Begriff	Math. Def.	Text Def.
Graph	$G = (V, E)$	Menge von Knoten V und Kanten E
ungerichtet	$e = \{u, v\} \in E$	Kanten verbinden Knoten ohne Richtung
gerichtet	$e = (u, v) \in E$	Kanten verbinden Knoten mit Richtung
zusammen-hängend	G ist zusammenhängend	Es gibt einen Pfad zwischen allen Knoten im Graphen G
isomorph	$G_1 \cong G_2$	Zwei Graphen sind isomorph, wenn es eine bijektive Abbildung zwischen ihren Knoten gibt.
Brücke	$e \in E$ ist eine Brücke	Entfernen von e trennt den Graphen
Wanderung	$W = (v_1, v_2, \ldots, v_n)$	Folge von Knoten, die durch Kanten verbunden sind
Weg	$P = (v_1, v_2, \ldots, v_n)$	Wanderung ohne Knotenwiederholung
Pfad	$P = (v_1, v_2, \ldots, v_n)$	Weg ohne Kantenwiederholung
Kreis	$C = (v_1, v_2, \ldots, v_n)$	Pfad, der am Startknoten endet
Hamiltonpfad	$P = (v_1, v_2, \ldots, v_n)$	Pfad, der alle Knoten genau einmal besucht
Eulerpfad	$P = (v_1, v_2, \ldots, v_n)$	Pfad, der alle Kanten genau einmal besucht
Eulerkreis	$C = (v_1, v_2, \ldots, v_n)$	Kreis, der alle Kanten genau einmal besucht
Handshaking Lemma	$\sum_{v \in V} \deg(v) = 2\lvert E\rvert$	Summe der Grade aller Knoten ist gleich der doppelten Anzahl der Kanten
bipartit	$V = V_1 \cup V_2$ $E \subseteq V_1 \times V_2$	Knotenmenge in zwei disjunkte Mengen Kanten verbinden nur Knoten aus verschiedenen Mengen
Wald	$G = (V, E)$	Graph ohne Zyklen
Baum	$G = (V, E)$ $\lvert E\rvert = \lvert V\rvert - 1$	Zusammenhängender, azyklischer Graph.
MST	$T = (V, E')$	Minimaler Spannbaum eines Graphen G beinhaltet alle Knoten von G
Fluss-netzwerk	$N = (G, w, s, t)$	Netzwerk N mit Graph G Kapazitätsfunktion w, Startknoten s und Zielknoten t

Algorithm 2 Bellman-Moore-Ford, Dijkstra, A*

```
Require: Graph G = (V, E), ein Startknoten s, ein Zielknoten t
Ensure: Graph ist gewichtet, zusammenhängend und hat keine negativen Zyklen
Require: A* benötigt eine Heuristik für die Abschätzung der Distanz.
Ensure: Die Heuristik muss monoton und optimistisch sein.(h(v) ≤ dist(v,t))
  dist(s) ← 0                                   ▷ Abstand vom Startknoten
  W(s) ← {s}                                    ▷ Weg vom Startknoten
  F ← ∅                                         ▷ Nur in Dijkstra und A*
  for all v ∈ V \ {s} do
      dist(v) ← ∞
      W(v) ← ∅
  end for
  push(stack, s)                                ▷ Stack für Bellman-Moore-Ford
  while stack ≠ ∅ do                            ▷ In Dijkstra und A* ist die Bedingung t ∉ F
      v* ← pop(stack)                           ▷ Nur in Bellman-Moore-Ford
      v* ← arg min_{v∈V} dist(v)                ▷ In Dijkstra
      v* ← arg min_{v∈V}(dist(v) + h(v))        ▷ In A* mit Heuristik h
      F ← F ∪ {v*}                              ▷ Nur in Dijkstra und A*
      for all v ∈ N(v*) do                      ▷ Gehe alle Nachbarn von v* durch
          if dist(v*) + l(v*, v) < dist(v) then
              dist(v) ← dist(v*) + l(v*, v)
              W(v) ← W(v*) ∪ {v}
              push(stack, v)                    ▷ Nur in Bellman-Moore-Ford
          end if
      end for
  end while
  return dist, W   ▷ dist enthält alle Abstände, W alle Wege. Man kann auch nur den
  Abstand zum Zielknoten t zurückgeben oder nur die Wege, je nach Nutzung.
```

- **Welche Arten von Graphen kennen Sie?**
  - Ungerichtete Graphen
  - Gerichtete Graphen
  - Gewichtete Graphen
  - Zusammenhängende Graphen
  - Azyklische Graphen (Wald)
  - Azyklische zusammenhängende Graphen (Bäume)
  - Bipartite Graphen
  - Isomorphe Graphen
  - Eulergraphen
  - Hamiltongraphen
- **Welche verschiedenen Darstellungsformen von Graphen kennen Sie?**
  - Adjazenzmatrix => $A_{ij} = 1$ oder $A_{ij} = Gewicht$ wenn Kante zwischen i und j existiert, sonst 0
  - Adjazenzliste => Liste von Knoten, die jeweils ihre Nachbarn enthalten.
  - Kantenliste => Liste aller Kanten, z.B. $(u, v), (v, w)$

Algorithm 1 Fleury

```
Require: Graph G = (V, E)
Ensure: Graph ist zusammenhängend und hat entweder 0 oder 2 Knoten mit ungeradem
  Grad
  C ← ∅                                                        ▷ Kreis
  v ← getAnyNodeWithOddDegreeOrAnyNode(G)                      ▷ Startknoten
  while E ≠ ∅ do
      e ← getNeighboringNonBridge(v, E)
      if e = notfound then
          e ← getAnyNeighboringEdge(v, E)
      end if
      if e = notfound then
          return C                                 ▷ Error: Kein Kreis gefunden
      end if
      C ← C ∪ {e}                                  ▷ Füge Kante zum Kreis hinzu
      E ← E \ {e}                                  ▷ Entferne Kante aus dem Graphen
      v ← getOtherNode(e, v)                       ▷ Wechsel zum anderen Knoten der Kante
  end while
  return C                                         ▷ Gibt den Kreis zurück
```

Algorithm 3 Prim

```
Require: Graph G = (V, E), ein Wurzelknoten r
Ensure: Graph ist zusammenhängend, ungerichtet und gewichtet
  Q ← V
  for all u ∈ Q do
      dist(u) ← ∞                                  ▷ Abstand zur Wurzel
      pred(u) ← 0                                  ▷ Vorgängerknoten im MST
  end for
  dist(r) ← 0
  while Q ≠ ∅ do
      u ← arg min_{v∈Q} dist(v)                    ▷ Knoten mit minimalem Abstand
      Q ← Q \ {u}
      for all v ∈ N(u) do                          ▷ Gehe alle Nachbarn von u durch
          if v ∈ Q ∧ l(u, v) < dist(v) then
              dist(v) ← l(u, v)
              pred(v) ← u
          end if
      end for
  end while
```

Algorithm 4 Kruskal

```
Require: Graph G = (V, E)
Ensure: Graph ist zusammenhängend, ungerichtet und gewichtet
  E' ← ∅                                           ▷ MST Kanten
  L ← sort(E)                                      ▷ Sortiere Kanten nach Gewicht
  while L ≠ ∅ do
      e ← pop(L)                                   ▷ Nimm die leichteste Kante
      if (V, E' ∪ {e}) hat keinen Kreis then
          E' ← E' ∪ {e}                            ▷ Füge Kante zum MST hinzu
      end if
  end while
  return (V, E')                                   ▷ MST
```

Algorithm 5 Ford-Fulkerson

```
Require: Netzwerk N = (G, w, s, t) mit Graph G, Kapazitätsfunktion w, Startknoten s
  und Zielknoten t
Ensure: N ist ein Flussnetzwerk
  f ← 0                                            ▷ Aktueller Fluss
  while es gibt einen augmentierenden Pfad p von s nach t in G_f do
      p ← findAugmentingPath(s, t, G_f)            ▷ beliebiger augmentierenden Pfad
      c ← min_{e∈p}(w(e))                          ▷ Bestimme die Flusskapazität
      f ← f + c
      for all e ∈ p do
          w(e) ← w(e) − c
          f(e_rev) ← f(e_rev) + c                  ▷ Füge den Fluss in die Rückwärtskante ein
      end for
  end while
  return f                                         ▷ Gibt den maximalen Fluss zurück
```

- **Wie wird das Kantengewicht mathematisch definiert?**
  - Das Kantengewicht ist eine Funktion $l : E \to \mathbb{R}$, die jedem Kantenpaar (u, v) ein Gewicht zuordnet.
  - In gewichteten Graphen wird das Gewicht oft als Distanz oder Kosten interpretiert.
- **Was ist Nachbarschaft in Graphen?**
  - Die Nachbarschaft eines Knotens v in einem Graphen $G = (V, E)$ ist die Menge aller Knoten, die direkt mit v durch eine Kante verbunden sind.
  - In gerichteten Graphen gibt es auch eingehende und ausgehende Nachbarn:
    * Eingehende Nachbarn: $N_{in}(v) = \{u \in V \mid (u, v) \in E\}$
    * Ausgehende Nachbarn: $N_{out}(v) = \{u \in V \mid (v, u) \in E\}$
    * Nachbarn: $N(v) = N_{in}(v) \cup N_{out}(v)$
- **Was ist ein regulärer Graph?**
  - Ein regulärer Graph ist ein Graph, in dem alle Knoten den gleichen Grad haben.
  - Formal: Ein Graph G ist k-regulär, wenn $\deg(v) = k$ für alle $v \in V$ gilt.
  - Beispiele:
    * Ein 3-regulärer Graph hat für jeden Knoten genau 3 Nachbarn.
    * Ein vollständiger Graph K_n ist $(n-1)$-regulär, da jeder Knoten mit allen anderen Knoten verbunden ist.

- **Wann ist ein Graph eulersch?**
  - Ein Graph ist eulersch, wenn er einen Eulerpfad enthält, der alle Kanten genau einmal besucht.
  - Ein Graph ist eulersch, wenn er entweder:
    * keinen Knoten mit ungeradem Grad hat (Eulerkreis) oder
    * genau zwei Knoten mit ungeradem Grad hat (Eulerpfad).
- **Wann ist ein Graph hamiltonisch?**
  - Ein Graph ist hamiltonisch, wenn er einen Hamiltonpfad enthält, der alle Knoten genau einmal besucht.
  - Ein Graph ist hamiltonisch, wenn er einen Hamiltonkreis enthält, der alle Knoten genau einmal besucht und am Startknoten endet.
- **Gegeben ein beliebiger Graph G, ist es hier möglich einen Eulerkreis zu konstruieren?**
  - Nein, nicht immer.
  - Ein Eulerkreis benötigt folgende Bedingungen:
    * Der Graph muss zusammenhängend sein.
    * Alle Knoten müssen einen geraden Grad haben.
- **Welche Algorithmen kennen Sie zur Berechnung von Eulerkreisen? Was sind die Unterschiede?**
  - Fleury's Algorithmus:
    * Geht Kanten durch und vermeidet Brücken, wenn möglich.
    * Einfach zu implementieren, aber ineffizient für große Graphen.
  - Hierholzer's Algorithmus:
    * Baut den Eulerkreis rekursiv auf.
    * Effizienter und schneller als Fleury's Algorithmus.
- **Diskutieren Sie die Euler- bzw. Hamiltoneigenschaft für vollständige Graphen K_n mit $n > 3$!**
  - Vollständige Graphen K_n sind hamiltonisch und manchmal eulerisch für $n > 3$.
  - Euler: Jeder Knoten hat einen geraden Grad, falls n ungerade ist(Kanten pro Knoten ist $n - 1$). Wenn n gerade ist, hat jeder Knoten einen ungeraden Grad und es gibt keinen Eulerkreis.
  - Hamilton: Jeder Knoten ist mit jedem anderen Knoten verbunden, was einen Hamiltonkreis ermöglicht.
- **Diskutieren Sie die Lösbarkeit des Eulerkreisproblems mit dem des Hamiltonkreisproblems!**
  - Das Eulerkreisproblem ist in polynomialer Zeit lösbar, während das Hamiltonkreisproblem NP-vollständig ist.
  - Es gibt effiziente Algorithmen für spezielle Fälle des Eulerkreisproblems, während das Hamiltonkreisproblem in der Regel heuristische Ansätze erfordert.
- **Wie lautet die Abbruchbedingung der bidirektionalen Suche?**
  - Die naïve Abbruchbedingung der bidirektionalen Suche ist erreicht, wenn die Suchfronten von Start- und Zielknoten sich treffen.
  - Das bedeutet, dass ein Pfad zwischen den beiden Knoten gefunden wurde.
  -
- **In einem MST ist jede Verbindung die kürzeste Verbindung.**
  - Falsch.
  - Ein MST minimiert die Summe der Kantengewichte, aber nicht unbedingt die einzelnen Punkt-zu-Punkt-Verbindungen.
- **Ist der Spannbaum aus dem Dijkstra Algorithmus minimal?**
  - Falsch.
  - Der Dijkstra-Algorithmus findet den kürzesten Pfad von einem Startknoten zu allen anderen Knoten, aber der resultierende Baum ist nicht unbedingt minimal im Sinne des Spannbaums.
- **Welche Algorithmen für die Berechnung von Spannbäumen kennen Sie? Diskutieren Sie die Unterschiede!**
  - Prim's Algorithmus:
    * Startet bei einem Knoten und fügt iterativ die leichteste Kante hinzu, die einen neuen Knoten verbindet.
    * Effizient für dichte Graphen.
  - Kruskal's Algorithmus:
    * Sortiert die Kanten nach Gewicht und fügt sie zum MST hinzu, solange sie keinen Kreis bilden.
    * Effizient für spärliche Graphen.
- **Wann ist ein MST eindeutig?**
  - Ein MST ist eindeutig, wenn alle Kanten unterschiedliche Gewichte haben.
  - Wenn zwei Kanten das gleiche Gewicht haben, kann es mehrere mögliche MSTs geben.

- **Was unterscheidet eine Arboreszenz von einem gerichteten zyklenfreien Graphen?**
  - Eine Arboreszenz ist ein gerichteter, zyklenfreier Graph, der von einem Wurzelknoten ausgeht und alle anderen Knoten erreicht.
  - Ein gerichteter zyklenfreier Graph (DAG) kann mehrere Wurzelknoten haben und muss nicht notwendigerweise alle Knoten erreichen.
- **Mit welchem Algorithmus findet man minimale Arboreszenzen?**
  - Der Edmonds-Karp-Algorithmus ist ein bekannter Algorithmus zur Berechnung minimaler Arboreszenzen.
- **Was ist ein Webgraph?**
  - Ein Webgraph ist ein gerichteter Graph, der die Struktur des Internets oder eines Netzwerks von Webseiten darstellt.
  - Knoten repräsentieren Webseiten, und Kanten repräsentieren Hyperlinks zwischen diesen Webseiten.
- **Erklären Sie das Random Surfer Modell, welche Annahmen sind in dem Modell enthalten?**
  - Das Random Surfer Modell beschreibt das Verhalten von Nutzern im Internet, die zufällig Links zwischen Webseiten folgen.
  - Der Page Rank beschreibt die Chance, dass der Random Surfer auf einer bestimmten Seite landet.
- **Was bedeutet der Dämpfungsfaktor im Random Surfer Modell?**
  - Der Dämpfungsfaktor ist ein Wert zwischen 0 und 1, der die Wahrscheinlichkeit angibt, dass der Random Surfer einem Link folgt.
  - Ein Dämpfungsfaktor von 0,85 bedeutet beispielsweise, dass 85% der Zeit einem Link gefolgt wird und 15% der Zeit eine zufällige Seite besucht wird.
  - Dies verhindert, dass der Random Surfer in endlosen Schleifen stecken bleibt und sorgt für eine gleichmäßige Verteilung der Seitenbesuche.
- **Welches Verfahren wird für die Berechnung des Page Rank verwendet?**
  - Dafür wird das Random Surfer Modell verwendet, welches die Wahrscheinlichkeit beschreibt, dass ein Nutzer auf einer bestimmten Seite landet.
  - Der Page Rank wird iterativ berechnet, indem die Adjazenzmatrix des Webgraphen verwendet wird.
  - Der Dämpfungsfaktor wird dabei berücksichtigt, um die Wahrscheinlichkeit zu steuern, dass der Random Surfer einem Link folgt oder eine zufällige Seite besucht.
  - Die Berechnung erfolgt durch die Lösung des Eigenwertproblems der Adjazenzmatrix, wobei der Page Rank als Eigenvektor des Matrixprodukts interpretiert wird.
- **Ist eine Matrix irreduzibel und nicht-negativ, so gibt es auch einen positiven Eigenvektor.**
  - Wahr.
  - Eine irreduzible Matrix hat einen positiven Eigenvektor, da sie eine starke Verbindung zwischen den Zuständen darstellt.
- **Beschreiben Sie ein weiteres Zentralitätsmaß außer die Eigenvektorzentralität für Graphen!**
  - **Page Rank:** Misst die Wichtigkeit eines Knotens basierend auf der Anzahl und Qualität der eingehenden Verbindungen.
  - **Zwischenzentralität:** Misst, wie oft ein Knoten auf dem kürzesten Pfad zwischen anderen Knoten liegt.
  - **Nähezentralität:** Misst die durchschnittliche Entfernung eines Knotens zu allen anderen Knoten im Graphen.
- **Was ist ein Netzwerk und was ist ein Fluss?**
  - Ein Netzwerk ist ein Graph, wobei die Kanten Kapazitäten haben, die den maximalen Fluss zwischen den Knoten begrenzen.
  - Ein Fluss nutzt die Kapazität aus, muss aber die Flusserhaltung und die Kapazitätsbeschränkungen der Kanten respektieren.
  - Das Flusserhaltungsgesetz besagt, dass die Summe der eingehenden Flüsse gleich der Summe der ausgehenden Flüsse an jedem Knoten ist, außer am Start- und Zielknoten.
  - $f(u) = \sum_{v \in N_{in}(u)} f(v) - \sum_{v \in N_{out}(u)} f(v)$, wobei $f(u)$ der Fluss am Knoten u ist.
- **Was ist ein Flussnetzwerk?**
  - Ein Flussnetzwerk ist ein Netzwerk, in dessen Graphen es keine Retourkanten gibt, d.h. es gibt keine Kanten, die von einem Zielknoten zurück zu einem Startknoten führen.
- **Wie lässt sich ein augmentierender Pfad finden?**
  - Ein augmentierender Pfad ist ein Pfad im Flussnetzwerk, der von einem Startknoten zu einem Zielknoten führt und in dem noch Kapazität für zusätzlichen Fluss vorhanden ist.
  - Er kann durch Tiefensuche (DFS) oder Breitensuche (BFS) gefunden werden.
- **Lineare Programmierung ist nur dann effizient lösbar wenn alle Variablen kontinuierlich sind.**
  - Falsch.
  - Lineare Programmierung kann auch mit ganzzahligen Variablen gelöst werden, aber die Effizienz kann variieren.