1 Definitionen

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 Kantenwiederholung
Pfad	$P = (v_1, v_2, \ldots, v_n)$	Weg ohne Knotenwiederholung
Kreis	$C = (v_1, v_2, \ldots, v_n)$	Weg, der am Startknoten endet
Hamiltonpfad	$P = (v_1, v_2, \ldots, v_n)$	Pfad, der alle Knoten genau einmal besucht
Eulerweg	$P = (v_1, v_2, \ldots, v_n)$	Weg, der alle Kanten genau einmal besucht
Eulerkreis	$C = (v_1, v_2, \ldots, v_n)$	Kreis, der alle Kanten genau einmal besucht
Handshaking Lemma	$\sum_{v \in V} \deg(v) = 2\|E\|$	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	$G = (V, E)$ $\|E\| = \|V\| - 1$	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

2 Algorithmen

Algorithm 1 Fleury

Require: Graph $G = (V, E)$
Ensure: Graph ist zusammenhängend und hat entweder 0 oder 2 Knoten mit ungeradem Grad

$C \leftarrow \emptyset$ ▷ Kreis
$v \leftarrow getAnyNodeWithOddDegreeOrAnyNode(G)$ ▷ Startknoten
while $E \neq \emptyset$ **do**
  $e \leftarrow getNeighboringNonBridge(v, E)$
  if $e = notfound$ **then**
    $e \leftarrow getAnyNeighboringEdge(v, E)$
  end if
  if $e = notfound$ **then**
    return C ▷ Error: Kein Kreis gefunden
  end if
  $C \leftarrow C \cup \{e\}$ ▷ Füge Kante zum Kreis hinzu
  $E \leftarrow E \setminus \{e\}$ ▷ Entferne Kante aus dem Graphen
  $v \leftarrow getOtherNode(e, v)$ ▷ Wechsel zum anderen Knoten der Kante
end while
return C ▷ Gibt den Kreis zurück

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.($h(v) \leq dist(v, t)$)

$dist(s) \leftarrow 0$ ▷ Abstand vom Startknoten
$W(s) \leftarrow \{s\}$ ▷ Weg vom Startknoten
$F \leftarrow \emptyset$ ▷ Nur in Dijkstra und A*
for all $v \in V \setminus \{s\}$ **do**
  $dist(v) \leftarrow \infty$
  $W(v) \leftarrow \emptyset$
end for
$push(stack, s)$ ▷ Stack für Bellman-Moore-Ford
while $stack \neq \emptyset$ **do** ▷ In Dijkstra und A* ist die Bedingung $t \notin F$
  $v^* \leftarrow pop(stack)$ ▷ Nur in Bellman-Moore-Ford
  $v^* \leftarrow \arg\min_{v \in V} dist(v)$ ▷ In Dijkstra
  $v^* \leftarrow \arg\min_{v \in V}(dist(v) + h(v))$ ▷ In A* mit Heuristik h
  $F \leftarrow F \cup \{v^*\}$ ▷ Nur in Dijkstra und A*
  for all $v \in N(v^*)$ **do** ▷ Gehe alle Nachbarn von v^* durch
    if $dist(v^*) + l(v^*, v) < dist(v)$ **then**
      $dist(v) \leftarrow dist(v^*) + l(v^*, v)$
      $W(v) \leftarrow W(v^*) \cup \{v\}$
      $push(stack, v)$ ▷ 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.

Algorithm 3 Prim

Require: Graph $G = (V, E)$, ein Wurzelknoten r
Ensure: Graph ist zusammenhängend, ungerichtet und gewichtet

$Q \leftarrow V$
for all $u \in Q$ **do**
  $dist(u) \leftarrow \infty$ ▷ Abstand zur Wurzel
  $pred(u) \leftarrow 0$ ▷ Vorgängerknoten im MST
end for
dist(r) $\leftarrow 0$
while $Q \neq \emptyset$ **do**
  $u \leftarrow \arg\min_{v \in Q} dist(v)$ ▷ Knoten mit minimalem Abstand
  $Q \leftarrow Q \setminus \{u\}$
  for all $v \in N(u)$ **do** ▷ Gehe alle Nachbarn von u durch
    if $v \in Q \wedge l(u, v) < dist(v)$ **then**
      $dist(v) \leftarrow l(u, v)$
      $pred(v) \leftarrow u$
    end if
  end for
end while

Algorithm 4 Kruskal

Require: Graph $G = (V, E)$
Ensure: Graph ist zusammenhängend, ungerichtet und gewichtet

$E' \leftarrow \emptyset$ ▷ MST Kanten
$L \leftarrow sort(E)$ ▷ Sortiere Kanten nach Gewicht
while $L \neq \emptyset$ **do**
  $e \leftarrow pop(L)$ ▷ Nimm die leichteste Kante
  if $(V, E' \cup \{e\})$ hat keinen Kreis **then**
    $E' \leftarrow E' \cup \{e\}$ ▷ 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
```

3 Ausgewählte Theoriefragen

1. **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
2. **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. $(u, v), (v, w)$
3. **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.
4. **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: $N_{in}(v) = \{u \in V | (u, v) \in E\}$
     - Ausgehende Nachbarn: $N_{out}(v) = \{u \in V | (v, u) \in E\}$
     - Nachbarn: $N(v) = N_{in}(v) \cup N_{out}(v)$
5. **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.

6. **Wann ist ein Graph eulersch?**
   - Ein Graph ist eulersch, wenn er einen Eulerweg 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 (Eulerweg).
7. **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.
8. **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.
9. **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.
10. **Diskutieren Sie die Euler- bzw. Hamiltoneigenschaft für vollständige Graphen K_n mit $n > 3$!**
    - Vollständige Graphen K_n sind hamiltonisch und manchmal eulersch 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.

11. **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.
12. **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.
    -
13. **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.
14. **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.
15. **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.
16. **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.

17. **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.
18. **Mit welchem Algorithmus findet man minimale Arboreszenzen?**
    - Der Edmonds-Algorithmus findet minimale Arboreszenzen in gerichteten Graphen.
19. **Was ist ein Webgraph?**
    - Ein Webgraph ist ein gerichteter Graph, der die Struktur des Internets oder eines Netzwerks von Webseiten abbildet.
    - Knoten repräsentieren Webseiten, Kanten repräsentieren Hyperlinks zwischen diesen Webseiten.
20. **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 oder zufällig auf eine neue Seite springen.
    - Der PageRank gibt die Wahrscheinlichkeit an, dass der Random Surfer auf einer bestimmten Seite landet.
21. **Was bedeutet der Dämpfungsfaktor im Random Surfer Modell?**
    - Der Dämpfungsfaktor ist ein Wert zwischen 0 und 1, der angibt, mit welcher Wahrscheinlichkeit der Random Surfer einem Link folgt, anstatt zufällig zu springen.
    - 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.
22. **Welches Verfahren wird für die Berechnung des PageRank verwendet?**
    - Der PageRank basiert auf dem Random-Surfer-Modell: Mit Wahrscheinlichkeit α folgt ein Nutzer einem zufällig gewählten Link, mit Wahrscheinlichkeit $1 - \alpha$ springt er auf eine beliebige Seite.
    - Die Berechnung erfolgt iterativ: Jeder Knoten verteilt seinen aktuellen PageRank anteilig auf seine ausgehenden Kanten.
    - Die Werte werden iterativ aktualisiert, bis sich der PageRank aller Knoten kaum noch ändert (Konvergenz).
    - Der Dämpfungsfaktor (typisch $\alpha = 0{,}85$) verhindert, dass der Surfer in Sackgassen stecken bleibt.

23. **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.
24. **Beschreiben Sie ein weiteres Zentralitätsmaß außer die Eigenvektorzentralität für Graphen!**
    - **PageRank:** 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 (Betweenness Centrality).
    - **Nähezentralität:** Misst die durchschnittliche Entfernung eines Knotens zu allen anderen Knoten im Graphen (Closeness Centrality).
25. **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.
26. **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.
27. **Lineare Programmierung ist nur dann effizient lösbar wenn alle Variablen kontinuierlich sind.**
    - Wahr.
    - Lineare Programmierung kann auch mit ganzzahligen Variablen gelöst werden, aber die Effizienz leidet darunter.
    - Ganzzahlige lineare Programmierung ist NP-vollständig, während kontinuierliche lineare Programmierung in polynomialer Zeit gelöst werden kann.