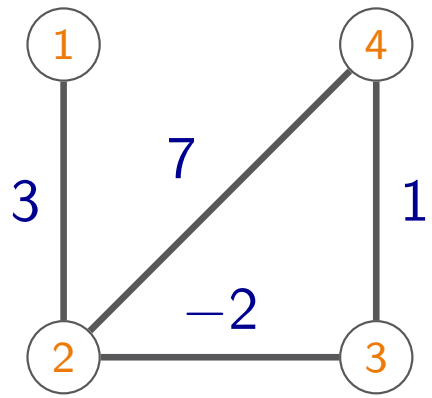
$$A = \begin{array}{c} \begin{matrix} 1 & 2 & 3 & 4 \end{matrix} \\ \begin{pmatrix} 0 & 3 & 0 & 0 \\ 3 & 0 & -2 & 7 \\ 0 & -2 & 0 & 1 \\ 0 & 7 & 1 & 0 \end{pmatrix} \end{array}$$

Kodierung von Graphen

Adjazenzmatrix

Sei $G = (V, E, (w))$ ein (un)gerichteter (gewichteter) einfacher Graph. Die zugehörige Adjazenzmatrix $A \in \mathbb{N}^{n \times n}$ ist definiert mit

$$a_{ij} = \begin{cases} 1 & \text{falls } (i,j) \in E \\ 0 & \text{sonst} \end{cases} \quad \text{bzw.} \quad a_{ij} = \begin{cases} w_{ij} & \text{falls } (i,j) \in E \\ 0 & \text{sonst.} \end{cases}$$



$$A = \begin{array}{c} \begin{matrix} 1 & 2 & 3 & 4 \end{matrix} \\ \begin{pmatrix} 0 & 3 & 0 & 0 \\ 3 & 0 & -2 & 7 \\ 0 & -2 & 0 & 1 \\ 0 & 7 & 1 & 0 \end{pmatrix} \end{array}$$

Adjazenzmatritzen ungerichteter Graphen sind symmetrisch.

Vor- und Nachteile

Kantenliste

- – Test ob $(u, v) \in E$: $O(m)$
- – $N(v)$ berechnen: $O(m)$
- \+ Speicherplatz $\Theta(m)$

Vor- und Nachteile

Kantenliste

- – Test ob $(u, v) \in E$: $O(m)$
- – $N(v)$ berechnen: $O(m)$
- \+ Speicherplatz $\Theta(m)$

Adjazenzlisten

- – Test ob $(u, v) \in E$: $O(\min\{\deg(u), \deg(v)\})$
- \+ $N(v)$ berechnen: $O(1)$ ($N(v)$ liegt als Liste bereits vor)
- \+ Speicherplatz $\Theta(n + m)$

Vor- und Nachteile

Kantenliste

- – Test ob $(u, v) \in E$: $O(m)$
- – $N(v)$ berechnen: $O(m)$
- \+ Speicherplatz $\Theta(m)$

Adjazenzlisten

- – Test ob $(u, v) \in E$: $O(\min\{\deg(u), \deg(v)\})$
- \+ $N(v)$ berechnen: $O(1)$ ($N(v)$ liegt als Liste bereits vor)
- \+ Speicherplatz $\Theta(n + m)$

Adjazenzmatrix

- \+ Test ob $(u, v) \in E$: $O(1)$
- – $N(v)$ berechnen: $O(n)$
- – Speicherplatz $\Theta(n^2)$

Vor- und Nachteile

Kantenliste

- – Test ob $(u, v) \in E$: $O(m)$
- – $N(v)$ berechnen: $O(m)$
- \+ Speicherplatz $\Theta(m)$

Adjazenzlisten

- – Test ob $(u, v) \in E$: $O(\min\{\deg(u), \deg(v)\})$
- \+ $N(v)$ berechnen: $O(1)$ ($N(v)$ liegt als Liste bereits vor)
- \+ Speicherplatz $\Theta(n + m)$

Adjazenzmatrix

- \+ Test ob $(u, v) \in E$: $O(1)$
- – $N(v)$ berechnen: $O(n)$
- – Speicherplatz $\Theta(n^2)$

Graphendurchläufe

Graphendurchlauf (graph traversal)

Wie können wir die Knoten oder Kanten eines Graphen systematisch durchlaufen? (Erreichbarkeitsanalyse)

- Breitensuche (breadth-first search, BFS)
- Tiefensuche (depth-first search, DFS)

Breitensuche (BFS)

Definition

Eine Warteschlange Q (Queue) ist eine Folge von Elementen, für die die folgenden Operationen ausführbar sind:

- $\mathrm{Enqueue}(Q, u)$ füge eine Element u am Ende ein
- $v := \mathrm{Dequeue}(Q)$ speichere das erste Element von Q als v ab und entferne es aus Q.

Breitensuche (BFS)

Definition

Eine Warteschlange Q (Queue) ist eine Folge von Elementen, für die die folgenden Operationen ausführbar sind:

- $\mathrm{Enqueue}(Q, u)$ füge eine Element u am Ende ein
- $v := \mathrm{Dequeue}(Q)$ speichere das erste Element von Q als v ab und entferne es aus Q.

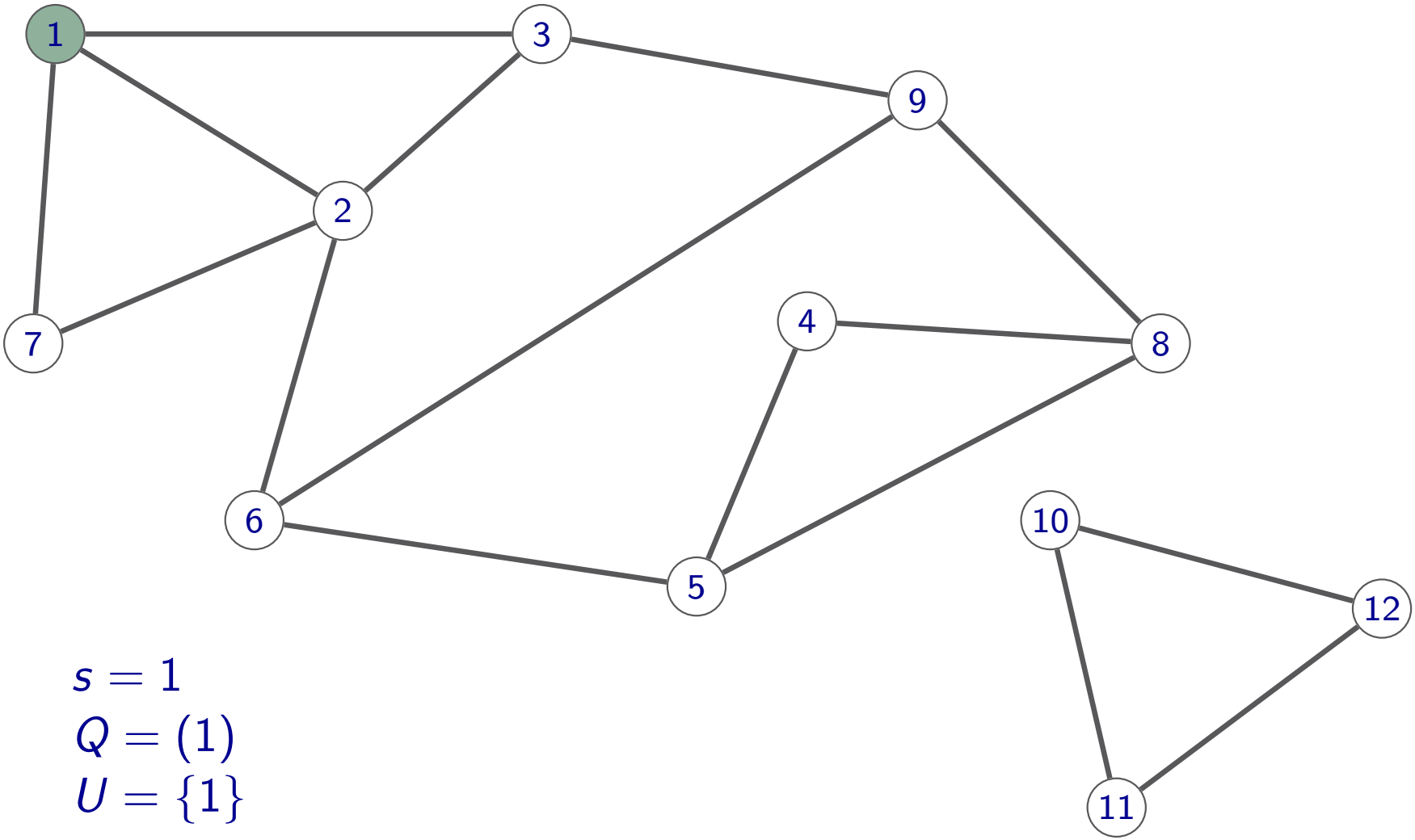
Idee: Beginnend beim Startknoten, speichere die Nachbarn eines bearbeiteten Knoten in die Queue, und arbeite die Queue nach first-in first-out Prinzip ab.

Breitensuche (BFS)

Input : Ein Graph $G = (V, E)$ und ein Startknoten $s \in V$.
Output : Menge $U \subseteq V$ aller Knoten, die von s erreichbar sind.

```
1 Initialisierung: U = {s}, Q = (s) Warteschlange
2 while Q ≠ ∅ do
3     v ← Dequeue(Q)
4     foreach u ∈ N(v) do
5         if u ∉ U then
6             Enqueue(Q, u)
7             U ← U ∪ {u}
8 return U
```

BFS: Beispiel

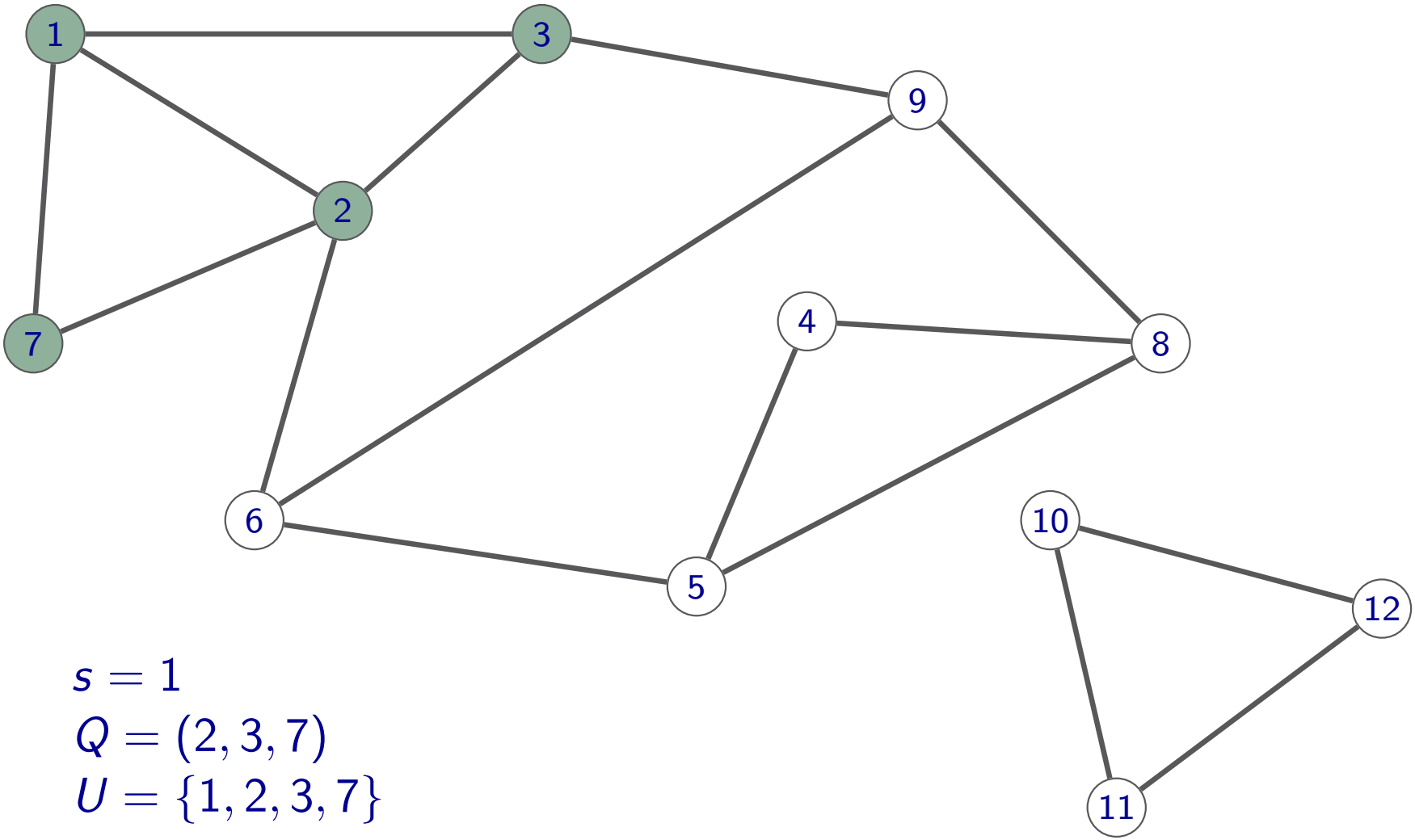


$s = 1$

$Q = (1)$

$U = \{1\}$

BFS: Beispiel

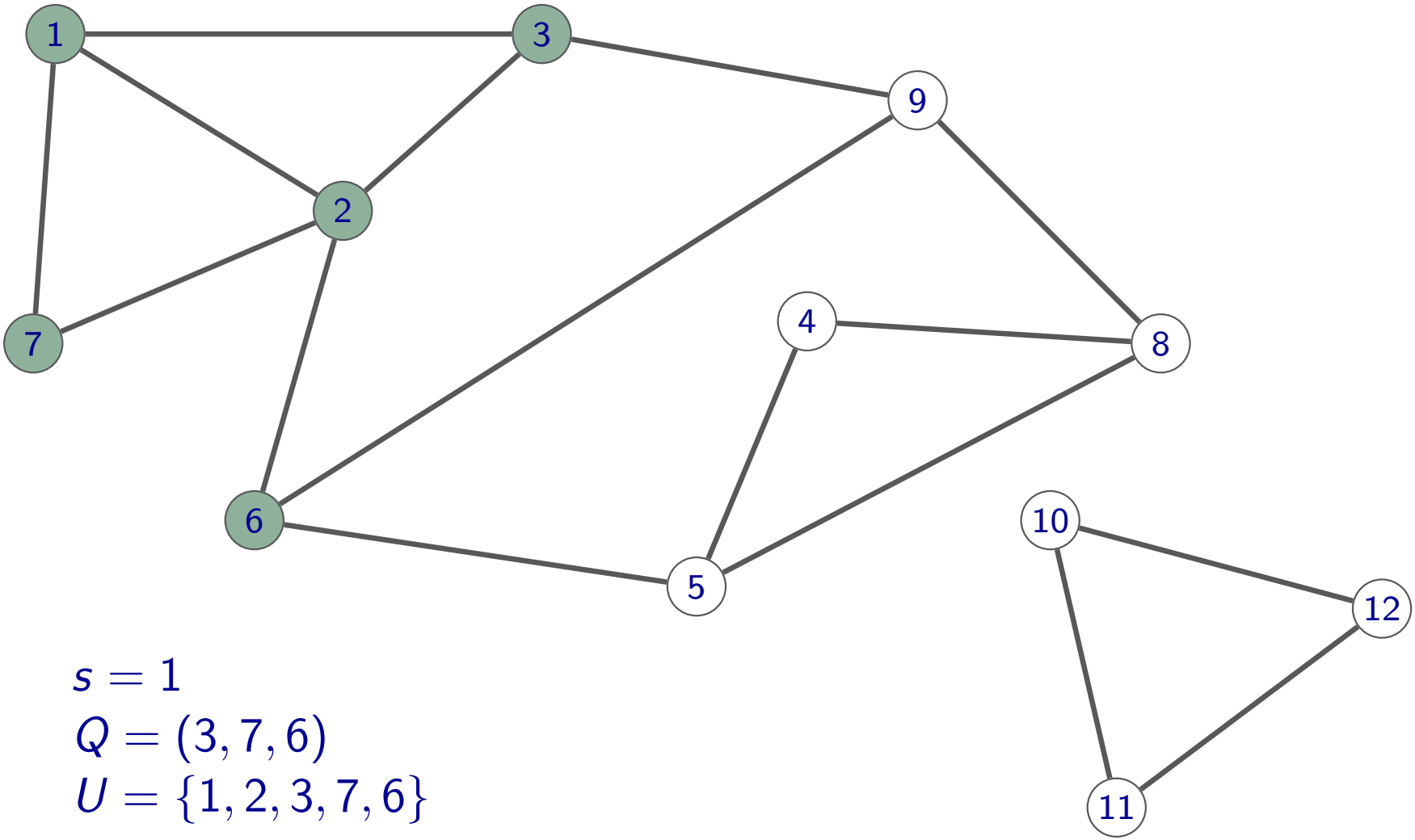


$s = 1$

$Q = (2, 3, 7)$

$U = \{1, 2, 3, 7\}$

BFS: Beispiel

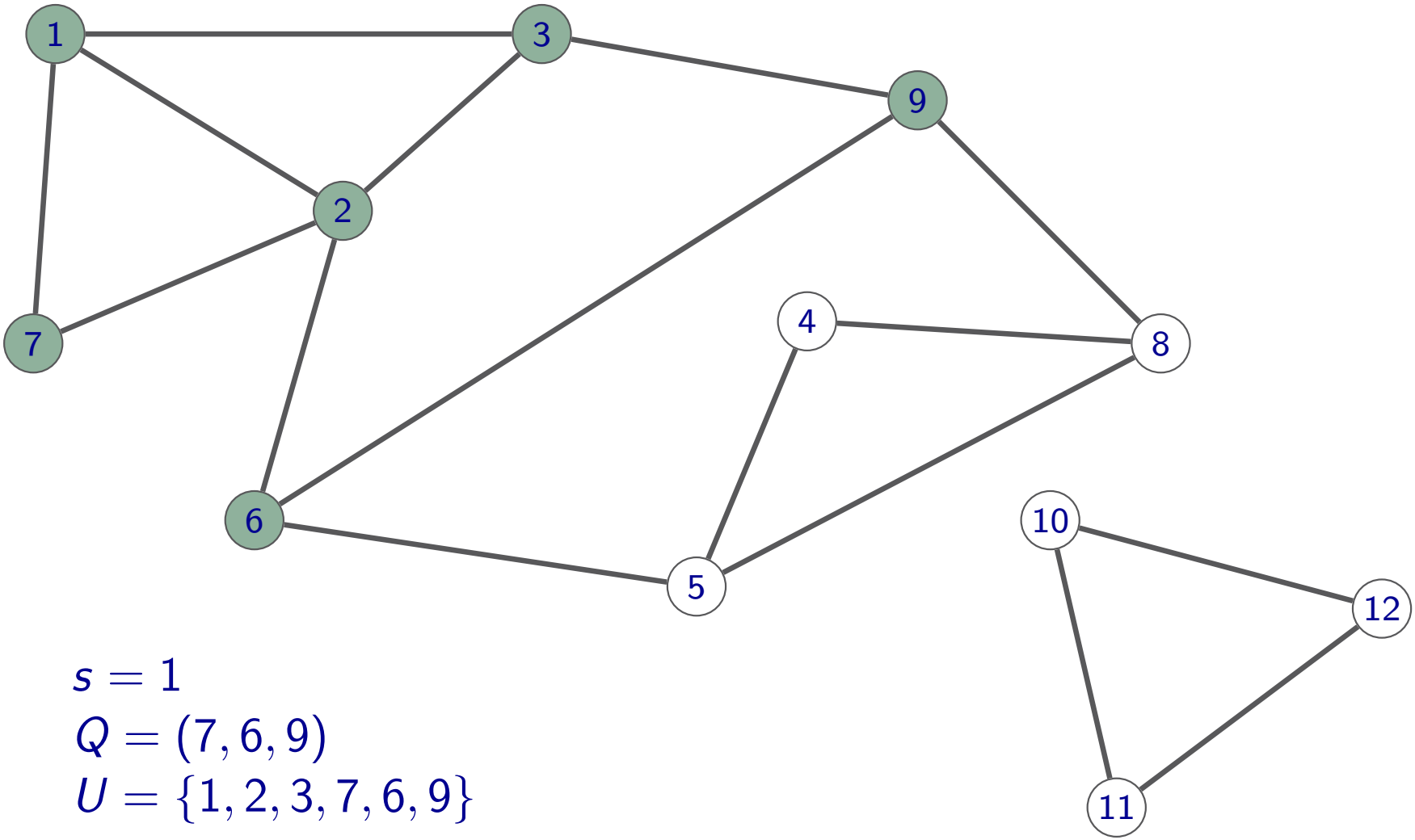


$s = 1$

$Q = (3, 7, 6)$

$U = \{1, 2, 3, 7, 6\}$

BFS: Beispiel

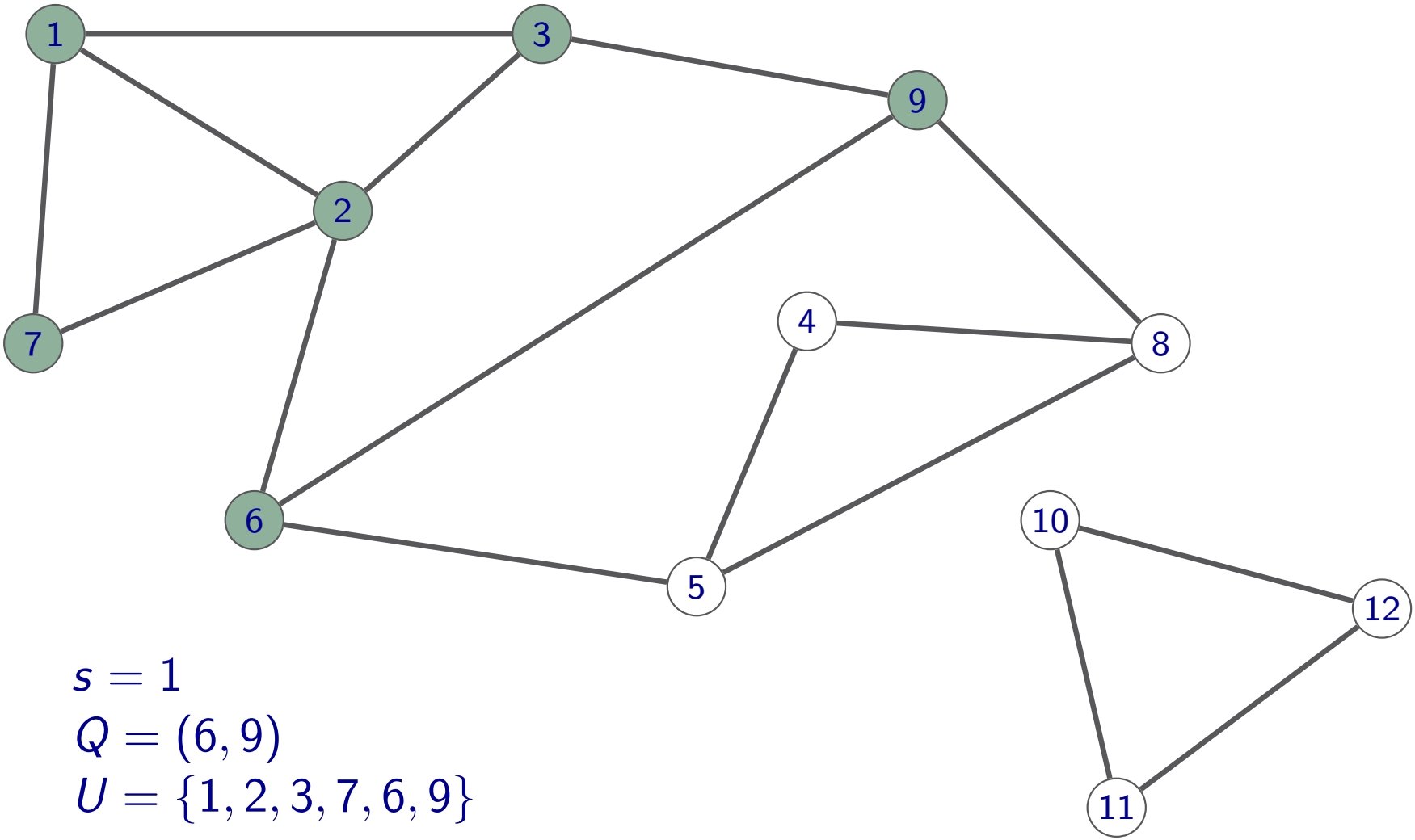


$s = 1$

$Q = (7, 6, 9)$

$U = \{1, 2, 3, 7, 6, 9\}$

BFS: Beispiel

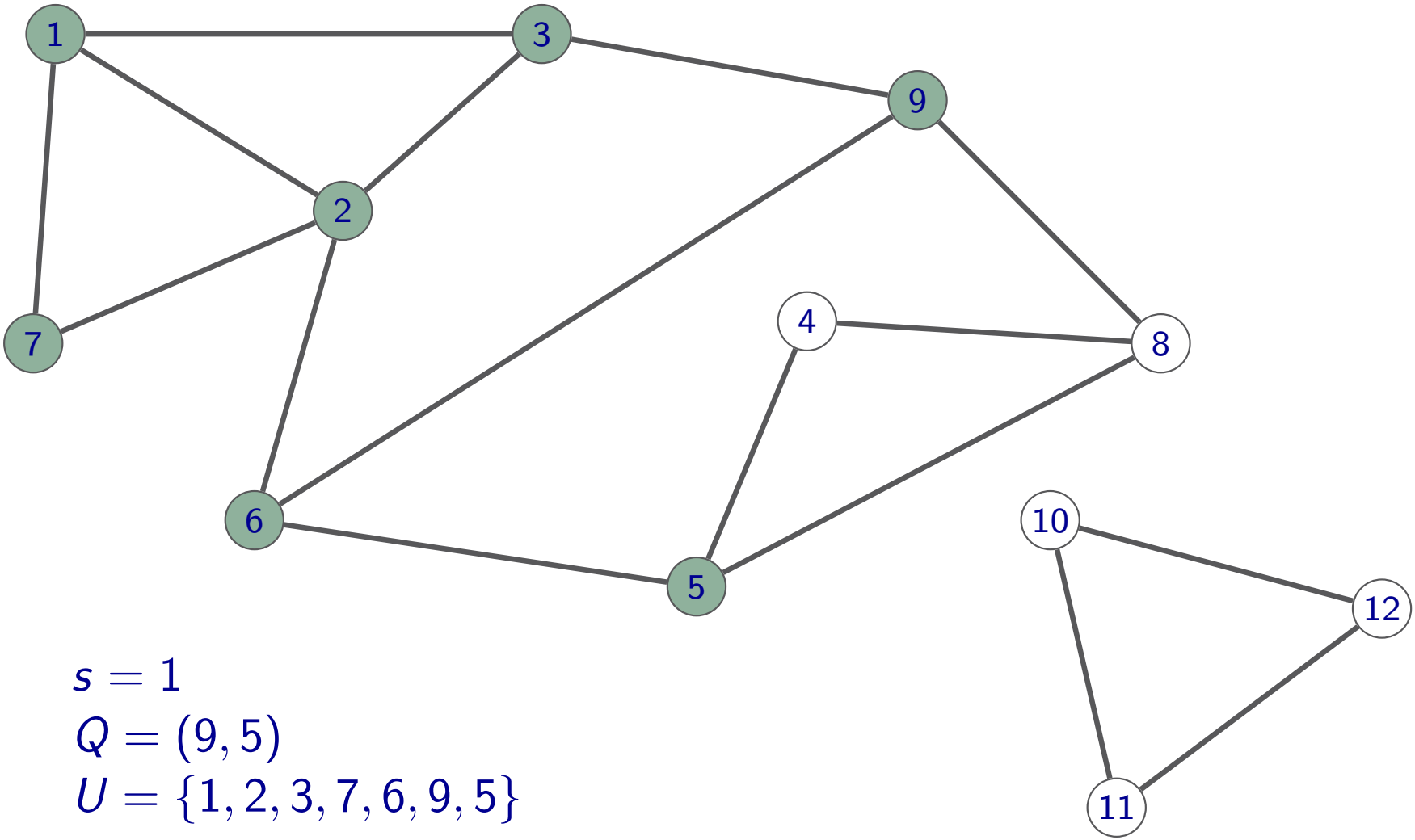


$s = 1$

$Q = (6, 9)$

$U = \{1, 2, 3, 7, 6, 9\}$

BFS: Beispiel

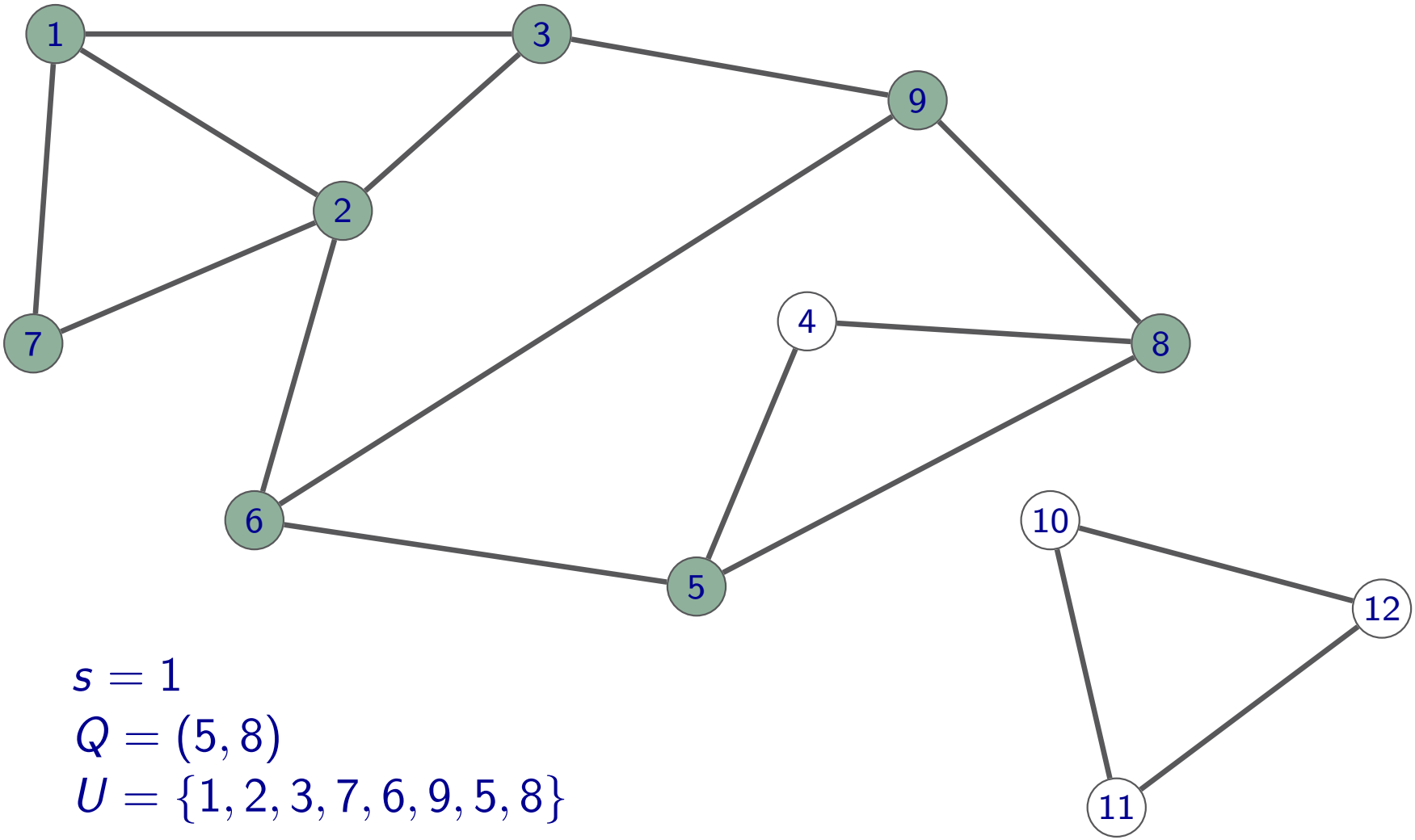


$s = 1$

$Q = (9, 5)$

$U = \{1, 2, 3, 7, 6, 9, 5\}$

BFS: Beispiel

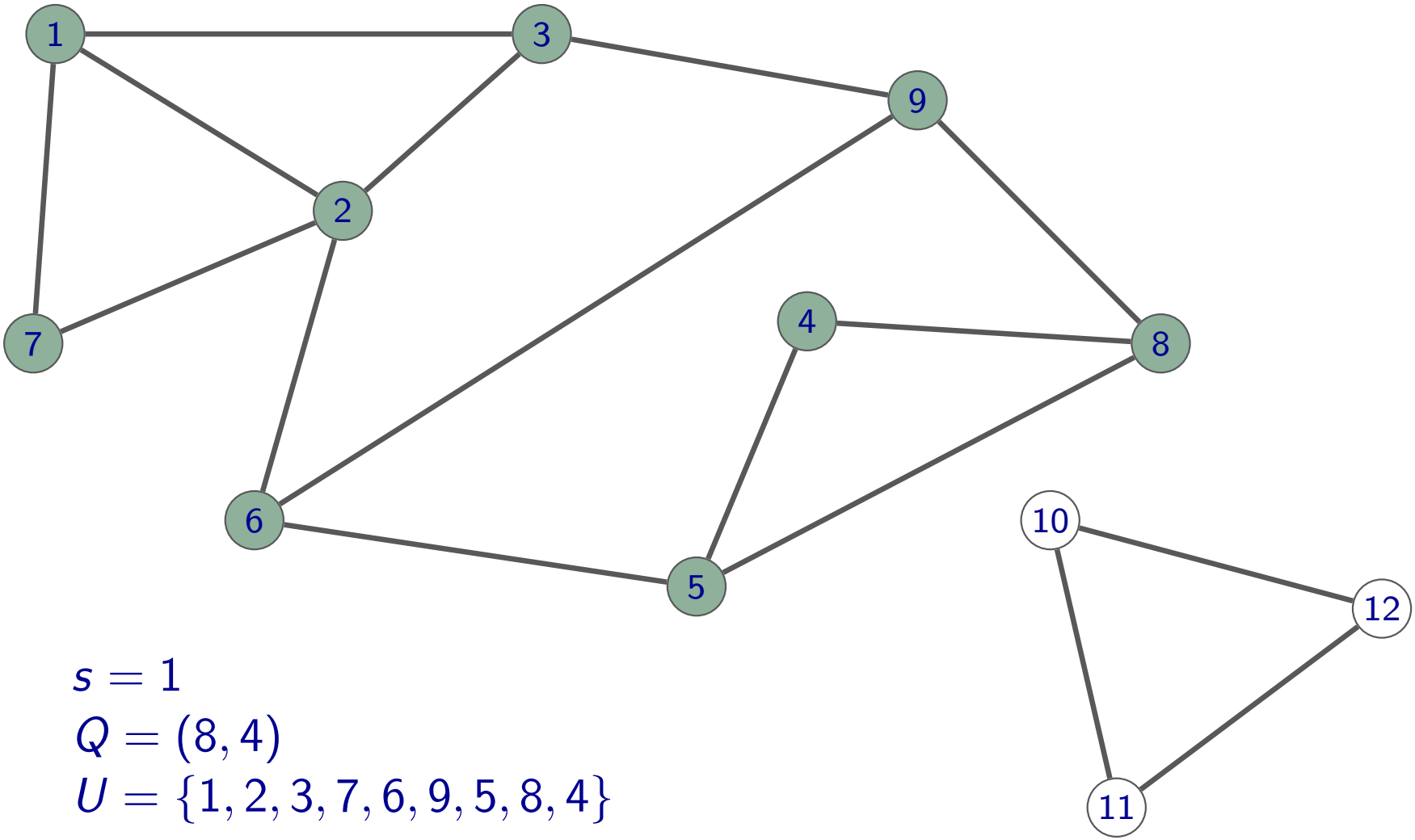


$s = 1$

$Q = (5, 8)$

$U = \{1, 2, 3, 7, 6, 9, 5, 8\}$

BFS: Beispiel

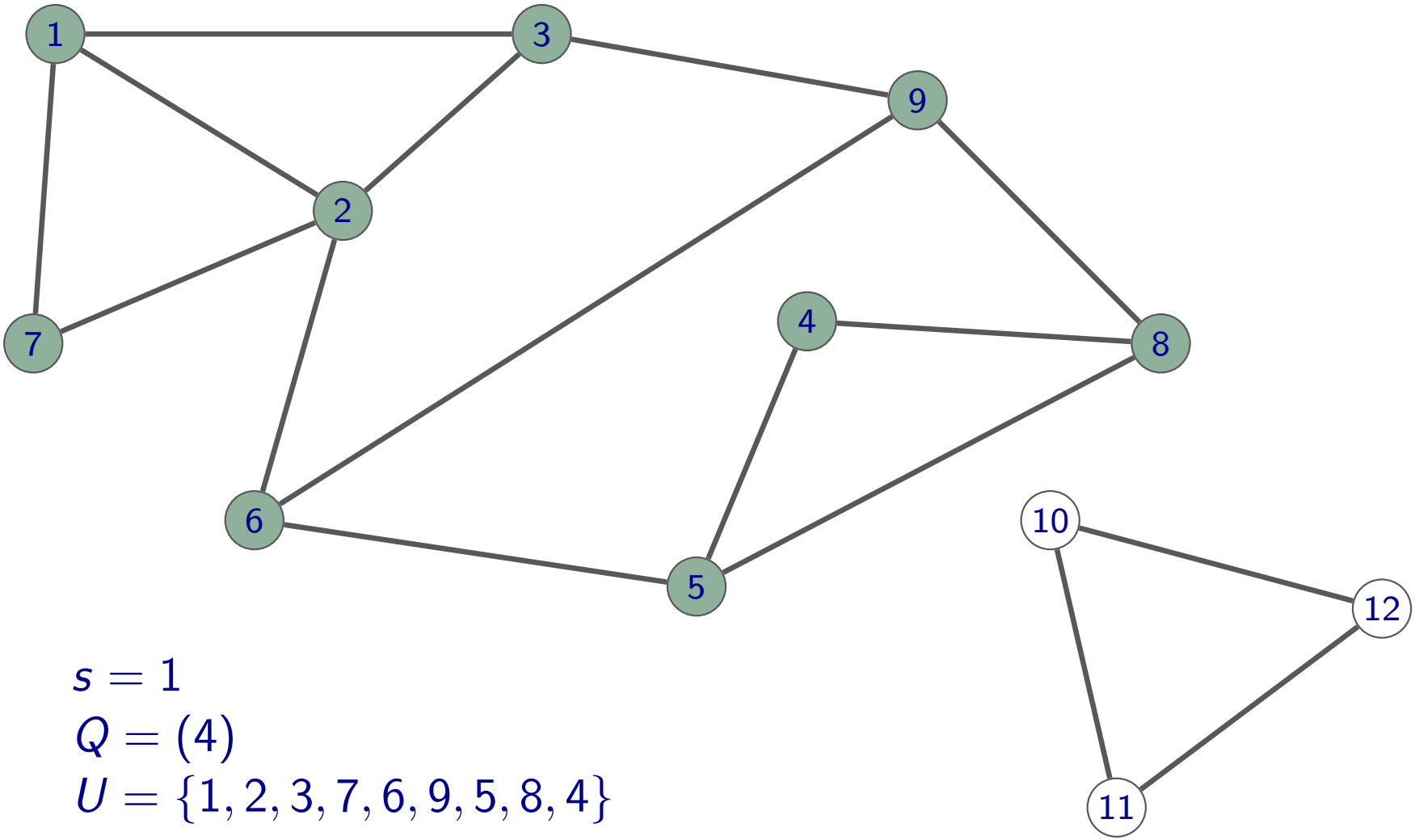


$s = 1$

$Q = (8, 4)$

$U = \{1, 2, 3, 7, 6, 9, 5, 8, 4\}$

BFS: Beispiel

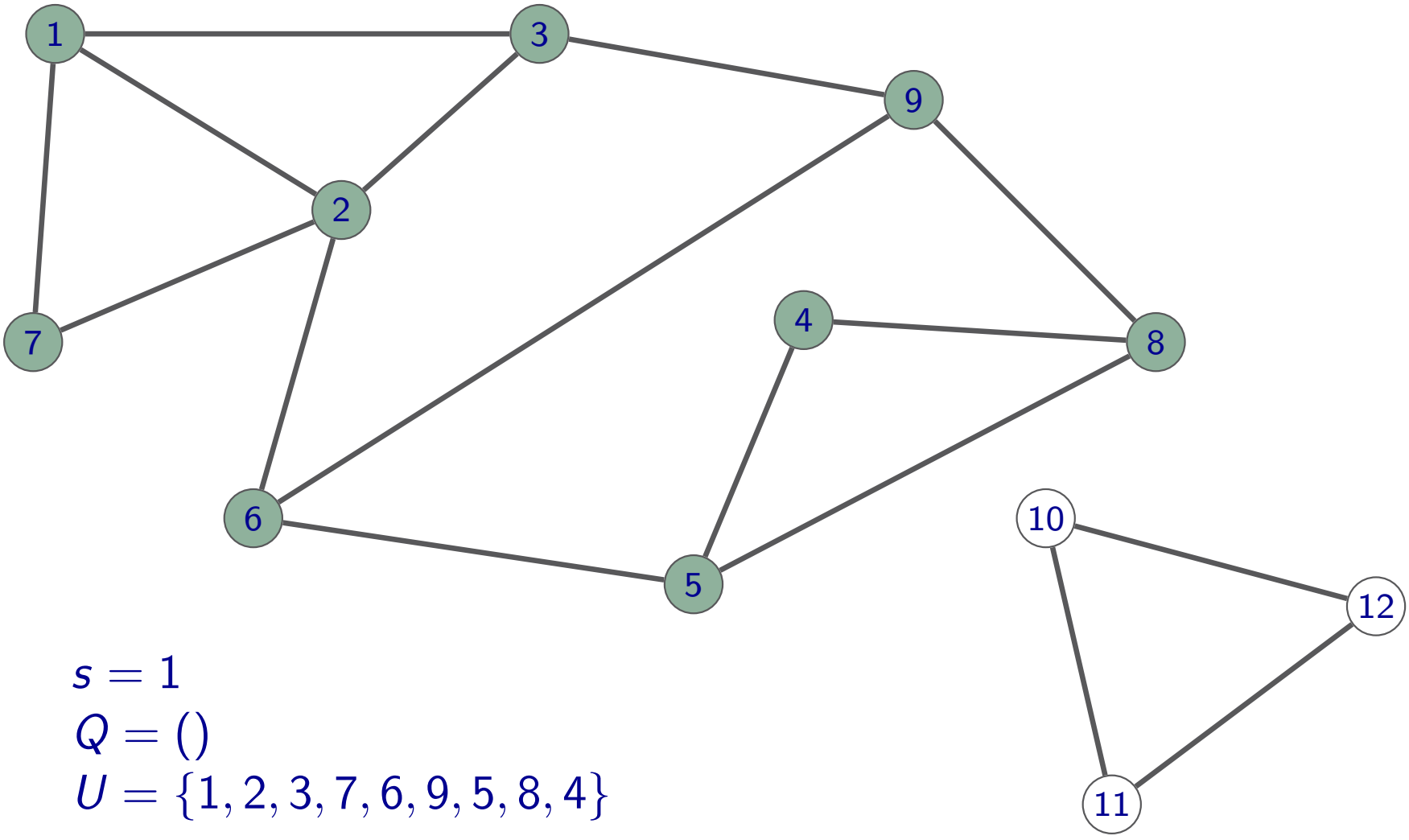


$s = 1$

$Q = (4)$

$U = \{1, 2, 3, 7, 6, 9, 5, 8, 4\}$

BFS: Beispiel



$s = 1$

$Q = ()$

$U = \{1, 2, 3, 7, 6, 9, 5, 8, 4\}$

BFS: Laufzeit

Laufzeit: Alle atomaren Operationen in Zeilen 1-7 benötigen Zeit $\mathcal{O}(1)$ (Adjazenzliste).

```
1 Initialisierung: U = {s}, Q = (s) Warteschlange
2 while Q ≠ ∅ do
3     v ← Dequeue(Q)
4     foreach u ∈ N(v) do
5         if u ∉ U then
6             Enqueue(Q, u)
7             U ← U ∪ {u}
8 return U
```

BFS: Laufzeit

```
1 Initialisierung: U = {s}, Q = (s) Warteschlange
2 while Q ≠ ∅ do
3     v ← Dequeue(Q)
4     foreach u ∈ N(v) do
5         if u ∉ U then
6             Enqueue(Q, u)
7             U ← U ∪ {u}
8 return U
```

Laufzeit: Alle atomaren Operationen in Zeilen 1-7 benötigen Zeit $\mathcal{O}(1)$ (Adjazenzliste).

– Jeder Knoten nur einmal an Q angehängt (Zeilen 5-7).

BFS: Laufzeit

```
1 Initialisierung: U = {s}, Q = (s) Warteschlange
2 while Q ≠ ∅ do
3     v ← Dequeue(Q)
4     foreach u ∈ N(v) do
5         if u ∉ U then
6             Enqueue(Q, u)
7             U ← U ∪ {u}
8 return U
```

Laufzeit: Alle atomaren Operationen in Zeilen 1-7 benötigen Zeit $\mathcal{O}(1)$ (Adjazenzliste).

- Jeder Knoten nur einmal an Q angehängt (Zeilen 5-7).
- Zeile 3 nur einmal pro Knoten, also $n = |V|$ mal.

BFS: Laufzeit

```
1 Initialisierung: U = {s}, Q = (s) Warteschlange
2 while Q ≠ ∅ do
3     v ← Dequeue(Q)
4     foreach u ∈ N(v) do
5         if u ∉ U then
6             Enqueue(Q, u)
7             U ← U ∪ {u}
8 return U
```

Laufzeit: Alle atomaren Operationen in Zeilen 1-7 benötigen Zeit $\mathcal{O}(1)$ (Adjazenzliste).

- Jeder Knoten nur einmal an Q angehängt (Zeilen 5-7).
- Zeile 3 nur einmal pro Knoten, also $n = |V|$ mal.
- Schleife 4-7 wird für jeden gefundenen Knoten v genau $\deg(v)$-mal durchlaufen.

BFS: Laufzeit

```
1 Initialisierung: U = {s}, Q = (s) Warteschlange
2 while Q ≠ ∅ do
3     v ← Dequeue(Q)
4     foreach u ∈ N(v) do
5         if u ∉ U then
6             Enqueue(Q, u)
7             U ← U ∪ {u}
8 return U
```

Laufzeit: Alle atomaren Operationen in Zeilen 1-7 benötigen Zeit $\mathcal{O}(1)$ (Adjazenzliste).

- Jeder Knoten nur einmal an Q angehängt (Zeilen 5-7).
- Zeile 3 nur einmal pro Knoten, also $n = |V|$ mal.
- Schleife 4-7 wird für jeden gefundenen Knoten v genau $\deg(v)$-mal durchlaufen.

$\Rightarrow$ Laufzeit: $\mathcal{O}(n + \sum_{v \in V} \deg(v)) \overset{\text{Handshake-L.}}{=} \mathcal{O}(n + 2|E|) = \mathcal{O}(n + m)$.

BFS: Laufzeit

```
1 Initialisierung: U = {s}, Q = (s) Warteschlange
2 while Q ≠ ∅ do
3     v ← Dequeue(Q)
4     foreach u ∈ N(v) do
5         if u ∉ U then
6             Enqueue(Q, u)
7             U ← U ∪ {u}
8 return U
```

Laufzeit: Alle atomaren Operationen in Zeilen 1-7 benötigen Zeit $\mathcal{O}(1)$ (Adjazenzliste).

- Jeder Knoten nur einmal an Q angehängt (Zeilen 5-7).
- Zeile 3 nur einmal pro Knoten, also $n = |V|$ mal.
- Schleife 4-7 wird für jeden gefundenen Knoten v genau $\deg(v)$-mal durchlaufen.

$\Rightarrow$ Laufzeit: $\mathcal{O}(n + \sum_{v \in V} \deg(v)) \overset{\text{Handshake-L.}}{=} \mathcal{O}(n + 2|E|) = \mathcal{O}(n + m)$.

Da der Input Größe $\mathcal{O}(n + m)$ hat, ist BFS ein Linearzeit-Algorithmus.

BFS: Korrektheit

1. Algorithmus terminiert (siehe Laufzeitanalyse).

BFS: Korrektheit

1. Algorithmus terminiert (siehe Laufzeitanalyse).

2. Bleibt zu zeigen, dass U die Menge der von s aus erreichbaren Knoten ist.

BFS: Korrektheit

1. Algorithmus terminiert (siehe Laufzeitanalyse).

2. Bleibt zu zeigen, dass U die Menge der von s aus erreichbaren Knoten ist.

Definition. Sei $dist(u, v)$ die Länge eines kürzesten Pfades von u nach v (Zahl der Kanten auf dem Pfad) bzw. ∞ falls kein Pfad existiert.

BFS: Korrektheit

1. Algorithmus terminiert (siehe Laufzeitanalyse).

2. Bleibt zu zeigen, dass U die Menge der von s aus erreichbaren Knoten ist.

Definition. Sei $dist(u, v)$ die Länge eines kürzesten Pfades von u nach v (Zahl der Kanten auf dem Pfad) bzw. ∞ falls kein Pfad existiert.

Sei $U_k = \{v \in V \mid dist(s, v) \leq k\}$ für $k \in \mathbb{N}$.

BFS: Korrektheit

1. Algorithmus terminiert (siehe Laufzeitanalyse).

2. Bleibt zu zeigen, dass U die Menge der von s aus erreichbaren Knoten ist.

Definition. Sei $dist(u, v)$ die Länge eines kürzesten Pfades von u nach v (Zahl der Kanten auf dem Pfad) bzw. ∞ falls kein Pfad existiert.

Sei $U_k = \{v \in V \mid dist(s, v) \leq k\}$ für $k \in \mathbb{N}$.
Zu zeigen: $\bigcup_{k \in \mathbb{N}} U_k = U$.

BFS: Korrektheit

1. Algorithmus terminiert (siehe Laufzeitanalyse).

2. Bleibt zu zeigen, dass U die Menge der von s aus erreichbaren Knoten ist.

Definition. Sei $dist(u, v)$ die Länge eines kürzesten Pfades von u nach v (Zahl der Kanten auf dem Pfad) bzw. ∞ falls kein Pfad existiert.

Sei $U_k = \{v \in V \mid dist(s, v) \leq k\}$ für $k \in \mathbb{N}$.
Zu zeigen: $\bigcup_{k \in \mathbb{N}} U_k = U$.

„$\supseteq$“: Angenommen u ist erster Knoten mit $dist(s, u) = \infty$, der U zugewiesen wird (in Zeile 7).

BFS: Korrektheit

1. Algorithmus terminiert (siehe Laufzeitanalyse).

2. Bleibt zu zeigen, dass U die Menge der von s aus erreichbaren Knoten ist.

Definition. Sei $dist(u, v)$ die Länge eines kürzesten Pfades von u nach v (Zahl der Kanten auf dem Pfad) bzw. ∞ falls kein Pfad existiert.

Sei $U_k = \{v \in V \mid dist(s, v) \leq k\}$ für $k \in \mathbb{N}$.
Zu zeigen: $\bigcup_{k \in \mathbb{N}} U_k = U$.

„$\supseteq$“: Angenommen u ist erster Knoten mit $dist(s, u) = \infty$, der U zugewiesen wird (in Zeile 7). Dann ist er Nachbar von $v \in U$, also $dist(s, v) < \infty$. $\Rightarrow dist(s, u) \leq dist(s, v) + 1 < \infty$, Widerspruch.

BFS: Korrektheit

1. Algorithmus terminiert (siehe Laufzeitanalyse).

2. Bleibt zu zeigen, dass U die Menge der von s aus erreichbaren Knoten ist.

Definition. Sei $dist(u, v)$ die Länge eines kürzesten Pfades von u nach v (Zahl der Kanten auf dem Pfad) bzw. ∞ falls kein Pfad existiert.

Sei $U_k = \{v \in V \mid dist(s, v) \leq k\}$ für $k \in \mathbb{N}$.
Zu zeigen: $\bigcup_{k \in \mathbb{N}} U_k = U$.

„$\supseteq$“: Angenommen u ist erster Knoten mit $dist(s, u) = \infty$, der U zugewiesen wird (in Zeile 7). Dann ist er Nachbar von $v \in U$, also $dist(s, v) < \infty$. $\Rightarrow dist(s, u) \leq dist(s, v) + 1 < \infty$, Widerspruch.

BFS: Korrektheit

„$\subseteq$“: Induktion über k.

IA: $k = 0$: $s \in U$ ✓

BFS: Korrektheit

„$\subseteq$“: Induktion über k.

IA: $k = 0$: $s \in U$ ✓

IV: Sei k beliebig, aber fest und angenommen es gilt $U_k \subseteq U$.

BFS: Korrektheit

„$\subseteq$“: Induktion über k.

IA: $k = 0$: $s \in U$ ✓

IV: Sei k beliebig, aber fest und angenommen es gilt $U_k \subseteq U$.

IS: Sei $v \in V$ mit $dist(s, v) = k+1$.

BFS: Korrektheit

„$\subseteq$“: Induktion über k.

IA: $k = 0$: $s \in U$ ✓

IV: Sei k beliebig, aber fest und angenommen es gilt $U_k \subseteq U$.

IS: Sei $v \in V$ mit $dist(s, v) = k+1$. Dann $\exists$ Pfad $s, v_1, \ldots v_k, v$ mit $v_k \in U_k \subseteq U$ (wegen IV).

BFS: Korrektheit

„$\subseteq$“: Induktion über k.

IA: $k = 0$: $s \in U$ ✓

IV: Sei k beliebig, aber fest und angenommen es gilt $U_k \subseteq U$.

IS: Sei $v \in V$ mit $dist(s, v) = k+1$. Dann $\exists$ Pfad $s, v_1, \ldots v_k, v$ mit $v_k \in U_k \subseteq U$ (wegen IV). Dazu muss v_k in Q gewesen sein.

BFS: Korrektheit

„$\subseteq$“: Induktion über k.

IA: $k = 0$: $s \in U$ ✓

IV: Sei k beliebig, aber fest und angenommen es gilt $U_k \subseteq U$.

IS: Sei $v \in V$ mit $dist(s, v) = k+1$. Dann $\exists$ Pfad $s, v_1, \ldots v_k, v$ mit $v_k \in U_k \subseteq U$ (wegen IV). Dazu muss v_k in Q gewesen sein.

Betrachte Schleife, in der v_k abgearbeitet und Nachbarn betrachtet werden. Da $(v_k, v) \in E$, wird v in Zeile 4 betrachtet und falls es noch nicht in U war in Zeilen 5-7 zu U hinzugefügt.

BFS: Korrektheit

„$\subseteq$“: Induktion über k.

IA: $k = 0$: $s \in U$ ✓

IV: Sei k beliebig, aber fest und angenommen es gilt $U_k \subseteq U$.

IS: Sei $v \in V$ mit $dist(s, v) = k+1$. Dann $\exists$ Pfad $s, v_1, \ldots v_k, v$ mit $v_k \in U_k \subseteq U$ (wegen IV). Dazu muss v_k in Q gewesen sein.

Betrachte Schleife, in der v_k abgearbeitet und Nachbarn betrachtet werden. Da $(v_k, v) \in E$, wird v in Zeile 4 betrachtet und falls es noch nicht in U war in Zeilen 5-7 zu U hinzugefügt.
$\Rightarrow v \in U$ und damit gilt $U_{k+1} \subseteq U$. □

Anwendungen von **Breitensuche**:

- Erreichbarkeitsanalyse
- Distanzen in Verkehrsnetzen berechnen
- Graph auf Kreisfreiheit, Zusammenhang, Bipartitheit testen

  $\rightarrow$ Übungsaufgabe

Anwendungen von **Breitensuche**:

- Erreichbarkeitsanalyse
- Distanzen in Verkehrsnetzen berechnen
- Graph auf Kreisfreiheit, Zusammenhang, Bipartitheit testen

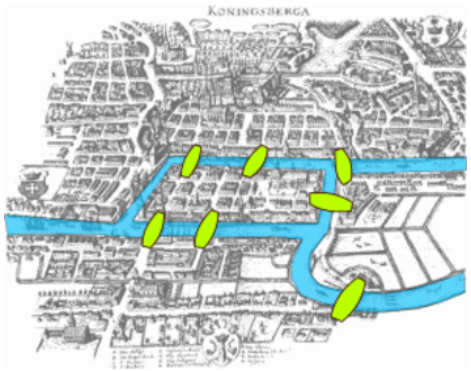
  $\rightarrow$ Übungsaufgabe

Für **Tiefensuche**:

- Ersetze Warteschlange (Queue, first-in first-out) durch Stapel (Stack, last-in first-out).

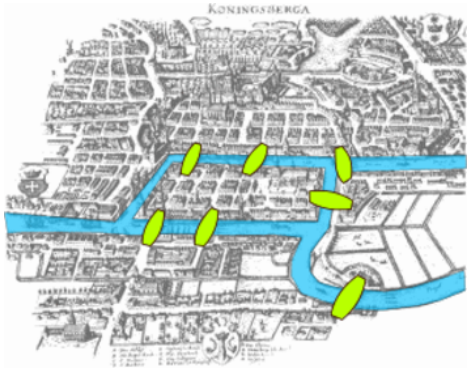
Das Königsberger Brückenproblem

Das Königsberger Brückenproblem



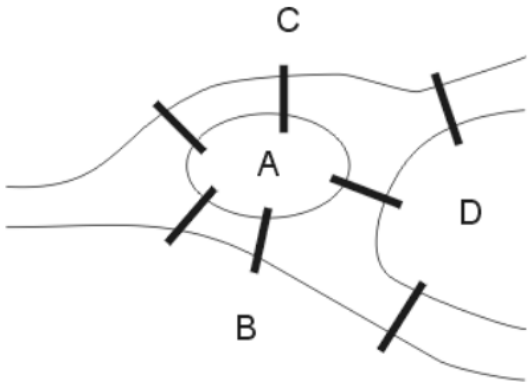
L. Euler: Gibt es einen (Rund)Weg, bei dem man jede der sieben Brücken genau einmal überquert?
$\rightarrow$ Frage nach Eulerweg (-tour) in Graphen

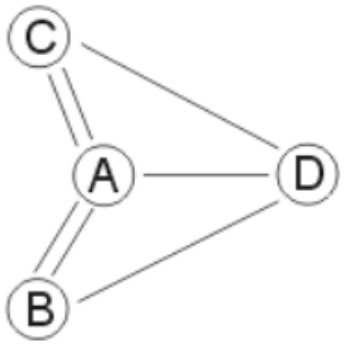
Das Königsberger Brückenproblem



L. Euler: Gibt es einen (Rund)Weg, bei dem man jede der sieben Brücken genau einmal überquert?
→ Frage nach Eulerweg (-tour) in Graphen

Geburtsstunde der Graphentheorie





L. Euler (1707-1783)

Eulersche Graphen

Definition

Sei $G = (V, E)$ ein (un-)gerichteter Graph.

- Ein Eulerweg in G ist ein Weg, der jede Kante in G genau einmal enthält. (Haus des Nikolaus)
- Eine Eulertour (Eulerkreis) ist ein geschlossener Eulerweg.
- Ein Graph heißt Eulersch, wenn er eine Eulertour enthält.

Eulersche Graphen

Definition

Sei $G = (V, E)$ ein (un-)gerichteter Graph.

- Ein Eulerweg in G ist ein Weg, der jede Kante in G genau einmal enthält. (Haus des Nikolaus)
- Eine Eulertour (Eulerkreis) ist ein geschlossener Eulerweg.
- Ein Graph heißt Eulersch, wenn er eine Eulertour enthält.

Satz (Euler 1736, Hierholzer 1873)

Sei $G = (V, E)$ ein ungerichteter, zusammenhängender Graph. G ist Eulersch genau dann wenn alle Knoten in V geraden Grad haben.

Eulersche Graphen

Definition

Sei $G = (V, E)$ ein (un-)gerichteter Graph.

- Ein Eulerweg in G ist ein Weg, der jede Kante in G genau einmal enthält. (Haus des Nikolaus)
- Eine Eulertour (Eulerkreis) ist ein geschlossener Eulerweg.
- Ein Graph heißt Eulersch, wenn er eine Eulertour enthält.

Satz (Euler 1736, Hierholzer 1873)

Sei $G = (V, E)$ ein ungerichteter, zusammenhängender Graph. G ist Eulersch genau dann wenn alle Knoten in V geraden Grad haben.

Beweis. → an Tafel

Wie findet man eine Eulertour?

Sei G Eulersch.

Wir haben gezeigt:

- Es existiert ein geschlossener Weg C in G.
- Jede Zusammenhangskomponente hat geraden Knotengrad auch ohne Kanten aus C und mind. einen Knoten mit C gemeinsam.

$\Rightarrow$ **Rekursion auf Zusammenhangskomponenten**

Hierholzer Algorithmus

Input : Ein Graph $G = (V, E)$
Output : Eine Eulertour K, oder false, falls keine existiert.

1 Falls Graph nicht Eulersch (check Knotengrad $\forall v \in V$), **return** false.
2 Wähle $v_0 \in V$ beliebig. Setze $K = \emptyset$.
3 **while** true **do**
4 Konstruiere geschlossenen Weg K' in G, der in v_0 startet und keine Kante doppelt enthält.
5 $K \leftarrow K \cup K'$, d.h. durchlaufe K bis v_0, dann K', dann Rest von K.
6 Setze $E \leftarrow E \setminus E(K)$.
7 **if** $E = \emptyset$ **then**
8 **return** K
9 **else**
10 $v_0 \leftarrow$ erster Knoten in K mit Grad > 0 (bzgl. verbliebener Kanten)

Hierholzer Algorithmus

Input : Ein Graph $G = (V, E)$
Output : Eine Eulertour K, oder false, falls keine existiert.

1 Falls Graph nicht Eulersch (check Knotengrad $\forall v \in V$), **return** false.
2 Wähle $v_0 \in V$ beliebig. Setze $K = \emptyset$.
3 **while** true **do**
4 Konstruiere geschlossenen Weg K' in G, der in v_0 startet und keine Kante doppelt enthält.
5 $K \leftarrow K \cup K'$, d.h. durchlaufe K bis v_0, dann K', dann Rest von K.
6 Setze $E \leftarrow E \setminus E(K)$.
7 **if** $E = \emptyset$ **then**
8 **return** K
9 **else**
10 $v_0 \leftarrow$ erster Knoten in K mit Grad > 0 (bzgl. verbliebener Kanten)

Korrektheit: Folgt aus konstruktivem Beweis des Satzes von Euler/Hierholzer.

Insbesondere terminiert der Algorithmus, da in jeder Iteration mindestens eine Kante entfernt wird.

Hierholzer Algorithmus - Laufzeit

Initialisierung (Schritte 1 und 2) in Zeit $\mathcal{O}(n + m)$.

Hierholzer Algorithmus - Laufzeit

Initialisierung (Schritte 1 und 2) in Zeit $\mathcal{O}(n+m)$.

Jede Iteration in der while-Schleife läuft in Zeit $\mathcal{O}(|K'|)$

Hierholzer Algorithmus - Laufzeit

Initialisierung (Schritte 1 und 2) in Zeit $\mathcal{O}(n+m)$.

Jede Iteration in der while-Schleife läuft in Zeit $\mathcal{O}(|K'|)$

- Adjazenzlisten enthalten nur verbliebene Kanten

Hierholzer Algorithmus - Laufzeit

Initialisierung (Schritte 1 und 2) in Zeit $\mathcal{O}(n+m)$.

Jede Iteration in der while-Schleife läuft in Zeit $\mathcal{O}(|K'|)$

- Adjazenzlisten enthalten nur verbliebene Kanten
- Nutze immer die erste Kante in der Liste

Hierholzer Algorithmus - Laufzeit

Initialisierung (Schritte 1 und 2) in Zeit $\mathcal{O}(n+m)$.

Jede Iteration in der while-Schleife läuft in Zeit $\mathcal{O}(|K'|)$

- Adjazenzlisten enthalten nur verbliebene Kanten
- Nutze immer die erste Kante in der Liste
- Doubly Linked List zum Löschen der Kanten in Zeit $\mathcal{O}(1)$

Hierholzer Algorithmus - Laufzeit

Initialisierung (Schritte 1 und 2) in Zeit $\mathcal{O}(n+m)$.

Jede Iteration in der while-Schleife läuft in Zeit $\mathcal{O}(|K'|)$

- Adjazenzlisten enthalten nur verbliebene Kanten
- Nutze immer die erste Kante in der Liste
- Doubly Linked List zum Löschen der Kanten in Zeit $\mathcal{O}(1)$

In jeder Iteration werden $|K'|$ Kanten gelöscht, also terminiert die while-Scheife nach $\mathcal{O}(m)$ Schritten.

Hierholzer Algorithmus - Laufzeit

Initialisierung (Schritte 1 und 2) in Zeit $\mathcal{O}(n+m)$.

Jede Iteration in der while-Schleife läuft in Zeit $\mathcal{O}(|K'|)$

- Adjazenzlisten enthalten nur verbliebene Kanten
- Nutze immer die erste Kante in der Liste
- Doubly Linked List zum Löschen der Kanten in Zeit $\mathcal{O}(1)$

In jeder Iteration werden $|K'|$ Kanten gelöscht, also terminiert die while-Scheife nach $\mathcal{O}(m)$ Schritten.

$\rightarrow \mathcal{O}(n+m)$ Linearzeit

Hamiltonsche Graphen

Definition

Sei $G = (V, E)$ ein ungerichteter Graph.

- Ein Kreis in G ist ein Hamiltonkreis, wenn er jeden Knoten in V genau einmal enthält.
- Ein Graph heisst hamiltonsch, wenn er einen Hamiltonkreis enthält.

Hamiltonsche Graphen

Definition

Sei $G = (V, E)$ ein ungerichteter Graph.

- Ein Kreis in G ist ein Hamiltonkreis, wenn er jeden Knoten in V genau einmal enthält.
- Ein Graph heisst hamiltonsch, wenn er einen Hamiltonkreis enthält.

Frage: Ist ein gegebener Graph hamiltonsch?

Hamiltonsche Graphen

Definition

Sei $G = (V, E)$ ein ungerichteter Graph.

- Ein Kreis in G ist ein Hamiltonkreis, wenn er jeden Knoten in V genau einmal enthält.
- Ein Graph heisst hamiltonsch, wenn er einen Hamiltonkreis enthält.

Frage: Ist ein gegebener Graph hamiltonsch?

- Vermeintlich ähnlich zu Frage, ob Graph Eulersch.
- Andere Komplexitätsklasse $\rightarrow$ Problem ist NP-vollständig
  Man nimmt an, dass es keinen effizienten Test gibt. (später mehr)

Zusammenfassung

- Begriffe der Graphentheorie
- Graphdurchläufe:
  - Breitensuche (BFS) in $\mathcal{O}(n + m)$
  - Tiefensuche (DFS) in $\mathcal{O}(n + m)$
  - Eulertour finden: Hierholzer Algorithmus in $\mathcal{O}(n + m)$