Prof. Dr. Ir. Joost-Pieter Katoen

Sebastian Junges, Benjamin Kaminski, David Korzeniewski, Tim Quatmann

Übung 9

– Musterlösung –

Hinweise:

- Die Lösungen müssen bis **Donnerstag, den 28. Juni um 16:00 Uhr** in den entsprechenden Übungskasten eingeworfen werden. Sie finden die Kästen am Eingang Halifaxstr. des Informatikzentrums (Ahornstr. 55).
- Die Übungsblätter **müssen** in Gruppen von je 3 Studierenden aus der gleichen Kleingruppenübung abgegeben werden.
- Drucken Sie ggf. digital angefertigte Lösungen aus. Abgaben z.B. per Email sind nicht zulässig.
- Namen und Matrikelnummer sowie die **Nummer der Übungsgruppe** sind auf jedes Blatt der Abgabe zu schreiben. Abgaben, die aus mehreren Blättern bestehen **müssen geheftet bzw. getackert** werden! Die **Gruppennummer muss sich auf der ersten Seite oben links** befinden.
- **Bei Nichtbeachten der obigen Hinweise müssen Sie mit erheblichen Punktabzügen rechnen!**

Aufgabe 1 (Graphen Terminologie): (10 mal 2 Punkte = 20 Punkte)

a) Wie viele gerichtete Graphen mit genau $n \in \mathbb{N}$ Knoten gibt es? Begründen Sie Ihre Antwort kurz.

b) Wie viele ungerichtete Graphen mit genau $n \in \mathbb{N}$ Knoten gibt es? Begründen Sie Ihre Antwort kurz.

c) Wie viele einfache Pfade der Länge genau $k \in \{0, 1, \ldots, n-1\}$ hat ein vollständiger ungerichteter Graph mit $n \in \mathbb{N}$ Knoten? Begründen Sie Ihre Antwort kurz.

d) Wie viele Zykel der Länge höchstens 4 hat ein vollständiger ungerichteter Graph mit $n \in \mathbb{N}$ Knoten? Begründen Sie Ihre Antwort kurz.

e) Sei $G = (V, E)$ ein gerichteter Graph mit $G^T = (V, E')$. Betrachte den Graphen $\hat{G} = (V, \hat{E})$ mit $\hat{E} = E \cup E'$. Beweisen oder widerlegen Sie folgende Aussagen:

i) $\hat{G}$ ist symmetrisch.

ii) Falls $\hat{G}$ stark zusammenhängend ist, dann ist G oder G^T stark zusammenhängend.

iii) Falls G oder G^T stark zusammenhängend ist, dann ist auch $\hat{G}$ stark zusammenhängend.

iv) G ist schwach zusammenhängend genau dann, wenn G^T schwach zusammenhängend ist.

f) Beweisen oder widerlegen Sie folgende Aussage: Jeder knotengewichteter DAG hat genau einen kritischen Pfad.

Lösung:

a) Es gibt $2^{|M|}$ viele Teilmengen einer endlichen Menge M. Für die Kanten eines gerichteten Graphen $G = (V, E)$ gilt, dass $E \subseteq V \times V = \{(u, v) \mid u, v \in V\}$. Mit $|V \times V| = |V| \cdot |V| = n^2$ folgt, dass es

$$2^{n^2}$$

viele gerichtete Graphen mit n Knoten gibt.

b) Für die Kanten eines ungerichteten Graphen $G = (V, E)$ gilt, dass $E \subseteq \{\{u, v\} \subseteq V \mid u \neq v\}$. Es gibt $\binom{m}{k}$ viele k-elementige Teilmengen einer m-elementigen Menge. Daher gibt es $\binom{n}{2}$ viele mögliche Kanten. Es gibt insgesamt also

$$2^{\binom{n}{2}} = 2^{\frac{n\cdot(n-1)}{2}} = 2^{\frac{n^2}{2}-\frac{n}{2}}$$

viele ungerichtete Graphen mit n Knoten.

Hinweise:

- Man kann bei a) und b) z.B. auch über die Anzahl verschiedener Adjazenzmatrizen argumentieren.

c) Bei einem vollständigen Graphen ist jede Folge von Knoten ein gültiger Pfad. Die Anzahl der Folgen mit k verschiedenen Knoten beträgt

$$\frac{n!}{(n-k-1)!} = \prod_{i=0}^{k}(n-i)$$

Hinweise:

- Beachten Sie, dass für den Fall $k > n$ das obige Produkt immer zu 0 auswertet, da einer der Faktoren 0 ist.

d) In einem vollständigen Graphen gilt für $k \geq 2$:

$$v_0 v_1 \ldots v_k \text{ ist ein einfachr Pfad} \iff v_0 v_1 \ldots v_k v_0 \text{ ist ein Zykel}$$

Wir können also jeden Pfad der Länge 2 (bzw. 3) zu einem Zykel der Länge 3 (bzw. 4) eindeutig zuordnen. Da Zykel in ungerichteten Graphen mindestens Länge 3 haben müssen folgt mit c): Die Anzahl der Zykel der Länge höchstens 4 beträgt

$$\begin{aligned}&\underbrace{n\cdot(n-1)\cdot(n-2)}_{\text{Zykel der Länge genau 3}} + \underbrace{n\cdot(n-1)\cdot(n-2)\cdot(n-3)}_{\text{Zykel der Länge genau 4}}\\ &= n\cdot(n-1)\cdot(n-2)\cdot(1+(n-3))\\ &= n\cdot(n-1)\cdot(n-2)^2\end{aligned}$$

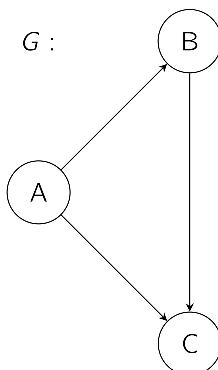
e) i) Die Aussage ist Wahr: Falls $(u, v) \in \hat{E}$ dann gibt es zwei Fälle:

- Falls $(u, v) \in E$ ist nach Definition von G^T auch $(v, u) \in E'$ und daher $(v, u) \in \hat{E}$.
- Falls $(u, v) \in E'$ ist nach Definition von G^T auch $(v, u) \in E$ und daher $(v, u) \in \hat{E}$.

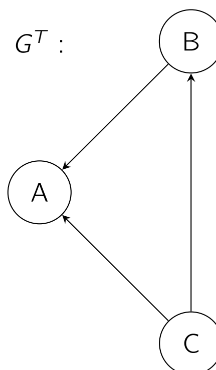
In beiden Fällen folgt also $(v, u) \in \hat{E}$.

ii) Die Aussage ist Falsch. Betrachte das folgende Gegenbeispiel. Weder G noch G^T sind stark zusammenhängend; $\hat{G}$ jedoch schon.

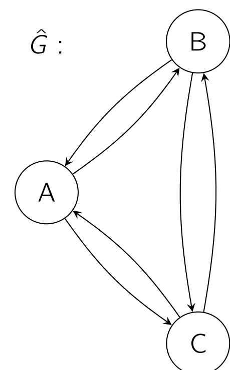
G :



G^T :



$\hat{G}$:



iii) Die Aussage ist wahr. Wir nehmen an, dass G stark zusammenhängend ist (der Fall dass G^T stark zusammenhängend ist ist analog[1]).

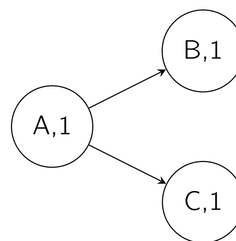
Angenommen $\hat{G}$ sei nicht stark zusammenhängend. Dann gibt es zwei Knoten $u, v \in V$, so dass u nicht von v erreicht werden kann (in $\hat{G}$). Da G stark zusammenhängend ist, gibt es jedoch einen Pfad $v_0 v_1 \dots v_k$ mit $v_0 = v$ und $v_k = u$ in G. Wegen $(v_i, v_{i+1}) \in E$ für alle $i \in \{0, \dots, k-1\}$ gilt auch $(v_i, v_{i+1}) \in \hat{E}$ für alle $i \in \{0, \dots, k-1\}$. Entsprechend ist $v_0 v_1 \dots v_k$ auch in $\hat{G}$ ein Pfad von v nach u. Widerspruch.

iv) Die Aussage ist Wahr.

" $\Longrightarrow$ ": Sei G schwach zusammenhängend und seien $v, u \in V$ zwei beliebige Knoten. Es gibt eine Folge $v_0 v_1 \dots v_k$ mit $v_0 = v$, $v_k = u$ und für alle $i \in \{0, \dots, k-1\}$ gilt $(v_i, v_{i+1}) \in E$ oder $(v_{i+1}, v_i) \in E$. Nach Definition von G^T gilt ebenfalls für alle $i \in \{0, \dots, k-1\}$, dass $(v_{i+1}, v_i) \in E'$ oder $(v_i, v_{i+1}) \in E'$. Es folgt, dass u von v auch in G^T über einen (ungerichteten) Pfad erreichbar ist. G^T ist also schwach zusammenhängend.

" $\Longleftarrow$ ": (analog)

f) Die Aussage ist Falsch! Betrachte das folgende Gegenbeispiel. Es gibt zwei kritische Pfade: AB und AC.



Aufgabe 2 (Tiefensuche): (4+4=8 Punkte)

Betrachten Sie folgende Implementierung einer Tiefensuche. Beachten Sie, dass der Graph als Adjazenzmatrix gegeben ist, wobei `adj[v][w]` genau dann wahr ist, wenn es eine Kante von `v` nach `w` gibt.

a) Bestimmen Sie die asymptotische Laufzeit der Methode `completeDFS` in Abhängigkeit der Knotenanzahl $n := |V|$ und der Kantenanzahl $m := |E|$ des gegebenen Graphen $G = (V, E)$. Nehmen Sie dazu an, dass nur **Vergleiche**, **Zuweisungen** und `print` **Anweisungen** jeweils eine Zeiteinheit benötigen. Begründen Sie ihre Antwort kurz.

b) Bestimmen und begründen Sie die asymptotische Laufzeit der Methode `completeDFS` analog zu Aufgabenteil a). Nehmen Sie dieses mal jedoch an, dass lediglich `print` **Anweisungen** eine Zeiteinheit benötigen.

```
1  void DFS(bool adj[n][n], int v, int &color[n]) {
2     print("Faerbe Knoten " + toString(v) + " grau.");
3     color[v] = GRAY;
4     for (int w = 0; w < n; w++) {
5        if (adj[v][w] == true) {
6           if (color[w] == WHITE) {
7              DFS(adj, w, color);
8           }
9        }
10    }
11    print("Faerbe Knoten " + toString(v) + " schwarz.");
12    color[v] = BLACK;
```

[1] Tatsächlich wurde in der Vorlesung gezeigt, dass G genau dann stark zusammenhängend ist, wenn G^T stark zusammenhängend ist.

```
13  }
14
15  void completeDFS(bool adj[n][n], int n) {
16      int color[n] = WHITE;
17      for (int v = 0; v < n; v++) {
18          if (color[v] == WHITE) {
19              DFS(adj, v, color);
20          }
21      }
22  }
```

Lösung:

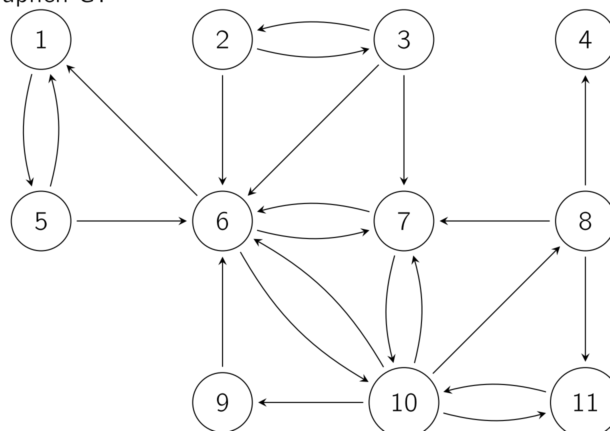
a) Die Laufzeit ist in $\Theta(n^2)$: Die Methode `DFS` wird für jeden der n Knoten **genau einmal ausgeführt**. Jeder einzelne dieser Aufrufe benötigt (ohne Berücksichtigung der rekursiven Aufrufe) $\Theta(n)$ Zeit. Die Methode `DFS` benötigt insgesamt also $\Theta(n^2)$ Zeit.

Die Methode `completeDFS` benötigt $\Theta(n + n^2) = \Theta(n^2)$ Zeit.

b) Jeder der n Knoten wird genau einmal grau und genau einmal schwarz gefärbt. Die Laufzeit beträgt daher $2 \cdot n \in \Theta(n)$.

Aufgabe 3 (SCCs in gerichteten Graphen): (13+3=16 Punkte)

Betrachten Sie folgenden Graphen G:



a) Wenden Sie den *Kosaraju-Sharir Algorithmus* aus der Vorlesung an, um die starken Zusammenhangskomponenten des Graphen G zu finden. Geben Sie dabei folgende Informationen an:

- In Phase 1 (Zeile 16 und 17) soll das Array `color` und der Stack `S` am Ende jeder Schleifeniteration, in der `DFS1` ausgeführt wurde, angegeben werden.
- In Phase 3 (Zeile 20 bis 23) soll das Array `color`, der Stack `S` und das Array `scc` am Ende jeder Schleifeniteration, in der `DFS2` ausgeführt wurde, angegeben werden.

Nehmen Sie hierbei an, dass `scc` initial mit Nullen gefüllt ist und die Knoten im Graphen und in allen Adjazenzlisten aufsteigend nach ihren Schlüsselwerten sortiert sind, also der Knoten mit Schlüssel 1 vom Algorithmus als erstes berücksichtig wird usw.

b) Geben Sie den Kondensationsgraph $G{\downarrow}$ an.

Lösung:

a) Phase 1:

S: 4, 11, 8, 9, 10, 7, 6, 5, 1

color: (1, s), (2, w), (3, w), (4, s), (5, s), (6, s), (7, s), (8, s), (9, s), (10, s), (11, s)

S: 4, 11, 8, 9, 10, 7, 6, 5, 1, 3, 2

color: (1, s), (2, s), (3, s), (4, s), (5, s), (6, s), (7, s), (8, s), (9, s), (10, s), (11, s)

Phase 3:

S: 4, 11, 8, 9, 10, 7, 6, 5, 1, 3

color: (1, w), (2, s), (3, s), (4, w), (5, w), (6, w), (7, w), (8, w), (9, w), (10, w), (11, w)

scc: (1, 0), (2, 2), (3, 2), (4, 0), (5, 0), (6, 0), (7, 0), (8, 0), (9, 0), (10, 0), (11, 0)

S: 4, 11, 8, 9, 10, 7, 6, 5

color: (1, s), (2, s), (3, s), (4, w), (5, s), (6, s), (7, s), (8, s), (9, s), (10, s), (11, s)

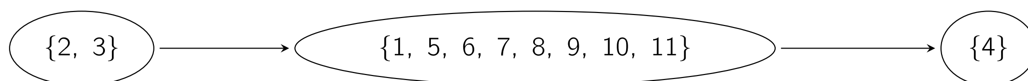
scc: (1, 1), (2, 2), (3, 2), (4, 0), (5, 1), (6, 1), (7, 1), (8, 1), (9, 1), (10, 1), (11, 1)

S:

color: (1, s), (2, s), (3, s), (4, s), (5, s), (6, s), (7, s), (8, s), (9, s), (10, s), (11, s)

scc: (1, 1), (2, 2), (3, 2), (4, 4), (5, 1), (6, 1), (7, 1), (8, 1), (9, 1), (10, 1), (11, 1)

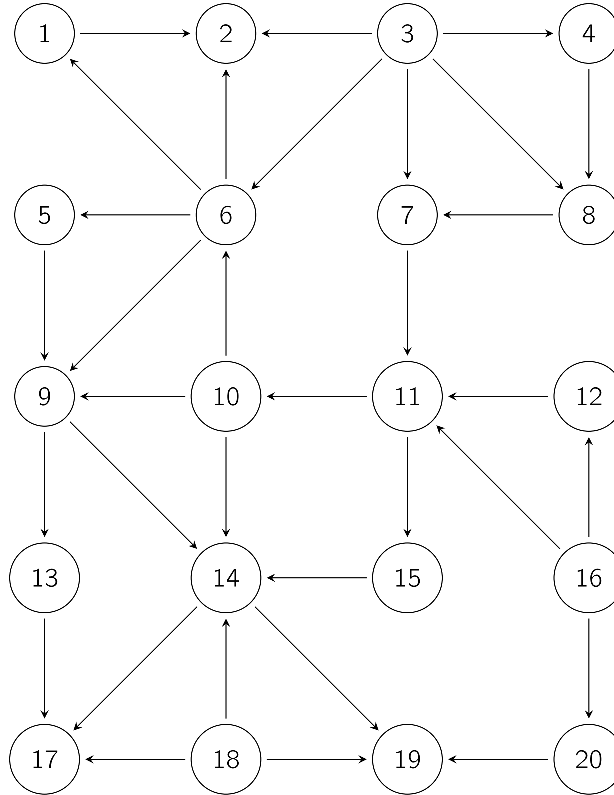
b) Der Kondensationsgraph $G{\downarrow}$ ist wie folgt:



Aufgabe 4 (Topologische Sortierung): (9+15=24 Punkte)

Wir betrachten den Algorithmus `topoSort` aus der Vorlesung (Vorlesung14+15, Folie 69).

a) Bestimmen Sie eine *topologische Sortierung* unter Verwendung des in der Vorlesung vorgestellten Algorithmus für den folgenden Graphen. Im gesamten Algorithmus werden Knoten in aufsteigender Reihenfolge ihrer Schlüssel berücksichtigt (d.h. die Adjazenzlisten sind aufsteigend nach Ihrem Knotenschlüssel sortiert). Als Ergebnis geben Sie an, welcher Knoten v welchen Topologiewert `topo`(v) erhält.



Ergebnis:

`topo`(v)	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
`Knoten` v	**2**	**1**																		

b) Geben Sie eine äquivalente Implementierung zum Algorithmus `topoSort` an, die keine Rekursion benutzt. Ihre Implementierung soll exakt die gleichen Topologiewerte generieren, wie die aus der Vorlesung bekannte Implementierung.

Hinweise:

- Sie können davon ausgehen, dass der gegebene Graph azyklisch ist.
- Sie dürfen die Funktionssignatur von `topoSort(List adj[n], int n, int &topo[n])` nicht verändern.
- Sie dürfen die Methode `List reverse(List l)` benutzen, welche die gegebene Liste `l` in umgekehrter Reihenfolge zurückgibt.

Lösung:

a) Ergebnis:

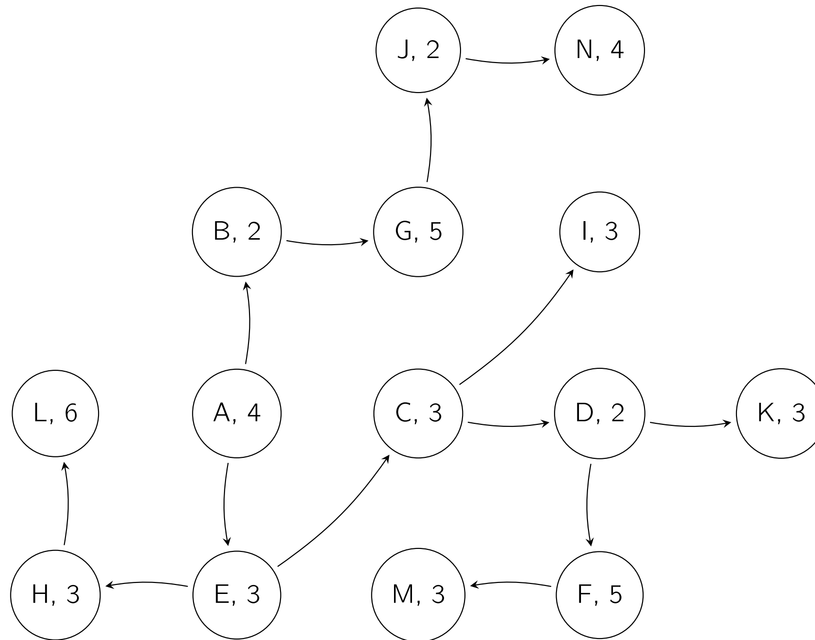
topo(v)	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
Knoten v	**2**	**1**	17	13	19	14	9	5	6	10	15	11	7	8	4	3	12	20	16	18

b) Eine Beispiellösung:

```
1  void DFS(List adj[n], int start, int &color[n], int &topoNum, int &topo[n]) {
2      Stack wait; // Enthaelt noch zu verarbeitende Knoten
3      wait.push(start);
4      while (!wait.isEmpty()) {
5          v = wait.pop();
6          if (color[v] == WHITE) {
7              // v wird zum ersten mal besucht: Untersuche Nachfolger von v
8              color[v] = GRAY;
9              // Um die Nachfolger in der gleichen Reihenfolge zu behandeln wie in der
10             // Implementierung aus der Vorlesung, drehen wir die Liste der Nachfolger um.
11             List successors = reverse(adj[v]);
12             // Fuege v wieder auf den Stapel ein, damit der Knoten nach Behandlung aller
13             // Nachfolger erneut betrachtet wird
14             wait.push(v);
15             foreach (w in successors) {
16                 // Da der Graph azyklisch ist, ist w immer WHITE
17                 wait.push(w);
18             }
19         } else if (color[v] == GRAY) {
20             // Alle Nachfolger wurden behandelt
21             color[v] = BLACK;
22             topo[v] = ++topoNum;
23         }
24     }
25 }
26
27 void topoSort(List adj[n], int n, int &topo[n]) {
28     int color[n] = WHITE; topoNum = 0;
29     for (int v = 0; v < n; v++) {
30         if (color[v] == WHITE) {
31             DFS(adj, v, color, topoNum, topo);
32         }
33     }
34 }
```

Aufgabe 5 (Kritischer Pfad): (16 Punkte)

Betrachten Sie den folgenden knotengewichteten Graphen mit Knotenmenge $V = \{A, B, \ldots, N\}$.



Nutzen Sie den in der Vorlesung vorgestellten Algorithmus um einen kritischen Pfad in obigen Graphen zu finden. Im gesamten Algorithmus werden dabei Knoten in aufsteigender alphabetischer Reihenfolge ihrer Schlüssel berücksichtigt.
Geben Sie die vom Algorithmus ermittelten frühesten Start- und Endzeitpunkte (`est` bzw. `eft`) sowie die kritische Abhängigkeit (`critDep`) für jeden Knoten an. Geben Sie außerdem an, in welcher Reihenfolge der Algorithmus die Knoten schwarz färbt. Geben Sie am Ende die ermittelte Gesamtdauer so wie den kritischen Pfad an.

Knoten	A	B	C	D	E	F	G	H	I	J	K	L	M	N
`est`														
`critDep`														
`eft`														

Reihenfolge Schwarzfärbung:

Gesamtdauer:

Kritischer Pfad:

Lösung: ______

______.

Knoten	A	B	C	D	E	F	G	H	I	J	K	L	M	N
`est`	16	11	10	8	13	3	6	6	0	4	0	0	0	0
`critDep`	E	G	D	F	C	M	J	L	-1	N	-1	-1	-1	-1
`eft`	**20**	13	13	10	16	8	11	9	3	6	3	6	3	4

Reihenfolge Schwarzfärbung: N, J, G, B, M, F, K, D, I, C, L, H, E, A
Gesamtdauer: 20
Kritischer Pfad: A E C D F M

Aufgabe 6 (Bipartite Graphen): (16 Punkte)

Ein ungerichteter Graph $G = (V, E)$ heißt **bipartit**, falls es zwei Mengen U und W gibt mit

- $U \cup W = V$
- $U \cap W = \emptyset$
- $E \subseteq \big\{\{v, v'\} \mid v \in U \text{ und } v' \in W\big\}$, d.h. es gibt keine Kanten zwischen zwei Knoten in U bzw. zwei Knoten in W.

Aufgabe: Schreiben sie eine Funktion `bool isBipartit(List adj[n])`, die für einen ungerichteten Graphen (gegeben als Adjazenzlisten) entscheidet, ob dieser bipartit ist. Ihre Funktion soll auf Breitensuche oder Tiefensuche basieren. Begründen Sie die Korrektheit Ihrer Funktion. Ein formaler Korrektheitsbeweis ist jedoch nicht notwendig.

Hinweise:

- Sie dürfen folgenden Satz aus der Globalübung verwenden:
  Satz: Ein ungerichteter Graph ist bipartit genau dann, wenn er keine Zykel ungerader Länge enthält.

Lösung:

Eine Version basierend auf **Breitensuche**:

```
1  bool BFS(List adj[n], int start, int &color[n]) {
2      Queue wait; // zu verarbeitende Knoten
3      wait.enqueue(start);
4      color[start] = RED;
5      while (!wait.isEmpty()) { // es gibt noch unverarbeitete Knoten
6          int v = wait.dequeue();
7          foreach (w in adj[v]) {
8              if (color[w] == WHITE) { // neuer ("ungefundener") Knoten
9                  // Faerbe Knoten 'abwechselnd' Rot bzw. Blau
10                 if (color[v] == RED) color[w] = BLUE;
11                 if (color[v] == BLUE) color[w] = RED;
12                 wait.enqueue(w);
13             } else if (color[v] == color[w]) {
14                 // Da w bereits gefunden wurde, muss es einen Pfad von v nach w geben,
15                 // der nicht die Kante (v,w) benutzt und der abwechselnd rote und blaue Knoten besucht.
16                 // Dieser Pfad zusammen mit der Kante (v,w) bilden einen Zykel ungerader Laenge.
17                 return false;
18             }
19         }
20     }
21     return true;
22 }
23
24 bool isBipartit(List adj[n]) {
25     int color[n] = WHITE;
26     for (int i = 0; i < n; i++) {
27         if (color[i] == WHITE) {
28             if (!BFS(adj, i, color)) return false;
29         }
30     }
31     // Wird diese Stelle erreicht, hat jeder Knoten entweder die Farbe Rot oder Blau.
32     // Ausserdem wurden alle Kanten untersucht und dabei wurde keine Kante zwischen
33     // zwei Knoten der gleichen Farbe gefunden.
34     return true;
35 }
```

Eine Version basierend auf **Tiefensuche**:

```
1  bool DFS(List adj[n], int v, int &color[n]) {
2     foreach (w in adj[v]) {
3        if (color[w] == WHITE) { // neuer ("ungefundener") Knoten
4           // Faerbe Knoten 'abwechselnd' Rot bzw. Blau
5           if (color[v] == RED) color[w] = BLUE;
6           if (color[v] == BLUE) color[w] = RED;
7           if (!DFS(adj, w, color)) return false;
8        } else if (color[v] == color[w]) {
9           // Da w bereits gefunden wurde, muss es einen Pfad von v nach w geben,
10          // der nicht die Kante (v,w) benutzt und der abwechselnd rote und blaue Knoten besucht.
11          // Dieser Pfad zusammen mit der Kante (v,w) bilden einen Zykel ungerader Laenge.
12          return false;
13       }
14 return true;
15 }
16
17 bool isBipartit(List adj[n]) {
18    int color[n] = WHITE;
19    for (int i = 0; i < n; i++) {
20       if (color[i] == WHITE) {
21          color[i] = RED;
22          if (!DFS(adj, i, color)) return false;
23       }
24    }
25    // Wird diese Stelle erreicht, hat jeder Knoten entweder die Farbe Rot oder Blau.
26    // Ausserdem wurden alle Kanten untersucht und dabei wurde keine Kante zwischen
27    // zwei Knoten der gleichen Farbe gefunden.
28    return true;
29 }
```