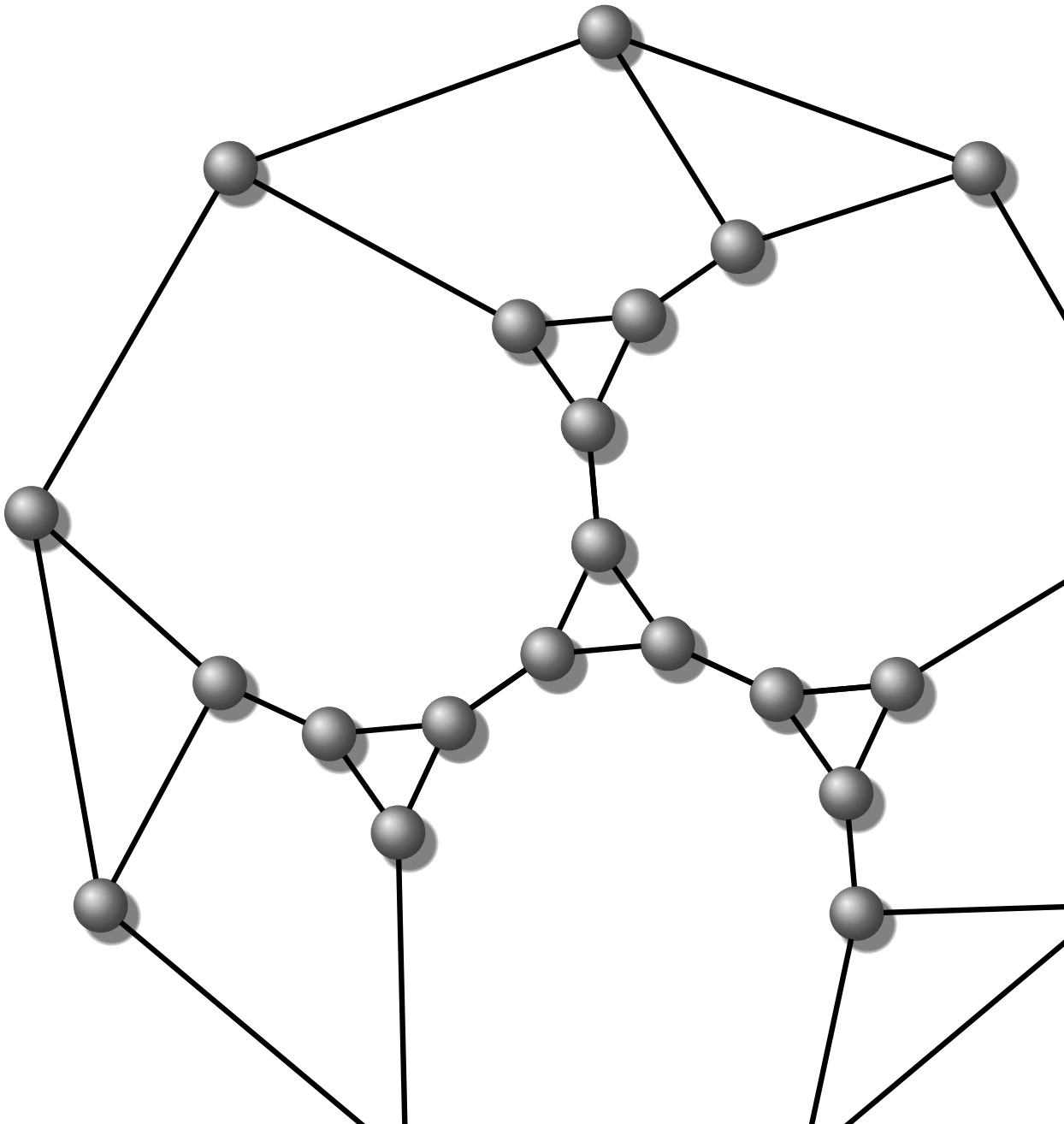
Volker Turau und Christoph Weyer

Lösungen für das Buch

Algorithmische Graphentheorie

4. erweiterte und überarbeitete Auflage



Dieses Dokument enthält die Lösungen zu den Aufgaben aus dem Buch Algorithmische Graphentheorie. Die hier angegebenen Verweise beziehen sich auf die 4. erweiterte und überarbeitete Auflage die 2015 im De Gruyter Verlag erschienen ist. Anmerkungen zu den Lösungen der Aufgaben schicken Sie bitte per e-mail an die Autoren.

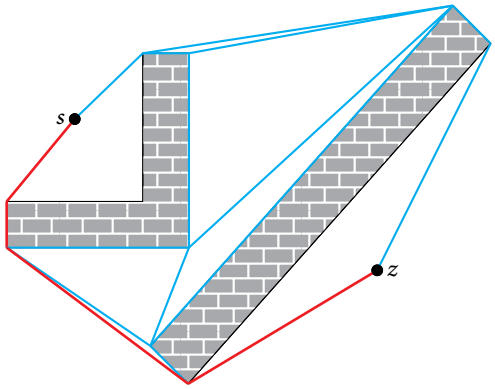
Version 1 — Stand August 2015

Inhaltsverzeichnis

Kapitel 1 – Einleitung

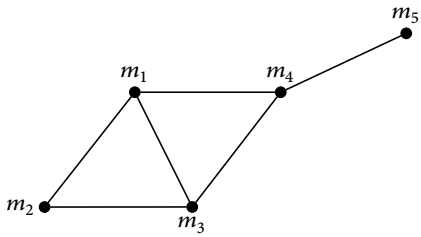
1. Der Verbindungszusammenhang der drei Netzwerke von links nach rechts ist 3, 3 und 2 und der Knotenzusammenhang ist 3, 3 und 1.

2. Die farbig gezeichneten Linien bilden das System der Geradensegmente für den gesuchten Weg. Der kürzeste Weg von s nach z führt über die Segmente auf der unteren konvexen Hülle, die entsprechenden Geradensegmente sind rot dargestellt.



3. Die optimale Reihenfolge der Produktionsaufträge ist 1–3–2–4 mit einer Gesamtumrüstzeit von 5,5.

4. Im Folgenden ist zunächst der Konfliktgraph und dann die Dispatchtabelle für die gegebene Klassenhierarchie und die angegebenen Methoden dargestellt.



Methode	m_1	m_2	m_3	m_4	m_5
Zeile	1	2	3	2	3

	A	B	C	D	E	F	G	H	I
1	$m_1 \mid A$	$m_1 \mid B$	$m_1 \mid C$	$m_1 \mid D$	$m_1 \mid A$	$m_1 \mid F$	$m_1 \mid G$		
2		$m_4 \mid B$	$m_4 \mid B$	$m_4 \mid B$	$m_2 \mid E$	$m_2 \mid E$	$m_2 \mid E$	$m_4 \mid H$	$m_4 \mid H$
3			$m_3 \mid C$	$m_3 \mid C$	$m_3 \mid E$	$m_3 \mid E$	$m_3 \mid E$		$m_5 \mid I$

5. Es gilt

$$\sum_{W \in M} PR(W) = |M|(1-d) + d \sum_{W \in M} \sum_{D \in PL(W)} \frac{PR(D)}{LC(D)}.$$

Da jede Web-Seite W aus M genau $LC(W)$-mal in der rechten Doppelsumme vorkommt, gilt:

$$\sum_{W \in M} PR(W) = |M|(1-d) + d \sum_{W \in M} LC(W) \frac{PR(W)}{LC(W)}$$

und somit ist

$$\sum_{W \in M} PR(W) = |M|.$$

D. h., die Summe der Werte des PageRank aller Seiten ist gleich der Anzahl der Seiten. Für die Menge aller existierenden Web-Seiten gilt diese Aussage nicht.

6. Es ergibt sich folgendes Gleichungssystem

$$\begin{aligned} PR(A) &= 0{,}5 + 0{,}5\,(PR(B) + PR(D)/2) \\ PR(B) &= 0{,}5 + 0{,}5\,PR(C) \\ PR(C) &= 0{,}5 + 0{,}5\,(PR(A)/2 + PR(D)/2) \\ PR(D) &= 0{,}5 + 0{,}5\,PR(A)/2 \end{aligned}$$

mit dieser Lösung:

$$PR(A) = 6/5 \quad PR(B) = 1 \quad PR(C) = 1 \quad PR/D) = 4/5.$$

7. Die Dichte des Netzwerks beträgt $1/3$. Die Werte der Zentralitätsmaße sind in der folgenden Tabelle enthalten.

	A	B	C	D	E	F	G	H	I
Degree	0,375	0,375	0,25	0,25	0,5	0,5	0,375	0,25	0,125
Betweenness	0,07	0,07	0	0	0,29	0,29	0,43	0,25	0
Closeness	2,25	2,25	2,63	2,63	1,88	1,88	1,63	2,25	2,63

8. Für jede Ecke e von G gilt: $C_B(e) = 0$, $C_D(e) = 1$ und $C_C(e) = 1$.

9. Für das soziale Netzwerk mit den Akteuren $\{e, e_1, e_2, \ldots, e_{n-1}\}$ und den dazugehörigen Kanten $\{(e, e_1), (e, e_2), \ldots, (e, e_{n-1})\}$ gilt $C_B(e) = 1$.

Kapitel 2 – Einführung

1. Über die Anzahl der Ecken geraden Grades kann keine allgemeine Aussage gemacht werden. Als Beispiel betrachte man die Graphen C_n.

2. Es sei G ein ungerichteter Graph mit n Ecken und m Kanten. Nach Voraussetzung gilt $3n = 2m$. Somit ist 2 ein Teiler von n.

3. Es sei G ein bipartiter Graph mit Eckenmenge $E_1 \cup E_2$ und $|E_1| = n_1$. Dann gilt $m \leq n_1(n - n_1)$. Die rechte Seite nimmt ihren größten Wert für $n_1 = n/2$ (n gerade) bzw. $n_1 = (n + 1)/2$ (n ungerade) an. In beiden Fällen folgt $4m \leq n^2$.

4. Ist Δ der Grad einer Ecke, so gilt $\Delta|E_1| = \Delta|E_2|$ bzw. $|E_1| = |E_2|$.

5. Es sei e eine Ecke mit $g(e) = \Delta(G)$ und e' ein beliebiger Nachbar von e. Dann gilt $n \geq \Delta(G) + g(e') \geq \Delta(G) + \delta(G)$. Für einen vollständigen Graph mit $n > 2$ Ecken gilt $\Delta(K_n) + \delta(K_n) = 2(n - 1) > n$.

6. Es sei $W = <e_1, \ldots, e_s>$ mit $s \geq 3$ ein einfacher Weg maximaler Länge in G. Wegen $g(e_1) \geq 2$ gibt es eine Ecke $e' \in N(e_1)\backslash\{e_2\}$. Da W maximale Länge hat ist $e' \in \{e_3, \ldots, e_s\}$. Somit enthält G den geschlossenen Weg $<e', e_1, \ldots, e_j, e'>$.

7. Offensichtlich ist G_P bipartit. Aus der Maximalität von P folgt $g_{G_P}(e) \geq g_G(e)/2$ für jede Ecke e von G. Die Aussage ergibt sich nun direkt aus der in Kapitel 2 bewiesenen Beziehung zwischen der Summe der Eckengrade und der Anzahl der Kanten eines Graphen.

8. Es sei G ein ungerichteter Graph mit $m > (n - 1)(n - 2)/2$. Angenommen, die Eckenmenge von G lässt sich so in zwei nichtleere Teilmengen E_1 und E_2 zerlegen, dass keine Kante Ecken aus E_1 und E_2 verbindet. Ist $|E_1| = n_1$, so gilt

   $$2m \leq n_1(n_1 - 1) + (n - n_1)(n - n_1 - 1)$$

   (Gleichheit gilt genau dann, wenn beide Teilgraphen vollständig sind). Die rechte Seite dieser Ungleichung wird maximal für $n_1 = 1$ bzw. $n_1 = n-1$. Dies führt in beiden Fällen zum Widerspruch.

   Die Graphen $K_{n-1} \cup K_1$ haben die angegebene Anzahl von Ecken und sind nicht zusammenhängend.

9. $K_{n_1} \cup K_{n_2}$.

10. Für jede Ecke e aus G gilt $g(e) + \bar{g}(e) = 5$, wobei $\bar{g}(e)$ den Eckengrad von e im Komplement $\overline{G}$ von G bezeichnet. Somit kann man annehmen, dass es in G eine Ecke e mit $g(e) \geq 3$ gibt. Sind die Nachbarn von e paarweise nicht benachbart, so enthält $\overline{G}$ den Graphen C_3 und ansonsten gilt dies für G.

11. Es sei $W = <e_1, \ldots, e_s, e_1>$ ein geschlossener Weg. Von e_1 aus startend, verfolge man W, bis man zum ersten Mal an eine Ecke e_j kommt, an der man schon einmal war, d. h. $e_j = e_i$ für $i < j$. Da W geschlossen ist, gibt es auf jeden Fall eine solche Ecke. Dann ist $<e_i, \ldots, e_j>$ ein einfacher geschlossener Weg.

12. Es sei e eine Ecke mit Eckengrad $\Delta(G)$ und L die Menge der Ecken von G, welche nicht zu e benachbart sind (einschließlich e). Dann ist nach Voraussetzung jede Kante von G zu einer Ecke von L inzident. Nun folgt:

$$m \leq \sum_{a \in L} g(a) \leq \Delta(G)|L| = \Delta(G)(n - \Delta(G)).$$

13. Die Bestimmung des Herzens erfolgt in zwei Phasen. Die erste Phase bestimmt eine Teilmenge F von E.

```
F := ∅;
while G.E ≠ ∅ do
    Wähle eine Ecke e aus G.E und füge e in F ein;
    Entferne e und alle Vorgänger von e in G aus G.E;
```

Von jeder Ecke $e \in E \backslash F$ kann mit einer Kante eine Ecke in F erreicht werden. Man beachte, dass es für eine neu in F eingefügte Ecke keine Nachfolger in F geben kann. In der zweiten Phase werden nun die Ecken von F in umgekehrter Reihenfolge, in der sie in F eingefügt wurden, betrachtet.

```
H := ∅;
while F ≠ ∅ do
    Wähle aus F die zuletzt eingefügte Ecke f aus;
    Füge f in H ein und entferne f und alle Vorgänger von f in G aus F;
```

Zwischen den Ecken von H gibt es keine Kanten und von jeder Ecke $e \in E \backslash H$ gibt es einen Weg mit höchstens zwei Kanten zu einer Ecke in H.

14. Angenommen die Aussage ist falsch. Dann gibt es eine Aufteilung der Eckenmenge in disjunkte Teilmengen E_1 und E_2, so dass nur eine einzige Kante k zwischen E_1 und E_2 existiert. Es sei $k = (e, e')$ mit $e \in E_1$ und $e' \in E_2$. Nach dem Entfernen von k haben die von E_1 und E_2 induzierten Graphen genau eine Ecke mit ungeradem Eckengrad. Dies steht im Widerspruch zu den Ergebnissen aus Abschnitt 2.1.

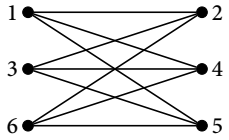
15. Die Adjazenzmatrix, die Adjazenzliste und die Kantenliste sehen wie folgt aus.

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

$1 \to 2$
$2 \to 4,6$
$3 \to 4$
$4 \to 5$
$5 \to 3$
$6 \to 1,4,5$

1	2	3	4	5	6	7	8	9
1	2	2	3	4	5	6	6	6
2	4	6	4	5	3	1	4	5

16. (a)



(b) Die Aufteilung der Eckemenge in $E_1 = \{e_1, e_3, e_6\}$ und $E_2 = \{e_2, e_4, e_5\}$ zeigt, dass der Graph bipartit ist.

17. Die Funktion `nachfolger(e_i, e_j)` überprüft, ob es eine Kante von e_i nach e_j gibt. (Zuerst folgt die einfache, danach die verbesserte Version)

```
function nachfolger(e_i : Ecke, e_j : Ecke) : Boolean
   var l, obergrenze : Integer;

   l := N[i];
   if i = n then
      obergrenze := m;
   else
      obergrenze := N[i+1] - 1;
   while l ≤ obergrenze do
      if A[l] = j then
         return true;
      l := l+1;
   return false

function nachfolger(e_i : Ecke, e_j : Ecke) : Boolean
   var l : Integer;

   l := N[i];
   while l ≤ N[i+1] - 1 do
      if A[l] = j then
         return true;
      l := l+1;
   return false;
```

Die Funktion `ausgrad(e_i)` bestimmt den Ausgrad der Ecke `e_i` (zuerst folgt die einfache, danach die verbesserte Version).

```
function ausgrad(e_i : Ecke) : Boolean
   if i = n then
      return m+1-N[i];
   else
      return N[i+1]-N[i];

function ausgrad(e_i : Ecke) : Boolean
   return N[i+1]-N[i];
```

Die Funktion `eingrad(e_i)` bestimmt den Eingrad der Ecke `e_i`. Hier führt die Einführung des zusätzlichen Eintrags zu keiner Vereinfachung.

```
function eingrad(e_i : Ecke) : Boolean
   var grad, l : Integer;

   grad := 0;
   for l := 1 to m do
      if A[l] = i then
         grad := grad+1;
   return grad;
```

18. Es sei B das Feld der Länge $(n^2 - n)/2$ und A die Adjazenzmatrix des Graphen. Für natürliche Zahlen i, j setze:

$$f(i,j) = (i-1)n - i(i+1)/2 + j.$$

Dann gilt $A[i,j] = B[f(i,j)]$ für $i < j$ und $A[i,j] = B[f(j,i)]$ für $i > j$. Ist $i > j$, so sind die Ecken e_i und e_j genau dann benachbart, wenn $B[f(i,j)] \neq 0$ ist. Der Eckengrad $g(e_i)$ einer Ecke e_i bestimmt sich wie folgt:

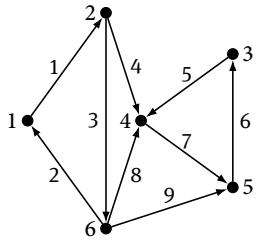
$$g(e_i) = \sum_{k=1}^{i-1} B[f(k,i)] + \sum_{k=i+1}^{n} B[f(i,k)].$$

19. Um festzustellen, ob zwei Ecken e_i und e_j benachbart sind, muss für den Fall $j \geq i$ die Nachbarliste von e_i und im anderen Fall die von e_j durchsucht werden. Die Bestimmung der Anzahl der Nachbarn von e_i erfordert das Durchsuchen der Nachbarlisten der Ecken $e_1, \ldots, e_i$. Gegenüber der normalen Adjazenzliste ändert sich der Aufwand für die erste Aufgabe nicht. Für die zweite Aufgabe ist der Aufwand jedoch höher. Für Ecke n müssen zum Beispiel alle Nachbarschaftslisten durchsucht werden, d. h., der Aufwand ist $O(n + m)$ gegenüber $O(g(n))$ bei normalen Adjazenzlisten.

20. Die Funktion `nachfolger` verwendet die Datenstruktur `Listenelement`:

```
type ListenElement = Struktur
   wert : Integer;
   nachfolger : Zeiger ListenElement;

var A : Array[1..n] von Zeiger ListenElement;

function nachfolger(e_i : Ecke, e_j : Ecke) : Boolean
   var eintrag : ListenElement;

   eintrag := A[i];
   while eintrag ≠ null and eintrag→wert ≤ j do
      if eintrag→wert = j then
         return true;
      eintrag := eintrag→nachfolger;
   return false;
```

21. Nummeriert man die Kanten des Graphen aus Aufgabe 15 wie links dargestellt, so ergibt sich folgende Inzidenzmatrix.



$$\begin{pmatrix} 1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ -1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & -1 & 0 & 1 & -1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & -1 & 0 & -1 \\ 0 & 1 & -1 & 0 & 0 & 0 & 0 & 1 & 1 \end{pmatrix}$$

Für ungerichtete Graphen entscheidet folgende Funktionen, ob e_j ein Nachfolger von e_i ist.

```
function nachfolger(e_i : Ecke, e_j : Ecke) : Boolean
   var k : Integer;

   k := 1;
   while k ≤ m do
      if Z[j, k] = 1 and Z[i, k] = -1 then
         return true;
      k := k + 1;
   return false;
```

Für ungerichtete Graphen muss in der **if**-Bedingung die Zahl -1 durch 1 ersetzt werden.

Der Ausgrad einer Ecke e_i in einem gerichteten Graphen ist gleich der Anzahl der Einträge mit Wert 1 in der i-ten Zeile und der Eingrad gleich der Einträge mit Wert -1 in der i-ten Zeile. Der Grad einer Ecke e_i in einem ungerichteten Graphen ist gleich der Anzahl der Einträge mit Wert 1 in der i-ten Zeile.

22. Adjazenzlisten basierend auf Feldern.

23. Eine Datei schließt sich genau dann selbst ein, wenn die zugehörige Ecke in dem gerichteten Graphen auf einem geschlossenen Weg liegt. Dies kann an den Diagonaleinträgen der Erreichbarkeitsmatrix erkannt werden.

24. (*a*) $n^2 + 2n + 3 \leq 6n^2$ für $n \geq 1$.

    (*b*) $\log n^2 = 2 \log n$ für $n \geq 1$.

    (*c*) $\sum_{i=1}^{n} i = n(n+1)/2 \leq n^2$ für $n \geq 1$.

    (*d*) $\log n! = \sum_{i=1}^{n} \log i \leq n \log n$ für $n \geq 1$.

25. Es sei A die Adjazenzmatrix des Graphen. Eine Ecke e_i ist genau dann eine Senke, wenn die i-te Zeile von A nur Nullen enthält und die i-te Spalte bis auf den i-ten Eintrag keine Nullen enthält. Für einen Eintrag a_{ji} von A mit $j \neq i$ bedeutet dies: Ist $a_{ji} = 0$, so ist e_i keine Senke und ist $a_{ji} \neq 0$, so ist e_j keine Senke. Der folgende Algorithmus entdeckt in jedem Schritt eine Ecke, welche keine Senke ist.

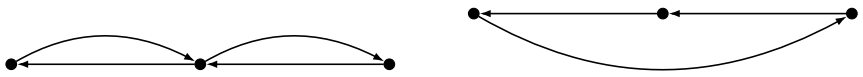
```
i := 1;
for j := 2 to n do
   if A[j, i] = 0 then
      i := j;
```

    Nachdem diese Schleife durchlaufen wurde, gibt es nur noch einen Kandidaten für die Senke, den letzten Wert von `i`. Die Überprüfung, ob `e`$_{\texttt{i}}$ eine Senke ist, kann mit Aufwand $O(n)$ durchgeführt werden.

    Liegt die Adjazenzliste des Graphen vor, so stellt man mit Aufwand $O(n)$ fest, ob es genau eine Ecke mit Ausgrad 0 gibt. Falls nicht, so gibt es keine Senke. Andernfalls überprüft man mit Aufwand $O(m)$, ob e in den Nachfolgelisten aller anderen Ecken vorkommt. Falls ja, so ist e eine Senke und ansonsten gibt es keine Senke.

26. Folgende einfache Eigenschaft von transitiven Reduktionen wird im folgenden verwendet: Es sei $k = (e, e')$ eine Kante in einer transitiven Reduktion R. Dann verwendet jeder Weg in R von e nach e' die Kante k.

    (*a*)



    (*b*) Angenommen es gibt zwei verschiedene transitive Reduktionen R_1 und R_2. Es sei $k = (e_i, e_j)$ eine Kante in $R_1 \backslash R_2$. Da e_j von e_i in R_2 erreichbar sein muss, gibt es eine Ecke e, so dass es in R_2 einen Weg von e_i über e nach e_j gibt. Da es in R_1 keine geschlossenen Wege gibt, existieren dort Wege von e_i nach e und von e nach e_j, welche k nicht enthalten. Dies kann aber nicht sein.

    (*c*) Es sei G der vollständig bipartite Graph mit Eckenmenge $E_1 \cup E_2$ und $|E_1| = |E_2| = n/2$. Jede Kante (e_1, e_2) mit $e_1 \in E_1$ und $e_2 \in E_2$ wird nun in Richtung von den Ecken aus E_2 orientiert. Dieser Graph hat genau die angegebene Eigenschaft.

    (*d*) Es sei R die transitive Reduktion eines gerichteten Graphen ohne geschlossene Wege und K_R die Kantenmenge von R. Dann ist $K' \subseteq K_R$. Angenommen es gilt

$K' \subset K_R$. Dann existiert eine Kante $k = (e, e')$ in K_R, mit maxlen$(e, e') > 1$ ist, d. h., es gibt einen Weg W von e nach e', welcher mindestens zwei Kanten enthält. Da der Graph keine geschlossenen Wege besitzt, ist k nicht in W enthalten. Dies führt zum Widerspruch und somit gilt $K_R = K'$.

27. (a) Der Algorithmus verwendet die in Aufgabe 18 angegebene Funktion f, um Paare von Ecken in einem eindimensionalen Feld A zu indizieren. Das Feld wird mit 0 initialisiert. Haben zwei Ecken $\mathtt{e_i}$ und $\mathtt{e_j}$ einen gemeinsamen Nachbarn, so wird dies an der Stelle $A[f(i,j)]$ eingetragen.

```
function vierWeg(G : Graph) : Boolean
   var vierKreis : Array[1..(n-1)n/2] von Integer;

   Initialisiere vierKreis mit 0;
   foreach e in G.E do
      foreach e_i in N(e) do
         foreach e_j in N(e) \ {e_i} do
            if vierKreis[f(i, j)] ≠ 0 then
               return true;
            else
               vierKreis[f(i, j)] := 1
   return false;
```

(b) Die Initialisierung des Feldes `vierKreis` erfordert $n(n-1)/2$ Schritte. In der anschließenden Suche wird jede Komponente dieses Feldes maximal einmal verändert. Der Aufwand ist somit $O(n^2)$.

(c) Um festzustellen, ob ein Graph einen Untergraph vom Typ $K_{s,s}$ hat, muss ein Feld der Länge $\binom{n}{s}$ betrachtet werden. Ferner werden für jede Ecke e alle s-elementigen Teilmengen der Nachbarn von e betrachtet. Daraus ergibt sich ein Aufwand von $O(n^s)$.

28. Es sei $k = (e_i, e_j)$ die zusätzliche Kante und $k' = (e_{i'}, e_{j'})$ eine Kante, welche neu im transitiven Abschluss liegt (d. h. $E'[i', j'] = 1$ und $E[i', j'] = 0$). Dann muss es einen einfachen Weg von $e_{i'}$ nach $e_{j'}$ geben, welcher k enthält. Da E die Erreichbarkeitsmatrix ist, gilt: $E[i', i] = 1$ und $E[j, j'] = 1$. D. h., jede neue Kante geht von einem Vorgänger von e_i zu einem Nachfolger von e_j. Folgender Algorithmus bestimmt E' mit Aufwand $O(n^2)$.

```
var E' : Array[1..n, 1..n] von Integer;

Initialisiere E' mit E;
for l := 1 to n do
   if E[l, i] = 1 then
      for s := 1 to n do
         if E[j, s] = 1 then
            E'[l, s] := 1;
```

Es sei G ein Graph, welcher aus zwei Teilgraphen G_1 und G_2 mit je $n/2$ Ecken besteht, so dass es keine Kante von G_2 nach G_1 gibt. Ferner besitze G_1 eine Ecke e_1, von der aus jede Ecke aus G_1 erreichbar ist, und in G_2 eine Ecke e_2, welche von jeder Ecke aus G_2 erreichbar ist. Wird die Kante (e_2, e_1) neu in G eingefügt, so wird jede Ecke von G_1 von jeder Ecke aus G_2 erreichbar. D. h., der neue transitive Abschluss hat $n^2/4$ zusätzliche Kanten.

29. Die eine Richtung der Behauptung ist trivialerweise erfüllt. Es sei $<e_1,\ldots,e_s,e_1>$ ein geschlossener Weg minimaler Länge. Angenommen es gilt $s > 3$. Gibt es die Kante (e_3,e_1), dann ist $<e_1,e_2,e_3,e_1>$ ein geschlossener Weg der Länge 3. Andernfalls gibt es die Kante (e_1,e_3), dann aber ist $<e_3,e_4,\ldots,e_s,e_1,e_3>$ ein geschlossener Weg der Länge $s-1$. Dieser Widerspruch zeigt, dass es einen geschlossenen Weg der Länge 3 gibt.

30. (*a*) Angenommen G enthält eine Ecke e und Nachbarn e_1, …, e_6, so dass der von $\{e,e_1,\ldots,e_6\}$ induzierte Graph vom Typ $K_{1,6}$ ist. Dies bedeutet, dass keine der Ecken e_1, …, e_6 benachbart sind. Es sei $\widehat{ee_i}$ die Strecke, welche die zu e und e_i gehörenden Punkte verbindet. Dann muss es $i,j \in \{1,\ldots,6\}$ geben, so dass der Winkel zwischen $\widehat{ee_i}$ und $\widehat{ee_j}$ maximal 60° ist. Hieraus folgt, dass der Abstand zwischen den zu e_i und e_j gehörenden Punkten maximal a ist. Somit sind e_i und e_j entgegen der Annahme benachbart.

(*b*) Es sei e eine Ecke von G vom Grad Δ. Die zu den Δ Nachbarn von e gehörenden Punkte liegen innerhalb des Kreises mit Radius a um den zu e gehörenden Punkt. Dann muss es mindestens $\lceil \Delta/6 \rceil$ Punkte geben, welche innerhalb eines Kreissektors mit Winkel 60° liegen. Die entsprechenden $\lceil \Delta/6 \rceil$ Nachbarn von e sind paarweise benachbart, da die Distanz der zugehörenden Punkte maximal a ist. Somit bilden diese Ecken zusammen mit e eine Clique mit $\lceil \Delta(G)/6 \rceil + 1$ Ecken.

31. (*a*) Q_s hat 2^s Ecken, hieraus folgt, dass es $s2^{s-1}$ Kanten gibt.

(*b*) Unterscheiden sich zwei beliebige Ecken e_1 und e_2 von Q_s in genau l Positionen, so ist $d(e_1,e_2) = l$. Somit gilt $D(Q_s) \leq s$. Da der Abstand zwischen den Ecken $(0,\ldots,0)$ und $(1,\ldots,1)$ gleich s ist, gilt $D(Q_s) = s$.

32. (*a*) Für $n = 3$ ist die Aussage trivialerweise richtig. Sei nun $n > 3$. Falls G Hamiltonsch ist, so gilt dies auch für G'. Sei nun umgekehrt G' Hamiltonsch und H ein Hamiltonscher Kreis von G'. Liegt die Kante (e,e') nicht auf H, dann ist auch H ein Hamiltonscher Kreis für G. Andernfalls ergibt sich aus H ein einfacher Weg $<e_1,\ldots,e_n>$ mit $e_1 = e$ und $e_n = e'$ in G, auf dem alle Ecken von G liegen. Im Folgenden werden die Mengen

$$A = \{e_i \mid (e',e_{i-1}) \text{ ist Kante in } G \text{ und } 3 \leq i \leq n-1\}$$

und

$$B = \{e_i \mid (e,e_i) \text{ ist Kante in } G \text{ und } 3 \leq i \leq n-1\}$$

betrachtet. Es gilt $\{e_1,e_2,e_n\} \cap A = \emptyset$, $\{e_1,e_2,e_n\} \cap B = \emptyset$, $|A| \geq |N(e')| - 1$ und $|B| = |N(e)| - 1$. Nun folgt aus der Voraussetzung:

$$n-3 \geq |A \cup B| = |A| + |B| - |A \cap B| \geq n-2-|A \cap B|.$$

Somit ist $|A \cap B| > 0$, d. h., es gibt eine Ecke e_i, so dass (e,e_i) und (e',e_{i-1}) Kanten in G sind. Dann ist $<e_1,e_2,\ldots,e_{i-1},e_n,e_{n-1},\ldots,e_i,e_1>$ ein Hamiltonscher Kreis in G.

(*b*) (*i*) Da vollständige Graphen Hamiltonsch sind, kann die Aussage leicht mittels Teil *(a)* bewiesen werden.

(*ii*) Folgt direkt aus *(i)*.

33. Der Petersen-Graph besteht aus zwei Kopien von C_5, die Ecken dieser beiden Graphen sind durch fünf *Speichen* verbunden (siehe Abbildung 2.10). Man überlegt sich schnell, dass in einem Hamiltonschen Kreis entweder genau zwei oder genau vier Speichen vorkommen müssen. Im ersten Fall müssten dann jeweils vier Kanten jeder Kopie des zyklischen Graphen C_5 auf dem Hamiltonschen Kreis vorkommen. Dies geht jedoch nicht. Im zweiten Fall müssten von der einen Kopie zwei und von der anderen Kopie drei Kanten auf dem Hamiltonschen Kreis liegen. Man sieht sofort, dass auch dies unmöglich ist.

Kapitel 3 – Bäume

1. (*a*) Ist G ein Baum, so gilt $m = n - 1 < n$. Gilt umgekehrt $n > m$, so betrachte man einen aufspannenden Baum B von G. Da B $n - 1$ Kanten hat, gilt $m = n - 1$ und somit $G = B$.

   (*b*) Wenn G einen einzigen geschlossenen Weg W enthält, so erhält man einen aufspannenden Baum, falls man eine beliebige Kante aus W entfernt. Somit hat G genau $n - 1 + 1 = n$ Kanten. Ist umgekehrt die Anzahl der Kanten in G gleich der Anzahl der Ecken, so besteht G aus einem aufspannenden Baum und einer zusätzlichen Kante. Diese bewirkt, dass es in G genau einen geschlossenen Weg gibt.

   (*c*) Es sei G ein Wald und U ein induzierter Untergraph mit n Ecken und m Kanten. Da U ein Wald ist, gilt $m = n - z < n$ nach den Ergebnissen aus Abschnitt 3.1, hierbei bezeichnet z die Anzahl der Zusammenhangskomponenten von U. Hätte jede Ecke mindestens den Grad 2, so wäre $m \geq n$. Dieser Widerspruch zeigt, dass U eine Ecke e mit $g(e) \leq 1$ besitzt.

   Es sei nun G ein ungerichteter Graph in dem jeder induzierte Untergraph eine Ecke e mit $g(e) \leq 1$ besitzt. Angenommen G enthält einen geschlossenen Weg W. Es sei W_I der von den Ecken von W induzierte Untergraph. Dann gilt $g(e) \geq 2$ für jede Ecke von W_I. Dieser Widerspruch zeigt, dass G ein Wald ist.

   (*d*) Es sei G ein Wald und U ein zusammenhängender Untergraph. Dann ist U ein Baum mit Eckenmenge E_U. Angenommen der von E_U induzierte Untergraph enthält eine Kante k, welche nicht in U enthalten ist. Fügt man k in U ein, so entsteht ein geschlossener Weg. Dies ist aber unmöglich, da G ein Wald ist. Somit ist U ein induzierter Untergraph.

   Es sei G ein Graph, in dem jeder zusammenhängende Untergraph ein induzierter Untergraph ist. Angenommen G enthält einen geschlossenen Weg W. Ist e eine Kante von W, dann ist $W \backslash \{e\}$ kein induzierter Untergraph. Somit enthält G keinen geschlossenen Weg, d. h., G ist ein Wald.

2. Es sei e eine beliebige Ecke und G_e der von e und den Nachbarn von e induzierte Untergraph. Man sieht leicht, dass G_e ein Windmühlengraph ist. Falls alle Ecken den Grad 2 haben, so ist $G = W_3 = K_3$. Hat G eine Ecke e mit $g(e) > 2$, so dass alle anderen Ecken zu e inzident sind, so ist G ebenfalls ein Windmühlengraph. Angenommen G hat keine dieser beiden Eigenschaften. Es sei e eine Ecke mit maximalem Eckengrad Δ. Dann ist $\Delta > 2$. Es sei N die Menge der Nachbarn von e und $e' \neq e$ eine Ecke, welche nicht in N ist. Für jede Ecke $u \in N$ gibt es genau eine Ecke e_u, so dass $<e', e_u, u>$ ein Weg ist. Für $u_1 \neq u_2$ gilt $e_{u_1} \neq e_{u_2}$, sonst wäre $<e, u_1, e_{u_1}, u_2, e>$ ein geschlossener Weg der Länge vier. Somit gilt:

$$\Delta \geq g(e') \geq g(e) = \Delta, \text{ bzw. } g(e') = g(e).$$

   Im Folgenden wird gezeigt, dass G regulär ist. Dazu muss noch gezeigt werden, dass $g(e) = g(u)$ für jede Ecke $u \in N$ gilt. Wiederholt man die Argumentation aus dem letzten Abschnitt mit e' anstelle von e, so folgt dass der Eckengrad jeder nicht zu e' benachbarten Ecke ebenfalls Δ ist. Somit verbleibt maximal ein Nachbar u von e, von dem

noch nicht gezeigt wurde, dass $g(u) = \Delta$ ist. Wiederholt man die letzte Argumentation für einen Nachbarn $u' \neq u$ von e, so folgt auch $g(u) = \Delta$, d. h., G ist regulär.

Es sei A die Adjazenzmatrix von G. Da G regulär ist, folgt aus der Voraussetzung $A^2 = (\Delta - 1)I + H$. Hierbei ist I die Einheitsmatrix und H die Matrix, deren Einträge alle gleich 1 sind. Die Matrizen A und H sind symmetrisch und es gilt $AH = HA = \Delta I$. Somit sind A und H simultan diagonalisierbar. Die Eigenwerte von H sind 1 und 0 mit den Vielfachheiten 1 und $n - 1$. Der Eigenvektor zum Eigenwert 1 ist $v_1 = (1, \ldots, 1)$. Als Eigenvektor von A hat v_1 den Eigenwert Δ. Es sei v ein weiterer gemeinsamer Eigenvektor. Dann ist $Hv = 0$ und somit $A^2v = (\Delta - 1)v$ bzw. $Av = \pm\sqrt{\Delta - 1}v$. Somit sind Δ, $\sqrt{\Delta - 1}$ und $-\sqrt{\Delta - 1}$ die Eigenwerte von A mit den Vielfachheiten $1, r$ und s. Es gilt $n = 1 + r + s$. Betrachtet man die Spur von A, so folgt $s - r = \Delta/\sqrt{\Delta - 1}$. Da $s - r$ eine ganze Zahl ist, muss $\Delta - 1$ eine Quadratzahl sein, deren Wurzel Δ teilt. Hieraus folgt $\Delta = 2$ im Widerspruch zur Annahme. Somit ist G ein Windmühlengraph.

3. Die Aussage wird durch vollständige Induktion bewiesen. Für $n = 2$ ist $d_1 = d_2 = 1$ und der Graph C_2 ist der gesuchte Baum. Sei nun $n > 2$. Es muss Indizes i, j mit $d_i = 1$ und $d_j > 1$ geben. Entfernt man d_i und d_j aus der Zahlenfolge und fügt die Zahl $d_j - 1$ ein, so kann die Induktionsvoraussetzung angewendet werden. An den so erhaltenen Baum muss nur noch eine zusätzliche Kante an die Ecke mit Grad $d_j - 1$ angehängt werden.

4. Würde es in $\overline{B}$ drei Zusammenhangskomponenten geben, so gäbe es in B einen geschlossenen Weg der Länge drei. Da B ein Baum ist, kann das nicht sein. Nach Voraussetzung besteht B somit aus zwei Zusammenhangskomponenten. Gäbe es in einer der beiden Zusammenhangskomponenten zwei nicht inzidente Ecken, so würde dies wieder zu einem Widerspruch führen. Somit sind beide Zusammenhangskomponenten vollständige Graphen. Hätten beide Komponenten mehr als eine Ecke, so gäbe es einen geschlossenen Weg der Länge vier in B. D. h., $\overline{B}$ besteht aus einer isolierten Ecke und einem vollständigen Graphen. Hieraus folgt, dass B ein Sterngraph ist.

5. Ein Binärbaum mit der angegebenen Eigenschaft hat $2b - 1$ Ecken. Der Beweis erfolgt durch vollständige Induktion. Für $b = 1$ ist die Aussage klar. Es sei nun $b > 1$ und e eine Ecke auf dem vorletzten Niveau von B. Dann hat e genau zwei Nachfolger. Entfernt man diese aus B, so erhält man einen Binärbaum mit $b - 1$ Blätter für den die Voraussetzung erfüllt ist. Somit hat dieser Baum $2(b - 1) - 1$ Blätter. Daraus folgt die Aussage.

6. Der Beweis erfolgt durch vollständige Induktion nach der Anzahl n_i der inneren Ecken. Für $n_i = 1$ ist die Aussage klar. Sei nun $n_i > 1$ und e' eine Ecke auf dem vorletzten Niveau. Entfernt man alle Nachfolger von e', so erfüllt der entstandene Wurzelbaum $G(e')$ die Induktionsvoraussetzung. Es sei $E_B(e')$ die Menge der Blätter von $G(e')$. Dann gilt:
$$|E_B(e')| = \sum_{e \in E \setminus E_B, e \neq e'} (|N^+(e)| - 1) + 1.$$
Da $|E_B| = |E_B(e')| + |N^+(e')| - 1$ gilt, ist die Aussage bewiesen.

7. Die Anzahl der Blätter in einem Binärbaum ist genau dann maximal, wenn alle Niveaus voll besetzt sind. Somit hat ein Binärbaum der Höhe h maximal 2^h Blätter.

8. Es sei G quasi stark zusammenhängend. Folgendes Verfahren bestimmt eine Ecke w von G mit der angegebenen Eigenschaft. Es sei zunächst w eine beliebige Ecke von G. Gibt

es eine Ecke e, welche nicht von w erreichbar ist, so muss es eine Ecke e' geben, so dass e und w von e' erreichbar sind. Setze w gleich e' und wiederhole das Verfahren bis alle Ecken von w erreichbar sind. Besitzt G eine Ecke mit der angegebenen Eigenschaft, so folgt direkt, dass G auch quasi stark zusammenhängend ist.

9. Es werden folgende Typen zur Darstellung von Dateien und Verzeichnissen verwendet.

```
type Eintrag = Struktur
   name : String;
   verzeichnis : Boolean;
   inhalt : Zeiger Eintragliste;
   daten : Inhaltstyp;
```

```
type Eintragliste = Struktur
   inhalt : Zeiger Eintrag;
   nachbar : Zeiger Eintragliste;
```

Folgende Prozedur gibt die Namen aller Dateien im angegebenen Verzeichnis und allen darunterliegenden Unterverzeichnissen aus.

```
procedure ausgabeVerzeichnis(e : Zeiger Eintrag)
   var naechster : Zeiger Eintragliste;

   if e ≠ null then
      if e→verzeichnis then
         ausgabe("Verzeichnis : ␣", e→name);
         naechster := e→inhalt;
         while naechster ≠ null do
            ausgabeVerzeichnis(naechster→inhalt);
            naechster := naechster→nachbar;
      else
         ausgabe("Datei : ␣", e→name);
```

10. Um einen Eintrag mit dem Schlüssel x in einem binären Suchbaum zu löschen, muss zunächst die zugehörige Ecke e lokalisiert werden. Falls e ein Blatt oder genau einen Nachfolger hat, so kann das Löschen einfach durchgeführt werden. Falls jedoch e zwei Nachfolger hat, so ist der Vorgang aufwendiger. Ein Algorithmus zum Löschen von Einträgen arbeitet wie folgt.

    (1) Falls e keine Nachfolger hat, so lösche e.

    (2) Falls e genau einen Nachfolger hat, so ersetzte e durch seinen Nachfolger.

    (3) Falls e zwei Nachfolger hat, so ersetze e durch die Ecke mit dem größten Schlüssel im linken Teilbaum oder durch die Ecke mit dem kleinsten Schlüssel im rechten Teilbaum von e. Um die Ecke mit dem größten (kleinsten) Schlüssel im linken (rechten) Teilbaum zu finden, verfolge man vom linken (rechten) Nachfolger der Ecke startend immer den rechten (linken) Nachfolger bis ein Blatt erreicht ist, dies ist die gewünschte Ecke.

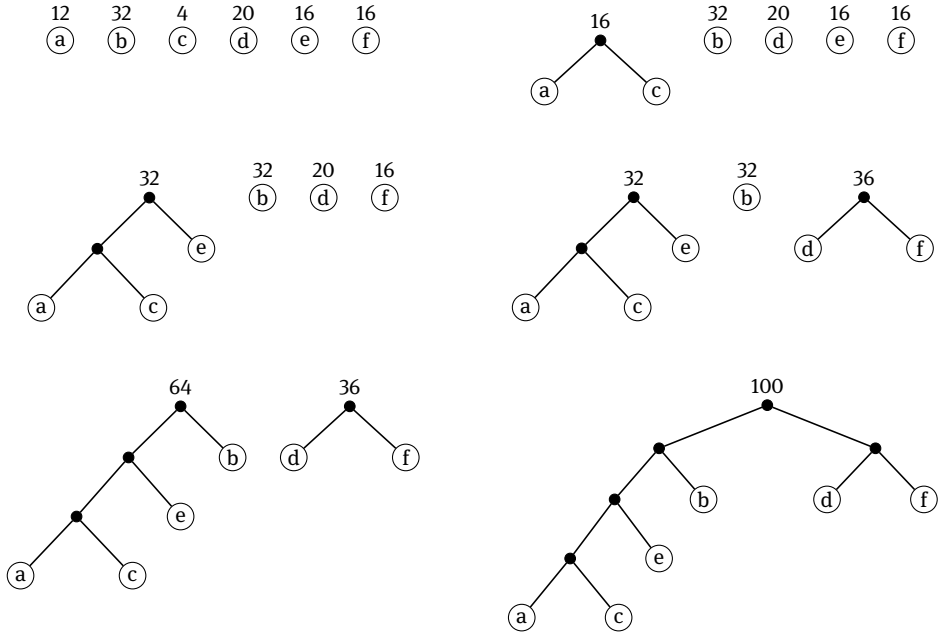
11. Die rekursive Prozedur `aufsteigendeReihenfolge` gibt die Objekte eines Suchbaumes in aufsteigender Reihenfolge aus.

```
procedure aufsteigendeReihenfolge(b : Suchbaum)
   if b ≠ null then
      aufsteigendeReihenfolge(b→links);
      ausgabe(b→daten);
      aufsteigendeReihenfolge(b→rechts);
```

12. Bei einem Binärbaum mit zehn Ecken minimaler Höhe sind die ersten drei Ebenen voll besetzt und auf der vierten Ebene sind drei Ecken. Der Baum hat also die Höhe drei. Entartet der Suchbaum zu einer linearen Liste, so liegt ein Binärbaum maximaler Höhe vor (vergleichen Sie Abbildung 3.7).

13. Liegt a auf einem höheren Niveau als b, so ist $l_a > l_b$. Wäre die Wahrscheinlichkeit von a echt kleiner als die von b, so könnte man die Codierung von a und b vertauschen und so einen Präfix-Code mit kleinerer mittlerer Wortlänge erhalten. Dies widerspricht der Optimalität des Huffman-Algorithmus.

14.



Der erzeugte Präfix-Code hat eine mittlere Codewortlänge von 2,48.

15. Die Prozedur `codeWörter` verwendet einen Stapel `S`. Dieser enthält von unten nach oben die Codierung des aktuellen Zeichens. Initial ist der Stapel leer.

```
var huffman : Wald;
var S : Stapel von Integer;

procedure codeWörter(e_w : Ecke)
   if huffman.ecken[w].links = 0 and huffman.ecken[w].rechts = 0  then
      ausgabe("Code␣für␣", huffman.ecken[w].bewertung, "␣ist : ␣");
         foreach b in S do
            ausgabe(b);
   else
      S.einfügen(0);
      codeWörter(huffman.ecken[w].links);
      S.entfernen();
      S.einfügen(1);
      codeWörter(huffman.ecken[w].rechts);
      S.entfernen();
```

Der Aufruf `codewörter(huffman.wurzeln[1])` gibt den Präfix-Code für alle Zeichen aus.

16. Die Kosten eines minimal aufspannenden Baumes betragen 20.

17. Ein aufspannender Baum B von G, der k enthält, heißt k-minimal aufspannender Baum von G, falls kein anderer aufspannender Baum B' von G existiert, dessen Kosten niedriger sind und der k enthält. Der Beweis der folgenden Aussage erfolgt analog zum Beweis des ersten Satzes in Abschnitt 3.6 auf Seite 79.

    Es sei G ein kantenbewerteter zusammenhängender Graph mit Eckenmenge E und $k = (e, e')$ eine Kante von G. Ferner sei U eine Teilmenge von E mit $e, e' \in U$ und (u, v) eine Kante mit minimalen Kosten mit $u \in U$ und $v \in E \backslash U$. Dann existiert ein k-minimal aufspannender Baum von G, der die Kante (u, v) enthält.

    Es seien nun k_0, k_1, …, k_{m-1} die Kanten von G mit $k_0 = k$, wobei die Kanten k_1, …, k_{m-1} nach aufsteigenden Bewertungen sortiert sind. Der Algorithmus von Kruskal wird auf G angewendet, wobei die Kanten in der angegebenen Reihenfolge betrachtet werden. Auf diese Art entsteht ein aufspannender Baum von G, welcher k enthält. Der Beweis, dass dies auch ein k-minimal aufspannender Baum ist, erfolgt analog zum Korrektheitsbeweis des Algorithmus von Kruskal.

18. Zur Konstruktion eines maximal aufspannenden Baumes kann jeder Algorithmus zur Bestimmung eines minimal aufspannenden Baumes verwendet werden. Dazu muss nur die Relation $\leq$ durch $\geq$ ersetzt werden. Eine andere Möglichkeit besteht darin, die Bewertung b_k jeder Kante k auf $C - b_k$ zu ändern, wobei C die höchste Bewertung aller Kanten ist. Danach wird für diese Bewertung ein minimal aufspannender Baum bestimmt. Dieser ist ein maximal aufspannender Baum für die ursprüngliche Bewertung.

19. Für eine Kante k aus einem aufspannenden Baum T besteht der Graph $T \backslash \{k\}$ aus zwei Zusammenhangskomponenten mit Eckenmengen E_1, E_2. Setze

$$K_k(T) = \{(a, b) \text{ Kante von } G \mid a \in E_1, b \in E_2\}.$$

    Es gilt $K_k(T) \cap T = \{k\}$. Es sei T' ein minimal aufspannender Baum von G und T der vom Algorithmus erzeugte Baum. Ferner sei $k \in T \backslash T'$ und W der geschlossene Weg in $T' \cup \{k\}$. Für alle $k' \in K_k(T) \cap W$ mit $k' \neq k$ gilt $k' \in T', k' \notin T$ und $k \in K_{k'}(T')$. Da k' nicht in T liegt, wurde k' durch den Algorithmus entfernt. D. h., k' war die Kante mit der höchsten Bewertung in einem geschlossenen Weg. Somit ist die Bewertung von k' mindestens so groß wie die Bewertung jeder von k' verschiedenen Kante des geschlossenen Weges in $T \cup \{k'\}$. Da $k' \in K_k(T)$ ist, muss k auf diesem Weg liegen. Somit gilt bewertung$(k') \geq$ bewertung(k). Da T' ein minimal aufspannender Baum ist, kann nicht bewertung$(k') >$ bewertung(k) gelten. Somit stimmen die Bewertungen von k' und k überein. Ersetzt man in T' die Kante k' durch k, so erhält man einen neuen minimalen aufspannenden Baum T'' mit $|T'' \cap T| > |T' \cap T|$. Durch Wiederholung dieser Vorgehensweise zeigt man, dass auch T ein minimal aufspannender Baum von G ist.

20. Hat ein Graph eine *Brücke* (vergleichen Sie Aufgabe 14 in Kapitel 2), deren Bewertung größer ist als die aller anderen Kanten, so liegt diese in jedem aufspannenden Baum.

21. Angenommen es gibt zwei verschiedene minimal aufspannende Bäume B_1 und B_2. Es sei k_1 eine Kante aus B_1, welche nicht in B_2 liegt. Fügt man k_1 in B_2 ein, so entsteht ein geschlossener Weg W. Da B_2 ein minimal aufspannender Baum ist und die Kanten verschiedene Bewertungen haben, sind die Bewertungen aller Kanten aus $W \cap B_2$ echt größer als die von k_1. Da B_1 ein Baum ist, muss es in $W \backslash B_1$ eine Kante k_2 mit bewertung(k_1) < bewertung(k_2) geben. Dieses Argument kann nun wiederholt werden. So entsteht eine unendliche Folge von Kanten k_1, k_2, ... aus G mit echt aufsteigenden Gewichten. Dieser Widerspruch zeigt, dass es nur einen minimal aufspannenden Baum gibt.

22. Für dichte Graphen ist der Algorithmus von Prim überlegen. Unter Verwendung von Fibonacci-Heaps ist die Laufzeit dieses Algorithmus $O(m + n \log n)$. Der Sortierschritt im Algorithmus von Kruskal hat allein schon einen Aufwand von $O(m \log n)$.

23. Die spezielle Eigenschaft der Bewertung kann ausgenutzt werden, um das Sortieren der Kanten nach ihren Bewertungen mit Aufwand $O(m + c)$ durchzuführen (Zeit und Speicher). Der Algorithmus von Kruskal hat in diesem Fall den Aufwand $O(m + c + n \log n)$.

24. Der Beweis erfolgt durch vollständige Induktion nach der Anzahl n der Ecken. Für $n = 2$ ist die Aussage klar. Es sei $n > 1$ und k die Kante mit der kleinsten Bewertung und G' der Graph, welcher durch das Verschmelzen der Endecken von k und das Ändern der Bewertungen entsteht. Nach Induktionsvoraussetzung bestimmt der Algorithmus einen minimal aufspannenden Baum B' von G'. Nach dem in Abschnitt 3.6 auf Seite 79 bewiesenen Satz gibt es einen minimal aufspannenden Baum B von G, welcher k enthält. Entfernt man k aus diesem Baum und ändert wie angegeben die Bewertungen der Kanten, so erhält man einen aufspannenden Baum für G'. Es gilt:

$$\text{kosten}(B') \leq \text{kosten}(B) - n \text{ bewertung}(k)$$

    B' ist auch ein Baum in G. Fügt man die Kante k zu B' hinzu und ändert wieder die Bewertungen, so erhält man einen aufspannenden Baum von G mit Kosten n bewertung(k)+ kosten(B'). Aus obiger Gleichung folgt, dass dieser Baum ein minimal aufspannender Baum von G ist. Dies ist gleichzeitig auch der Baum, den der Algorithmus konstruiert.

25. Es sei G' der Graph, welcher aus B, der neuen Ecke e und den neuen Kanten besteht. Wendet man den in Aufgabe 19 beschriebenen Algorithmus an, so müssen nur die neuen Kanten in B eingefügt und entsprechende Kanten entfernt werden. Dies kann mit Aufwand $O(n\, g(e))$ durchgeführt werden. Da der Graph G' $n - 1 + g(e) < 2n$ Kanten hat, bestimmt sowohl der Algorithmus von Kruskal als auch der von Prim in diesem Fall einen minimal aufspannenden Baum mit Aufwand $O(n \log n)$.

26. Auf den Kanten von G wird auf Basis der alten Bewertung w eine neue Kantenbewertung w' wie folgt definiert: $w'(k) = (w(k), 0)$, falls e nicht zu k inzident ist und $w'(k) = (w(k), 1)$ andernfalls. Die Bewertungen sind also keine reellen Zahlen, sondern Paare von Zahlen. Damit die Algorithmen von Prim bzw. Kruskal zur Anwendung kommen können, muss gezeigt werden, dass die neue Ordnung die gleiche Eigenschaft

wie eine numerische Bewertung hat. Auf den Paaren (a,b) wird eine totale Ordnung definiert: Es gilt $(a_1,b_1) < (a_2,b_2)$ genau dann, wenn $a_1 < a_2$ oder wenn $a_1 = a_2$ und $b_1 < b_2$ gilt. Ferner gilt $(a_1,b_1) = (a_2,b_2)$ genau dann, wenn $a_1 = a_2$ und $b_1 = b_2$ gilt. Zur Definition der Kosten eines aufspannenden Baumes bezüglich w' muss noch eine Addition definiert werden: $(a_1,b_1) + (a_2,b_2) = (a_1 + b_1, a_2 + b_2)$. Es sei T' ein bezüglich w' minimaler aufspannender Baum. Man zeigt nun leicht, dass T' die gewünschten Eigenschaften hat. Hierzu betrachte man einen bezüglich w minimalen aufspannenden Baum T von G, für den der Eckengrad von e minimal ist. Aus $\text{kosten}_w(T) < \text{kosten}_w(T')$ würde $\text{kosten}_{w'}(T) < \text{kosten}_{w'}(T')$ folgen, dies widerspricht der Minimalität von T' bezüglich w'. Somit ist T' ein minmal aufspannender Baum bezüglich w von G. Da $\text{kosten}_{w'}(T) = (\text{kosten}_w(T), g_T(e))$ gilt, ist der Eckengrad von e in T' minimal. Somit kann das beschriebene Problem mit den Algorithmen von Prim bzw. Kruskal gelöst werden.

27. Angenommen k liegt in jedem minimal aufspannenden Baum von G. Es sei B ein beliebiger minimal aufspannender Baum von G. Entfernt man k aus B, so zerfällt B in die beiden Zusammenhangskomponenten U_1 und U_2. Da W ein geschlossener Weg ist, gibt es auf W eine Kante $k' = (u_1,u_2) \neq k$, mit $u_1 \in U_1$ und $u_2 \in U_2$. Entfernt man k aus B und fügt dafür k' ein, so entsteht wegen $\text{kosten}(k) \geq \text{kosten}(k')$ ein minimal aufspannender Baum von G, der k nicht enthält.

28. Es sei B ein minimal aufspannender Baum von G und k eine der längsten Kanten von B, die Länge von k sei $l(k)$. Entfernt man k aus B, so zerfällt B in zwei Zusammenhangskomponenten mit den Eckenmengen E_1 und E_2. Nach dem in Abschnitt 3.6 auf Seite 79 bewiesenen Satz, ist die Länge jeder Kante in G zwischen E_1 und E_2 mindestens so lang wie k. Somit ist für $R < l(k)$ der Graph G_R nicht zusammenhängend. D. h., R_{crit} ist größer oder gleich der Länge der längsten Kante in B. Da B zusammenhängend ist, folgt $R_{crit} = l(k)$.

Kapitel 4 – Suchverfahren in Graphen

1. (*a*) Es genügt, den Fall `TSB[i] < TSB[j]` zu betrachten. Gilt `TSE[i] < TSB[j]`, so sind die Intervalle disjunkt. Andernfalls erfolgt der Aufruf von `tiefensuche(e`$_j$`)` innerhalb des Aufrufs `tiefensuche(e`$_i$`)`. Somit muss `TSE[j] < TSE[i]` gelten, d. h., es gilt die dritte Bedingung.

   (*b*) Ist k eine Baum- oder Vorwärtskante, so erfolgt der Aufruf von `tiefensuche(e`$_j$`)` innerhalb des Aufrufs `tiefensuche(e`$_i$`)` und endet deshalb auch vor diesem, d. h.

   $$\texttt{TSB[i]} < \texttt{TSB[j]} < \texttt{TSE[j]} < \texttt{TSE[i]}.$$

   Gilt umgekehrt diese Ungleichungskette, so erfolgte der Aufruf `tiefensuche(e`$_j$`)` innerhalb der Prozedur `tiefensuche(e`$_i$`)`. Erfolgte der Aufruf direkt, so ist k eine Baumkante und andernfalls eine Vorwärtskante.

   Ist k eine Rückwärtskante, so erfolgt der Aufruf `tiefensuche(e`$_i$`)` innerhalb der Prozedur `tiefensuche(e`$_j$`)` und endet deshalb auch vor diesem, d. h.

   $$\texttt{TSB[j]} < \texttt{TSB[i]} < \texttt{TSE[i]} < \texttt{TSE[j]}.$$

   Gilt umgekehrt diese Ungleichungskette, so erfolgte der Aufruf `tiefensuche(e`$_i$`)` innerhalb der Prozedur `tiefensuche(e`$_j$`)`, d. h., e_j ist Vorgänger von e_i und k ist somit eine Rückwärtskante.

   Die Kante k ist genau dann eine Querkante, wenn der Aufruf `tiefensuche(e`$_j$`)` schon vor dem Aufruf der Prozedur `tiefensuche(e`$_i$`)` endet. D. h., k ist genau dann eine Querkante, wenn

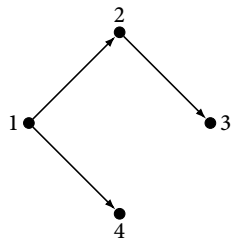
   $$\texttt{TSB[j]} < \texttt{TSE[j]} < \texttt{TSB[i]} < \texttt{TSE[i]}$$

   gilt.

2. Die Aussage ist wahr. Innerhalb des Aufrufs `tiefensuche(e`$_i$`)` werden alle von e_i erreichbaren unbesuchten Ecken besucht. Wegen `tsNummer[i] < tsNummer[j]`, muss e_j im Tiefensuchebaum von e_i erreichbar sein.

3. Die Adjazenzmatrix hat auf und unterhalb der Diagonalen nur Nullen.

4. Nein, dies zeigt eine topologische Sortierung, die zwischen zwei Teilgraphen *hin- und herspringt*. Betrachten Sie folgendes Beispiel:



Ecke	e_1	e_2	e_3	e_4
Sortierungsnummer	1	2	4	3

5. Ist w die Wurzel des Baumes, so ist `höhe(w)` die Höhe des Wurzelbaumes.

```
function höhe(e_i : Ecke) : Integer;
   var hmax : Integer;
   hmax := 0;
   foreach e_j in N(e_i) do
      hmax := max(hmax, höhe(e_j) + 1);
   return hmax;
```

6. {2,4,5,6}, {1}, {3}, {7}, {8}

7. Es sei T ein Tiefesuchebaum von G' mit Wurzel e_1 und e_i, e_j Ecken aus T. Als Ecke e_1 besucht wurde, war e_i noch nicht besucht worden. Somit ist `TSE[1] > TSE[i]`. D. h., der Aufruf von `tiefensuche(e_i)` war vor dem Aufruf von `tiefensuche(e_1)` beendet. Da e_i in G' von e_1 erreichbar ist, ist e_1 in G von e_i erreichbar. Wäre e_i vor e_1 in G betrachtet worden, so wäre `TSE[i] > TSE[1]`. Somit wurde e_1 vor e_i in G betrachtet. Aus `TSE[1] > TSE[i]` folgt nun, dass der Aufruf von `tiefensuche(e_i)` in G innerhalb des Aufrufs `tiefensuche(e_1)` erfolgte. Somit ist e_i in G von e_1 erreichbar, d. h., e_i und e_1 liegen auf einem geschlossenen Weg in G. Das gleiche gilt für e_j und e_1 und somit auch für e_i und e_j. Hieraus folgt, dass die Ecken von T in einer starken Zusammenhangskomponente von G liegen.

   Sind umgekehrt e_1 und e_2 Ecken von G, welche in einer starken Zusammenhangskomponente liegen, so liegen e_1 und e_2 in G und G' auf einem geschlossenen Weg. Somit müssen auch e_1 und e_2 im gleichen Tiefesuchebaum von G' liegen. Hieraus folgt, dass die Ecken einer starken Zusammenhangskomponente von G in einem Tiefesuchebaum von G' liegen.

8. Es wird eine topologische Sortierung des DAG's betrachtet. Die Ecke mit Sortierungsnummer 1 hat Eingrad 0 und die mit Sortierungsnummer n hat Ausgrad 0. Eine Ecke mit Ausgrad 0 findet man, indem man an einer beliebigen Ecke einen einfachen Weg startet. Da der Graph keine geschlossenen Wege hat, endet der Weg an einer Ecke mit Ausgrad 0. Ein Algorithmus zur Bestimmung einer topologischen Sortierung sucht wiederholt eine Ecke mit Ausgrad 0, entfernt diese und alle inzidenten Kanten aus G und nummeriert die entfernten Ecken in absteigender Reihenfolge. Da die Bestimmung einer Ecke mit Ausgrad 0 den Aufwand $O(n)$ hat, ergibt sich ein Gesamtaufwand von $O(n^2)$. Der auf der Tiefensuche aufbauende Algorithmus arbeitet mit Aufwand $O(n + m)$.

9. Zwei Ecken e_i und e_j sind genau dann in einer starken Zusammenhangskomponenten, wenn sie auf einem geschlossenen Weg liegen. Dies ist genau dann der Fall, wenn e_{ji} und e_{ij} beide ungleich 0 sind, bzw. wenn $e_{ji}e_{ij}$ ungleich 0 ist. Hierbei ist $E = (e_{ij})$. Für den i-ten Diagonaleintrag d_i von E^2 gilt:

$$d_i = e_{1i}e_{i1} + \ldots + e_{ji}e_{ij} + \ldots + e_{ni}e_{in}.$$

   Ist $d_i = 0$, so bildet Ecke e eine komplette Zusammenhangskomponente. Ist $d_i \neq 0$, so ist $e_{ii} = 1$. Ferner gibt es noch $d_i - 1$ weitere Ecken e_j mit $e_{ji}e_{ij} \neq 0$. Hieraus folgt sofort die Aussage.

10. Der Algorithmus basiert auf der Tiefensuche.

```
var besucht : Array[0..n] von Boolean;
var vorgänger : Array[0..n] von Ecke;
```

```
function geschlossenerWeg(G : Graph) : Boolean
   var e_i : Ecke;
   Initialisiere besucht mit false und vorgänger mit ⊥;
   foreach e_i in G.E do
      if not besucht[i] then
         if tiefensuche(e_i) then
            return true;
   return false;

function tiefensuche(e_i : Ecke) : Boolean
   var e_j : Ecke;
   besucht[i] := true;
   foreach e_j in N(e_i) do
      if not besucht[j] then
         vorgänger[j] := e_i;
         if tiefensuche(e_j)
            return true;
      else if vorgänger[j] ≠ e_i then
         return true;
   return false;
```

Am Ende des Algorithmus hat der Algorithmus einen Baum durchlaufen, d. h., die Anzahl der betrachteten Kanten ist kleiner als n. Somit ist die Laufzeit $O(n)$.

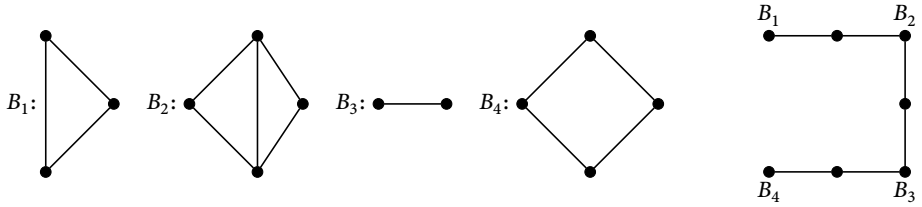
11. Es werden zwei Tiefensuchedurchgänge gestartet mit den Startecken e_i bzw. e_j. Hierbei wird das gleiche Feld `tsNummer` benutzt (im zweiten Durchgang wird es nicht mehr initialisiert). Jede nicht besuchte Ecke ist weder von e_i noch von e_j erreichbar und muss somit entfernt werden. Bei der Verwendung von Adjazenzlisten basierend auf Zeigern müssen nur die entsprechenden Listen komplett entfernt werden. Somit ist der Aufwand $O(n + m)$.

12. Das Feld `trennendeEcken` zeigt nach Aufruf der gleichnamigen Prozedur an, welche Ecken trennende Ecken sind.

```
var tsNummer, minNummer : Array[1..n] von Integer;
var vorgänger : Array[1..n] von Ecke
var zähler : Integer;
var trennendeEcken : Array[1..n] von Boolean;

procedure trennendeEcken(G : Graph)
   var e_i : Ecke;
   Initialisiere tsNummer und zähler mit 0;
   Initialisiere trennendeEcken mit false;
   foreach e_i in G.E do
      if tsNummer[i] = 0 then
         blöcke(e_i);
         if e_i ist mehr als einmal in vorgänger enthalten then
            trennendeEcken[i] := true;
         else
            trennendeEcken[i] := false;

procedure blöcke(e_i : Ecke)
   var e_j : Ecke
   zähler := zähler + 1;
```

```
tsNummer[i] := minNummer[i] := zähler;
foreach e_j in N(e_i) do
   if tsNummer[j] = 0 then
      vorgänger[j] := e_i;
      blöcke(e_j);
      if minNummer[j] ≥ tsNummer[i] then
         trennendeEcken[i] := true;
      else
         minNummer[i] := min(minNummer[i], minNummer[j]);
   else
      if e_j ≠ vorgänger[i] then
         minNummer[i] := min(minNummer[i], tsNummer[j]);
```

13. Links sind die Blöcke des Graphen und rechts ist der Blockbaum dargestellt.



14. Ist (e_i, e_j) eine Brücke, so muss e_i oder e_j eine trennende Ecke sein. Ist eine der beiden Ecken keine trennende Ecke, so kann diese keine weiteren Nachbarn haben, d. h., der Grad ist gleich 1. Die umgekehrte Aussage ist offensichtlich.

15. Der Beweis erfolgt durch vollständige Induktion nach b. Ist $b = 1$, so ist $t = 0$ und die Aussage ist wahr. Sei $b > 1$, B ein Block, welcher zu einem Blatt im Blockbaum gehört, und e die zugehörige Ecke. Entferne aus G alle Ecken in $B \backslash \{e\}$. Der neue Graph hat $b - 1$ Blöcke und mindestens $t - 1$ trennende Ecken. Die Aussage folgt nun aus der Induktionsvoraussetzung.

16. Ist der Graph zweifach zusammenhängend, so ist keine Ecke eine trennende Ecke. Andernfalls hat der Blockbaum mindestens zwei Blätter (jeder Baum mit mindestens zwei Ecken hat auch mindestens zwei Blätter). Ein Block, welcher zu einem Blatt im Blockbaum gehört, hat mindestens eine Ecke, welche nicht trennend ist.

17. Eine Kante liegt genau dann auf einem geschlossenen Weg, wenn sie zu einer starken Zusammenhangskomponente von G gehört. Mit dem in Abschnitt 4.5 beschriebenen Verfahren werden zunächst die starken Zusammenhangskomponenten bestimmt. Die gesuchte Kantenmenge besteht aus allen Kanten von G, welche in keiner starken Zusammenhangskomponente liegen.

18. Wendet man die Breitensuche n-mal auf den Graphen an, wobei jedesmal eine andere Startecke gewählt wird, so werden für jede Ecke die erreichbaren Ecken bestimmt. Der Aufwand ist $O(n\,m)$.

19. Es sei e eine beliebige Ecke und l_e die Länge des kürzesten Weges, der e enthält. Es wird eine Breitensuche mit Startecke e gestartet. Sei e_j die erste Ecke, welche die Breitensuche zum zweiten Mal besucht und e_i der aktuelle Vorgänger von e_j. Dann gilt

$$\mathrm{niv}(e_j) = \mathrm{niv}(e_i) \text{ oder } \mathrm{niv}(e_j) = \mathrm{niv}(e_i) + 1.$$

Im ersten Fall liegt ein geschlossener Weg der Länge $2\,\mathrm{niv}(e_i)+1$ und im zweiten Fall der Länge $2\,\mathrm{niv}(e_i)+2$ vor. Im ersten Fall gilt $l_e = 2\,\mathrm{niv}(e_i)+1$ und im zweiten $l_e = 2\,\mathrm{niv}(e_i)+2$ oder $l_e = 2\,\mathrm{niv}(e_i) + 1$. Um eine Unterscheidung zu treffen, müssen alle Ecken e' mit $\mathrm{niv}(e') = \mathrm{niv}(e_i)$ abgearbeitet werden. Danach kann die Breitensuche beendet werden. Trifft man dabei auf eine weitere schon besuchte Ecke e_j mit $\mathrm{niv}(e_j) = \mathrm{niv}(e_i)$, so gilt $l_e = 2\,\mathrm{niv}(i) + 1$ und andernfalls $l_e = 2\,\mathrm{niv}(e_i) + 2$.

Ein Algorithmus zur Bestimmung der Länge eines kürzesten Weges startet von jeder Ecke e eine Breitensuche. Diese wird wie beschrieben benutzt, um die Länge des kürzesten geschlossenen Weges W mit $e \in W$ zu bestimmen. So erhält man die Länge des kürzesten geschlossenen Weges des Graphen mit Aufwand $O(n\,m)$.

20. Zunächst wird in linearer Zeit eine topologische Sortierung des Graphen bestimmt. Danach werden die Ecken in umgekehrter Reihenfolge ihrer topologischen Sortierungsnummern bearbeitet und die Menge der erreichbaren Ecken wie folgt bestimmt:

$$\mathrm{erreichbar}(e_i) := \bigcup_{e_j \in N^+(e_i)} \mathrm{erreichbar}(e_j) \cup \{e_j\}.$$

21. Hat der Tiefensuchebaum die Höhe 1, so ist der Graph ein Sterngraph, wobei die Tiefensuche an der zentralen Ecke startete. In diesem Fall stimmen Tiefensuche- und Breitensuchebaum überein. Für den vollständigen Graph hat der Tiefensuchebaum die Höhe $n - 1$. In diesem Fall ist der Breitensuchebaum ebenfalls ein Sterngraph, d. h., er hat die Höhe 1.

22. Zur Bestimmung der Zusammenhangskomponenten eines ungerichteten Graphen mit Hilfe der Breitensuche kann die Prozedur `zhkGraph` aus Abbildung 4.14 verwendet werden. Für die Änderung wird der Aufruf der Prozedur `zhk(e_i)` durch den Aufruf `breitensuche(G, e_i)` ersetzt. Die Prozedur `breitensuche` aus Abbildung 4.25 muss dabei geringfügig geändert werden. Anstatt dem Feld `niveau` wird das Feld `zhkNummer` verwendet. Die Prozedur `breitensuche` sieht dann wie folgt aus.

```
procedure breitensuche(G : Graph, e_s : Ecke)
   var e_i, e_j : Ecke;
   var W : Warteschlange von Ecke;

   zhkNummer[s] := zähler;
   W.einfügen(e_s);
   while W ≠ ∅ do
      e_i := W.entfernen();
      foreach e_j in N(e_i) do
         if zhkNummer[j] = 0 then
            zhkNummer[j] := zähler;
            W.einfügen(e_j);
```

23. Die Änderungen an der Funktion `breitensuche` lassen den Aufwand von $O(m + n)$ unverändert.

```
function breitensuche(G : Graph, e_s : Ecke) : Graph
   var e_i, e_j : Ecke;
   var W : Warteschlange von Ecke;
   var niveau : Array[1..n] von Integer;
   var B  :  Graph;
```

```
Initialisiere niveau mit -1;
Initialisiere B mit den Ecken von G;
niveau[s] := 0;
W.einfügen(e_s);
while W ≠ ∅ do
   e_i := W.entfernen();
   foreach e_j in N(e_i) do
      if niveau[j] = -1 then
         niveau[j] := niveau[i] + 1;
         W.einfügen(e_j);
         Füge Kante (e_i, e_j) in B ein;
      else if niveau[j] = niveau[i] + 1 then
         Füge Kante (e_i, e_yj) in B ein;
return B;
```

24. (*a*) Der Beweis erfolgt durch vollständige Induktion nach n. Für $n = 1$ ist die Aussage wahr. Sei nun $n > 1$ und e eine Ecke von B mit $g^+(e) = 0$. Setze $G' = G\backslash\{e\}$ und $B' = B\backslash\{e\}$. Da B' die gleichen Eigenschaften wie B hat, folgt aus der Induktionsvoraussetzung, dass B' ein Tiefensuchebaum von G' ist. Es sei nun e' der Vorgänger von e in B und (e,e'') mit $e'' \neq e'$ eine Kante in G. Nach Voraussetzung ist e'' Vorgänger von e in B, d. h., wenn die Tiefensuche bei Ecke e' ankommt, sind schon alle Nachbarn von e besucht worden. Somit wird nach e' die Ecke e besucht und sofort wieder zu e' zurückgekehrt. Also ist B ein Tiefensuchebaum von G.

    (*b*) Für eine beliebige Ecke e' von G bezeichne $d_B(e,e')$ die Länge des Weges von e nach e' in B. Mittels vollständiger Induktion nach $d(e,e')$ wird gezeigt, dass $d_B(e,e') = d(e,e')$ für alle Ecken e' von G gilt. Ist $d(e,e') = 0$, so ist $e' = e$ und die Aussage ist wahr. Sei nun $d(e,e') > 0$ und e'' der Vorgänger von e' in G auf dem Weg von e nach e'. Dann ist $d(e,e'') < d(e,e')$ und somit ist $d_B(e,e'') = d(e,e'')$ nach Induktionsvoraussetzung. Also gilt: $d_B(e,e') \geq d(e,e') = d(e,e'') + 1 = d_B(e,e'') + 1$. Da (e',e'') eine Kante in G ist, unterscheiden sich $d_B(e,e')$ und $d_B(e,e'')$ maximal um 1. Hieraus folgt $d_B(e,e') = d(e,e')$. Man beachte, dass B nicht notwendigerweise durch die in Abschnitt 4.8 beschriebene Realisierung der Breitensuche erzeugt werden kann.

25. Da B keine geschlossenen Wege enthält, können die Verfahren etwas vereinfacht werden. Wird die Tiefensuche wie in Abschnitt 4.2 beschrieben mit Hilfe eines Stapels realisiert, so ist der Speicheraufwand proportional zur maximalen Stapeltiefe. Da im ungünstigsten Fall bis zu den Blättern gesucht werden muss, ist der Speicheraufwand $O(H)$. Bei der iterativen Tiefensuche wird keine Ecke jenseits von Tiefe von e betrachtet. Somit ist der Speicheraufwand $O(d_e)$ mit $d_e = d(w,e)$. Der Speicheraufwand der Breitensuche ist proportional zur Länge der Warteschlange. Im ungünstigsten Fall wird die gesuchte Ecke als letzte Ecke in dem entsprechenden Niveau betrachtet. Da in diesem Fall alle Ecken aus Niveau d_e in der Warteschlange sind, ist der Speicheraufwand $O(b^{d_e})$.

    Die Laufzeit der drei Suchverfahren ist proportional zur Anzahl der betrachteten Ecken. Bei der Tiefensuche müssen alle Ecken bis auf die Nachfolger von e besucht werden. Bei der Breitensuche alle Ecken im gleichen oder in vorhergehenden Niveaus. Für die Tiefensuche sind dies $(b^{H-d_e+1} - 1)/(b-1)$ und für die Breitensuche $(b^{d_e+1} - 1)/(b-1)$ Ecken. Bei der iterativen Tiefensuche werden maximal $b^{d_e+2}/(b-1)^2$ Ecken betrachtet (vergleichen Sie Abschnitt 4.10).

26. Ist (e, e') eine Rückwärtskante eines ungerichteten Graphen, so ist entweder e ein Vorgänger oder ein Nachfolger von e' im Tiefensuchewald, d. h., Rückwärtskanten verbinden niemals Blätter des Tiefensuchebaums. Da auch Vorwärtskanten keine Blätter verbinden, ist die Aussage bewiesen.

27. Der Algorithmus ist eine Variante der Tiefensuche angewendet auf die Ausgangsecke des inversen Graph des Schaltkreises (d. h., die Richtung jeder Kante des Graphen wird umgedreht).

```
function schaltkreis(e : Ecke) : Boolean
   switch e.art begin
      case Eingabeecke :
         return Wert(e);
      case Konjunktion :
         return schaltkreis(Nachfolger1(e)) and schaltkreis(Nachfolger2(e));
      case Disjunktion :
         return schaltkreis(Nachfolger1(e)) or schaltkreis(Nachfolger2(e));
      case Negation :
         return not schaltkreis(Nachfolger(e));
      case Ausgabeecke :
         return schaltkreis(Nachfolger(e));
```

28. (*a*) Die Aussage folgt direkt aus der Beobachtung, dass stark zusammenhängende Graphen auch halbzusammenhängend sind.

    (*b*) Man bestimme eine topologische Sortierung des kreisfreien Strukturgraphen $\hat{G}$. Die Ecken e_1, e_2, ..., e_n seien gemäß ihrer Sortierungsnummern nummeriert. Zunächst wird folgende Aussage bewiesen: $\hat{G}$ ist genau dann halbzusammenhängend, wenn für $i > 0$ Ecke e_i ein direkter Vorgänger von e_{i+1} ist. Gilt diese Bedingung, so existiert der Weg $<e_1, \ldots, e_n>$, d. h., $\hat{G}$ ist halbzusammenhängend. Sei nun $\hat{G}$ halbzusammenhängend. Für $i > 0$ muss es in $\hat{G}$ einen Weg von e_i nach e_{i+1} geben. Auf diesem Weg kann aber keine andere Ecke liegen, diese müsste nämlich eine Sortierungsnummer zwischen $t + 1$ und t haben. Dies zeigt, dass e_i ein direkter Vorgänger von e_{i+1} ist.

    Diese Aussagen führen unmittelbar zu einem Algorithmus mit Laufzeit $O(n + m)$. Dazu beachte man, dass die Erstellung des Strukturgraphen, die Bestimmung der topologischen Sortierung und die Überprüfung der angegebenen Bedingung mit linearem Aufwand durchgeführt werden kann.

29. (*a*) Man bestimme eine topologische Sortierung von G und führe folgenden Algorithmus zur Bestimmung der Menge H aus.

```
H := ∅;
E' := G.E;
while E' ≠ ∅ do
    Wähle e aus E' mit der größten Sortierungsnummer und füge e in H ein;
    Entferne e und alle Vorgänger von e in G aus E';
```

    Aus der Konstruktion ergibt sich sofort, dass H eine unabhängige Menge ist.

    (*b*) Es sei $s > 0$ und G ein gerichteter Graph mit $2s + 1$ Ecken, der genau aus einem geschlossenen Weg besteht. Für jede unabhängige Menge H von G gilt $|H| \leq s$ bzw. $|E \backslash H| \geq s + 1$. Da jede Ecke genau einen Nachfolger und einen Vorgänger hat, kann nicht jede Ecke aus $E \backslash H$ einen Nachfolger in H haben.

Kapitel 5 – Entwurfsmethoden für die algorithmische Graphentheorie

1. Die Nebenbedingungen lassen zu, dass eine Lösung des ILP-Problems aus zwei oder mehr disjunkten geschlossenen Wegen besteht.

2. (*a*) Es sei I eine maximale unabhängige Menge von G. Ist $e \notin I$, so ist I eine maximale unabhängige Menge von $G_{E\setminus\{e\}}$. Ist $e \in I$, so ist $I\setminus\{e\}$ eine unabhängige Menge von $G_{E\setminus N[e]}$. Wäre $I\setminus\{e\}$ keine maximale unabhängige Menge von $G_{E\setminus N[e]}$, dann gäbe es eine unabhängige Menge I' von $G_{E\setminus N[e]}$ mit $|I'| > |I| - 1$. Somit wäre auch $I' \cup \{e\}$ eine unabhängige Menge mit mehr als $|I| + 1$ Ecken. Hieraus folgt die Behauptung.

   (*b*) Es genügt, eine maximale unabhängige Menge für jede Zusammenhangskomponente zu bestimmen. Ist $\Delta(G) \leq 2$, so sind die Zusammenhangskomponenten von G isolierte Ecken, Pfade oder geschlossene Wege. In jedem dieser Fälle lässt sich eine maximale unabhängige Menge in linearer Zeit bestimmen.

   (*c*) Die Korrektheit folgt direkt aus Teil *(a)*.

   (*d*) Bezeichne mit $T(n)$ die Anzahl der Schritte, die die Funktion `unabhängig` für Graphen mit n Ecken benötigt. Dann gibt es eine Konstante $C > 0$, so dass für $n > 4$ wegen $\Delta(G) \geq 3$ folgende Beziehung gilt:

$$T(n) \leq T(n-1) + T(n-(\Delta(G)+1)) + Cn \leq T(n-1) + T(n-4) + Cn.$$

   Schätzt man $T(n)$ durch λa^n mit $a > 1$ ab und wählt a, so dass

$$a^n \approx a^{n-1} + a^{n-4} + o(1)$$

   erfüllt ist, dann ist folgende Ungleichung zu lösen:

$$a^4 \geq a^3 + 1 + \epsilon.$$

   Der Wert $a \approx 1{,}39$ erfüllt diese Ungleichung. Somit ist $O(1{,}39^n)$ der Aufwand der Funktion `unabhängig`.

3. (*a*) Es sei k eine beliebige Kante des Graphen. Die Menge R der Rundreisen kann in zwei Teilmengen aufgeteilt werden: R_1 enthält die Rundreisen, die k verwenden, und R_2 die Rundereisen ohne die Kante k. Nun werden zwei neue Graphen G_1 und G_2 gebildet. Für Graph G_1 werden die Endecken von k in G zu einer Ecke verschmolzen. Für Graph G_2 wird Kante k aus G entfernt. Das Problem des Handlungsreisenden wird nun für G_1 und G_2 getrennt gelöst.

   (*b*) Jede Rundreise verwendet für jede Ecke e genau zwei zu e inzidente Kanten. Somit ist der Beitrag, welchen die zu e inzidenten Kanten der Rundreise zur Gesamtlänge beitragen, mindestens so hoch wie die Summe der Längen der beiden kürzesten zu e inzidenten Kanten.

   Nachdem die Länge l einer ersten Rundreise bekannt ist, werden nur noch Rundreisen kürzerer Länge gesucht. Deshalb müssen Graphen, bei denen die untere Grenze für eine Rundreise über l liegt, nicht betrachtet werden.

4. Für $|S| = 1$ besteht der minimale Steiner Baum genau aus einer Ecke. In diesem Fall gibt es keine Menge U, welche die Anforderungen der Nebenbedingungen erfüllt. Somit ist $x_k = 0$ für alle $k \in K$ eine Lösung des ILP-Problems, diese entspricht dem leeren Baum.

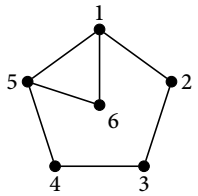
   Sei nun $|S| > 1$. Für eine Lösung des ILP-Problems sei T_S der von den Kanten k mit $x_k = 1$ gebildete Graph. Für $s \in S$ muss $\beta(\{s\}) \geq 1$ sein. Somit ist jede Ecke von S eine Ecke von T_S. Jede Zusammenhangskomponente von T_S muss mindestens eine Ecke von S enthalten, sonst würde die Zielfunktion für die Lösung keinen minimalen Wert annehmen. Angenommen T_S ist nicht zusammenhängend. Es sei U die Menge der Ecken einer Zusammenhangskomponente. Dann ist $\beta(U) \geq 1$, d. h., es gibt eine Ecke außerhalb von U, welche durch eine Kante k mit $x_k = 1$ mit einer Ecke von U verbunden ist. Dieser Widerspruch zeigt, dass T_S zusammenhängend ist. Angenommen T_S enthält einen geschlossen Weg. Es sei k eine beliebige Kante dieses Weges. Dann könnte x_k auf den Wert 0 gesetzt werden und es wäre immer noch eine Lösung des ILP-Problems. Dies zeigt, dass T_S ein Steiner Baum für S ist. Da ein minimaler Steiner Baum für S eine Lösung des ILP-Problems ist, ist der Beweis vollständig.

5. Der angegebene Algorithmus ist ein Greedy-Algorithmus. Eine Implementierung mit Aufwand $O(n + m)$ kann mit den in Abschnitt 5.2 vorgestellten Datenstrukturen erreicht werden. Der einzige Unterschied besteht darin, dass die Ecke mit kleinstem Eckengrad entfernt wird, hierzu wird ein Zeiger auf das Ende der Liste `EckengradListe` verwaltet. Ferner werden noch Zähler für die Anzahl der in C verbliebenen Kanten und Ecken verwaltet. Mit diesen Zählern kann einfach überprüft werden, ob C eine Clique ist.

6. Es sei $e_1, \ldots, e_n$ eine Reihenfolge der Ecken. Der Algorithmus bestimmt zuerst alle nicht-erweiterenbaren Cliquen die e_1 enthalten. Danach die, welche e_2 und nicht e_1 enthalten (diese sind in $P = N(e_2)\backslash\{e_1\}$ enthalten). Allgemein bestimmt ein Aufruf die nicht-erweiterenbaren Cliquen welche e_i aber nicht $e_1, \ldots, e_{i-1}$ enthalten.

   Für jeden Aufruf `bronKerbosch(P, R, X)` gilt, dass jede Ecke $e \in R$ zu allen Ecken aus $P \cup X$ benachbart ist. Dies gilt trivialerweise für den initialen Aufruf der Prozedur `bronKerbosch(G.E, ∅, ∅)`. Da in jedem folgenden Aufruf R durch $R \cup \{e\}$ mit $e \in P$, P durch $P \cap N(e)$ und X durch $X \cap N(e)$ ersetzt wird, gilt diese Aussage zu jedem Zeitpunkt. Da immer nur Ecken aus P in R eingefügt werden, folgt daraus, dass der von R induzierte Graph immer eine Clique ist. Aus dem gleichen Grund ist jede Ecke, die zu allen Ecken in R benachbart ist, in $P \cup X$ enthalten. Man beachte, dass in der Schleife nur Ecken von P nach X verschoben werden. Hieraus folgt, dass R genau dann eine nicht-erweiterbare Clique ist, wenn $P \cup X = \emptyset$. Da der Algorithmus gemäß der Backtracking Methode den gesamten Suchbaum durchläuft, werden alle nicht-erweiterbaren Cliquen gefunden.

7. Betrachtet der Algorithmus für den folgenden Graph zuerst Ecke e_3, dann verbleiben noch die drei Ecken e_1,e_5,e_6 mit Grad 2. Somit besteht die berechnete unabhängige Menge aus zwei Ecken. Wird hingegen zuerst Ecke e_4 betrachtet, so bleiben die Ecken e_1,e_2,e_6 über. In diesem Fall wird die Menge e_4,e_6,e_2 als unabhängige Menge mit drei Ecken bestimmt.



8. Gehört e zu einer unabhängigen Menge von B_e mit Gewicht $\mathrm{OPT}(e)$ so ist

$$\mathrm{OPT}(e) = w(e) + \sum_{e' \in N^{++}(e)} \mathrm{OPT}(e').$$

Andernfalls gilt $\mathrm{OPT}(e) = \sum_{e' \in N^{+}(e)} \mathrm{OPT}(e')$. Hieraus ergibt sich die angegebene Gleichung.

Ein Algorithmus auf Basis der dynamischen Programmierung verwaltet für jede Ecke e von B zwei Variablen $e.o$ und $e.w$ mit folgenden Werten:

- $e.o = \mathrm{OPT}(e)$,
- $e.w = \sum_{e' \in N^{+}(e)} \mathrm{OPT}(e')$.

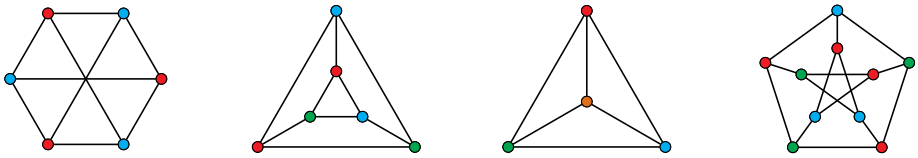
Initial sind alle Werte gleich 0. Der Baum wird gemäß der Tiefensuche durchlaufen. Immer wenn ein Knoten endgültig verlassen wird, wird zuerst $e.w$ und anschließend $e.o$ gemäß folgender Gleichung bestimmt:

$$e.o = \max\left(\sum_{e' \in N^{+}(e)} e'.o,\ w(e) + \sum_{e' \in N^{+}(e)} e'.w \right).$$

Der Zeitaufwand ist $O(\sum_{e \in B} |N^{+}(e)|) = O(n)$. Der Speicheraufwand ist ebenfalls linear.

Kapitel 6 – Färbung von Graphen

1. Die chromatische Zahl der vier Graphen ist 2, 3, 4 und 3.



2. Es gilt $\Delta(G_n) = n - 1$, $\chi(G_n) = n - 2$ und $\omega(G_n) = n - 3$.

3. Angenommen es gibt in G ein Paar $\{e, e'\}$ von nicht benachbarten Ecken. Dann kann jede Färbung von $G\backslash\{e, e'\}$ mit $\chi(G) - 2$ Farben zu einer Färbung von G mit $\chi(G) - 1$ Farben erweitert werden (e und e' werden mit einer noch nicht verwendeten Farbe gefärbt). Dieser Widerspruch zeigt, dass alle Ecken von G paarweise benachbart sind, d. h., G ist vollständig.

4. Es gilt $\chi(L_{z,s}) = 2$ (man färbe die Ecken wie ein Schachbrett) und $\omega(L_{z,s}) = 2$.

5. Man unterteile die Ecken von Q_s in zwei Teilmengen, je nachdem ob die Anzahl der Einsen gerade oder ungerade ist. Da die beiden Mengen unabhängige Mengen sind, ist $\chi(Q_s) = 2$. Da Q_s keine Dreiecke enthält, ist $\omega(Q_s) = 2$.

6. Es sei B ein beliebiger Tiefensuchebaum von G der Höhe h. Es gilt $h \leq m$. Die Ecken von G werden derart gefärbt, so dass die Ecken auf einem Niveau jeweils die gleiche Farbe bekommen. Dazu werden $h + 1$ Farben verwendet. Da nach den Ergebnissen aus Abschnitt 4.7 eine Kante niemals Ecken aus dem gleichen Niveau verbindet, liegt eine zulässige Färbung vor. Somit gilt $\chi(G) \leq m + 1$.

7. Man betrachte den in Abschnitt 4.7.4 definierten Blockbaum G_B und erzeuge für einen beliebigen Block B eine minimale Färbung. Danach wird G_B anhand der Tiefensuche mit Startecke B durchlaufen. Da G_B ein Baum ist, ist in jedem neuen Block genau eine Ecke schon gefärbt. Diese wird jeweils zu einer minimalen Färbung erweitert. Hieraus folgt die Behauptung. Färbungsalgorithmen können diese Eigenschaft nutzen und zunächst in linearer Zeit die Blöcke bestimmen und danach wie beschrieben weiter verfahren. Der Vorteil liegt darin, dass gegebenenfalls kleinere Graphen gefärbt werden müssen.

8. Es sei A die Menge der Ecken von G, deren Grad mindestens $\chi(G) - 1$ ist. Angenommen es gilt $|A| \leq \chi(G) - 1$. Dann färbe man die Ecken von G mit dem Greedy-Algorithmus, wobei zuerst die Ecken aus A betrachtet werden. Hierfür werden maximal $\chi(G) - 1$ Farben verwendet. Da die restlichen Ecken maximal den Eckengrad $\chi(G) - 2$ haben, vergibt der Greedy-Algorithmus insgesamt höchstens $\chi(G) - 1$ Farben. Dieser Widerspruch beweist $|A| \geq \chi(G)$.

9. Es sei e eine beliebige Ecke von G. Dann ist nach Voraussetzung $N(e)$ eine unabhängige Menge von G. Also gilt $\alpha(G) \geq \Delta(G)$. Somit folgt

$$(\alpha(G) + 1)^2 > \alpha(G)(\alpha(G) + 1) \geq \alpha(G)(\Delta(G) + 1) \geq \alpha(G)\chi(G) \geq n.$$

Dies beweist die Behauptung.

10. Da es zwischen den Ecken einer unabhängigen Menge keine Kanten gibt, gilt

$$m \leq n(n-1)/2 - \alpha(G)(\alpha(G)-1)/2 = (n^2 - \alpha(G)^2 - (n-\alpha(G)))/2$$

bzw. $2m \leq n^2 - \alpha(G)^2$. Hieraus folgt die Behauptung. Gilt $\alpha(G) = \sqrt{n^2 - 2m}$, so folgt aus obiger Ungleichung, dass $\alpha(G) = n$ gilt. Da G zusammenhängend ist, besteht G in diesem Fall nur aus einer Ecke.

11. (a) Für jede Ecke e gilt $\chi(G\backslash\{e\}) \geq \chi(G)-1$. Da nach Voraussetzung $\chi(G\backslash\{e\}) < \chi(G)$ für jede Ecke gilt, muss $\chi(G\backslash\{e\}) = \chi(G) - 1$ sein.

(b) Kritisch sind C_2 und C_n mit ungeradem $n > 1$.

(c) Kritische Graphen sind offensichtlich zusammenhängend. Ist G ein 2-kritischer Graph, so hat $G\backslash\{e\}$ keine Kanten. Daraus folgt $G = C_2$. Ein 3-kritischer Graph G ist nicht bipartit und enthält somit einen geschlossenen Weg W ungerader Länge. Alle Ecken von G müssen auf W liegen. Gäbe es Kanten, welche nicht auf W liegen, so würde es eine Ecke e geben, so dass $G\backslash\{e\}$ immer noch einen geschlossenen Weg ungerader Länge enthält. Da $\chi(G\backslash\{e\}) = 2$ ist, kann dies nicht sein. Somit ist G vom Typ C_{2i+1} mit $i \geq 1$.

(d) Es sei G ein kritischer Graph und B_i die Blöcke von G. Nach Aufgabe 7 gilt:

$$\chi(G) = \max\{\chi(B_i) \mid i = 1,\ldots,s\}.$$

Angenommen es ist $s > 1$. Ohne Einschränkung kann man annehmen, dass $\chi(G) = \chi(B_1)$ ist. Dann gibt es eine Ecke e in $B_2\backslash B_1$. Somit ist $\chi(G\backslash\{e\}) = \chi(B_1) = \chi(G)$. Dieser Widerspruch zeigt, dass $s = 1$ gilt, d. h., G ist zweifach zusammenhängend.

(e) Es sei e eine Ecke mit $g(e) = \delta(G)$ und $G' = G - \{e\}$. Dann ist $\chi(G') = c - 1$. Bezeichne mit $F_1,\ldots,F_{c-1}$ die Farbklassen einer minimalen Färbung von G'. Wäre $\delta(G) < c-1$, dann würde es eine Farbklasse F_j geben, so dass e zu keiner Ecke aus F_j benachbart ist. Eine Färbung von e mit der zu F_j gehörenden Farbe würde nun zu einer $c-1$-Färbung von G führen. Dieser Widerspruch zeigt, dass $\delta(G) \geq c-1$ ist.

(f) Der Graph ist 4-kritisch.

12. Analog zur Lösung von Aufgabe 11 (e) in diesem Kapitel. Man beachte, dass G' weniger Kanten als G hat und somit $\chi(G') \leq c-1$ gilt.

13. Für eine minimale Färbung muss es zu jedem Paar von Farben eine Kante geben, welche Ecken mit dieser Farbe verbindet. Somit gilt: $\chi(G)(\chi(G)-1)/2 \leq m$ bzw. $(\chi(G) - 1/2)^2 \leq 2m + 1/4$. Hieraus folgt die erste Ungleichung. Gleichheit gilt für vollständige Graphen.

Als nächstes wird die Ungleichung $2\sqrt{n} \leq \chi(G) + \chi(\overline{G})$ betrachtet. Es seien f_1 bzw. f_2 minimale Färbungen von G bzw. $\overline{G}$. Ordne jeder Ecke e aus G das Paar $(f_1(e), f_2(e))$ zu. Angenommen es gibt zwei Ecken $e' \neq e''$, welche das gleiche Paar zugeordnet bekommen. Dann ist $f_1(e') = f_1(e'')$ und $f_2(e') = f_2(e'')$. Somit können e' und e'' weder in G noch in $\overline{G}$ benachbart sein. Dieser Widerspruch beweist, dass $n \leq \chi(G)\chi(\overline{G})$ gilt. Hieraus folgt

$$4n \leq 4\chi(G)\chi(\overline{G}) \leq (\chi(G) + \chi(\overline{G}))^2$$

und somit $2\sqrt{n} \leq \chi(G) + \chi(\bar{G})$. Für C_4 gilt $\chi(C_4) + \chi(\overline{C_4}) = 4 = 2\sqrt{4}$ und für $K_{s,s}$ mit $s > 2$ gilt $\chi(K_{s,s}) + \chi(\overline{K_{s,s}}) = 2 + s > 2\sqrt{2s} = 2\sqrt{n}$.

Der Beweis der Ungleichung $\chi(G) + \chi(\bar{G}) \leq n + 1$ wird durch vollständige Induktion nach n geführt. Für $n = 2$ ist die Aussage wahr. Sei nun $n > 2$, e eine Ecke von G und $G_e = G\backslash\{e\}$. Dann gilt $\chi(G_e) + 1 \geq \chi(G)$ und $\chi(\overline{G_e}) + 1 \geq \chi(\bar{G})$. Nach Induktionsvoraussetzung ist $\chi(G_e) + \chi(\overline{G_e}) \leq n$. Gilt sogar $\chi(G_e) + \chi(\overline{G_e}) < n$, so folgt die Aussage direkt. Somit bleibt noch der Fall $\chi(G_e) + \chi(\overline{G_e}) = n$. Ist $g(e) < \chi(G_e)$, so folgt $\chi(G) = \chi(G_e)$ und hieraus ergibt sich $\chi(G) + \chi(\bar{G}) \leq n + 1$. Ist $g(e) \geq \chi(G_e)$, so gilt

$$\bar{g}(e) = n - 1 - g(e) \leq n - 1 - \chi(G_e) = n - 1 - (n - \chi(\overline{G_e})) = \chi(\overline{G_e}) - 1$$

und somit $\bar{g}(e) < \chi(\overline{G_e})$. Also ist $\chi(\bar{G}) = \chi(\overline{G_e})$, woraus die Aussage folgt.

Für die Graphen $K_{s,s}$ gilt $\chi(K_{s,s}) + \chi(\overline{K_{s,s}}) = 2 + s < 2s + 1 = n + 1$ und für die Graphen K_n gilt $\chi(K_n) + \chi(\overline{K_n}) = n + 1$.

14. Es gilt $\chi(R_n) = 1 + \chi(C_n)$ für alle $n \geq 3$. Somit ist $\chi(R_{2n}) = 3$ und $\chi(R_{2n+1}) = 4$.

15. Da I_5 vollständig ist, gilt $\chi(I_5) = 5$. Für $n \geq 6$ gilt nach dem Satz von Brooks $\chi(I_n) \leq 4$ (siehe Seite 169). Da I_n einen geschlossenen Weg der Länge drei enthält ist $\chi(I_n) \geq 3$. Ist $n \equiv 0(3)$, so können die Ecken im Uhrzeigersinn mit den Farben 1, 2, 3, 1, 2, 3, ... gefärbt werden. Somit ist $\chi(I_{3n}) = 3$ für $n \geq 2$. Ist $n \not\equiv 0(3)$, so legen die Farben von zwei auf C_n benachbarten Ecken eine Färbung fest. In diesem Fall kommt man nicht mit drei Farben aus, d. h., die chromatische Zahl ist 4.

16. Der Beweis wird durch vollständige Induktion nach s geführt. Für $s = 3$ ist die Aussage wahr. Sei nun $s > 3$. Es ist zu zeigen, dass G_s keinen geschlossen Weg der Länge drei enthält. Dazu beachte man zunächst, dass der von den neuen Ecken induzierte Untergraph diese Eigenschaft hat. Nach Induktionsvoraussetzung und Konstruktion bilden die Nachbarn der Ecken von G_{s-1} in G_s jeweils eine unabhängige Menge. Somit ist $\omega(G_s) = 2$. Eine minimale Färbung von G_{s-1} kann zu einer Färbung von G_s erweitert werden, indem die Ecke e' die gleiche Farbe wie e und die zusätzliche Ecke e'' eine neue Farbe bekommt. Somit gilt $\chi(G_s) \leq s$ nach Induktionsvoraussetzung. Angenommen es gibt eine Färbung von G_s, welche nur $s - 1$ Farben verwendet. Da Ecke e'' zu jeder neuen Ecke benachbart ist, ist keine dieser Ecken mit der gleichen Farbe gefärbt. Nun kann für den Untergraphen G_{s-1} von G_s eine Färbung mit $s - 2$ Farben erzeugt werden. Dazu bekommt jede Ecke e die Farbe von e'. Dies widerspricht der Induktionsvoraussetzung und somit ist $\chi(G_s) = s$.

17. Es werden maximal zwei weitere Farben benötigt. Man wende die Breitensuche auf B an und färbe die noch nicht gefärbten Ecken in ungeraden Niveaus mit der Farbe 2 und die in geraden Niveaus mit der Farbe 3.

18. Es wird ein Graph gebildet, in dem jede Radiostation einer Ecke entspricht und zwei Ecken verbunden sind, wenn die Entfernung der zugehörigen Stationen unter dem vorgegebenen Wert liegt. Radiostationen, welche in einer festen minimalen Färbung dieses Graphen die gleiche Farbe haben, können die gleiche Sendefrequenz benutzen.

19. Nein. Die Prozedur stützt sich bei der Färbung einer Ecke e mit $g(e) < 5$ wesentlich auf die Planarität des Graphen. Der Graph K_6 erfüllt die angegebene Bedingung, aber es gilt $\chi(K_6) > 5$.

20. Es sei G ein ungerichteter Graph, so dass G und $\overline{G}$ planar sind. Bezeichne die Anzahl der Kanten von G mit m und die von $\overline{G}$ mit $\overline{m}$. Dann gilt nach den Ergebnissen aus Abschnitt 6.5: $n(n-1)/2 = m + \overline{m} \leq 2(3n-6)$. Hieraus folgt $n(n-1)/(n-2) \leq 12$. Diese Ungleichung ist für $n > 10$ nicht erfüllt.

21. Es sei s die Anzahl der Farbklassen in einer nicht trivialen Färbung, welche genau eine Ecke enthalten. Diese bilden eine Clique. Somit ist $s \leq \chi(G)$. Da die restlichen Farbklassen mindestens zwei Farben enthalten, gilt $\chi_n(G) \leq s + (n-s)/2 \leq (n + \chi(G))/2$ bzw. $2(n-\chi_n(G)) \geq (n-\chi(G))$. Hieraus folgt die Behauptung. Der Greedy-Algorithmus erzeugt eine nicht triviale Färbung.

22. Es genügt folgende Aussage zu beweisen: Für jede Ecke e_i gilt: $h(e_i) + f(e_i) \leq h + 1$. Hierbei bezeichnet $f(e_i)$ die vom Greedy-Algorithmus vergebene Farbnummer und $h(e_i)$ die Höhe der Ecke e_i. Der Beweis erfolgt durch vollständige Induktion nach der Farbnummer. Für Ecken e_i mit $f(e_i) = 1$ ist die Aussage klar. Sei nun e_i eine Ecke mit $f(e_i) > 1$. Da e_i nicht mit der Farbnummer $f(e_i) - 1$ gefärbt wurde, muss es einen Nachbarn e_j von e_i mit $j < i$ und $f(e_j) = f(e_i) - 1$ geben. Dann folgt $h(e_j) \geq h(e_i) + 1$. Nach Induktionsvoraussetzung gilt nun:

$$h + 1 \geq h(e_j) + f(e_j) \geq h(e_i) + 1 + f(e_i) - 1 = h(e_i) + f(e_i).$$

23. Für $\omega(G) = \Delta(G)+1$ ist die Vermutung für jeden Wert von ϵ korrekt. Ist $\omega(G) < \Delta(G)+1$ dann ist der gesuchte Wert von ϵ nach oben beschränkt:

$$\epsilon \leq \frac{\Delta(G) + 1 - \chi(G)}{\Delta(G) + 1 - \omega(G)}.$$

Es sei G_k das Komplement des Graphen bestehend aus k Kopien von C_5. Dann gilt $\omega(G_k) = 2k$, $\chi(G_k) = 3k$ und $\Delta(G_k) = 2+5(k-1)$ und damit $\epsilon \leq (2k-2)/(3k-2) < 2/3$.

24. Der Graph G_S wird nicht explizit abgespeichert. Die Nachbarschaftsrelation wird durch die Funktion `nachbar` implementiert. In dem Feld `aufteilung` der Länge 81 werden die vergebenen Ziffern abgespeichert. Die vorgegebenen Ziffern werden in dem Feld `vorgabe` der gleichen Länge abgespeichert. Nicht belegte Zellen des Gitters werden mit 0 initialisiert. Durch die Vorgabe einiger Ziffern müssen auch für die erste Ecke alle Ziffern durchprobiert werden. Deshalb lautet der initiale Aufruf `partition(-1)`. Vor diesem Aufruf wird das Feld `aufteilung` mit den Werten des Feldes `vorgabe` initialisiert. Folgende Änderungen gegenüber dem in Kapitel 5 vorgegebenen Code müssen umgesetzt werden:

    (*a*) Vorgegebene Ziffern dürfen nicht geändert werden. Die Funktion `partition` muss dazu erweitert werden.

    (*b*) Für alle nicht vorgegebenen Zellen müssen alle Ziffern durchprobiert werden, d. h., die Verwendung des Feldes `maximum` entfällt.

    (*c*) Bei der Überprüfung einer Teillösung müssen immer alle vorbelegten Ziffern mitberücksichtigt werden. Dazu muss die Funktion `korrekteTeillösung` erweitert werden.

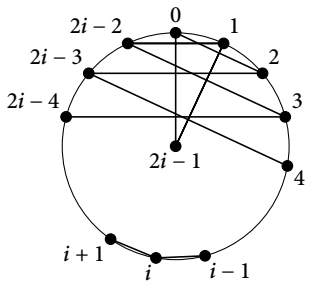
```
const DIM = 9;
var aufteilung : Array[0..DIM*DIM] von Integer;
var vorgabe : Array[0..DIM*DIM] von Integer;

function partition(i : Integer) : Boolean
   while vorgabe[i+1]! = 0 and i ≤ DIM*DIM-1 do
      i := i+1;
   if erweitereTeillösung(i) then
      do
         if korrekteTeillösung(i+1) then
            if i+1 = DIM*DIM-1 then
               return true;
            if partition(i+1)
               return true;
      while variiereTeillösung(i+1);
   return false;

function erweitereTeillösung(i : Integer) : Boolean
   if i = DIM*DIM-1 then
      return false;
   else
      aufteilung[i+1] :=  1;
      return true;

function variiereTeillösung(i : Integer) : Boolean
   if aufteilung[i] ≥ DIM then
      return false;
   else
      aufteilung[i] := aufteilung[i] + 1;
      return true;

function nachbar(i : Integer, j : Integer) : Boolean;
   var i1, i2, j1, j2 : Integer;
   if i = j then
      return false;
   i2 := i mod DIM; i1 := (i-i2)/DIM;
   j2 := j mod DIM; j1 := (j-j2)/DIM;
   if i1 = j1 or i2 = j2 then
      return true;
   if i1/3 = j1/3 and i2/3 = j2/3 then
      return true;
   return false;

function korrekteTeillösung(i : Integer) : Boolean
   for j := 0 to i-1 do
      if aufteilung[j] = aufteilung[i] and nachbar(i, j) then
         return false;
   for j := 0 to DIM*DIM-1 do
      if vorgabe[j] = aufteilung[i] and nachbar(i, j) then
         return false;
   return true;
```

Gibt der Aufruf von `partition(-1)` den Wert **`false`** zurück, dann hat das Sudoku keine Lösung. Andernfalls steht in dem Feld `aufteilung` die zuerst gefundene Lösung.

25. Die kantenchromatischen Zahlen der vier Graphen aus Aufgabe 1 sind von links nach rechts 3, 3, 3 und 4. Für die Graphen I_n gilt: $\chi'(I_{2n+1}) = 5$ und $\chi'(I_{2n}) = 4$. Für die

Graphen R_n gilt: $\chi'(R_n) = n$.

26. Das Problem kann als Kantenfärbungsproblem formuliert werden. Jede Mannschaft entspricht einer Ecke in einem vollständigen Graphen. Die Anzahl der Farben in einer minimalen Kantenfärbung entspricht der Anzahl der Tage des Wettbewerbs. Die Kanten mit gleicher Farbe entsprechen den Partien eines Spieltages. Um die minimale Dauer des Wettbewerbs zu bestimmen, muss $\chi'(K_n)$ bestimmt werden.



Zunächst wird gezeigt, dass $\chi'(K_{2i}) = 2i-1$ ist. Dazu werden die Ecken von K_{2i} mit den Zahlen $\{0,\ldots,2i-1\}$ nummeriert und eine graphische Darstellung von K_{2i} betrachtet, bei der die Ecken $\{0,\ldots,2i-2\}$ gleichmäßig auf einem Kreis angeordnet sind und die Ecke $2i-1$ im Mittelpunkt liegt. Folgende Kanten können mit Farbe 1 gefärbt werden: $(2i-1,0),(1,2i-2),(2,2i-3),\ldots,(i-1,i)$. In der graphischen Darstellung sieht man, dass alle Kanten bis auf die erste parallel sind und somit keine gemeinsame Ecke haben. Eine neue Farbklasse bekommt man, indem man zu jeder Ecke jeder Kante, ausgenommen der ersten Ecke der ersten Kante, 1 modulo $2i-1$ addiert. Die neue Farbklasse besteht aus den Kanten

$$(2i-1,1),(2,0),(3,2i-2),\ldots,(i,i+1)$$

und ist durch fette Kanten dargestellt. Man erhält diese Kanten, indem man die Kanten der ersten Farbklasse um den Mittelpunkt um den Winkel $2\pi/(2i-1)$ dreht. Auf diese Art erhält man $2i-1$ Farbklassen mit je i Kanten. Da alle Kanten gefärbt sind und alle Farbklassen die maximale Anzahl von Kanten enthalten, ist $\chi'(K_{2i}) = 2i-1$.

Es bleibt noch $\chi'(K_{2i+1})$ zu bestimmen. Eine Farbklasse in K_{2i+1} kann maximal i Kanten enthalten. Somit ist $\chi'(K_{2i+1}) \geq 2i+1$. Da K_{2i+1} ein Untergraph von K_{2i+2} ist, induziert eine Kantenfärbung von K_{2i+2} mit $2i+1$ Farben eine Kantenfärbung mit $2i+1$ Farben auf K_{2i+1}. Somit ist $\chi'(K_{2i+1}) = 2i+1$.

27. Aus den $n!$ möglichen Reihenfolgen sind die zu bestimmen, die zu einer 2-Färbung führen. Für $n \leq 4$ wird immer eine 2-Färbung erzeugt. Sei nun $n \geq 6$. Bezeichne die Ecken mit ungeraden Nummern mit E_1 und die mit geraden Nummern mit E_2. Sind die ersten beiden Ecken aus E_1, so bekommen diese die Farbe 1. Daraufhin kann keine der Ecken aus E_2 die Farbe 1 bekommen. Somit wird in diesem Fall eine 2-Färbung erzeugt. Die gleiche Aussage gilt für E_2. Dies sind insgesamt $n(n-2)(n-2)!/2$ verschiedene Reihenfolgen. Sind die ersten beiden Ecken durch eine Kante verbunden und die dritte Ecke entweder aus der gleichen Menge wie die erste Ecke oder mit der ersten Ecke verbunden, so wird ebenfalls eine 2-Färbung erzeugt. Dies sind noch einmal $n(n-2)(n-3)(n-3)!/2$ verschiedene Reihenfolgen. In allen anderen Fällen bekommen eine Ecke aus E_1 und eine Ecke aus E_2 die gleiche Farbe und dadurch werden mindestens 3 Farben vergeben.

Somit ist die Wahrscheinlichkeit, dass der Greedy-Algorithmus eine 2-Färbung erzeugt, gleich

$$\frac{n(n-2)(n-2)! + n(n-2)(n-3)(n-3)!}{2n!} = \frac{2n-5}{2n-2}.$$

Die Wahrscheinlichkeit, dass der Algorithmus für diesen Graphen eine optimale Färbung erzielt, strebt also mit wachsendem n gegen 1. Für $n = 20$ liegt diese Wahrscheinlichkeit schon über 92 %. Mit Wahrscheinlichkeit $2^{n/2}(n/2)!/n!$ erzeugt der Algorithmus eine Färbung mit $n/2$ Farben. Für $n = 10$ liegt dieser Wert unter 0,11 %.

28. Die folgende Tabelle zeigt die erzeugten Färbungen. Es ist $\chi = 3$.

Ecke	e_1	e_2	e_3	e_4	e_5	e_6
Greedy-Algorithmus	1	2	1	3	2	4
Johnson Algorithmus	1	1	2	2	3	4
Optimale Färbung	1	2	3	3	2	1

29. Mittels vollständiger Induktion nach i wird gezeigt, dass Ecken aus $F_{\pi(i)}$ eine Farbe mit der Nummer i oder kleiner bekommen. Hieraus folgt dann direkt die Behauptung. Die Ecken aus $F_{\pi(i)}$ haben bezüglich f alle die gleiche Farbe, d. h., sie sind nicht benachbart. Somit bekommen die Ecken aus $F_{\pi(1)}$ alle die Farbe 1 zugeordnet. Angenommen eine Ecke e aus $F_{\pi(i)}$ bekommt die Farbe $i+1$ zugeordnet. Somit muss e zu einer Ecke e' mit Farbe i benachbart sein. Nach Induktionsvoraussetzung kann e' nicht in $F_{\pi(1)}, \ldots, F_{\pi(i-1)}$ liegen. Also liegen e und e' in $F_{\pi(i)}$. Dies bedeutet aber, dass e und e' nicht benachbart sind. Dieser Widerspruch zeigt die Behauptung.

Wendet man den Greedy-Algorithmus in der angegebenen Reihenfolge auf die Ecken des Graphen aus Aufgabe 28 an, so erhält man eine Färbung mit vier Farben. Ordnet man die Ecken nach absteigenden Farbklassen um, d. h., man betrachtet die Ecken in der Reihenfolge $e_6, e_4, e_5, e_2, e_3, e_1$, dann erhält man hingegen eine Färbung mit drei Farben.

Es sei G ein bipartiter Graph mit Eckenmenge $E = \{a_1, \ldots, a_n\} \cup \{b_1, \ldots, b_n\}$ und Kantenmenge $K = \{(a_i, b_j) \mid 1 \leq i, j \leq n,\ i \neq j\}$. Im ungünstigsten Fall ordnet der Greedy-Algorithmus für jedes i den Ecken a_i und b_i die gleiche Farbe zu. Dann vergibt der Algorithmus n Farben, obwohl 2 ausreichen. In diesem Fall wird für jede Permutation der Farbklassen ebenfalls eine Färbung mit n Farben erzeugt.

30. (*a*) Der Greedy-Algorithmus betrachtet bei der Bestimmung der Farbe einer Ecke e die schon gefärbten Nachbarn und wählt dann die kleinste Farbnummer aus, welche noch nicht für diese Nachbarn verwendet wurde. Zur Färbung der i-ten Ecke e_i werden $\min\{i-1,\ g(e_i)\}$ Nachbarn betrachtet, d. h., die Farbnummer von e_i ist maximal $\min\{i,\ g(e_i)+1\}$. Eine Obergrenze für die Anzahl der durch den Greedy-Algorithmus vergebenen Farben ist:

$$\max\{\min\{i,\ g(e_i)+1\} \mid i = 1, \ldots, n\}.$$

(*b*) Die Ecken $\{e_i \mid g(e_i) < k\} \cup \{e_1, \ldots, e_k\}$ liegen innerhalb der ersten k Farbklassen. Es gilt also, diese Menge für festes k möglichst groß zu machen. Dies erreicht man, indem Ecken mit kleinem Eckengrad über die erste Teilmenge und Ecken mit großem Eckengrad über die zweite Teilmenge abgedeckt werden. Hierzu betrachtet man beispielsweise die Ecken in der Reihenfolge absteigender Eckengrade.

(*c*) Alle Ecken von H_i haben den Grad i. Betrachtet man die Ecken von H_i in der Reihenfolge absteigender Eckennummern (d. h. $2i+2, 2i+1, \ldots, 1$), so vergibt der Greedy-Algorithmus $i + 1$ Farben.

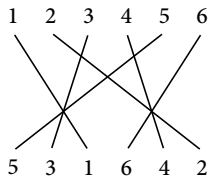
Kapitel 7 – Perfekte Graphen

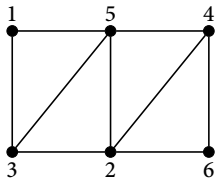
1. Enthält jede Clique von G weniger als $\sqrt{n}$ Ecken, so folgt aus dem ersten Lemma aus Abschnitt 7.4.2 auf Seite 205, dass $\chi(G) < \sqrt{n}$ ist. Somit gilt $n/\chi(G) > \sqrt{n}$. Aus dem gleichen Lemma folgt, dass G eine unabhängige Menge der Größe $\sqrt{n}$ hat. Somit hat $\overline{G}$ eine Clique der Größe $\sqrt{n}$.

2. Orientiert man jede Kante des Permutationsgraphen in Richtung der höheren Eckennummern, so entsteht eine kreisfreie transitive Orientierung. Die chromatische Zahl des Permutationsgraphen ist $h + 1$, wobei h die maximale Höhe einer Ecke ist. Aus Aufgabe 22 aus Kapitel 6 folgt nun, dass der Greedy-Algorithmus für Permutationsgraphen eine minimale Färbung erzeugt.

3. Definiere eine Permutation ρ durch $\rho(i) = \pi(n + 1 - i)$ für $i = 1, \ldots, n$. Es gilt $\rho^{-1}(i) = n + 1 - \pi^{-1}(i)$. Ferner ist $\pi^{-1}(i) < \pi^{-1}(j)$ genau dann, wenn $\rho^{-1}(i) > \rho^{-1}(j)$ ist. Somit ist G_ρ das Komplement von G_π.

4. Im Folgenden ist das Permutationsdiagramm, der Graph und die minimale Färbung dargestellt.





Farbe:

1	2	3	4	5	6
2	2	1	1	0	0

5. Eine Ecke j ist zu $\pi(1)$ benachbart, wenn π^{-1} die Reihenfolge von j und $\pi(1)$ vertauscht. Für $j \neq \pi(1)$ gilt $\pi^{-1}(j) > \pi^{-1}(\pi(1)) = 1$. Somit sind die Ecken $j = 1, \ldots, \pi(1) - 1$ zu $\pi(1)$ benachbart und es gilt $g(\pi(1)) = \pi(1) - 1$. Eine Ecke j ist zu $\pi(n)$ benachbart, wenn π^{-1} die Reihenfolge von j und $\pi(n)$ vertauscht. Für $j \neq \pi(n)$ gilt $\pi^{-1}(j) < \pi^{-1}(\pi(n)) = n$. Somit sind die Ecken $j = n, \ldots, \pi(1) + 1$ zu $\pi(n)$ benachbart und es gilt $g(\pi(n)) = n - \pi(n)$.

6. Es sei $I = \{I_1, \ldots, I_n\}$ eine Menge von Intervallen der Form $I_i = [a_i, b_i]$ und G der zugehörige Intervallgraph. Auf der Menge I kann eine partielle Ordnung eingeführt werden: $I_i < I_j$ genau dann, wenn $I_i \cap I_j = \emptyset$ und $b_i < a_j$ gilt. Das Komplement $\overline{G}$ hat die Kantenmenge $K = \{(I_i, I_j) \mid I_i < I_j \text{ oder } I_i > I_j\}$. Wird jede Kante in K in Richtung des größeren Intervalles ausgerichtet, so erhält man eine transitive Orientierung von $\overline{G}$.

7. Die Intervalle, welche zu einer Farbklasse einer Färbung des Schnittgraphen gehören, überlappen sich paarweise nicht. Somit kann ein Arbeiter diese Arbeitsaufträge ausführen. Umgekehrt induziert eine Zuordnung von Arbeitern zu den Aufträgen eine Färbung von G. Somit ist $\chi(G)$ die minimale Anzahl von Arbeitern zur Ausführung der Aufträge. Auf der Menge der Intervalle wird eine partielle Ordnung definiert: $I_i < I_j$, falls $b_i < a_j$. Ohne Einschränkung der Allgemeinheit kann man annehmen, dass die Intervalle in der Reihenfolge $I_1, I_2, \ldots$ gefärbt werden. Es seien $\{I_1, \ldots, I_s\}$ die mit Farbe 1 gefärbten Intervalle und G' der von den restlichen Intervallen induzierte Graph. Im Folgenden wird gezeigt, dass $\omega(G') < \omega(G)$ gilt. Daraus kann gefolgert werden, dass der Algorithmus

maximal $\omega(G)$ Farben vergibt. Wegen $\omega(G) \leq \chi(G)$ bestimmt der Algorithmus somit eine minimale Färbung. Angenommen es gilt $\omega(G') = \omega(G)$. Dann gibt es in G' eine Clique C mit $\omega(G)$ Intervallen. Es sei I_C der Schnitt aller Intervalle aus C. Es ist $I_C \neq \emptyset$. Angenommen es gilt $I_i \cap I_C = \emptyset$ für alle $i = 1, \ldots, s$. Sei j maximal mit $I_j < I_C$ (man beachte, dass $I_C < I_1$ nicht gelten kann). Somit gibt es in C ein Intervall I mit $I_j < I$ und es ist $I_C < I_{j+1}$. Dies steht im Widerspruch zur Wahl von I_j. Also gibt es ein Intervall $I \in \{I_1, \ldots, I_s\}$ mit $I \cap I_C \neq \emptyset$. Da C eine maximale Clique ist, muss $I \in C$ sein. Somit liegt C nicht in G'. Dieser Widerspruch zeigt $\omega(G') < \omega(G)$.

Eine Implementierung mit Aufwand $O(n \log n)$ erfolgt analog zur Lösung des gewichteten Intervallplanungsproblems.

Kapitel 8 – Flüsse in Netzwerken

1. Es sei f ein maximaler Fluss auf G und f' ein maximaler Fluss auf G'. Da f' ein zulässiger Fluss für G ist, gilt: $|f| \geq |f'|$. Es sei $(X,\overline{X})$ ein q-s-Schnitt von G' mit $|f'| = \kappa_{G'}(X,\overline{X})$. Da keine Kante aus H in dem Schnitt $(X,\overline{X})$ liegt, gilt $|f'| = \kappa_G(X,\overline{X})$. Analog zu dem Beweis des Lemmas aus Abschnitt 8.2 auf Seite 222 folgt nun:

$$|f| = f(X,\overline{X}) - f(\overline{X},X) \leq \kappa_G(X,\overline{X}) = |f'|.$$

   Man beachte hierzu die in Abschnitt 8.1 angegebene Definition von $|f|$. Hieraus folgt $|f'| = |f|$.

2. Es sei X die Menge der vier Ecken auf der linken Seite des Netzwerkes. Dann gilt $\kappa(X,\overline{X}) = 1 + \lambda + \lambda^2 = 2$. Somit ist der Fluss, welcher die drei mittleren waagrechten Kanten sättigt, maximal und hat den Wert 2. Wählt man die Erweiterungswege W_i wie in dem Hinweis angegeben, so ist für $i > 0$ jeweils eine der drei mittleren Kanten eine Rückwärtskante und die anderen beiden sind Vorwärtskanten. Die Wahl der Rückwärtskante ändert sich dabei zyklisch: jeweils die darüberliegende Kante ist im nächsten Erweiterungsweg die Rückwärtskante. Unter Ausnutzung der im Hinweis angegebenen Beziehung $\lambda^{i+2} = \lambda^i - \lambda^{i+1}$ folgt, dass die entstehenden Flüsse $|f_i|$ folgende Werte haben:

$$|f_i| = 1 + \sum_{j=2}^{i+1} \lambda^j = -\lambda + \sum_{j=0}^{i+1} \lambda^j < -\lambda + \lambda + 2 = 2.$$

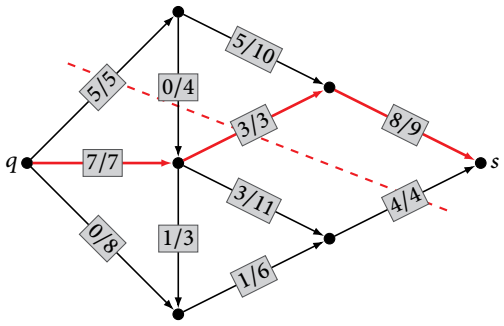
   Ferner gilt $\lim\limits_{i\to\infty} |f_i| = 2$. Fügt man noch eine zusätzliche Kante von q nach s mit Kapazität 1 ein, so ist der maximale Fluss dieses Netzwerkes gleich 3. Somit konvergiert die Folge der Flüsse f_i noch nicht einmal gegen den maximalen Fluss.

3. (*a*) Der Wert von f ist 11.

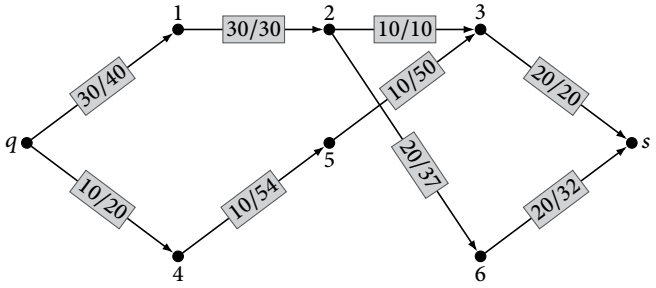
   (*b*) Die drei rot markierten Kanten bilden einen Erweiterungsweg für f mit $f_\Delta = 1$. Der erhöhte Fluss hat den Wert 12 und ist unten dargestellt.

   (*c*) Es sei X die Menge der Ecken unterhalb der rot gestrichelten Linie. Dann ist $\kappa(X,\overline{X}) = 12$ und somit ist $\kappa(X,\overline{X})$ ein Schnitt mit minimaler Kapazität und der erhöhte Fluss ist maximal.

   (*d*) Der unten dargestellte Fluss ist maximal und hat den Wert 12.

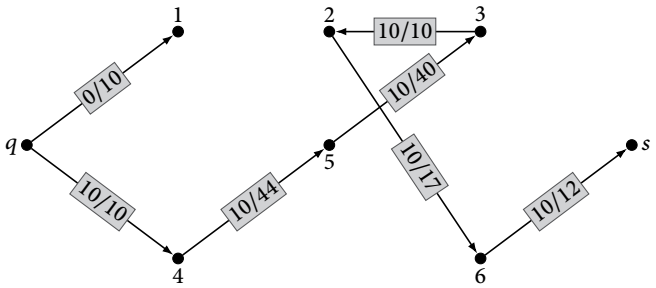


4. Das zu dem trivialen Fluss f_0 gehörende geschichtete Hilfsnetzwerk G'_{f_0} sieht wie folgt aus. Die Bewertungen der Kanten geben den blockierenden Fluss f_1 und die Kapazitäten an.



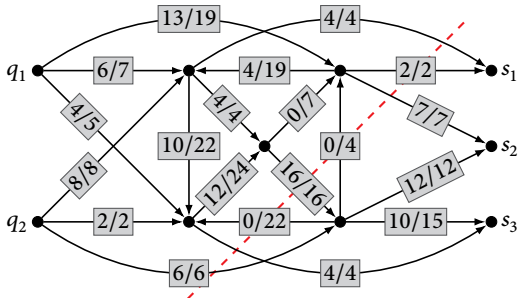
Ecke	q	e_1	e_2	e_3	e_4	e_5	e_6	s
Durchsatz	60	30	30	20	20	50	32	52

Bei der Bestimmung von f_1 hat zunächst Ecke e_3 und danach Ecke e_1 minimalen Durchsatz. Der blockierende Fluss f_1 hat den Wert 40. Das zu f_1 gehörende geschichtete Hilfsnetzwerk G'_{f_1} sieht wie folgt aus.



Hieraus ergibt sich ein blockierender Fluss f_2 mit dem Wert 10 und somit ein maximaler Fluss f mit dem Wert 50. Im Gegensatz zum ursprünglichen Netzwerk wird die Prozedur `blockFluss` nur zweimal aufgerufen.

5. Die zusätzlichen Kanten müssen eine unbeschränkte Kapazität haben. Der unten dargestellte Fluss hat den Wert 39. Der dargestellte Schnitt besitzt eine minimale Kapazität.



6. Ist $X_1 \subseteq X_2$ oder $X_2 \subseteq X_1$, so ist die Aussage wahr. Im Folgenden wird der Fall betrachtet, dass $X_1 \not\subseteq X_2 \not\subseteq X_1$ gilt. Dann sind folgende Mengen nicht leer und paarweise disjunkt:

$$A = X_1 \cap X_2, B = \overline{X}_1 \cap X_2, C = X_1 \cap \overline{X}_2, D = \overline{X}_1 \cap \overline{X}_2.$$

Ferner gilt $q \in A, s \in D, \overline{X_1 \cap X_2} = B \cup C \cup D, D = \overline{X_1 \cup X_2}$ und $X_1 \cup X_2 = A \cup B \cup C$. Da $(X_1, \overline{X}_1)$ und $(X_2, \overline{X}_2)$ minimale Schnitte sind, gilt:

$$\begin{aligned}
\kappa(X_1, \overline{X}_1) &\leq \kappa(A, B \cup C \cup D) \\
\kappa(X_1, \overline{X}_1) &\leq \kappa(A \cup B \cup C, D) \\
\kappa(X_2, \overline{X}_2) &\leq \kappa(A, B \cup C \cup D) \\
\kappa(X_2, \overline{X}_2) &\leq \kappa(A \cup B \cup C, D)
\end{aligned}$$

Wegen $X_1 = A \cup C$ und $\overline{X}_1 = B \cup D$ folgt aus der ersten Ungleichung

$$\kappa(A,B) + \kappa(A,D) + \kappa(C,B) + \kappa(C,D) \leq \kappa(A,B) + \kappa(A,C) + \kappa(A,D)$$

bzw.

$$\kappa(C,B) + \kappa(C,D) \leq \kappa(A,C).$$

Aus den anderen drei Ungleichungen ergibt sich analog:

$$\begin{aligned}
\kappa(A,B) + \kappa(C,B) &\leq \kappa(B,D) \\
\kappa(B,C) + \kappa(B,D) &\leq \kappa(A,B) \\
\kappa(A,C) + \kappa(B,C) &\leq \kappa(C,D)
\end{aligned}$$

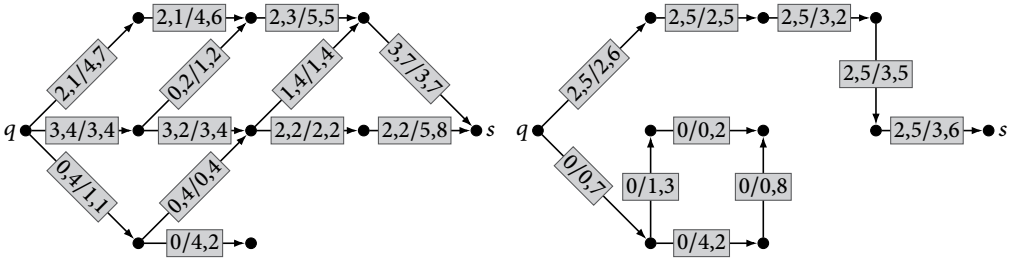
Addiert man die letzten vier Ungleichungen, so folgt $\kappa(C,B) = \kappa(B,C) = 0$, da alle Kapazitäten nicht negativ sind. Setzt man dies in die letzten vier Ungleichungen ein, so ergibt sich $\kappa(A,B) = \kappa(B,D)$ und $\kappa(A,C) = \kappa(C,D)$. Schließlich gilt:

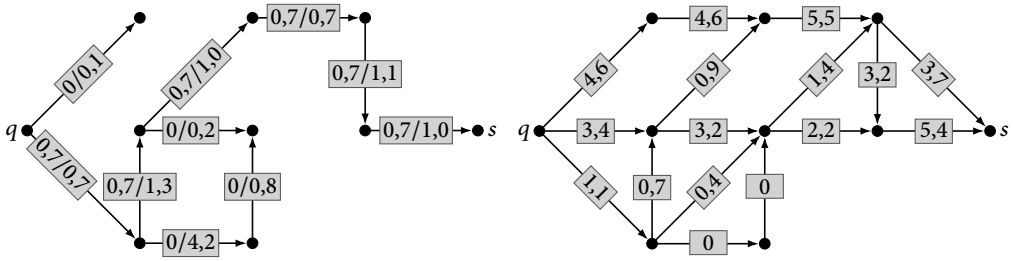
$$\begin{aligned}
\kappa(X_1, \overline{X}_1) &= \kappa(A,B) + \kappa(A,D) + \kappa(C,D) \\
&= \kappa(A,B) + \kappa(A,D) + \kappa(A,C) \\
&= \kappa(A, B \cup D \cup C) \\
&= \kappa(X_1 \cap X_2, \overline{X_1 \cap X_2})
\end{aligned}$$

und

$$\begin{aligned}
\kappa(X_1, \overline{X}_1) &= \kappa(A,B) + \kappa(A,D) + \kappa(C,D) \\
&= \kappa(B,D) + \kappa(A,D) + \kappa(C,D) \\
&= \kappa(B \cup A \cup C, D) \\
&= \kappa(X_1 \cup X_2, \overline{X_1 \cup X_2})
\end{aligned}$$

7. Im Folgenden sind die drei konstruierten geschichteten Hilfsnetzwerke und die dazugehörenden blockierenden Flüsse dargestellt. Die letzte Abbildung zeigt den maximalen Fluss mit Wert 9,1.





8. Es sei f ein maximaler Fluss und W ein Weg von q nach s mit $f(k) > 0$ für alle Kanten k von W. Ferner sei k_0 eine Kante von W mit $f(k_0) \leq f(k)$ für alle Kanten k von W. Man entferne k_0 und setze $f'(k) = f(k) - f(k_0)$ für alle k auf W und ansonsten sei $f'(k) = f(k)$. Dieses Verfahren kann maximal m mal wiederholt werden. Die so entstehenden Wege bilden eine Folge von Erweiterungswegen, die zu einem maximalen Fluss führen. Leider ergibt sich hieraus kein neuer Algorithmus zur Bestimmung eines maximalen Flusses, da ein solcher Fluss schon für die Bestimmung der Wege benötigt wird.

9. Die Prozedur `blockFluss` findet unabhängig vom Wert von b einen blockierenden Fluss mit Wert 4. Dieser verwendet die beiden oberen Kanten. Ist $b = 0$, so ist der blockierende Fluss maximal. Ein maximaler Fluss hat den Wert $\min\{4 + b,\ 10\}$.

10. Enthält ein Netzwerk einen geschlossenen Weg W, so dass $|f| < \kappa(k)$ für alle Kanten k von W gilt, so gibt es einen maximalen Fluss f' und eine Kante k auf W mit $f'(k) > |f'| = |f|$.

11. Der Algorithmus konstruiere die Flüsse $f_0, f_1, \ldots, f_k$, wobei f_0 der triviale und f_k ein maximaler Fluss ist. Setze $\Delta_i = |f_{i+1}| - |f_i|$. Nach Aufgabe 11 kann mit maximal m Erweiterungswegen ein maximaler Fluss gefunden werde (mit f_i startend). Somit gilt $\Delta_i \geq (|f_{max}| - |f_i|)/m$. Hieraus folgt:

$$|f_{max}| - |f_i| \leq m\Delta_i = m(|f_{i+1}| - |f_i|) = m(|f_{i+1}| - |f_{max}|) + m(|f_{max}| - |f_i|).$$

Somit ergibt sich $|f_{max}| - |f_{i+1}| \leq (|f_{max}| - |f_i|)(1 - 1/m)$. Da f_0 der triviale Fluss ist, kann mittels vollständiger Induktion gezeigt werden, dass folgende Ungleichung gilt:

$$|f_{max}| - |f_i| \leq |f_{max}|\,(1 - 1/m)^i.$$

Da alle Kapazitäten ganzzahlig sind, folgt:

$$1 \leq |f_{max}| - |f_{k-1}| \leq |f_{max}|\,(1 - 1/m)^{k-1}.$$

Unter Verwendung der Ungleichung $m(\log m - \log(m-1)) \geq 1$ zeigt man nun $k - 1 \leq m \log |f_{max}|$. Hieraus folgt die Aussage.

12. Es sei f ein maximaler Fluss des Netzwerkes.

    (a) Wird die Kapazität jeder Kante um c erhöht, so erhöht sich auch der Wert eines maximalen Flusses um mindestens c. Dazu wird der Wert von f auf den Kanten eines Weges W von q nach s jeweils um c erhöht. Gibt es noch andere Wege W' von q nach s, welche mit W keine gemeinsamen Kanten haben, so kann der Wert von f noch weiter erhöht werden.

(*b*) Wird die Kapazität jeder Kante um den Faktor c erhöht, so setze man $f'(k) = cf(k)$ für jede Kante k des Netzwerkes. Dann ist f' ein maximaler Fluss in dem neuen Netzwerk und es gilt $|f'| = c\,|f|$.

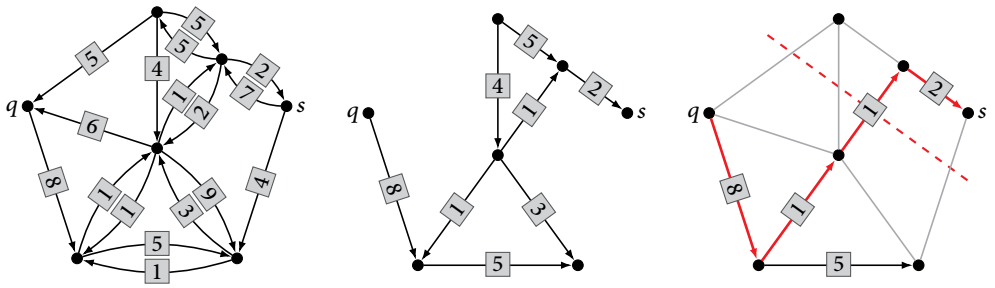
13. Die Kapazitäten der Kanten seien z_i/n_i für $i = 1, \ldots, m$, wobei z_i und n_i positive ganze Zahlen sind. Ferner sei V das kleinste gemeinsame Vielfache der Nenner $n_1, \ldots, n_m$. Dann ist $f_\Delta \geq 1/V$ für jeden Erweiterungsweg jedes Flusses f. Ist f_{max} ein maximaler Fluss, so endet der in Abschnitt 8.2 beschriebene Algorithmus spätestens nach $|f_{max}|V$ Schritten.

14. Der Wert eines Schnittes ist in beiden Netzwerken gleich. Die Behauptung folgt aus dem Satz von Ford-Fulkerson auf Seite 225.

15. (*a*) Im Folgenden ist zunächst der Graph G_f dargestellt. Daneben der Graph, welcher von den Vorwärtskanten von G_f induziert wird. Da es in diesem Graph keinen Weg von q nach s gibt, ist f ein blockierender Fluss.

(*b*) Ganz rechts ist der Graph G'_f abgebildet.

(*c*) Die in G'_f rot gezeichneten Kanten bilden einen blockierenden Fluss h mit Wert 1. Der neue Fluss $f + h$ hat den Wert 12 und ist maximal. Dazu betrachte man den in G'_f durch die rot gestrichelte Linie dargestellten Schnitt $(X, \overline{X})$. Es gilt $\kappa(X, \overline{X}) = 12$.



16. Da f_1 und f_2 Flüsse auf G sind, gilt $f(k) \geq 0$ für jede Kante k von G und die Flusserhaltungsbedingung ist für f erfüllt. Somit ist f genau dann ein Fluss von G, wenn $f_1(k) + f_2(k) \leq \kappa(k)$ für jede Kante k von G gilt.

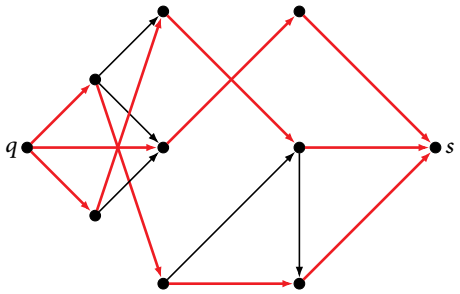
17. Wird die Kapazität einer Kante um 1 erhöht, so erhöht sich der maximale Fluss höchstens um 1. Da die Kapazitäten ganzzahlig sind, findet der Algorithmus von Edmonds und Karp in einem Schritt (mit f startend) einen maximalen Fluss. Somit ist der Aufwand $O(n + m)$.

Wird die Kapazität einer Kante um 1 erniedrigt, so erniedrigt sich der maximale Fluss höchstens um 1. Ist $f(k) < \kappa(k)$, so folgt aus der Ganzzahligkeit der Kapazitäten, dass f auch in dem neuen Netzwerk ein maximaler Fluss ist. Ist $f(k) = \kappa(k)$, so findet man mit Aufwand $O(n+m)$ einen Weg W von q nach s, welcher die Kante k enthält und der Fluss durch jede Kante von W positiv ist. Bilde einen neuen Fluss f' durch $f'(k) = f(k)-1$ für alle Kanten k auf W und $f'(k) = f(k)$ für alle Kanten, welche nicht auf W liegen. Dann ist f' ein zulässiger Fluss auf dem neuen Netzwerk mit $|f'| = |f| - 1$. Der Algorithmus von Edmonds und Karp findet in einem Schritt (mit f' startend) einen maximalen Fluss. Somit ist der Aufwand $O(n + m)$.

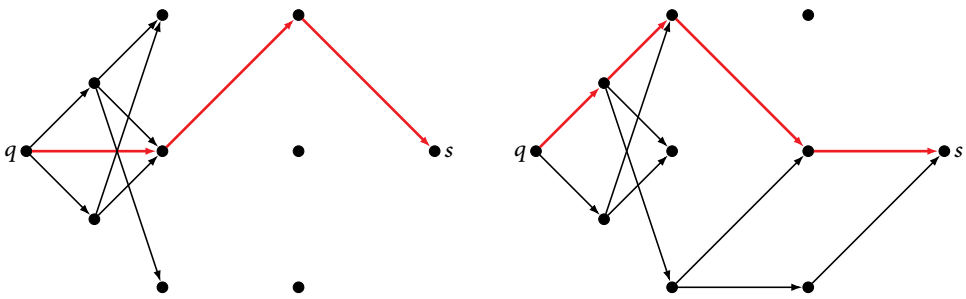
18. Die Prozeduren `erweitereVorwärts` und `erweitereRückwärts` können nicht direkt verwendet werden, sondern müssen noch angepasst werden. Das folgende Programm ist eine Variante der Prozedur `erweitereRückwärts`. Man beachte, dass am Ende der **`while`**-Anweisung die Flusserhaltungsbedingung für die entfernte Ecke e_i erfüllt ist (nur für $e_i \neq e$). Die Prozedur `erweitereVorwärts` muss entsprechend abgeändert werden.

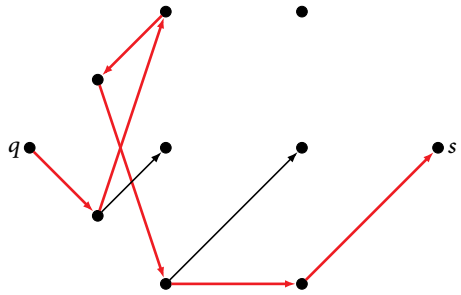
```
procedure erweitereRückwärts(var G : Netzwerk, e : Ecke, var f : Fluss)
   var durchfluss : Array[1..n] von Real;
   var W : Warteschlange von Ecke;
   var m : Real;
   var ei, ej : Ecke;

   Initialisiere durchfluss mit 0;
   durchfluss[G.index(e)] := fe;
   W.einfügen(e);
   repeat
      ei := W.entfernen();
      while durchfluss[i] ≠ 0 do
         Sei k = (ei, ej) eine Kante mit f(k) > 0;
         m := min(f(k), durchfluss[i]);
         f[i, j] := f[i, j] - m;
         if ej ≠ s then
            W.einfügen(ej);
         durchfluss[j] := durchfluss[j] + m;
         durchfluss[i] := durchfluss[i] - m;
   until W = ∅;
```

19. Die rot dargestellten Kanten bilden einen maximalen Fluss mit Wert 3.



Hierzu werden die im Folgenden dargestellten drei blockierenden Flüsse gebildet. Dargestellt sind jeweils die geschichteten Hilfsnetzwerke und die blockierenden Flüsse (rote Kanten).





20. Wegen $\alpha \in [0,1]$ gilt für alle Kanten k:

$$0 \leq f_\alpha(k) = \alpha f_1(k) + (1-\alpha) f_2(k) \leq \alpha\kappa(k) + (1-\alpha)\kappa(k) = \kappa(k).$$

Für jede Ecke $e \neq q, s$ von G gilt:

$$\begin{aligned}
\sum_{k=(j,e)\in K} f_\alpha(k) &= \sum_{k=(j,e)\in K} \alpha f_1(k) + (1-\alpha) f_2(k) \\
&= \alpha \sum_{k=(j,e)\in K} f_1(k) + (1-\alpha) \sum_{k=(j,e)\in K} f_2(k) \\
&= \alpha \sum_{k=(e,j)\in K} f_1(k) + (1-\alpha) \sum_{k=(e,j)\in K} f_2(k) \\
&= \sum_{k=(e,j)\in K} \alpha f_1(k) + (1-\alpha) f_2(k) \\
&= \sum_{k=(e,j)\in K} f_\alpha(k)
\end{aligned}$$

Es gilt $|f_\alpha| = \alpha |f_1| + (1-\alpha)|f_2|$.

21. Es sei k eine Kante eines minimalen Schnittes von G mit positiver Kapazität. Dann ist die Kapazität eines minimalen Schnittes von $G\backslash\{k\}$ echt kleiner als die eines minimalen Schnittes von G. Die Aussage folgt nun aus dem Satz von Ford und Fulkerson auf Seite 225.

22. Der Beweis der Aussagen erfolgt analog zu den Beweisen in Abschnitt 8.6. Der Beweis für Aussage *(a)* ergibt sich aus dem Beweis des Lemmas aus diesem Abschnitt auf Seite 243 und der für Aussage *(b)* aus dem Beweis des nachfolgenden Satzes auf Seite 244.

Kapitel 9 – Anwendungen von Netzwerkalgorithmen

1. Für jede Kante $k \in K$ wird eine binäre Variable x_k eingeführt, d. h. $x_k \in \{0, 1\}$. Desweiteren wird folgende Nebenbedingung definiert

$$\sum_{e \sim k} x_k \leq 1 \text{ für jede Ecke } e \text{ von } G.$$

   Hierbei bedeutet $e \sim k$ dass Ecke e zur Kante k inzident ist. Die zu maximierende Zielfunktion lautet

$$\sum_{k \in K} x_k.$$

2. In Abschnitt 9.1 wurde gezeigt, dass es eine eineindeutige Beziehung zwischen den Zuordnungen eines bipartiten Graphen G und den binären Flüssen des zugehörigen 0-1-Netzwerkes N_G gibt. Es sei Z eine Zuordnung eines bipartiten Graphen G. Ein Weg W, welcher zwei nicht zugeordnete Ecken verbindet, heißt Erweiterungsweg für Z, falls jede zweite Kante von W in Z liegt. Die Erweiterungswege für Z in G entsprechen eineindeutig den Erweiterungswegen bezüglich f_Z in N_G. Der Begriff des Erweiterungsweges lässt sich auch auf beliebige ungerichtete Graphen übertragen. Es gilt folgender Satz.

   Satz. *Eine Zuordnung Z in einem ungerichteten Graphen G ist genau dann maximal, wenn es in G für Z keinen Erweiterungsweg gibt.*

   Beweis. Es sei Z eine Zuordnung und W ein Erweiterungsweg für Z. Ferner sei Z' die symmetrische Differenz von Z und W (Z' besteht aus den Kanten, die entweder nur in Z oder nur in W liegen, aber nicht in beiden Mengen). Dann ist Z' eine Zuordnung für G und es gilt $|Z'| = |Z| + 1$. Somit kann es für eine maximale Zuordnung keinen Erweiterungsweg geben.

   Es sei nun Z eine Zuordnung, welche nicht maximal ist, und Z' eine maximale Zuordnung. Bezeichne mit G' den von den Kanten aus $Z \cup Z'$ induzierten Untergraphen von G. Für jede Ecke e von G' ist $g(e) \leq 2$. Ist $g(e) = 2$, so ist e zu genau einer Kante aus Z und einer Kante aus Z' inzident. Somit können die Zusammenhangskomponenten von G' in zwei Gruppen aufgeteilt werden: Geschlossene und einfache Wege. Auf diesen Wegen liegen abwechselnd Kanten aus Z und Z'. Da geschlossene Wege jeweils gleichviel Kanten aus Z und Z' enthalten und es in Z' mehr Kanten als in Z gibt, muss es in G' einen Weg geben, welcher zwei bezüglich Z nicht zugeordnete Ecken verbindet. Dies ist ein Erweiterungsweg für Z in G. ∎

3. Mit einem Greedy-Verfahren wird eine nicht erweiterbare Zuordnung Z bestimmt (Aufwand $O(n+m)$). Ist Z noch keine vollständige Zuordnung, so ist $|Z| < n/2$. Dann gibt es Ecken e und e' von G, welche zu keiner Kante aus Z inzident sind. Da Z nicht erweiterbar ist, gibt es für jeden Nachbarn e_i von e oder e' genau eine zu e_i inzidente Kante aus Z. Wegen $|Z| < n/2$ und $g(e) + g(e') \geq n$ gibt es eine Kante $k = (e_i, e_j) \in Z$, so dass (e, e_i) und (e_j, e') Kanten in G sind. Somit ist auch

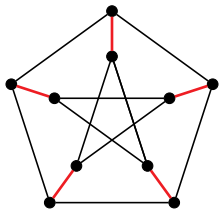
$$Z' = Z \setminus \{k\} \cup \{(e, e_i), (e_j, e')\}$$

eine Zuordnung von G. Dieses Verfahren wiederholt man bis eine vollständige Zuordnung vorliegt. Der Aufwand des folgenden Algorithmus ist $O(n+m)$.

```
var L : Array[1..n] von Ecke;

Bestimme mit einem Greedy – Verfahren nicht erweiterbare Zuordnung Z;
Initialisiere L mit ⊥;
for alle nicht zugeordneten Ecken e, e' do
    foreach e_i in N(e) do
        Sei (e_i, e_j) eine Kante in Z;
        L[j] = e_i;
    Suche Nachbarn e_i von e' mit L[i] ≠ 0;
    Z := Z \ {(e_i, L[i])} ∪ {(e, L[i]), (e_i, e')};
    foreach e_i in N(e) do
        Sei (e_i, e_j) eine Kante in Z;
        L[j] := ⊥;
```

4. Die rot markierten Kanten bilden jeweils eine maximale Zuordnung.

5. Die rot markierten Kanten bilden eine vollständige Zuordnung.



6. Wegen $\alpha(G) = 2$ besteht jede Farbklasse einer Färbung von G aus maximal zwei Ecken. Es sei a die Anzahl der Farbklassen mit zwei Ecken einer minimalen Färbung von G. Dann ist $\chi(G) = a + (n-2a) = n-a$. Die Farbklassen mit zwei Ecken induzieren eine Zuordnung auf $\overline{G}$ mit a Kanten. Somit gilt $a \leq z$ und $\chi(G) \geq n-z$. Jede maximale Zuordnung auf $\overline{G}$ induziert eine Färbung mit $n-z$ Farben auf G. Somit gilt $\chi(G) \leq n-z$.

7. Angenommen es gibt einen Baum B mit zwei verschiedenen vollständigen Zuordnungen M_1 und M_2. Es sei M die Menge der Kanten, welche entweder in M_1 oder in M_2 enthalten sind (aber nicht in beiden Zuordnungen). Da M_1 und M_2 verschieden sind, ist $M \neq \emptyset$. Jede Ecke e von B ist entweder zu zwei Kanten aus M oder zu keiner Kante aus M inzident. Somit muss M einen geschlossenen Weg enthalten. Dieser Widerspruch zeigt die Behauptung.

8. Es sei $T \subseteq E_1$. Ferner sei K_1 die Menge der Kanten von G, welche zu einer Ecke aus T inzident sind, und K_2 die Menge der Kanten von G, welche zu einer Ecke aus $N(T)$ inzident sind. Es gilt $K_1 \subseteq K_2$. Aus der Voraussetzung folgt $|T|\,k \leq |K_1|$ und $|N(T)|\,k \geq |K_2|$. Somit gilt $|N(T)| \geq |T|$ für alle Teilmengen T von E_1. Die Aussage folgt nun aus dem Satz von Hall (siehe Seite 259).

9. Es sei U' bzw. U die Menge der Ecken, welche zu keiner Kante aus Z' bzw. Z inzident sind. Dann sind U' und U unabhängige Mengen mit $|U'| = n-2|Z'|$ und $|U| = n-2|Z|$. Eine maximale unabhängige Menge von G hat maximal $|Z| + |U|$ Ecken. Somit gilt:

$$n - 2|Z'| = |U'| \leq \alpha(G) \leq |Z| + |U| = n - |Z|.$$

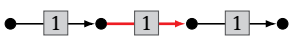
Hieraus folgt $|Z| \leq 2|Z'|$.

10. Nach dem Satz von Hall (siehe Seite 259) hat G genau dann keine Zuordnung Z mit $|Z| = |E_1|$, wenn es eine Teilmenge D von E_1 mit $|D| > |N(D)|$ gibt. Zunächst wird ein maximaler binärer Fluss f auf dem zu G gehörenden 0-1-Netzwerk N_G bestimmt. Gilt $|f| = |E_1|$, so existiert keine Teilmenge D von E_1 mit $|D| > |N(D)|$. Gilt $|f| < |E_1|$, so wird die Menge X aller Ecken aus N_G bestimmt, welche von q über Erweiterungswege bezüglich f erreichbar sind. Es sei $D = X \cap E_1$. Wie im Beweis des Satzes von Hall in Abschnitt 9.1 zeigt man $|f| = |E_1 \backslash X| + |N(D)|$. Wegen $|E_1| = |E_1 \backslash X| + |D|$ folgt nun $|D| > |N(D)|$. Dieser Algorithmus hat eine Laufzeit von $O(\sqrt{n}m)$.

11. Es sei G ein kantenbewerteter, vollständig bipartiter Graph und e eine beliebige Ecke von G. Es sei G' eine Kopie von G, in der die Bewertungen aller zu e inzidenten Kanten um b geändert sind. Dann ist jede vollständige Zuordnung von G auch eine vollständige Zuordnung von G' (und umgekehrt). Die Kosten der beiden Bewertungen unterscheiden sich um b. Es sei $B = b_1 + \ldots b_{|E_1|}$, wobei $b_i = \min\{\text{kosten}(k) \mid k \text{ zu } e_i \in E_1 \text{ inzidente Kante}\}$ ist. Ist die maximale Zuordnung für G_0 auch für G vollständig, so hat man eine vollständige Zuordnung für G mit minimalen Kosten B. Andernfalls bestimme man, wie in Aufgabe 10 beschrieben, eine Teilmenge X von E_1 mit $|N_{G_0}(X)| < |X|$ und verändere den Graphen G_0. Man beachte, dass G_0 mindestens eine zusätzliche Kante besitzt. Ist die maximale Zuordnung für G_0 auch für G vollständig, so hat man eine vollständige Zuordnung für G mit minimalen Kosten $B + (|X| - |N_{G_0}(X)|)b_{min}$. Andernfalls wird der letzte Schritt wiederholt. Da in jedem Schritt G_0 mindestens eine zusätzliche Kante bekommt, sind maximal $|E_1|(|E_1| - 1) < n^2$ Durchgänge notwendig.

12. Es sei G ein kantenbewerteter bipartiter Graph und T die Summe der Bewertungen aller Kanten von G. Durch Hinzufügen neuer Ecken und Kanten mit Bewertung T erhält man einen Graph G' mit den in Aufgabe 11 angegebenen Eigenschaften. Jede Zuordnung von G kann zu einer Zuordnung von G' ergänzt werden. Die Wahl von T bedingt, dass aus einer Zuordnung mit minimalen Kosten für G eine vollständige Zuordnung mit minimalen Kosten für G' entsteht. Es sei Z' eine vollständige Zuordnung von G' mit minimalen Kosten. Entfernt man alle Kanten mit Bewertung T aus Z', so erhält man eine Zuordnung Z von G. Dies ist eine maximale Zuordnung mit minimalen Kosten. Der in Aufgabe 11 angegebene Algorithmus kann also zur Bestimmung maximaler Zuordnungen mit minimalen Kosten in beliebigen kantenbewerteten bipartiten Graphen verwendet werden.

Eine nicht erweiterbare Zuordnung mit minimalen Kosten ist nicht notwendigerweise eine maximale Zuordnung wie der folgende Graph zeigt. Die rot markierte Kante bildet eine nicht erweiterbare Zuordnung mit minimalen Kosten. Diese Zuordnung ist aber nicht maximal.



Der Zusammenhang zwischen maximalen Zuordnungen mit minimalen Kosten in kantenbewerteten, vollständig bipartiten Graphen vom Typ $G_{n,n}$ und maximalen Zuordnungen in beliebigen kantenbewerteten bipartiten Graphen ist nicht auf nicht erweiterbare Zuordnungen übertragbar. Somit kann der angegebene Algorithmus nicht zur Bestimmung nicht erweiterbarer Zuordnungen mit minimalen Kosten verwendet werden.

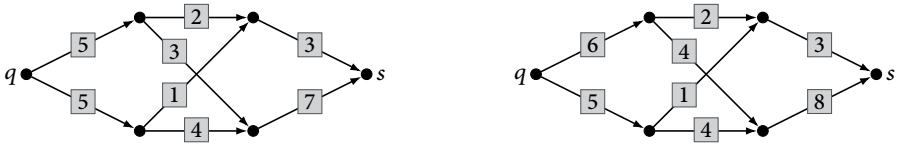
13. Es sei G' das angegebene neue Netzwerk. Jedem q-s-Fluss f auf G entspricht ein s-q-Fluss f' auf G' und umgekehrt. Dazu definiert man

$$f((e,e')) = -f'((e',e))$$

für alle Kanten (e,e'). Man beweist direkt, dass f' genau dann zulässig für G' ist, wenn f zulässig für G ist. Ferner ist f genau dann minimal zulässig, wenn f' maximal zulässig ist. Somit genügt es einen maximalen Fluss für das Netzwerk G' zu konstruieren. Hierzu konstruiert man zunächst wie in Abschnitt 9.2 angegeben einen zulässigen Fluss für G, dieser induziert einen zulässigen Fluss auf G'. Mit Hilfe von Erweiterungswegen kann dieser zu einem zulässigen maximalen Fluss für G' erweitert werden. Dieser entspricht dann einen minimalen zulässigen Fluss auf G.

14. Der Fluss mit Wert 2 auf der Kante (e_1,e_2) und dem Wert 1 auf allen anderen Kanten hat unter den zulässigen Flüssen des Netzwerkes aus Abbildung 9.7 den kleinsten Wert.

15. Links ist ein minimaler Fluss dargestellt und rechts ein maximaler. Die Werte der Flüsse sind 10 und 11.



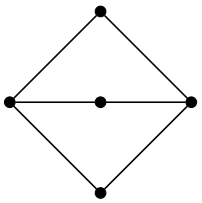
16. Genau dann gibt es einen zulässigen Fluss auf G, wenn jede Kante von G auf einem Weg von q nach s liegt.

17. In Aufgabe 13 wurde aus einem Netzwerk G mit oberen und unteren Kapazitätsgrenzen ein neues Netzwerk G' konstruiert. Der Wert eines minimalen zulässigen q-s-Flusses f_{min} auf G ist gleich dem negativen Wert eines maximal zulässigen s-q-Flusses f_{max} auf G'. Ist $(X,\overline{X})$ ein q-s-Schnitt von G, so ist $(\overline{X},X)$ ein s-q-Schnitt von G' und es gilt $\kappa_G(X,\overline{X}) = -\kappa_{G'}(\overline{X},X)$. Somit gilt:

$$\begin{aligned}
|f_{min}| &= -|f_{max}| \\
&= -\min\left\{\kappa_{G'}(X,\overline{X}) \;\middle|\; (X,\overline{X}) \text{ ist ein } s\text{-}q\text{-Schnitt von } G'\right\} \\
&= -\min\left\{-\kappa_G(\overline{X},X) \;\middle|\; (\overline{X},X) \text{ ist ein } q\text{-}s\text{-Schnitt von } G\right\} \\
&= -\min\left\{\kappa_G(X,\overline{X}) \;\middle|\; (\overline{X},X) \text{ ist ein } q\text{-}s\text{-Schnitt von } G\right\} \\
&= -\min\left\{\kappa_G(\overline{X},X) \;\middle|\; (X,\overline{X}) \text{ ist ein } q\text{-}s\text{-Schnitt von } G\right\}
\end{aligned}$$

18. Der Beweis aus Abschnitt 9.4 für die angegebene Aussage über den Kantenzusammenhang kann nicht auf den Eckenzusammenhang übertragen werden. In einem Graph G kann es eine Ecke e'' geben, welche in jeder trennenden Eckenmenge jedes Paares e, e'

von Ecken mit $Z^e(e,e') = Z^e(G)$ liegt. Dies ist z. B. gegeben, wenn eine Ecke e'' zu jeder anderen Ecke benachbart ist. Betrachten Sie z. B. die Ecken mit Grad 3 in folgendem Graph:



19. Es sei T^k eine minimale trennende Kantenmenge für e, e'. Der Graph $G \backslash T^k$ ist nicht mehr zusammenhängend und e und e' liegen in verschiedenen Zusammenhangskomponenten. Dann liegt e'' entweder in der gleichen Zusammenhangskomponente wie e oder e' oder in einer anderen Zusammenhangskomponente. Auf jeden Fall ist T^k eine trennende Eckenmenge für e, e'' oder e'', e'. Somit gilt $Z^k(e,e') \geq \min\left\{(Z^k(e',e''),\ Z^k(e'',e')\right\}$.

20. Es sei E die Eckenmenge von G und G' der vollständige Graph mit Eckenmenge E. Jede Kante (a,b) von G' wird mit $|f_{ab}|$ bewertet (man beachte, dass $|f_{ab}| = |f_{ba}|$ gilt, dies gilt nicht für unsymmetrische Netzwerke). Es sei B ein maximal aufspannender Baum von G' und e, e' beliebige Ecken aus E. Für die Kanten $k_i = (a_i, b_i)$ eines Weges W von e nach e' in B gilt somit:

$$|f_{ee'}| \leq \min\left\{|f_{a_i b_i}| \mid (a_i, b_i) \in W\right\}.$$

In Abschnitt 9.4 wurde bewiesen, dass $|f_{ab}| = Z^k(a,b)$ für jedes Paar von Ecken a, b aus G gilt. Aus Aufgabe 19 folgt somit $|f_{ee'}| = \min\{|f_{a_i b_i}| \mid (a_i, b_i) \in W\}$. Zu jedem Paar von Ecken a, b von G gibt es also eine Kante (e, e') von B mit $|f_{ab}| = |f_{ee'}|$. Da B $n-1$ und G' $n(n-1)/2$ Kanten hat, muss es mindestens $n/2$ Eckenpaare geben, so dass die Werte der entsprechenden maximalen Flüsse alle gleich sind.

21. Es sei N_G das zu G gehörende 0-1-Netzwerk und f ein binärer Fluss auf N_G. Ferner sei W ein einfacher Erweiterungsweg von N_G bezüglich f und f' der durch W entstandene erhöhte Fluss. Die spezielle Struktur von N_G bewirkt, dass sich auf W Vorwärts- und Rückwärtskanten abwechseln. Des Weiteren beginnt und endet W mit einer Vorwärtskante. Für jede Ecke e sei $f(e)$ bzw. $f'(e)$ der Fluss durch die Ecke e bezüglich f bzw. f'. Es gilt $f'(e) \geq f(e)$ für jede Ecke e von G.

Die Kanten aus Z induzieren einen binären Fluss f_Z auf N_G (vergleichen Sie hierzu den Beweis des Lemmas aus Abschnitt 9.1 auf Seite 257). Es sei f ein binärer maximaler Fluss von N_G, welcher durch eine Folge von Erweiterungswegen aus f_Z entsteht. Dann gilt $f(e) \geq f_Z(e)$ für jede Ecke e von G. Hieraus folgt die zu beweisende Aussage.

22. Der Beweis erfolgt durch vollständige Induktion nach s. Da Q_1 zusammenhängend ist, gilt die Aussage für $s = 1$. Sei nun $s > 1$. Bezeichne für $i = 0, 1$ den von den Ecken $\left\{(i, a_2, \ldots, a_s) \;\middle|\; a_j \in \{0,1\}\right\}$ induzierten Untergraph von Q_s mit Q^i. Diese beiden Untergraphen sind Hypercubes der Dimension $s-1$ und jede Ecke von Q^1 ist in Q_s mit genau einer Ecke in Q^0 verbunden. Sei nun M eine beliebige $(s-1)$-elementige Teilmenge der Eckenmenge E von Q_s. Es muss gezeigt werden, dass der von $E \backslash M$ induzierte Untergraph U von Q_s zusammenhängend ist. Ist $0 < |M \cap Q^0| < s-1$, so sind nach

Induktionsvoraussetzung die von $Q^0 \backslash M$ und $Q^1 \backslash M$ induzierten Untergraphen zusammenhängend und damit ist auch U zusammenhängend. Andernfalls ist M vollständig in Q^0 oder in Q^1 enthalten. Auch in diesem Fall ist U zusammenhängend.

23. Sowohl die Ecken- als auch Kantenzusammenhangszahl ist gleich 3.

24. Für beide Graphen gilt $Z^e(G) = 4$.

25. Es gilt $Z^e(G) = Z^k(G) = 3$.

26. Es sei G ein 2-fach kantenzusammenhängender Graph und X die Menge der Ecken, welche in G' von der Startecke s aus erreichbar sind. Ferner sei Y die Menge der restlichen Ecken. Angenommen Y ist nicht leer. Es sei $e \in Y$ die Ecke mit der kleinsten Tiefensuchenummer. Da G zusammenhängend ist, sind alle Ecken aus Y im Tiefensuchebaum Nachfolger von e. Es sei (e_i, e_j) eine Kante aus G mit $e_i \in X$ und $e_j \in Y$. Im Folgenden wird gezeigt, dass es nur eine solche Kante gibt. Dies steht aber im Widerspruch zu $Z^k(G) \geq 2$. Somit ist Y leer. Da e_j nicht in X liegt, ist (e_j, e_i) eine Kante in G'. Wäre (e_i, e_j) in G eine Rückwärtskante, so wäre `tsNummer[j] < tsNummer[i]`, d. h., e_i wäre Nachfolger von e_j im Tiefensuchebaum. Dann wäre aber e_j von e_i in G' erreichbar und somit auch e_j von s. Somit muss (e_i, e_j) in G eine Baumkante sein. Da e_j im Tiefensuchebaum ein Nachfolger von e ist, muss $e_j = e$ sein und e_i der Vorgänger von e im Tiefensuchebaum sein. Also ist (e_i, e_j) die einzige Kante in G, die Ecken aus X und Y verbindet.

    Sei nun umgekehrt jede Ecke in G' von der Startecke s aus erreichbar. Dann ist G zusammenhängend. Angenommen $Z^k(G) = 1$ und $\{k\}$ ist eine trennende Kantenmenge für G. Dann muss $k = (e_i, e_j)$ in G eine Baumkante sein. Also ist b in G' nicht von s aus erreichbar.

    Der Aufwand ist $O(m)$, da die Tiefensuche zweimal angewendet wird.

27. Es sei R das Rechteck, bei dem die entfernten Felder in diagonal gegenüberliegenden Ecken liegen. Eine Seite von R hat gerade und die andere ungerade Länge. Die Fläche des Schachbrettes außerhalb von R kann in 4 Rechtecke aufgeteilt werden. Mindestens eine Seite jedes Rechtecks hat gerade Länge (eventuell 0). Dieser Teil des Schachbrettes kann somit mit Rechtecken der Größe 1×2 vollständig abgedeckt werden. R ohne die beiden Felder kann in 3 Rechtecke aufgeteilt werden, bei denen jeweils eine Seite gerade Länge hat (eventuell 0). Somit gibt es auch eine Überdeckung für R.

    Die schwarzen und weißen Felder des Schachbrettes induzieren einen bipartiten Graph. Eine vollständige Überdeckung mit Rechtecken entspricht einer vollständigen Zuordnung dieses Graphen. Da zwei diagonal gegenüberliegende Felder die gleiche Farbe haben, besitzt der Restgraph keine vollständige Zuordnung. Somit gibt es auch keine vollständige Überdeckung der Restfläche.

28. Gibt es eine vollständige Zuordnung von G, so gilt $|E_1| = |E_2|$ und eine Eckenüberdeckung muss mindestens $|E_1| = (|E_1| + |E_2|)/2$ Ecken enthalten. Für die umgekehrte Beweisrichtung beachte man, dass sowohl E_1 als auch E_2 Eckenüberdeckungen von G sind. Da beide deshalb mindestens $(|E_1| + |E_2|)/2$ Ecken enthalten, folgt daraus $|E_1| = |E_2|$. Nun folgt aus dem Satz von König-Egerváry (siehe Seite 284), dass G eine vollständige Zuordnung besitzt.

29. Eine Ecke e in einer minimalen Eckenüberdeckung von G überdeckt maximal n Kanten. Hieraus folgt, dass eine minimale Eckenüberdeckung aus mindestens s Ecken bestehen muss. Die Aussage folgt nun aus dem Satz von König-Egerváry (siehe Seite 284).

30. Es sei $(X,\overline{X})$ ein nicht trivialer Schnitt eines c-kantenkritischen Graphen G und T^k die Menge der Kanten mit einer Ecke in X und einer Ecke in $\overline{X}$. Angenommen es gilt $|T^k| < c$. Nach Voraussetzung können die von X und $\overline{X}$ induzierten Untergraphen mit höchstens $c-1$ Farben gefärbt werden. Bezeichne mit $X_1, \ldots, X_{c-1}$ bzw. $\overline{X}_1, \ldots, \overline{X}_{c-1}$ die entsprechenden Farbklassen dieser Graphen (einige dieser Mengen sind eventuell leer). Es sei B der bipartite Graph mit Eckenmenge $\{X_1, \ldots, X_{c-1}\} \cup \{\overline{X}_1, \ldots, \overline{X}_{c-1}\}$. Genau dann gibt es eine Kante zwischen X_i und $\overline{X}_j$, falls es in G keine Kante gibt, welche eine Ecke aus X_i mit einer Ecke aus $\overline{X}_j$ verbindet. Nach Annahme besitzt B mindestens $(c-1)^2 - (c-2) = c^2 - 3c + 3$ Kanten. Da der Grad jeder Ecke von B maximal $c-1$ ist, überdeckt eine Menge von $c-2$ Ecken maximal $(c-2)(c-1) = c^2 - 3c + 2$ Kanten. Somit enthält jede Eckenüberdeckung von B mindestens $c-1$ Ecken. Nach dem Satz von König-Egerváry (siehe Seite 284) besitzt B eine vollständige Zuordnung mit $c-1$ Kanten. Es seien X_i und $\overline{X}_j$ die Ecken einer Kante einer solchen maximalen Zuordnung. Nach Definition von B ist $X_i \cup \overline{X}_j$ somit eine unabhängige Menge. Die Eckenmenge von G kann also in $c-1$ unabhängige Mengen aufgeteilt werden, d. h. $\chi(G) \leq c-1$. Dieser Widerspruch zeigt, dass $|T^k| \geq c-1$ ist, d. h., die Kantenzusammenhangszahl ist mindestens $c-1$.

31. Nach den Ergebnissen aus Abschnitt 6.5 gilt $\delta(G) \leq 5$. Nach dem ersten Satz in Abschnitt 9.4 auf Seite 274 gilt dann $Z^e(G) \leq \delta(G) \leq 5$.

32. Für die Graphen I_n mit $n \geq 5$ gilt $Z^k(I_n) = Z^e(I_n) = 4$.

33. Es seinen e, e' Ecken mit $Z^e(e,e') = 1$ und $\{e_x\}$ eine trennende Eckenmenge für a, b. Der Graph $G\backslash\{e_x\}$ zerfällt in die Zusammenhangskomponenten $Z_1, \ldots, Z_s$ mit $s > 1$. Es sei K_i die Menge der Kanten, welche e_x mit Ecken aus Z_i verbinden. Diese Mengen sind trennende Kantenmengen für e, e'. Da e_x den Grad Δ hat, gibt es ein K_i mit $|K_i| \leq \Delta/2$. Somit gilt $Z^k(G) \leq |K_i| \leq \Delta/2$.

34. Sowohl die Ecken- als auch Kantenzusammenhangszahl ist gleich 2.

35. Das Tiefesucheverfahren legt die Reihenfolge, in der die Nachbarn einer Ecke besucht werden, nicht fest. Es wird folgende Reihenfolge betrachtet: Für jede Ecke wird zunächst der unbesuchte Nachbar betrachtet, welcher über eine Kante aus Z erreichbar ist (falls vorhanden). Es sei B der hierdurch entstehende Tiefensuchebaum und (e,e') eine beliebige Kante aus Z. Ohne Einschränkung der Allgemeinheit kann angenommen werden, dass e vor e' besucht wurde. Somit wurde beim Besuch von e der Nachbar e' zuerst betrachtet. Da zu diesem Zeitpunkt e' noch nicht besucht wurde, wird (e,e') in B eingefügt. Somit sind alle Kanten aus Z auch in B enthalten.

36. Es sei e_i eine Ecke von B, welche zu keiner Kante aus Z inzident ist. Somit wurde e_i im Verlauf des Algorithmus nicht markiert. Beim Verlassen von e_i durch die Tiefensuche muss somit der Vorgänger e_j von e_i schon markiert sein. Da die Tiefensuche in diesem Moment e_j noch nicht verlassen hat, muss es einen Nachfolger e_s von e_j geben, so dass die Kante (e_j, e_s) in Z liegt.

Angenommen es gibt in B einen Erweiterungsweg $W = <e_1, \ldots, e_s>$ bezüglich Z (vergleichen Sie Aufgabe 11). Nach Konstruktion von Z ist $s \geq 4$. Ohne Einschränkung der Allgemeinheit kann angenommen werden, dass e_2 der Vorgänger von e_1 im Tiefensuchebaum ist (sonst ist e_{s-1} ein Vorgänger von e_s). Nach der oben bewiesenen Eigenschaft muss e_3 ein Nachfolger von e_2 sein. Somit ist $<e_s, e_{s-1}, \ldots, e_2>$ ein gerichteter Weg im Tiefensuchebaum. Analog zu oben zeigt man, dass e_{s-2} ein Nachfolger von e_{s-1} ist. Dieser Widerspruch zeigt, dass es bezüglich Z keinen Erweiterungsweg gibt. Somit ist Z eine maximale Zuordnung von B.

37. Es sei $U_{max} \in \mathcal{U}$ mit $f(U_{max}) = \max\{f(U) \mid U \in \mathcal{U}\}$. Wird eine Kante aus U_{max} entfernt, so entsteht eine isolierte Ecke (da die Anzahl der Ecken unverändert bleibt, würde andernfalls $f(U'_{max}) > f(U_{max})$ für den neuen Graphen U'_{max} gelten). Somit enthält U_{max} keinen geschlossenen Weg und U_{max} ist ein Wald. Aus dem gleichen Grund gibt es in U_{max} auch keinen Weg der Länge 3. Daraus folgt, dass jede Zusammenhangskomponente von U_{max} ein Sterngraph ist. Es sei s die Anzahl der Zusammenhangskomponenten von U_{max}. Bezeichnet man mit k die Anzahl der Kanten in U_{max}, so enthält U_{max} genau $k + s$ Ecken und es gilt $f(U_{max}) = (k + s - 1)/k$. Angenommen $s > 1$. Nun wird eine neuer Untergraph U'_{max} von G gebildet. U'_{max} enthält aus jeder Zusammenhangskomponente genau eine Kante. Dann gilt $U'_{max} \in \mathcal{U}$ und $f(U'_{max}) = (2s - 1)/s$. Aus $f(U_{max}) \geq f(U'_{max})$ folgt nun $s \geq k$, d. h., die Kanten in U_{max} bilden eine Zuordnung. Aus der Maximalität von $f(U_{max})$ folgt, dass diese Zuordnung maximal ist. Andernfalls ist $s = 1$ und man sieht direkt, dass G bis auf isolierte Ecken ein Sterngraph ist.

38. (*a*) Jede Ecke muss zu mindestens einer Ecke einer anderen Farbklasse benachbart sein, andernfalls würde es eine zweite Färbung von G geben.

    (*b*) Es seien F_1 und F_2 zwei Farbklassen. Angenommen der von $F_1 \cup F_2$ induzierte Untergraph U ist nicht zusammenhängend. Es seien U_1 und U_2 Zusammenhangskomponenten von U. Da jede Ecke zu mindestens einer Ecke einer anderen Farbklasse benachbart ist, enthalten U_1 und U_2 Ecken unterschiedlicher Farben. Vertauscht man die beiden Farben der Ecken in U_1, so gelangt man zu einer zweiten Färbung von G. Dieser Widerspruch zeigt die Behauptung.

    (*c*) Angenommen es gibt eine Menge M mit $c - 2$ Ecken, so dass der durch das Entfernen dieser Ecken entstehende Graph G' nicht mehr zusammenhängend ist. Dann gibt es zwei Farben f_1 und f_2, mit denen keine der Ecken in M gefärbt ist. Es seinen e_1 bzw. e_2 Ecken, die mit f_1 bzw. f_2 gefärbt sind. Nach Teil *(b)* gibt es zwischen diesen Ecken einen Pfad P, dessen Ecken nur mit f_1 und f_2 gefärbt sind. Somit liegen alle mit f_1 bzw. f_2 gefärbten Ecken in einer einzigen Zusammenhangskomponente G_1 von G'. Ändert man nun die Färbung einer Ecke in $G' \backslash G_1$ in f_1 um, so erhält man eine zweite Färbung von G. Dieser Widerspruch zeigt die Behauptung.

39. Es sei Z eine Zuordnung von G mit t Kanten, Z_1 die Menge der Ecken aus E_1, welche zu Kanten aus Z inzident sind, und $T \subseteq E_1$. Dann ist

$$|E_1| \geq |T \cup Z_1| = |T| + |Z_1| - |T \cap Z_1|.$$

    Wegen $|N(T)| \geq |T \cap Z_1|$ und $|Z_1| = t$ folgt:

$$|N(T)| \geq |T| + t - |E_1|.$$

Sei nun umgekehrt $|N(T)| \geq |T| + t - |E_1|$ für jede Teilmenge T von E_1. Man betrachte das zugehörige Netzwerk N_G und einen maximalen binären Fluss f. Wie in dem Beweis des Satzes von Hall (siehe Seite 259) zeigt man nun $|f| = \kappa(X, \overline{X}) = |E_1 \backslash X| + |N(X \cap E_1)|$. Nach Voraussetzung gilt nun $|f| \geq |E_1 \backslash X| + |X \cap E_1| + t - |E_1| = t$. Sei nun Z die Menge aller Kanten aus G, für die der Fluss f durch die entsprechende Kante in N_G gerade 1 ist. Da jede Ecke aus E_1 in N_G den Eingrad 1 und jede Ecke aus E_2 in N_G den Ausgrad 1 hat, folgt aus der Flusserhaltungsbedingung, dass Z eine Zuordnung mit $|f| \geq t$ Kanten ist.

40. Die Kanten einer Farbklasse einer Kantenfärbung bilden eine Zuordnung von G. Somit enthält jede Farbklasse maximal $|M|$ Kanten und G hat insgesamt maximal $\chi'(G)|M|$ Kanten. Aus der Voraussetzung folgt nun direkt $\chi'(G) > \Delta(G)$.

41. (*a*) $\rho(C_n) = 1$ und $\rho(K_n) = (n-1)/2$ für $n > 2$.

(*b*) Es sei K die Menge der Kanten und E die Menge der Ecken von G. Definiere einen bipartiten Graphen G_r mit Eckenmenge $E_1 \cup E_2$, wobei $E_1 = K$ ist und E_2 aus r Kopien von E besteht. Eine Ecke $(u, v) \in E_1$ ist zu den r Kopien von u und v inzident, d. h., für $x \in E_1$ gilt $g(x) = 2r$. Es sei H ein Untergraph von G mit Eckenmenge E_H und Kantenmenge K_H. Betrachtet man K_H als Teilmenge der Ecken von G_r, dann gilt $|N(K_H)| = r|E_H|$. Somit gilt $|N(K_H)| \geq |K_H|$ genau dann, wenn $|K_H|/|E_H| \leq r$ gilt. Mit Hilfe des Satzes von Hall (siehe Seite 259) folgt nun, dass genau dann $\rho(G) \leq r$ gilt, wenn G_r eine Zuordnung mit $|K|$ Kanten besitzt.

Hieraus ergibt sich sofort ein Algorithmus für das beschriebene Problem: Für einen gegebenen Graphen G und $r > 0$ konstruiere man den bipartiten Graphen G_r und stelle fest, ob dieser Graph eine Zuordnung mit r Kanten besitzt, genau dann gilt auch $\rho(G) \leq r$.

Der Graph G_r hat $m + rn$ Ecken und $2mr$ Kanten und eine maximale Zuordnung von G_r hat maximal m Kanten. Nach den Ergebnissen aus Abschnitt 9.1 kann eine maximale Zuordnung in einem bipartiten Graph mit Aufwand $O(\sqrt{z}m)$ bestimmt werden. Somit ist der Aufwand für das beschriebene Verfahren $O(m^{3/2}r)$. Da $r \leq (n-1)/2$ ist, ist der Aufwand gleich $O(m^{3/2}n)$.

(*c*) Mittels des in Teil (b) entwickelten Algorithmus in Kombination mit binärer Suche kann $\rho(G)$ mit Aufwand $O(\log_2 n(m^{3/2}n))$ bestimmt werden.

42. (*a*) Es sei O eine Orientierung von G mit der angegebenen Eigenschaft. Ist H ein Untergraph von G, dann gilt

$$m_H = \sum_{e \in E_H} g_H^+(e) \leq \sum_{e \in E_H} g_G^+(e) \leq r n_H$$

und somit ist $m_H/n_H \leq r$. Hieraus folgt $\rho(G) \leq r$.

Sei nun $\rho(G) \leq r$. Aus dem Beweis von Aufgabe 41 *(b)* folgt, dass G_r eine Zuordnung Z mit $|Z| = m$ hat. Mittels Z kann eine Orientierung von G mit der gewünschten Eigenschaft konstruiert werden. Ist k eine Kante von G und wird k durch Z der Ecke u zugeordnet, so wird k in Richtung von u weg orientiert. Da u nur r-mal als Ecke in G_r vorkommt, hat die konstruierte Orientierung die geforderte Eigenschaft.

(*b*) Aus Teil *(a)* und der Lösung von Aufgabe 41 *(c)* ergibt sich sofort ein Algorithmus mit Aufwand $O(\log_2 n(m^{3/2}n))$.

43. (*a*) Für jede Ecke e von G muss $g(e) \geq \theta(e)$ gelten, weiterhin muss folgende Gleichung erfüllt sein:

$$\sum_{i=1}^{n} \theta(e) = m.$$

(*b*) Beide Bedingungen sind nicht hinreichend. Hierzu betrachte man den Graphen K_4 und die Funktion $\theta(1) = \theta(2) = 0, \theta(3) = \theta(4) = 3$.

(*c*) Definiere einen bipartiten Graphen G_θ mit Eckenmenge $E_1 \cup E_2$, wobei E_1 gleich der Kantenmenge von G ist und E_2 enthält genau $\theta(e)$ Kopien jeder Ecke e aus G. Eine Ecke $(u, v) \in E_1$ ist zu den $\theta(u)$ Kopien von u und zu den $\theta(v)$ Kopien von v inzident, d. h., für $x = (u, v) \in E_1$ gilt $g(x) = \theta(u) + \theta(v)$. Man zeigt nun leicht, dass G genau dann die gewünschte Orientierung besitzt, wenn G_θ eine vollständige Zuordnung besitzt. Somit kann der in Abschnitt 9.1 vorgestellte Algorithmus verwendet werden.

44. Definiere für jede Kante k eine weitere Kantenbewertung $c'(k) = m\,c(k) + 1$. Es sei T ein bezüglich der Kantenbewertung c' ein minimaler Schnitt und X ein beliebiger Schnitt von G. Dann gilt:

$$\text{kosten}_{c'}(T) \leq \text{kosten}_{c'}(X).$$

Aus der Definition von c' ergibt sich folgende Ungleichung

$$m\,\text{kosten}_c(T) + |T| \leq m\,\text{kosten}_c(X) + |X|$$

Angenommen es gilt $\text{kosten}_c(X) < \text{kosten}_c(T)$. Da alle Kapazitäten ganzzahlig sind, gilt somit auch $\text{kosten}_c(X) + 1 \leq \text{kosten}_c(T)$. Hieraus folgt $m + |T| \leq |X|$, ein Widerspruch da $|X| < m$ gilt. Somit ist T auch bezüglich der Kantenbewertung c ein minimaler Schnitt. Gilt $\text{kosten}_c(X) = \text{kosten}_c(T)$, so folgt aus obiger Ungleichung $|T| \leq |X|$, d. h., T hat unter allen bezüglich der Kantenbewertung c minimalen Schnitten die wenigsten Kanten. Mittels des in Abschnitts 9.5 vorgestellten Algorithmus, angewendet auf die Bewertung c', kann das beschriebene Problem mit Aufwand $O(n\,m \log n)$ gelöst werden.

45. Es sei C eine Clique von G. Dann ist $E \backslash C$ eine Eckenüberdeckung von $\overline{G}$. Ist U eine Eckenüberdeckung von $\overline{G}$, so ist $E \backslash U$ eine Clique von G. Hieraus folgt $\omega(G) = \tau(\overline{G})$.

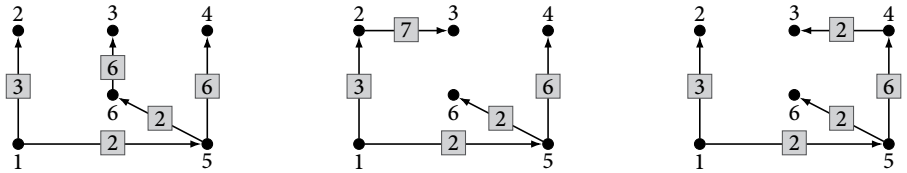
46. Ist eine Ecke e nicht in U enthalten, so müssen alle Nachbarn dieser Ecke in E enthalten sein (sonst sind die zu e inzidenten Kanten nicht überdeckt). Somit gilt $e \in U$, falls $g(e) > |U|$ ist.

47. Da nach Aufgabe 26 aus Kapitel 4 die Blätter eines Tiefensuchebaums eine unabhängige Menge sind, bildet die Menge der inneren Ecken eine Eckenüberdeckung. Mittels des in Aufgabe 36 beschriebenen Algorithmus kann eine maximale Zuordnung M des Baumes bestimmt werden. Man überzeugt sich leicht, dass die erste Ecke, die der Algorithmus besucht, durch M überdeckt wird. Ferner wird auch jede innere Ecke außer der Wurzel durch M überdeckt. Somit gilt $2|M| \geq |U|$. Zur Überdeckung der Kanten aus M werden mindestens $|M|$ Ecken benötigt, d. h. $\tau(G) \geq |M|$. Hieraus folgt die Behauptung.

Kapitel 10 – Kürzeste Wege

1. Es sei S die Menge der von der Startecke s aus erreichbaren Ecken und G_S der von S induzierte Untergraph von G. Nach Voraussetzung hat G_S die Eigenschaft $(*)$. Eine Anwendung des Algorithmus von Moore und Ford auf G verwendet nur Ecken und Kanten aus G_S, d. h., beide Anwendungen sind identisch. Somit arbeitet der Algorithmus auch auf G korrekt.

2. Beispiele für kW-Bäume.



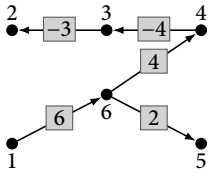
3. Der Graph besitzt keine geschlossenen Wege. Mit dem in Aufgabe 9 beschriebenen Verfahren können die kürzesten Wege in linearer Zeit bestimmt werden.

4. (a) Der kürzeste Weg von e durch eine vorgegebene dritte Ecke e_1 nach e' besteht aus dem kürzesten Weg von e nach e_1 gefolgt von dem kürzesten Weg von e_1 nach e'.

   (b) Für $i,j = 1, \ldots, l$ mit $i \neq j$ bestimme die kürzesten Wege von e nach e_i, e_i nach e_j und von e_i nach e' (insgesamt $l^2 + l$ Wege). Danach bestimme man unter allen $l!$ Reihenfolgen der Ecken $e_1, \ldots, e_l$ den kürzesten der Wege $<e, e_{i_1}, \ldots, e_{i_l} e'>$. Hierfür können die in Kapitel 5 beschriebenen Verfahren wie Branch & Bound oder Dynamische Programmierung verwendet werden.

5. Es sei B ein minimal aufspannender Baum von G, (e,e') eine der $n-1$ Kanten von B und W ein kürzester Weg von e nach e' in G. Wäre die Länge von W kleiner als die Länge von (e,e'), so würde dies zu einem aufspannenden Baum mit geringeren Kosten als B führen. Somit ist (e,e') der kürzeste Weg von e nach e' in G.

6. Die folgende Abbildung zeigt den entstehenden kW-Baum und die kürzesten Entfernungen zur Startecke.



Ecke	e_1	e_2	e_3	e_4	e_5	e_6
Entfernung	0	3	6	10	8	6

7. Die Behauptung ist falsch. Dazu wird ein Graph mit sieben Ecken betrachtet. Der Graph ist die Vereinigung der beiden Graphen C_3 und C_4. Die Kanten der ersten Zusammenhangskomponente tragen die Gewichte 25, 26 und 27 und die Kanten der zweiten Komponente die Gewichte 21, 22, 23 und 24.

8. Die Prozedur `kürzesteWege` aus Abbildung 10.6 muss wie folgt geändert werden.

```
var D : Array[1..n] von Real;
var vorgänger : Array[1..n] von Ecke;
```

```
procedure kürzesteWege(G : B-Graph, s : Ecke)
   var e_i , e_j : Ecke;
   var W : Warteschlange von Ecke;
   Initialisiere D mit ∞ und vorgänger mit ⊥;
   D[G.index(s)] := 0;
   W.einfügen(s);
   while W ≠ ∅ do
      e_i := W.entfernen();
      if not W.enthalten(vorgänger[i]) then
         foreach e_j in N+(e_i) do
            verkürze(e_i, e_j);
```

Es sei e_i die erste Ecke in der Warteschlange und e_j der momentane Vorgänger von e_i im kW-Baum. Dann wurde e_j nach e_i in die Warteschlange eingefügt, d. h., der aktuelle Wert von $D[e_j]$ ist kleiner als zu dem Zeitpunkt, als $D[e_i]$ zuletzt aktualisiert wurde. Somit ist der aktuelle Wert von $D[e_i]$ zu hoch. Spätestens bei der Abarbeitung von e_j wird der Wert von $D[e_i]$ erniedrigt und e_i erneut in die Warteschlange eingefügt. Somit ist es überflüssig, Ecke e_i vor ihrem aktuellen Vorgänger e_j zu bearbeiten. Die Korrektheit dieser Variante des Algorithmus von Moore und Ford ergibt sich aus dieser Beobachtung und der Korrektheit der Originalversion. Diese Variante hat in vielen Fällen eine geringere Laufzeit. Die *worst case* Komplexität bleibt aber unverändert. Dazu betrachte man die Folge von Graphen, welche durch die in Abbildung 10.8 angegebenen Adjazenzmatrizen M_n definiert ist. Der Algorithmus von Moore und Ford durchläuft die **while**-Schleife für diese Graphen $1 + (n-1)n/2$ mal.

9. In einem ersten Schritt wird eine topologische Sortierung des Graphen bestimmt. Die in Abbildung 10.5 dargestellte Prozedur `bellman` muss nur geringfügig abgeändert werden: Die Iteration über die Ecken erfolgt gemäß aufsteigenden Sortierungsnummern beginnend mit der Startecke.

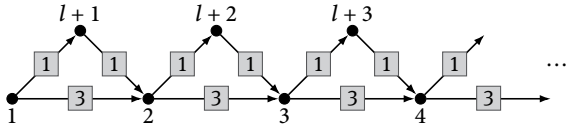
   Für den Korrektheitsbeweis des Algorithmus kann ohne Einschränkung der Allgemeinheit angenommen werden, dass die Ecken gemäß ihrer topologischen Sortierungsnummer nummeriert sind und dass die Startecke s die Sortierungsnummer 1 hat (d. h. $s = e_1$). Die Korrektheit ergibt sich aus folgender Aussage: Nach dem Ende des i-ten Durchlaufs gilt $D[j] = d(s, e_j)$ für $j = 1, \ldots, i+1$. Diese Aussage wird mit Hilfe von vollständiger Induktion nach i bewiesen. Zu jedem Zeitpunkt gilt $D[j] \geq d(s, e_j)$. Da die zweite Ecke nur einen Vorgänger hat, gilt nach dem ersten Durchlauf $D[j] = d(s, e_j)$ für $j = 1, 2$. Sei nun $i > 1$. Nach dem $i-1$-ten Durchgang werden die Werte für $D[1], \ldots, D[i]$ nicht mehr geändert. Es muss also nur gezeigt werden, dass nach dem i-ten Durchgang $D[i+1] = d(s, e_{i+1})$ gilt. Es sei e_j der Vorgänger von e_{i+1} auf einem kürzesten Weg von s nach e_{i+1}. Dann gilt $j < i+1$ bzw. $j \leq i$ und somit war $D[j] = d(s, e_j)$ vor dem j-ten Durchgang. Nach dem j-ten Durchgang gilt:

   $$D[i+1] \leq D[j] + B[j, i+1] = d(s, e_{i+1}).$$

   Wegen $d(s, e_{i+1}) \leq D[i+1]$ gilt also $D[i+1] = d(s, e_{i+1})$ schon nach dem j-ten und somit erst recht nach dem i-ten Durchgang.

10. Es sei n eine ungerade Zahl und $l = (n+1)/2$. Der unten dargestellte kantenbewertete gerichtete Graph mit n Ecken und $3(n-1)/2$ Kanten ist kreisfrei. Bei der Anwendung

des Algorithmus von Moore und Ford auf diesen Graph werden die Nachfolger jeder Ecke in der Reihenfolge aufsteigender Eckennummern betrachtet. Für $j = 1, \ldots, l-1$ gilt: Die Ecken e_j und e_{l+j} werden j-mal in die Warteschlange eingefügt. Somit ist die Laufzeit des Algorithmus für diesen Graph $O(n^2)$.



11. Die folgende Tabelle zeigt die Werte von `F`, `min`, `D` und `vorgänger` für die einzelnen Schritte.

F	min
3	
2, 4	3
2, 1, 5	4
1, 5, 8	2
5, 8	1
8, 6, 7	5
6, 7	8
6	7
∅	6

D							
1	2	3	4	5	6	7	8
∞	∞	0	∞	∞	∞	∞	∞
∞	3	0	2	∞	∞	∞	∞
6	3	0	2	10	∞	∞	∞
4	3	0	2	10	∞	∞	7
4	3	0	2	7	∞	∞	7
4	3	0	2	7	13	12	7
4	3	0	2	7	13	12	7
4	3	0	2	7	13	12	7
4	3	0	2	7	13	12	7

vorgänger							
1	2	3	4	5	6	7	8
⊥	⊥	⊥	⊥	⊥	⊥	⊥	⊥
⊥	3	⊥	3	⊥	⊥	⊥	⊥
4	3	⊥	3	4	⊥	⊥	⊥
2	3	⊥	3	4	⊥	⊥	2
2	3	⊥	3	1	⊥	⊥	2
2	3	⊥	3	1	5	5	2
2	3	⊥	3	1	5	5	2
2	3	⊥	3	1	5	5	2
2	3	⊥	3	1	5	5	2

12. Der folgende Algorithmus basiert auf der Tiefensuche und hat eine Laufzeit von $O(m)$. Eine neue Ecke e_i wird nur dann besucht, wenn Sie nicht schon als besucht markiert ist und wenn die Entfernung im Tiefensuchebaum von der Startecke e mit $d(e, e_i)$ übereinstimmt. Zum Beweis der Korrektheit der Prozedur `kWBaum` genügt es zu zeigen, dass alle Ecken erreicht werden. Es sei e' eine beliebige Ecke von G und W der Weg von der Startecke zu e' in einem beliebigen kW-Baum (da G die Eigenschaft $(*)$ hat, muss es einen kW-Baum geben). Angenommen e' ist nicht in dem erzeugten Baum B enthalten. Es sei e_j die erste Ecke auf W, welche nicht in B ist. Dann ist der Vorgänger e_i von e_j auf W in B enthalten und es gilt $d(e, e_j) = d(e, e_i) + B[e_i, e_j]$. Beim Besuch von e_i durch `aufbauen` wird dann e_j besucht. Dieser Widerspruch zeigt, dass B ein kW-Baum ist.

```
var besucht : Array[1..n] von Boolean;
var vorgänger : Array[1..n] von Ecke;

procedure kwBaum(G : B-Graph, e : Ecke)
   Initialisiere besucht mit false und vorgänger mit ⊥;
   aufbauen(e, e);

procedure aufbauen(ei : Ecke, e : Ecke)
   var ei, ej : Ecke;

   besucht[i] := true;
   foreach ej in N(ei) do
      if not besucht[j] and d(e, ej) = d(e, ei) + B[i, j] then
         vorgänger[j] := ei;
         aufbauen(ej, e);
```

13. Da G die Eigenschaft $(*)$ erfüllt, gilt das Optimalitätsprinzip. Dann ist der Teilweg W_1 von e'_i nach e_j auf W ein kürzester Weg von e'_i nach e_j. Für W_1 gilt wieder das Optimalitätsprinzip, somit gilt $L(\overline{W}) = d(e'_i, e'_j)$, d. h., $\overline{W}$ ist ein kürzester Weg.

14. Es seien a und b positive reelle Zahlen. Dann gilt: $ab = 1/e^{-\log a - \log b}$. Da die e-Funktion monoton steigend ist, ist das Produkt ab umso größer, umso kleiner $-\log a - \log b$ ist. Für $a \in [0,1]$ ist $-\log a > 0$. Ändert man die Bewertung jeder Kante von b_{ij} auf $-\log b_{ij}$, so kann der Algorithmus von Dijkstra angewendet werden. Dann entsprechen die kürzesten Wege bezüglich der neuen Bewertung den Wegen, bei denen das Produkt der einzelnen Wahrscheinlichkeiten maximal ist.

    Der Übertragungsweg von Sender 1 zum Empfänger 5 mit der größten Wahrscheinlichkeit einer korrekten Übertragung ist 1, 8, 9, 3, 4, 5. Die Übertragungswahrscheinlichkeit ist 0,40824.

15. Ohne Einschränkung der Allgemeinheit kann man annehmen, dass alle Kanten die Bewertung 1 haben (vergleichen Sie Aufgabe 23). Eine in Abschnitt 10.2 bewiesene Folgerung des Optimalitätsprinzip ist: Für alle Kanten (e, e') gilt:

$$d(s, e') \leq d(s, e) + B[e, e'].$$

    Die Aussage der Aufgabe folgt nun aus Aufgabe 24 in Kapitel 4.

16. Die folgende Tabelle zeigt die Werte von `D` und `vorgänger` für einen kW-Baum mit Startecke e_1.

D				
1	2	3	4	5
0	3	7	4	3

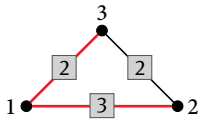
vorgänger				
1	2	3	4	5
0	1	2	3	4

17. Der Korrektheitsbeweis für diese Version des Algorithmus von Moore und Ford folgt dem Originalbeweis aus Abschnitt 10.3. Untersuchungen haben gezeigt, dass diese Variante in vielen Fällen eine geringere Laufzeit aufweist. Wendet man den neuen Algorithmus auf die Folge von Graphen an, welche durch die in Abbildung 10.8 angegebenen Adjazenzmatrizen M_n definiert ist, so stellt man allerdings fest, dass die Anzahl der Ausführungen der **`while`**-Schleife unverändert ist.

18. Vergleichen Sie hierzu Abschnitt 10.4.4. Sind die Kantenbewertungen reelle Zahlen, so haben die Ecken innerhalb eines Faches nicht unbedingt die gleichen Werte. Dies erhöht die Zugriffszeit und die in Abschnitt 10.4.4 angegebenen Laufzeiten werden nicht erreicht. Entscheidend ist die Verteilung der Ecken auf die Fächer. Je gleichmäßiger die Verteilung ist, desto größer ist der Vorteil von Bucket-Sort.

19. Es wird ein neuer Graph G' mit gleicher Ecken- und Kantenmenge gebildet. Die Kanten (e, e') von G' tragen die Bewertung $B'[e, e'] = B[e, e'] + (B[e] + B[e'])/2$. Genau dann ist W ein kürzester Weg von s nach z in G' bezüglich B', wenn W ein kürzester Weg von s nach z in G bezüglich B ist. Die Länge von W in G ist gleich $L'(W) - (B[s] + B[z])/2$, wobei $L'(W)$ die Länge von W in G' ist.

20. Die rot gezeichneten Kanten bilden einen kW-Baum für Ecke e_1, aber keinen minimal aufspannenden Baum.



21. (*a*) Distanz- und Vorgängermatrix des Graphen aus Aufgabe 6:

$$D = \begin{pmatrix} 0 & 3 & 6 & 10 & 8 & 6 \\ \infty & 0 & \infty & \infty & \infty & \infty \\ \infty & -3 & 0 & 5 & 3 & 1 \\ \infty & -7 & -4 & 0 & -1 & -3 \\ \infty & -5 & -2 & 2 & 0 & -1 \\ \infty & -3 & 0 & 4 & 2 & 0 \end{pmatrix} \qquad V = \begin{pmatrix} 0 & 3 & 4 & 6 & 6 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 3 & 0 & 5 & 6 & 3 \\ 0 & 3 & 4 & 0 & 6 & 3 \\ 0 & 3 & 4 & 5 & 0 & 3 \\ 0 & 2 & 4 & 6 & 6 & 0 \end{pmatrix}$$

(*b*) Die Werte für die Kantenbewertung B' sind: $b_{12} = 11$, $b_{15} = 10$, $b_{16} = 9$, $b_{32} = 0$, $b_{36} = 0$, $b_{43} = 0$, $b_{54} = 1$, $b_{64} = 1$, $b_{65} = 0$.

22. Zunächst wird der Abstand zwischen allen Paaren von Orten bestimmt. Danach wird für jeden Ort die weiteste Entfernung bestimmt (größter Wert in der entsprechenden Zeile der Distanzmatrix) und der Ort mit der kleinsten weitesten Entfernung ausgewählt. In diesem Fall ist es Ort 2 mit einer maximalen Entfernung von 180. Im Folgenden ist die Distanzmatrix dargestellt. Die weitesten Wegstrecken für die einzelnen Orte sind eingerahmt.

$$D = \begin{pmatrix} 0 & 100 & 250 & 280 & 140 & 260 \\ 100 & 0 & 150 & 180 & 40 & 160 \\ 250 & 150 & 0 & 30 & 190 & 70 \\ 280 & 180 & 30 & 0 & 220 & 100 \\ 140 & 40 & 190 & 220 & 0 & 120 \\ 260 & 160 & 70 & 100 & 120 & 0 \end{pmatrix}$$

23. Es sei W ein Weg bestehend aus l Kanten, $L(W)$ bzw. $L'(W)$ bezeichne die Länge von W vor bzw. nach der Erhöhung um c. Im ersten Fall gilt $L'(W) = L(W) + lc$ und im zweiten Fall $L'(W) = cL(W)$. Im zweiten Fall bleiben die kürzesten Wege die gleichen.

24. Der Graph hat die Eigenschaft $(*)$: Jeder geschlossene Weg hat mindestens eine Kante mit Bewertung 64, da alle Kanten (e_i, e_j) mit $i < j$ die Bewertung 64 haben. Der kW-Baum mit Startecke e_1 ist der Weg $<e_1, e_6, e_5, e_4, e_3, e_2>$ und die Längen der kürzesten Wege sind 64, 47, 38, 33, 30. Die **`while`**-Schleife wird 16 mal ausgeführt.

25. Die Werte für die Kantenbewertung B' sind: $b_{12} = 0$, $b_{14} = 4$, $b_{23} = 2$, $b_{31} = 1$, $b_{34} = 0$, $b_{45} = 0$, $b_{51} = 0$.

26. Der Graph hat nicht die Eigenschaft $(*)$, der geschlossene Weg $<e_6, e_5, e_3>$ hat die Länge -1. Wendet man die Prozedur `floyd` an, so wird der Diagonaleintrag $D[6,6]$ im vorletzten Durchgang negativ.

27. Mit Hilfe der Tiefensuche wird für jede Ecke e_i die Länge $d_B(e, e_i)$ des kürzesten Weges von e nach e_i in B bestimmt (Aufwand $O(n)$). Nach den Ergebnissen aus Abschnitt 10.2 ist B genau dann ein kW-Baum, wenn für jede Kante (e_i, e_j) von G die Ungleichungen $d_B(e, e_i) + B[e_i, e_j] \geq d_B(e, e_j)$ und $d_B(e, e_j) + B[e_i, e_j] \geq d_B(e, e_i)$ erfüllt sind. Dies kann mit Aufwand $O(m)$ überprüft werden.

28. Falls es einen geschlossenen Weg W mit negativer Länge gibt, so gibt es eine Ecke `es` auf W, so dass alle in `es` startenden Teilwege von W negative Länge haben. Die Funktion `negativerKreis` überprüft die Existenz eines solchen geschlossenen Weges für eine gegebene Ecke `es`. Wird ein solcher Weg gefunden, so wird die Funktion beendet. Der

Graph hat dann nicht die Eigenschaft (∗). In der Funktion `sternEigenschaft` wird diese Funktion für jede Ecke aufgerufen. Gibt es für keine Ecke des Graphen einen solchen Weg, so erfüllt der Graph die Eigenschaft (∗).

```
function sternEigenschaft(G : B-Graph) : Boolean
   foreach e_i in G.E do
      if negativerKreis(e_i, e_i, 0) then
         return false;
   return true;

function negativerKreis(e_s : Ecke, e_i : Ecke, länge : Integer) : Boolean
   foreach e_j in N+(e_i) do
      if länge+B[i, j] < 0 then
         if e_j = e_s then
            return true;
         else
            negativerKreis(e_s, e_j, länge+B[i, j]);
   return false;
```

29. Die folgende Tabelle zeigt die Werte von `D` und `vorgänger` für einen lW-Baum (*längste Wege Baum*) mit Startecke e_1.

D

1	2	3	4	5	6
0	1	−9	−3	3	−7

vorgänger

1	2	3	4	5	6
0	1	4	2	4	3

30. Die **`if`**-Anweisung in der Prozedur `floyd` aus Abbildung 10.25 muss wie folgt abgeändert werden.

```
if max(D[i, j], D[j, k]) < ∞ and D[i, k] > min(D[i, j], D[j, k]) then
   D[i, k] := min(D[i, j], D[j, k]);
   V[i, k] := V[j, k];
```

31. Da f_1 und f_2 konsistent sind, gilt $0 \leq f_i(e) \leq B[e,e'] + f_i(e')$ für $i = 1,2$ und jede Kante (e,e'). Multipliziert man die erste Ungleichung mit a und die zweite mit b und addiert diese dann, so erhält man:

$$0 \leq af_1(e) + bf_2(e) \leq (a+b)B[e,e'] + af_1(e') + bf_2(e').$$

Dividiert man diese Ungleichung durch $a+b$, so folgt die Konsistenz von $(af_1+bf_2)/(a+b)$.

32. Für einen beliebigen Brettzustand b und einen Zielzustand z sei

$f_1(b)$ die Anzahl der Plättchen, welche in b eine andere Position als in z haben;

$f_2(b)$ die Summe der für jedes Plättchen von b (ohne Beachtung der anderen Plättchen) mindestens notwendigen Verschiebungen, um die Position in z zu erreichen.

Für die in Abbildung 4.28 dargestellte Startstellung s gilt $f_1(s) = 8$ und $f_2(s) = 13$. Sowohl f_1 als auch f_2 sind zulässig. Beide Schätzfunktionen können mit Hilfe von Metriken auf dem Raum $Z \times Z$ definiert werden: für a,b aus $Z \times Z$ sei $d_1(a,b) = 0$, falls $a = b$, und 1, falls $a \neq b$ und $d_2(a,b) = |a_x - b_x| + |a_y - b_y|$. Bezeichnet man mit $P_a(i)$ die Position von Plättchen i im Zustand a, so gilt

$$f_i(a) = d_i(P_a(1),P_z(1)) + \ldots + d_i(P_a(8),P_z(8))$$

für $i = 1,2$. Aus der Dreiecksungleichung für d_1 und d_2 folgt:

$$f_i(a) - f_i(b) = \sum_{j=1}^{8} d_i(P_a(j), P_z(j)) - d_i(P_b(j), P_z(j)) \leq \sum_{j=1}^{8} d_i(P_a(j), P_b(j)).$$

Da die rechte Seite dieser Ungleichung höchstens so groß ist, wie die minimale Anzahl von Verschiebungen, um Zustand b von Zustand a aus zu erreichen, sind f_1 und f_2 konsistent. Wegen $f_1(a) \leq f_2(a)$ wird das Ziel mit Hilfe der zweiten Schätzfunktion im Allgemeinen schneller gefunden.

33. Startend mit dem trivialen Fluss f_0 werden mit Hilfe von Erweiterungswegen minimaler Kosten neue kostenminimale Flüsse f_i bestimmt. Dazu beachte man, dass der triviale Fluss ein Fluss mit Wert 0 und minimalen Kosten ist. Einen Erweiterungsweg mit minimalen Kosten findet man mit dem Algorithmus von Moore und Ford. Dieser Algorithmus ist anwendbar, da es in den Graphen G_{f_i} nach Teil (a) des Satzes aus Abschnitt 8.7 auf Seite 246 keine geschlossenen Wege negativer Länge gibt. Das Verfahren wird beendet, sobald Ecke s in G_{f_i} nicht mehr von q aus erreichbar ist. Dann liegt ein maximaler Fluss mit minimalen Kosten vor. Da alle Kapazitäten ganzzahlig sind, gilt $|f_{i+1}| \geq |f_i| + 1$, d. h., der Algorithmus terminiert nach maximal $|f_{max}|$ Durchgängen. Unter Verwendung des Algorithmus von Moore und Ford ergibt sich eine Laufzeit von $O(|f_{max}|nm)$.

34. Der in Abbildung 10.9 dargestellte Algorithmus von Dijkstra expandiert immer die Ecke aus dem kW-Baum, welche aktuell den kürzesten Abstand zur Startecke hat. Zur Lösung der gestellten Aufgabe muss die Ordnung auf den Ecken des kW-Baumes neu definiert werden. Ist die Entfernung zweier Ecken zur Startecke gleich, so wird die Ecke, bei der der Weg von der Startecke weniger Kanten enthält, als kleiner angesehen. Hierzu wird neben der eigentlichen Entfernung zur Startecke auch die Anzahl der Kanten des Weges im kW-Baum in einem Feld `kantenAnzahl` gespeichert. Dieses Feld wird mit ∞ initialisiert. Die Funktion `verkürze` muss wie folgt geändert werden.

```
function verkürze(e_i : Ecke, e_j : Ecke) : Boolean
   if D[i]+B[i, j] < D[j] then
      D[j] := D[i]+B[i, j];
      vorgänger[j] := e_i;
      kantenAnzahl[j] := kantenAnzahl[i]+1;
      return true;
   else
      if D[i]+B[i, j] = D[j] then
         if kantenAnzahl[j] > kantenAnzahl[i]+1 then
            vorgänger[j] := e_i;
            kantenAnzahl[j] := kantenAnzahl[i]+1;
            return true;
   return false;
```

Die Implementierung des Algorithmus von Dijkstra wird an zwei Stellen geändert. Die Auswahl einer Ecke aus `F` erfolgt gemäß der neuen Ordnung und beim Einfügen einer Ecke in `F` wird das Feld `kantenAnzahl` auf 1 gesetzt.

35. Es sei $e_1 = (a_1, \ldots, a_s)$ die Start- und $e_2 = (b_1, \ldots, b_s)$ die Zielecke. Für eine beliebige Ecke e von Q_s sei $e(i)$ die Ecke, welche sich von e genau an der i-ten Position unterscheidet.

```
for i := 1 to n do
   if a_i ≠ b_i then
      Schicke Nachricht von e nach e(i);
      e := e(i);
```

36. Nach dem die Breitensuche für eine Ecke *s* durchgeführt worden ist, enthält das Feld `niveau` die Längen der kürzesten Wege von *s* zu allen anderen Ecken. Somit genügt es, die Werte des Feldes `niveau` zu summieren.

```
C_C := 0;
foreach e_i in G.E do
   C_C := C_C + niveau[i]
return C_C/(n - 1);
```

Kapitel 11 – Approximative Algorithmen

1. (a) Ohne Einschränkung der Allgemeinheit kann angenommen werden, dass nach der Zuordnung der Programme zu Prozessoren Prozessor P_1 die höchste Last hat. Bezeichne mit T die Gesamtlaufzeit aller P_1 zugeordneter Programme. Das zuletzt an P_1 zugeordnete Programm habe die Laufzeit t_j. Da Programm j als letztes zugeordnet wurde, hat jeder Prozessor mindestens eine Laufzeit von $T - t_j$. Hieraus folgt:

$$\sum_{i=1}^{n} t_i \geq m(T - t_j) + t_j.$$

Ferner ist $A(n) = T$ und somit gilt

$$OPT(n) \geq \frac{\sum_{i=1}^{n} t_i}{m} \geq (T - t_j) + \frac{t_j}{m} = A(n) - (1 - \frac{1}{m})t_j.$$

Wegen $t_j \leq OPT(n)$ folgt

$$A(n) \leq OPT(n) + (1 - \frac{1}{m})t_j \leq OPT(n)(2 - \frac{1}{m}).$$

Hieraus ergibt sich die Behauptung.

(b) Die Lasten der einzelnen Prozessoren werden in einem Heap verwaltet. In jedem Schritt wird der Prozessor mit der geringsten Last ausgewählt und die Last dieses Prozessors wird um die Laufzeit des aktuellen Programms erhöht. Diese Operation hat den Aufwand $O(\log m)$. Für alle n Programme ergibt sich zusammen der Aufwand $O(n \log m)$.

(c) Es sei $n = m(m-1) + 1$, die ersten $n - 1$ Programme haben die Laufzeit 1 und das letzte Programm die Laufzeit m. Hieraus folgt $A(n) = 2m - 1$ und $OPT(n) = m$. Somit ist $\mathcal{W}_A(n) \leq 2 - \frac{1}{m}$.

(d) Da das zuletzt zugeordnete Programm die kürzeste Laufzeit hat, gilt

$$t_j \leq \frac{\sum_{i=1}^{n} t_i}{n} \leq \frac{m}{n} OPT(n).$$

Hieraus folgt

$$\mathcal{W}_A(n) \leq 1 + \frac{m}{n}(1 - \frac{1}{m}).$$

Wegen $n > m$ wird also der Wirkungsgrad verbessert. Mit einer genaueren Analyse kann gezeigt werden, dass $\mathcal{W}_A(n) \leq \frac{4}{3} - \frac{1}{3m}$ gilt.

2. (a) Es sei G ein Graph, der entsteht, wenn die Wurzel eines Baumes der Höhe 1 und $s + 1$ Ecken mit einer beliebigen Ecke des vollständigen Graphen K_s verbunden wird. G hat $2s + 1$ Ecken und $\omega(G) = s$. Die Wurzel des Baumes hat in G maximalen Eckengrad. Entfernt man diese und alle nicht zu ihr benachbarten Ecken, so verbleiben $s + 1$ isolierte Ecken. Somit erzeugt A_{C1} eine Clique der Größe 2 und es gilt $OPT(G)/A_{C1}(G) = s/2 = (n-1)/4$ bzw. $\mathcal{W}^{\infty}_{A_{C1}} = \infty$.

(*b*) Es sei G ein Graph G, der entsteht, wenn eine beliebige Ecke aus dem vollständigen Graphen K_s mit einer beliebigen Ecke aus dem vollständig bipartiten Graphen $K_{s,s}$ verbunden wird. G hat $3s$ Ecken und $\omega(G) = s$. In K_s gibt es $s-1$ Ecken mit Eckengrad $s-1$, diese werden von A_{C2} als Erstes aus G entfernt. Somit erzeugt A_{C2} eine Clique der Größe 2 und es gilt $OPT(G)/A_{C2}(G) = s/2 = n/6$ bzw. $\mathcal{W}^{\infty}_{A_{C2}} = \infty$.

3. In Kapitel 6 wurde gezeigt, dass die Größe einer maximalen Clique $\omega(G)$ gleich der Größe einer maximalen unabhängigen Menge $\alpha(\overline{G})$ des Komplements des Graphen ist. Somit ergibt sich aus jedem approximativen Algorithmus für das Optimierungsproblem von Cliquen direkt ein approximativer Algorithmus für das Optimierungsproblem der unabhängigen Menge mit gleichem Wirkungsgrad. Die in der Aufgabe gemachte Aussage folgt nun direkt aus dem letzten Satz aus Abschnitt 11.5.4 auf Seite 365.

4. Ein Erweiterungsweg bezüglich einer Zuordnung Z ist ein Weg, der bei einer nicht zugeordneten Ecke beginnt und endet und bei dem jede zweite Kante nicht in Z enthalten ist. Mit Hilfe eines Erweiterungsweges W kann eine gegebene Zuordnung vergrößert werden. Hierzu werden aus Z alle Kanten entfernt, welche auf W liegen. Die restlichen Kanten von W werden in Z eingefügt. Die so entstandene Zuordnung enthält eine Kante mehr als die ursprüngliche Zuordnung. Zum Beweis der Aussage in der Aufgabe wird der Graph G' betrachtet. Da die Kanten von G' aus zwei Zuordnungen stammen, ist der Eckengrad jeder Ecke von G' kleiner oder gleich 2. Es sei H eine Zusammenhangskomponente von G'. H ist entweder eine isolierte Ecke, ein einfacher, geschlossener Weg mit der gleichen Anzahl von Kanten aus Z_1 und Z_2 oder ein offener einfacher Weg, dessen Kanten abwechselnd aus Z_1 und Z_2 stammen. Komponenten vom letzten Typ mit einer ungeraden Anzahl von Kanten sind entweder Erweiterungswege bezüglich Z_1 oder Z_2 (je nachdem ob mehr Kanten aus Z_2 oder Z_1 stammen). Da alle Kanten aus Z_1 und Z_2 auf die Zusammenhangskomponenten verteilt sind, muss es $|Z_2| - |Z_1|$ Komponenten vom letzten Typ geben, bei denen die Mehrzahl der Kanten aus Z_2 kommt. Dies sind alles eckendisjunkte Erweiterungswege bezüglich Z_1.

5. Die folgende Funktion `greedyZuordnung` erzeugt offensichtlich eine nicht erweiterbare Zuordnung Z von G.

```
function greedyZuordnung(G : Graph) : Menge von Kante
   var Z : Menge von Kante;
   var ei , ej : Ecke;
   var k : Kante;

   foreach ei in G.E do
      if es gibt Kante k = (ei , ej), welche zu keiner Kante aus Z inzident ist then
         Z.einfügen(k);
   return Z;
```

Es sei M eine maximale Zuordnung von G. Dann gibt es nach der letzten Aufgabe $|M| - |Z|$ eckendisjunkte Erweiterungswege bezüglich Z. Nach Konstruktion enthält jeder dieser Erweiterungswege mindestens zwei Kanten aus M. Da die Wege eckendisjunkt sind, ist $|M|-|Z| \leq |M|/2$ bzw. $2|Z| \geq |M|$. Hieraus folgt, dass der Wirkungsgrad des angegebenen Algorithmus kleiner oder gleich 2 ist.

Es sei l eine gerade Zahl und G_l ein bipartiter Graph mit Eckenmenge

$$E = \{a_1, \ldots, a_l\} \cup \{b_1, \ldots, b_l\}$$

und Kantenmenge

$$K = \{(a_i, b_i) \mid i = 1,\ldots,l\} \cup \{(a_i, b_{i+1}) \mid i = 1,3,5,\ldots,l-1\} \cup \{(a_i, b_{i-1}) \mid i = 3,5,\ldots,l-1\}.$$

Die Kanten $\{(a_i, b_i) \mid i = 1,\ldots,l\}$ bilden eine maximale Zuordnung mit l Kanten. Die Kanten $\{(a_i, b_{i+1}) \mid i = 1,3,5,\ldots,l-1\}$ bilden eine Zuordnung, die durch die Funktion `greedyZuordnung` bestimmt wurden. Somit gilt

$$OPT(G_l)/A(G_l) = l/2l = 2,$$

d. h., der asymptotische Wirkungsgrad ist 2. Ist die von `greedyZuordnung` generierte Zuordnung Z nicht vollständig, dann gibt es eine nicht zugeordnete Ecke e. Es folgt, dass alle Nachbarn von e (dies sind mindestens $\delta(G)$ Ecken) durch Z abgedeckt sind. Somit muss Z mindestens $\delta(G)$ Kanten enthalten.

In der folgenden Implementierung von `greedyZuordnung` werden die Nachbarn jeder Ecke genau einmal betrachtet, deshalb ist die worst case Laufzeit gleich $O(n + m)$. Die Endecken der Kanten der Zuordnung können in linearer Zeit aus dem Feld `zuordnung` entnommen werden.

```
var zuordnung : Array[1..n] von Integer;

procedure greedyZuordnung(G : Graph)
   var e_i, e_j : Ecke;
   Initialisiere zuordnung mit ⊥;
   foreach e_i in G.E do
      if zuordnung[i] = ⊥ then
         foreach e_j in N(e_i) do
            if zuordnung[j] = ⊥ then
               zuordnung[i] := e_j;
               zuordnung[j] := e_i;
               break;
```

6. Der Algorithmus aus der letzten Aufgabe hat für das beschriebene Problem die worst case Laufzeit $O(n + \overline{m})$, wobei $\overline{m}$ die Anzahl der Kanten von $\overline{G}$ ist. Der folgende Algorithmus ist ebenfalls ein Greedy-Algorithmus, d. h., die Aussage über den Wirkungsgrad bleibt gültig. Er verwendet eine verkettete Liste zur effizienten Iteration über die Menge der noch nicht zugeordneten Ecken.

```
var zuordnung : Array[1..n] von Ecke;

procedure greedyZuordnung(G : Graph)
   var nachbarn : Array[1..n] von Boolean;
   var unbedeckt : Liste von Ecke;
   var e_i, e_j : Ecke;
   Initialisiere zuordnung mit ⊥ und nachbarn mit false;
   Füge alle Ecken von G in unbedeckt ein;
   while unbedeckt ≠ ∅ do
      e_i := unbedeckt.entfernen();
      foreach e_j in N(e_i) do
         nachbarn[j] := true;
      foreach e_j in unbedeckt do
         if not nachbarn[j] then
```

```
            unbedeckt.entferne(e_j);
            zuordnung[i] := e_j;
            zuordnung[j] := e_i;
            break;
        foreach e_j in N(e_i) do
            nachbarn[j] := false;
```

Für die Analyse der Laufzeit beachte man, dass jede der drei **for**-Schleifen innerhalb der **while**-Schleife den Aufwand $O(g(e_i))$ hat. Hieraus ergibt sich, dass der Algorithmus lineare Laufzeit $O(n+m)$ hat.

7. Nach Aufruf der Prozedur aus Aufgabe 5 werden die z Kanten der Zuordnung Z der Reihe nach betrachtet. Gibt es für die Endecken einer Kante (e_i, e_j) nicht zugeordnete Nachbarn e_i' und e_j' mit $e_i' \neq e_j'$, so wird (e_i, e_j) aus Z entfernt und die beiden neuen Kanten (e_i', e_i) und (e_j', e_j) werden in Z eingefügt. Man überzeugt sich schnell, dass nach Betrachtung der z Kanten jeder Erweiterungsweg bezüglich Z aus mindestens fünf Kanten besteht (drei davon gehören nicht zu Z). Die Laufzeit des Verfahrens bleibt weiterhin linear.

   Es sei M eine maximale Zuordnung von G. Dann gibt es nach der vorletzten Aufgabe $|M| - |Z|$ eckendisjunkte Erweiterungswege bezüglich Z. Nach Konstruktion enthält jeder dieser Erweiterungswege mindestens drei Kanten aus M. Da die Wege eckendisjunkt sind, ist $|M| - |Z| \leq |M|/3$ bzw. $3|Z| \geq 2|M|$. Somit ist der Wirkungsgrad des Algorithmus kleiner oder gleich $3/2$.

   Es sei l eine durch 3 teilbare Zahl und G_l ein bipartiter Graph mit Eckenmenge

$$E = \{a_1, \ldots, a_l\} \cup \{b_1, \ldots, b_l\}$$

   und Kantenmenge

$$K = \{(a_i, b_i) \mid i = 1, \ldots, l\} \cup \{(a_i, b_{i+1}) \mid 1 \leq i < l,\ i \text{ nicht durch 3 teilbar}\}$$
$$\cup\ \{(b_i, a_{i+1}) \mid 1 \leq i < l, i \text{ durch 3 teilbar}\}.$$

   Die Kanten $\{(a_i, b_i) \mid i = 1, \ldots, l\}$ bilden eine maximale Zuordnung mit l Kanten. Die Kanten $\{(a_i, b_{i+1}) \mid 1 \leq i < l,\ i \text{ nicht durch 3 teilbar}\}$ bilden eine Zuordnung, die durch den Algorithmus bestimmt wurde. Somit gilt

$$OPT(G_l)/A(G_l) = l/(2/3)l = 3/2$$

   und der asymptotische Wirkungsgrad ist gleich $3/2$.

8. (*a*) Angenommen es gibt eine Ecke e von G, welche nicht in M liegt und auch zu keiner Ecke aus M benachbart ist. Wegen $e \notin M$ sind alle Nachbarn von e untereinander benachbart, somit ist der von $N[e]$ induzierte Untergraph vollständig. Da G zusammenhängend aber nicht vollständig ist, gibt es eine Ecke a außerhalb von $N[e]$ die zu einer Ecke b aus $N[e]$ benachbart ist. Aus der Definition von M folgt, dass b in M liegt. Dieser Widerspruch zeigt, dass M eine dominierende Menge ist.

   Es seien $e, e' \in M$ und W ein kürzester Weg zwischen e und e' in G. Angenommen es gibt auf W eine Ecke e'' mit $e'' \notin M$. Da P ein kürzester Weg ist, sind die beiden Nachbarn von e'' auf W nicht benachbart (sonst gäbe es einen kürzeren Weg). Somit liegt e'' nach Definition in M. Dieser Widerspruch zeigt, dass M eine zusammenhängende dominierende Menge für G ist.

(*b*) Es seien e, e' beliebige Ecken von G und W ein kürzester Weg zwischen e und e'. Angenommen es gibt auf W eine innere Ecke e'' mit $e'' \notin M$. Da P ein kürzester Weg ist, sind die beiden Nachbarn von e'' auf P nicht benachbart (sonst gäbe es einen kürzeren Weg). Somit liegt e'' nach Definition in M. Dieser Widerspruch zeigt die Gültigkeit der Aussage.

(*c*) Der folgende einfache Algorithmus bestimmt M mit Aufwand $O(nm)$.

```
foreach (e, e') in G.K do
   foreach e_i in G.E \ {e, e'} do
      if e_i ∈ N(e) then
         if e_i ∉ N(e') then
            M.einfügen(e);
      else
         if e_i ∈ N(e') then
            M.einfügen(e);
```

Der nächste Algorithmus bestimmt M mit Aufwand $O(\sum_{i=1}^{n} g(i)^2) = O(m\Delta)$.

```
var Boolean : fertig;

foreach e in G.E do
   fertig := false;
   foreach e_i in N(e) do
      if fertig then
         break;
      foreach e_j in N(e) \ {e_i} do
         if e_i ∉ N(e_j) then
            M.einfügen(e);
            fertig := true;
            break;
```

9. Es seien $e_1, \ldots, e_s$ die Ecken, die in dieser Reihenfolge nach der angegebenen Regel entfernt werden. Ferner sei $M_i = M \backslash \{e_1, \ldots, e_i\}$ für $i = 0, \ldots, s$. Dann ist $M = M_0$ und $M' = M_s$. Im Folgenden wird mittels vollständiger Induktion nach i gezeigt, dass M_i die Eigenschaften (a) und (b) erfüllt. Der Induktionsanfang ergibt sich aus der letzten Aufgabe. Es sei nun $i > 0$, dann erfüllt M_{i-1} die Eigenschaften (a) und (b). Aus der Regel ergibt sich direkt, dass auch M_i eine zusammenhängende dominierende Menge von Ecken ist. Des Weiteren kann die entfernte Ecke e_i auf keinem kürzesten Weg von G liegen, hieraus ergibt sich auch die zweite Aussage für M_i. Damit ist der Beweis vollständig.

10. Es sei w die Wurzel des Breitensuchebaums und e eine Ecke auf dem b-ten Niveau. Damit diese beiden Ecken durch M abgedeckt sind, muss es Ecken $w_M, e_M \in M$ mit $h(w_M) \leq 1$ und $h(e_M) \geq b - 1$ geben. Da M zusammenhängend ist, muss es einen Weg zwischen w_M und e_M mit Ecken aus M geben. Somit enthält M mindestens $b - 1$ Ecken.

11. Für einen vollständig k-partiten Graphen G gilt $\chi(G) = k$. Nachdem die erste Ecke einer Teilmenge E_i durch den Greedy-Algorithmus die Farbe f_i erhalten hat, kann diese Farbe an keine Ecke einer anderen Teilmenge E_j mehr vergeben werden. Die restlichen Ecken von E_i werden jedoch unabhängig von ihrer Reihenfolge mit dieser Farbe gefärbt. Somit vergibt der Algorithmus genau $k = \chi(G)$ Farben. Der Greedy-Algorithmus färbt die Graphen C_n mit ungeradem n für jede Reihenfolge der Ecken mit 3 Farben, d. h., mit der minimalen Anzahl von Farben. Die Graphen C_n sind jedoch nicht vollständig k-partit.

12. In konstanter Zeit kann festgestellt werden, ob es sich um den vollständigen Graphen mit vier Ecken handelt. In diesem Fall ist $\chi(G) = 4$. Andernfalls folgt aus dem Satz von Brooks auf Seite 169, dass $\chi(G) \leq 3$ gilt. Mit linearem Aufwand kann geprüft werden, ob ein bipartiter Graph vorliegt, d. h., ob $\chi(G) = 2$ ist. Ist G nicht bipartit, dann besteht G aus genau einer Ecke oder es gilt $\chi(G) = 3$.

13. Angenommen G besitzt einen Hamiltonschen Kreis K. Da der Grad jeder Ecke gleich 3 ist, muss die Anzahl der Kanten von K gerade sein. Die Kanten von K lassen sich somit mit 2 Farben zulässig färben. Nun ist jede Ecke noch zu genau einer ungefärbten Kante inzident, somit lassen sich die restlichen Kanten von G alle mit einer Farbe färben. Hieraus folgt $\chi'(G) = 3$. Dieser Widerspruch zeigt, dass G nicht Hamiltonsch ist.

14. (a) Ein 1-Baum existiert nur, falls der von den Ecken $e_2, \ldots, e_n$ induzierte Untergraph G' zusammenhängend und $g(e_1) \geq 2$ ist. Es sei B ein minimal aufspannender Baum von G' und K_1 die Menge der zu Ecke e_1 inzidenten Kanten. Die beiden Kanten aus K_1 mit den geringsten Bewertungen bilden zusammen mit den Kanten aus B einen minimalen 1-Baum. Der Aufwand des Algorithmus hängt vom Aufwand der Bestimmung eines minimal aufspannenden Baumes für G' ab. Vergleichen Sie hierzu die in Kapitel 3 vorgestellten Algorithmen.

    (b) Man beachte, dass W ein 1-Baum ist.

15. Es seien W_1 und W_2 wie in der Aufgabe beschrieben. Gibt es eine Ecke e, welche mehr als einmal auf W_1 vorkommt, dann gibt es Kanten (e_1, e) und (e, e_2), welche nacheinander auf W_1 vorkommen. Ersetzt man diese beiden Kanten durch die Kante (e_1, e_2), so folgt aus der Dreiecksungleichung und der Minimalität von W_1, dass der neu entstandene Weg die gleiche Länge wie W_1 hat. Auf diese Art kann ein geschlossener einfacher Weg W_1' mit gleicher Länge wie W_1 erzeugt werden, auf dem alle Ecken von G liegen. Hieraus folgt:

$$L(W_2) \geq L(W_1) = L(W_1') \geq L(W_2).$$

    Dies zeigt die Behauptung.

16. Es sei L_i die Summe der Längen der Kanten, welche bis zum i-ten Schritt ausgewählt und in den Weg W_i eingefügt werden. Es sei $W_i = <e_1, \ldots, e_i, e_1>$ der konstruierte Weg. Mittels vollständiger Induktion nach i wird bewiesen, dass $L(W_i) \leq 2L_i$ für $i = 2, \ldots, n$ gilt. Für $i = 2$ ist die Aussage klar. Sei nun $i > 2$ und (e_s, f) die neu ausgewählte Kante. Da G die Dreiecksungleichung erfüllt, gilt

$$d(e_{s-1}, f) \leq d(e_{s-1}, e_s) + d(e_s, f)$$

    bzw.

$$d(e_{s-1}, f) + d(e_s, f) - d(e_{s-1}, e_s) \leq 2d(e_s, f).$$

    Mit Hilfe der Induktionsvoraussetzung folgt nun

$$\begin{aligned} L(W_i) &= L(W_{i-1}) + d(e_{s-1}, f) + d(e_s, f) - d(e_{s-1}, e_s) \\ &\leq 2L_{i-1} + 2d(e_s, f) \\ &= 2L_i. \end{aligned}$$

(Ist s = 1, so ersetze man in diesen Ungleichungen e_{s-1} durch e_i). Man beachte, dass die ausgewählten Kanten genau die Kanten sind, welche auch der Algorithmus von Prim auswählt. Somit ist $L(W_n) \leq 2K < 2OPT(G)$, wobei K die Kosten eines minimal aufspannenden Baumes von G sind. Hieraus folgt die Aussage über den Wirkungsgrad. Der Algorithmus hat die gleiche Laufzeit wie der Algorithmus von Prim.

17. Auf W liegen für jede Ecke e genau zwei verschiedene zu e inzidente Kanten. Diese haben zusammen mindestens die Länge $m(e)$. Da jede Kante auf W zu genau zwei Ecken inzident ist, gilt:
$$\frac{1}{2}\sum_{e \in E} m(e) \leq L(W).$$

18. Die Funktion kantenFärbung gibt die Anzahl a der vergebenen Farben zurück. Der Aufwand der Funktion ist gleich $O(a(n+m))$.

```
function kantenFärbung(G : Graph) : Integer
   var zuordnung : Array[1..n] von Integer;
   var e_i, e_j : Ecke;
   var farbe : Integer;

   Initialisiere zuordnung mit 0;
   farbe := 0;
   while G.K ≠ ∅ do
      farbe := farbe + 1;
      foreach e_i in G.E do
         if zuordnung[i] < farbe then
            foreach e_j in N(e_i) do
               if zuordnung[j] < farbe then
                  zuordnung[i] := farbe;
                  zuordnung[j] := farbe;
                  Färbe Kante (e_i, e_j) mit Farbe farbe;
                  Entferne Kante (e_i, e_j) aus G.K;
                  Entferne e_i bzw. e_j aus G.E wenn g(e_i) = 0 bzw. g(e_j) = 0;
                  break;
   return farbe;
```

19. Zur Bestimmung des Wirkungsgrads des dargestellten Algorithmus werden die in Abschnitt 11.4 konstruierten Graphen G_r betrachtet. Sie haben eine minimale Eckenüberdeckung mit r Ecken. Die Konstruktion der Graphen bewirkt, dass der Algorithmus der Reihe nach die Ecken aus $R_r, R_{r-1}, \ldots, R_1$ auswählt, d. h., die Menge R wird als Eckenüberdeckung konstruiert. Wie in Abschnitt 11.4 gezeigt, gilt auch für diesen Algorithmus $\mathcal{W}_A(n) = O(\log n)$ und $\mathcal{W}_A^{\infty} = \infty$. Mit Hilfe der in Abschnitt 5.2 entwickelten Datenstrukturen kann der Algorithmus mit Aufwand $O(n+m)$ umgesetzt werden.

20. Der Algorithmus verwaltet ein Feld überdeckung, in dem alle zur Überdeckung gehörenden Ecken markiert werden. Die Ecken des Baumes werden gemäß der Tiefensuche besucht. Bei diesem Algorithmus gehören Blätter nicht zur konstruierten Überdeckung. Die Funktion eckenÜberdeckung(e_i, vorgänger) zeigt beim Verlassen einer Ecke an, ob die Kante (e_i, vorgänger) noch überdeckt werden muss. Der Funktionsaufruf eckenÜberdeckung(e, ⊥) mit einer beliebigen Ecke e bestimmt eine minimale Eckenüberdeckung in linearer Zeit $O(n+m)$.

```
var überdeckung : Array[1..n] von Boolean;
```

```
function eckenÜberdeckung(e_i : Ecke, vorgänger : Ecke) : Boolean
   var e_j : Ecke;
   überdeckung[i] := false;
   foreach e_j in N(e_i) do
      if e_j ≠ vorgänger then
         überdeckung[i] := eckenÜberdeckung(e_j, e_i) or überdeckung[i];
   return not überdeckung[i];
```

Die Korrektheit des Verfahrens wird durch vollständige Induktion nach n, der Anzahl der Ecken des Baumes B, geführt. Für $n = 2$ erzeugt der Algorithmus offensichtlich eine minimale Eckenüberdeckung mit einer Ecke. Sei nun $n > 2$. Es sei e ein Blatt und e' der Vorgänger von e im Tiefensuchebaum. Hat e' keinen weiteren Nachfolger, so bestimmt der Algorithmus nach Induktionsvoraussetzung eine minimale Eckenüberdeckung U für den Baum $B\backslash\{e', e\}$. Somit ist $U \cup \{e'\}$ eine minimale Eckenüberdeckung von B. Wie man leicht sieht, ist dies aber genau die Eckenüberdeckung, welche der Algorithmus für B produziert. Gibt es in B kein Blatt mit dieser Eigenschaft, so muss es in B ein Blatt e geben, so dass der Vorgänger e' ein weiteres Blatt als Nachfolger hat. Nach Induktionsvoraussetzung bestimmt der Algorithmus für den Baum $B\backslash\{e\}$ eine minimale Eckenüberdeckung U. Wie man wieder leicht sieht, ist dies genau die Eckenüberdeckung, welche der Algorithmus für B produziert.

21. (*a*) Es sei U eine Eckenüberdeckung von G. Dann ist U auch eine Eckenüberdeckung von G_k mit $b_{G_k}(U) \leq b_G(U) - b(k)$ (mindestens eine der beiden Ecken e_i, e_j ist in U enthalten). Ist U eine kostenminimale Eckenüberdeckung von G so gilt:

$$\rho(G_k) \leq b_{G_k}(U) \leq b_G(U) - b(k) = \rho(G) - b(k).$$

(*b*) Der Algorithmus A verändert schrittweise die Bewertungen der Ecken von G. Bezeichne die dabei betrachteten Kanten mit $k_0, \ldots, k_{s-1}$ und die entstehenden eckenbewerteten Graphen mit $G_0 = G$, G_1 bis G_s. Durch vollständige Induktion wird bewiesen, dass $A(G_i) \leq 2\rho(G_i)$ für $i = 0, \ldots, s$ gilt. Wegen $A(G_0) = A(G)$ und $\rho(G_0) = \rho(G)$ ist damit die Aussage bewiesen. In G_s hat jede Kante mindestens eine Endecke, deren Bewertung gleich 0 ist. Somit ist $A(G_s) = \rho(G_s) = 0$, dies zeigt die Gültigkeit der Induktionsvoraussetzung. Aus Teil *(a)* der Aufgabe folgt für $i < s$ die Gültigkeit der Ungleichung $\rho(G_{i+1}) \leq \rho(G_i) - b_{G_i}(k_i)$. Ferner gilt $A(G_i) \leq A(G_{i+1}) + 2b_{G_i}(k_i)$ und nach Induktionsvoraussetzung ist $A(G_{i+1}) \leq 2\rho(G_{i+1})$. Aus diesen Ungleichungen folgt nun:

$$A(G_i) \leq A(G_{i+1}) + 2b_{G_i}(k_i) \leq 2\rho(G_{i+1}) + 2b_{G_i}(k_i) \leq 2\rho(G_i).$$

(*c*) Für einen beliebigen ungerichteten Graphen definiere eine nichtnegative Eckenbewertung durch $b(e) = 1$ für jede Ecke e. Mit dieser Bewertung bestimmt der angegebene Algorithmus die gleiche Eckenüberdeckung wie A_{EU2}.

(*d*) Die Kanten von G werden in einer beliebigen Reihenfolge betrachtet.

```
foreach k = (e, e') in G.K do
   b(e) := b(e) - b(k);
   b(e') := b(e') - b(k);
E := {e Ecke von G | b(e) = 0};
```

22. Ein polynomialer Algorithmus für das Optimierungsproblem löst natürlich auch das zugehörige Entscheidungsproblem. Es sei

```
function eckenÜberdeckung(G : Graph, wert : Integer) : Boolean
```

eine Funktion, welche mit polynomialem Aufwand entscheidet, ob G eine Eckenüberdeckung mit `wert` Ecken hat. Mit maximal OPT Aufrufen dieser Funktion kann die Größe OPT der kleinsten Eckenüberdeckung von G bestimmt werden. Es sei nun (e,e') eine Kante des Graphen G. Dann liegt e oder e' in einer minimalen Eckenüberdeckung. Es sei U_e bzw. $U_{e'}$ eine minimale Eckenüberdeckung des Graphen, der sich aus G ergibt, wenn alle zu e bzw. zu e' inzidenten Kanten entfernt werden. Ist $|U_e| = OPT - 1$, so ist $U_e \cup \{e\}$ eine minimale Eckenüberdeckung von G, andernfalls $U_{e'} \cup \{e'\}$. Der beschriebene Algorithmus wird durch den Aufruf `minÜberdeckung(G, OPT)` der folgenden rekursiven Funktion umgesetzt:

```
function minÜberdeckung(G : Graph, wert : Integer) : Menge von Kante
   if G.K = ∅ then
      return ∅;
   else
      Wähle beliebige Kante (e, e') aus G.K;
      G_e := G ohne die zu e inzidenten Kanten;
      if eckenÜbderdeckung(G_e, wert - 1) then
         return {e} ∪ minÜberdeckung(G_e, wert - 1);
      else
         G_e' := G ohne die zu e' inzidenten Kanten;
         return {e'} ∪ minÜberdeckung(G_e', wert - 1);
```

Insgesamt wird diese Funktion höchstens n-mal aufgerufen. Damit wird auch die Funktion `eckenÜberdeckung` höchstens n-mal aufgerufen, d. h., es liegt ein Algorithmus mit polynomialem Aufwand vor.

23. Ein polynomialer Algorithmus für das Optimierungsproblem löst natürlich auch das zugehörige Entscheidungsproblem. Es sei nun A ein polynomialer Algorithmus für das Entscheidungsproblem. In einer ersten Phase werden die Kosten C für eine optimale Lösung berechnet, die Bestimmung der optimalen Rundreise erfolgt dann in der zweiten Phase. Ist l die Länge der Kodierung einer Instanz des Problems des Handlungsreisenden, so liegt C zwischen 0 und 2^l. Mittels binärer Suche in Kombination mit Algorithmus A kann C in höchstens n Schritten bestimmt werden. In der zweiten Phase wird Algorithmus A für jede Kante einmal aufgerufen. Hierzu werden die Kosten der betrachteten Kante auf $C + 1$ gesetzt. Ergibt die Anwendung von A, dass dieser Graph keinen Hamiltonschen Kreis mit Kosten C besitzt, so wird die Bewertung der Kante wieder auf den ursprünglichen Wert zurückgesetzt, andernfalls wird die Kante entfernt. Am Ende dieser Phase bilden die verbliebenen Kanten einen Hamiltonschen Kreis mit Kosten C und das Optimierungsproblem ist gelöst. Insgesamt wurde Algorithmus A höchstens $n + m$-mal aufgerufen, d. h., das Verfahren hat polynomialen Aufwand.

24. Ist e eine Ecke von G und T eine beliebige Teilmenge der Ecken von G, so wird mit $N_T(e)$ im Folgenden die Menge der in T liegenden Nachbarn von e bezeichnet. Es sei e eine Ecke aus X mit $N_X(e) > N_{\overline{X}}(e)$. Wird nun e von X nach $\overline{X}$ verschoben, so erhält der Schnitt $(X,\overline{X})$ genau $N_{\overline{X}}(e) - N_X(e) > 0$ zusätzliche Kanten. Auch bei der Verschiebung einer Ecke e aus $\overline{X}$ mit $N_{\overline{X}}(e) > N_X(e)$ nach X wird die Anzahl der Kanten im Schnitt

erhöht. Da ein Schnitt maximal m Kanten enthält, endet das Verfahren spätestens nach m Schritten. Dann gilt $N_X(e) \leq N_{\overline{X}}(e)$ für alle Ecken e aus X, und $N_{\overline{X}}(e) \leq N_X(e)$ für alle Ecken aus $\overline{X}$. Nun kann der Wirkungsgrad leicht bestimmt werden. Es ist

$$2A(G) = \sum_{e \in X} N_{\overline{X}}(e) + \sum_{e \in \overline{X}} N_X(e).$$

Hieraus folgt

$$\begin{aligned} OPT(G) &\leq m \\ &= \frac{1}{2}\Big(\sum_{e \in X} \Big(N_X(e) + N_{\overline{X}}(e)\Big) + \sum_{e \in \overline{X}} \Big(N_X(e) + N_{\overline{X}}(e)\Big)\Big) \\ &\leq \sum_{e \in X} N_{\overline{X}}(e) + \sum_{e \in \overline{X}} N_X(e) \\ &= 2A(G). \end{aligned}$$

25. Das betrachtete Entscheidungsproblem liegt offenbar in $\mathcal{NP}$. Es wird ein Graph G mit Eckenmenge $E = X \cup Y \cup Z$ konstruiert. Zwei Ecken $a, b \in E$ sind genau dann benachbart, wenn es kein Element $m \in M$ gibt, welches a und b gleichzeitig als Komponenten enthält. Der Graph G hat $3q$ Ecken und lässt sich in polynomialer Zeit konstruieren. Ist $(x, y, z) \in M$, dann bilden x, y, z in G eine unabhängige Menge. Die von den Mengen X, Y und Z induzierten Untergraphen sind vollständig, somit ist $\chi(G) \geq q$. Es sei A eine vier-elementige Teilmenge von E. Dann muss es in A zwei Elemente geben, welche in der gleichen der drei Cliquen liegen. Somit ist $\alpha(G) \leq 3$. Im Folgenden wird bewiesen, dass M genau dann eine 3-dimensionale Zuordnung besitzt, wenn $\chi(G) = q$ gilt. Hieraus folgt unmittelbar die in der Aufgabe gemachte Aussage. Ist $\chi(G) = q$, so besteht G aus genau q Farbklassen mit je drei Elementen. Es sei $\{a, b, c\}$ mit $a \in X, b \in Y$ und $c \in Z$ eine solche Farbklasse. Nach Konstruktion von G gibt es $x \in X, y \in Y$ und $z \in Z$, so dass $(x, b, c), (a, y, c), (a, b, z)$ in M liegen. Wegen der paarweisen Konsistenz muss dann auch (a, b, c) in M liegen. Hieraus folgt, dass die aus den Farbklassen gebildeten Elemente aus M eine 3-dimensionale Zuordnung bilden. Umgekehrt gilt, dass jede 3-dimensionale Zuordnung auf G eine disjunkte Überdeckung mit 3-elementigen unabhängigen Mengen induziert, d. h., es gilt $\chi(G) = q$.

26. Der erste Teil des Algorithmus entspricht der Funktion `greedyZuordnung` aus Aufgabe 5. Besteht die erzeugte Zuordnung Z aus m' Kanten, so gilt $|I_1| = n - 2m'$. Wegen $m' < n/2$ gibt es eine Zahl $\gamma \in [0, 1]$ mit $m' = n(1 - \gamma)/2$ bzw. $|I_1| = n\gamma$. Nun bildet $|Z| + |I_1|$ eine obere Abschätzung für $\alpha(G)$, hieraus folgt:

$$\alpha(G) \leq \frac{n}{2}(1 + \gamma). \tag{1}$$

Nun wird der zweite Teil des Algorithmus betrachtet. Im ersten Durchlauf der **`while`**-Schleife werden $\delta + 1$ Ecken entfernt. In jedem weiteren Schritt werden höchstens Δ Ecken aus E entfernt. Dazu beachte man, dass es wegen des Zusammenhangs von G mindestens eine Ecke im Restgraphen geben muss, der Grad kleiner oder gleich $\Delta - 1$ ist. Somit gilt:

$$|I_2| \geq \frac{n - (\delta + 1)}{\Delta} + 1. \tag{2}$$

Aus Gleichung (1) und $|I_1| = n\gamma$ folgt:

$$\frac{\alpha(G)}{|I_1|} \leq \frac{\gamma+1}{2\gamma}. \tag{3}$$

Aus Gleichung (1) und (2) folgt:

$$\frac{\alpha(G)}{|I_2|} \leq \frac{\frac{n}{2}(1+\gamma)}{\frac{n-(\delta-1)+\Delta}{\Delta}} = \frac{\Delta n(1+\gamma)}{2(n-(\delta+1)+\Delta)} \leq \frac{\Delta n(1+\gamma)}{2(n-1)}. \tag{4}$$

Für die folgende Analyse beachte man, dass für $\gamma \in [0,1]$ die Funktion $(\gamma+1)/2\gamma$ monoton fallend und die Funktion $\Delta n(1+\gamma)/2(n-1)$ monoton steigend ist. Für die Bestimmung des Wirkungsgrads des Algorithmus ist der Schnittpunkt $S = (n-1)/\Delta n$ der beiden Funktionen von Bedeutung. Für $\gamma \leq S$ ist $|I_2| \geq |I_1|$ und für $\gamma \geq S$ ist $|I_1| \geq |I_2|$. Aus den Gleichungen (3) bzw. (4) ergibt sich die angegebene Schranke für den Wirkungsgrad.

27. Es sei d_i der Eckengrad der Ecke im Restgraphen, welche in Durchlauf i entfernt wurde. Dann gilt folgende Beziehung zwischen n und den d_i:

$$n = \sum_{i=1}^{|I_2|}(d_i+1).$$

Da jedesmal die Ecke mit dem kleinsten Eckengrad ausgewählt wird, haben im i-ten Schritt alle entfernten Ecken mindestens den Eckengrad d_i. Somit werden im i-ten Schritt mindestens $d_i(d_i+1)/2$ Kanten entfernt. Hieraus folgt:

$$\frac{\bar{d}n}{2} = |E| \geq \sum_{i=1}^{|I_2|} d_i(d_i+1)/2.$$

Addiert man das zweifache der letzten Gleichung zu der ersten Gleichung, so erhält man mittels der Cauchy-Schwartz Ungleichung und der ersten Gleichung:

$$(\bar{d}+1)n \geq \sum_{i=1}^{|I_2|}(d_i+1)^2 \geq \frac{1}{|I_2|}\left(\sum_{i=1}^{|I_2|}(d_i+1)\right)^2 \geq \frac{n^2}{|I_2|}.$$

Hieraus ergibt sich sofort die Behauptung.

28. Der Beweis erfolgt mittels vollständiger Induktion nach der Anzahl der Ecken des Graphen. Für $n = 1$ ist die Behauptung trivial. Es sei nun G ein Graph mit $n > 1$ Ecken. Die Funktion `unabhängigeMenge` bestimmt eine unabhängige Menge U mit $u(n)$ Ecken. Es sei $G' = G\backslash U$. Nach Induktionsvoraussetzung erzeugt `färbung` für G' eine Färbung mit höchstens

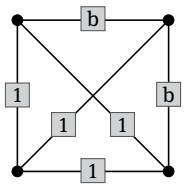
$$\sum_{i=1}^{n-u(n)} \frac{1}{u(i)}$$

Farben und für G wird genau eine zusätzliche Farbe benötigt. Da u monoton steigend ist, gilt

$$\sum_{i=n-u(n)+1}^{n} \frac{1}{u(i)} \geq \sum_{i=n-u(n)+1}^{n} \frac{1}{u(n)} = 1.$$

Hieraus folgt die Behauptung.

29. Erfüllen die Bewertungen der Kanten die Dreiecksungleichung, so gilt $L \leq 2K$. Im Allgemeinen gibt es aber keine solche Konstante, wie der folgende Graph zeigt, hierbei ist L von b abhängig und K von b unabhängig.



30. Das betrachtete Entscheidungsproblem liegt offenbar in $\mathcal{NP}$. Für einen gegebenen zusammenhängenden Graphen G wird wie im Hinweis zur Aufgabe ein neuer Graphen G' konstruiert. Es sei

$$S = \{x_{ee'} \mid (e,e') \; ist \; Kante \; von \; G\} \cup \{e_n\}.$$

Es sei s_{stei} das Gewicht eines minimalen Steiner Baums von G' für S und s_{deck} die Anzahl der Ecken in einer minimalen Eckenüberdeckung für G. Im Folgenden wird bewiesen, dass $s_{stei} = s_{deck}+m$ gilt. Hieraus folgt unmittelbar die in der Aufgabe gemachte Aussage.

Es sei M eine minimale Eckenüberdeckung von G. Dann ist der von $S \cup M$ induzierte Untergraph U von G' zusammenhängend. Hierzu beachte man, dass e_n zu jeder Ecke aus M benachbart ist und jede Ecke aus $S\backslash\{e_n\}$ zu einer Ecke aus M benachbart sein muss (M ist eine Eckenüberdeckung). Da alle Kanten die Bewertung 1 haben, hat ein aufspannender Baum von U das Gewicht $|S \cup M| = s_{deck} + m$. Hieraus folgt $s_{stei} \leq s_{deck} + m$. Sei nun T ein minimaler Steiner Baum von G' für S und S' die Menge der Steiner Ecken von T. Nach Konstruktion von G' bildet S' eine Eckenüberdeckung von G: Ist $k = (e,e')$ eine Kante von G, so gibt es in T einen Weg von e_n nach $x_{ee'}$, dieser verwendet eine der Ecken e und e', d. h., e oder e' liegt in S'. Hieraus folgt:

$$s_{deck} \leq |S'| = s_{stei} + 1 - |S| = s_{stei} - m.$$

31. Der Greedy-Algorithmus vergibt maximal $\Delta + 1$ Farben. Nach Aufgabe 30 aus Kapitel 2 enthält jeder Unit Disk Graph eine Clique mit mindestens $\lceil \Delta/6 \rceil + 1$ Ecken. Somit ist $\chi \geq \lceil \Delta/6 \rceil + 1$. Hieraus folgt:

$$\frac{OPT}{\chi} \leq \frac{\Delta + 1}{\lceil \Delta/6 \rceil + 1} \leq 6.$$