Prof. Dr. Ir. Joost-Pieter Katoen

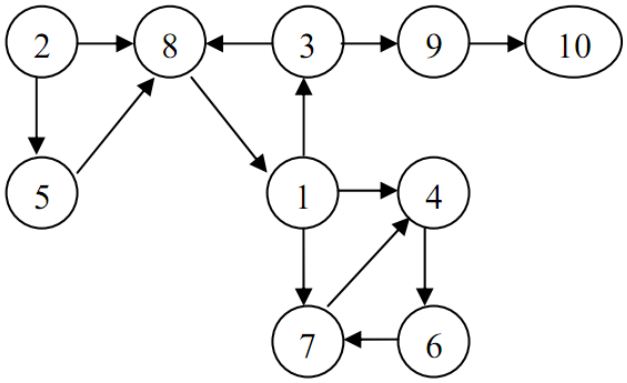
Jonathan Heinen, Sabrina von Styp

Hinweise:

- Die Übungsblätter sollen in Gruppen von je 3 Studierenden aus der gleichen Kleingruppenübung bearbeitet werden.
- Die Lösungen müssen bis Montag, den 28. Juni um 11:00 Uhr in den entsprechenden Übungskasten eingeworfen werden. Sie finden die Kästen am Eingang Halifaxstr. des Informatikzentrums (Ahornstr. 55).
- Namen und Matrikelnummern der Studierenden sowie die Nummer der Übungsgruppe sind auf jedes Blatt der Abgabe zu schreiben. Heften bzw. tackern Sie die Blätter!

Aufgabe 1 (Graphen): (4+4+8 Punkte)

Sei $G = (\{1, \ldots, 10\}, E)$ der folgende Graph:



a) Geben Sie den Graphen G als Adjazenzmatrix und in Adjazenzlistendarstellung an.

b) Führen Sie für den Graphen G die Breitensuche beginnend bei Knoten 1 durch. Geben Sie die Reihenfolge der Knotenbesuche an und heben Sie die entstehenden Baumkanten hervor.

c) Berechnen Sie mit Hilfe des in der Vorlesung vorgestellten Algorithmus die starken Zusammenhangskomponenten des obigen Graphen.

Aufgabe 2 (Backtracking): (4+10+4 Punkte)

Die Knoten eines ungerichteten Graphen sollen mit vier Farben so eingefärbt werden, dass zwei benachbarte Knoten (zwei Knoten, die durch eine Kante verbunden sind) nie die gleiche Farbe besitzen. Der Graph wird durch die Adjazenzmatrix `boolean[][] matrix` dargestellt. Das Array `int[] farbe` beinhaltet die Farbe, die dem jeweiligen Knoten zugewiesen ist. Die Farben werden durch die Zahlen 1, 2, 3 und 4 kodiert. Somit ist der Knoten `i` mit der Farbe `farbe[i]` eingefärbt. Mit der Zahl 0 kennzeichnen wir Knoten, die noch keine Färbung erhalten haben.

Hinweis: Knoten sind nie mit sich selbst über eine Kante verbunden.

a) Implementieren Sie die Java-Methode:

```
boolean test(boolean[][] matrix, int[] farbe),
```

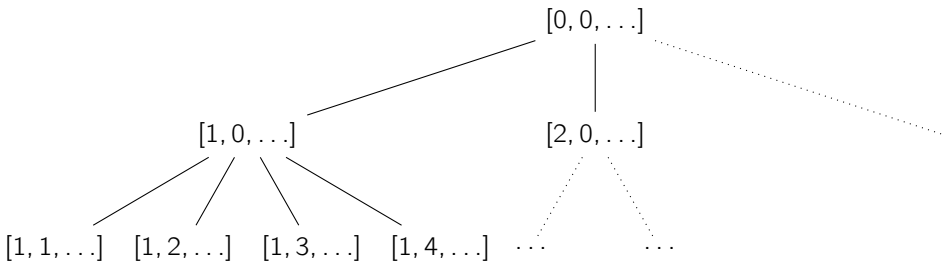
die testet, ob es sich um eine gültige Färbung handelt, d.h. durch Kanten verbundenen Knoten nie die gleiche Farbe besitzen.

b) In dieser Teilaufgabe soll nun ein Methode implementiert werden die, basierend auf Backtracking, einer gültige Färbung bestimmt.

Backtracking arbeitet nach dem Prinzip der Tiefensuche auf dem *Baum aller möglichen Lösungen*.

Ausgehend von einem vollständig ungefärbten Graphen wird ein Knoten nach dem anderen eingefärbt. Die Färbungen, die wir erhalten, indem wir einen weiteren Knoten einfärben, bezeichnen wir als Nachfolger. Für jeden Knoten haben wir vier mögliche Färbungen $(1 \ldots 4)$ und somit auch vier Nachfolger.

Diese Nachfolgerbeziehung, zusammen mit den vollständigen wie unvollständigen Färbungen, kann als Graph aufgefasst werden (siehe Abbildung). Es ergibt sich ein Baum, in dem die Wurzel den vollständig ungefärbten Graphen repräsentiert und jedes Blatt eine vollständige Färbung:



Eine gültige Färbung kann nun innerhalb dieses Baumes mit Hilfe der Tiefensuche gefunden werden. Hierbei wird der Baum nicht zuerst vollständig berechnet, sondern die Nachfolger werden während dem Durchlaufen berechnet.

Dies geschieht hier beim Absteigen durch das Hinzufügen einer Farbe für den nächsten (ungefärbten) Knoten, sowie beim Zurücklaufen (Backtracking) durch das Entfernen der Färbung des aktuellen (zuletzt gefärbten) Knoten. Implementieren Sie die Java-Methode

```
boolean loese(boolean[][] matrix, int[] farbe, int knoten),
```

die für den übergebenen Graphen eine gültige Färbung (ab Knoten n) berechnet, die entsprechend Tiefensuche den Baum traversiert und in den Blättern überprüft, ob es sich um eine gültige Färbung handelt. Existiert eine solche Färbung, so wird `true` zurückgegeben und `farbe` enthält eine gültige Lösung. Gibt es keine solche Färbung, so wird `false` zurückgegeben.

c) Während des Abstiegs im Baum werden Knoten besucht, die teilweise eingefärbte Graphen repräsentieren. Auch diese Teilfärbungen können bereits gleichgefärbte benachbarte Knoten besitzen. Ist dies der Fall, so sind alle vollständige Färbungen, die sich in den darunter liegenden Blättern befinden, ungültig und wir brauchen diesen gesamten Teilbaum nicht zu durchsuchen, sondern können unmittelbar im Baum zurückgehen (Backtracking-Schritt). Passen Sie Ihre Implementierung so an, dass beim bereits beim Erreichen einer unzulässigen Teilfärbung ein Backtracking-Schritt vollzogen wird, d.h. dass dieser Teilbaum nicht weiter durchsucht wird.

Hinweise:

1. Für die Tiefensuche im Baum ist es nicht nötig Knoten mit einer Farbe zu versehen, da sichergestellt ist, dass jeder Knoten nur einen, eindeutigen Vorgänger besitzt. Somit wird jeder Knoten nur exakt einmal besucht. Vergleichen Sie hierzu auch die Algorithmen zur Baumtraversierung aus der dritten Vorlesung, die der Tiefensuche entsprechen.
2. Im L2P, unter den Materialien zur Globalübung, finden Sie die Implementierung des Backtracking-Algorithmuses zur Lösung von Sudokus, der in der Globalübung am 21. Juni vorgestellt wurde.
3. Die in dieser Aufgabe zu implementierenden Algorithmen sollen in Java implementiert werden. Schicken Sie Ihren kompilierbaren und lauffähigen Quellcode per E-Mail an Ihren Tutor. Die E-Mail-Adresse Ihres Tutors finden Sie im L2P unter Teilnehmer.