Lösungen der Hausaufgaben von Übungsblatt 4

Algorithmen und Datenstrukturen (WS 2013, Ulrike von Luxburg)

Lösungen zu Aufgabe 1

(a) Nein. Zum Beispiel bei der Suche nach 8 im Array $[7, 8]$ ist anfangs $low = 0, high = 1$ und $mid = 0$. Danach wird wegen $7 < 8$ die untere Schranke auf $low := mid + 1 = 1 = high$ erhöht, dann aber aufgrund der Modifikation die Schleife abgebrochen, also 8 nicht gefunden.

(b) Eine Möglichkeit hierfür ist, die Ungleichungen in Zeile 6 und 8 umzudrehen.

(c) Der Beweis der Terminierung von Folie ≈ 20 gilt unverändert, da nach wie vor die Differenz $high - low$ in jedem Schleifendurchlauf um mindestens 1 verringert wird.

(d) Der Beweis ist analog zu den Folien $\approx$ 22ff. Die Invarianten sind nun $A[i] > value$ für alle $i < low$ und $value > A[i]$ für alle $i > high$. Der Rest gilt unverändert, außer der zweite kritische Fall: Dieser ist nun $A[low] = A[mid] > value$, aber wiederum wird nun low inkrementiert, so dass $low = high$ im nächsten Schleifendurchlauf gilt.

Lösungen zu Aufgabe 2

(a) (i) Es gibt generell nur eine mögliche 1-Färbung, nämlich $c_1 : V \to \{1\}, c_1(v) = 1$. Wir beweisen "$A \Leftrightarrow B$" per "$(B \Rightarrow A) \wedge (\neg B \Rightarrow \neg A)$": Enthält G tatsächlich keine Kante, so ist c_1 eine 1-Färbung, weil die Bedingung $i \sim j \Rightarrow c_1(i) \neq c_1(j)$ trivialerweise erfüllt ist (da $i \not\sim j$ für alle $i, j \in V$). Enthält G aber eine Kante (i, j), dann ist c_1 auch keine 1-Färbung, weil $c_1(i) = 1 = c_1(j)$.

(ii) Jede k-Färbung c_k kann ohne Änderung der Funktion auch als eine $k + 1$-Färbung $c_{k+1} := c_k$ aufgefasst werden, bei welcher die zusätzlich mögliche Farbe $k + 1$ einfach nicht genutzt wird (nicht im Bildbereich von c_{k+1} liegt). Trotzdem gilt weiterhin die Färbungseigenschaft $i \neq j \Rightarrow c_{k+1}(i) \neq c_{k+1}(j)$ wie bereits für c_k.

(iii) ObdA sei $V = \{1, \ldots, n\}$. Dann ist $c_n(i) = i$ offensichtlich eine n-Färbung, da jeder Knoten eine eigene Farbe hat.

(b) (i) "$A \Rightarrow B$": Wenn G bipartit bzgl. $V = V_1 \dot\cup V_2$ ist, dann ist c_2 mit $c_2(i) := \ell$ für $i \in V_\ell$ eine 2-Färbung, da jede Kante (i, j) nur zwischen V_1 und V_2 verläuft, also $1 = c_2(i) \neq c_2(j) = 2$ oder $2 = c_2(i) \neq c_2(j) = 1$ gilt. "$A \Leftarrow B$": Sei c_2 eine gültige 2-Färbung für G. Dann definiere $V_\ell := \{i \in V \mid c_2(i) = \ell\}$ mit offensichtlich $V_1 \dot\cup V_2 = V$. Aus der Färbungseigenschaft $i \sim j \Rightarrow c_2(i) \neq c_2(j)$ folgt dann, dass keine Kante "innerhalb" von V_1 bzw. V_2 liegt, sondern stets einen Knoten aus V_1 mit einem aus V_2 verbindet. Also ist G bipartit bzgl. V_1 und V_2.

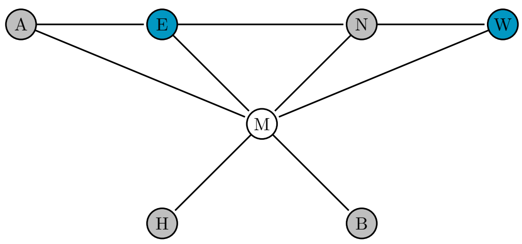
(ii) Folgender Pseudocode liefert das Gewünschte. Idee: Wähle irgendeinen Startknoten v und setze seine Farbe auf 1. Dann starte eine BFS und setze in jeder ungeraden Entfernung zu v die Farbe auf 2, in jeder geraden Entfernung zu v die Farbe auf 1, bis die aktuelle Zusammenhangskomponente komplett gefärbt ist. Fahre fort, bis alle Zusammenhangskomponenten gefärbt sind.

```
function Bipartite2Coloring(G)
    U ← V                                  ▷ U beinhaltet die noch ungefärbten Knoten
    while U ≠ ∅ do                         ▷ solange noch nicht alle Knoten gefärbt sind
        u ← ChooseAny(U)                   ▷ wähle irgendeinen ungefärbten Knoten u
        U ← U \ {u}                        ▷ entferne u aus U
        c_2(u) ← 1                         ▷ setze Farbe 1 für u
        Q ← [u]                            ▷ erstelle neue Liste aus u, genau wie bei BFS
        while Q ≠ ∅ do                     ▷ Nutze Q für BFS
            q ← Dequeue(Q)
            for x ∈ Adj(q) do              ▷ für jeden zu q benachbarten Knoten x
                if x ∈ U then              ▷ falls x noch ungefärbt ist
                    c_2(x) ← 3 − c_2(q)    ▷ setze andere Farbe 2 ↔ 1 für x als für q
                    U ← U \ {x}            ▷ entferne x aus U
                    Enqueue(Q, x)          ▷ füge x wie bei BFS hinzu
                end if
            end for
        end while
    end while
    return c_2
end function
```

(iii) Jede (ebenfalls bipartite) Zusammenhangskomponente kann unabhängig von allen anderen durch Vertauschen der Farben $1 \leftrightarrow 2$ auf genau 2 verschiedene Arten 2-gefärbt werden. Mit Z der Anzahl Zusammenhangskomponenten in G gibt es also insgesamt 2^Z verschiedene 2-Färbungen für G.

(c) Die Google-Suche nach "Landkarte Färbung minimal" o.ä. führt schnell zum *Vier-Farben-Satz*, siehe zum Beispiel Wikipedia. Damit ist $\ell = 4$.

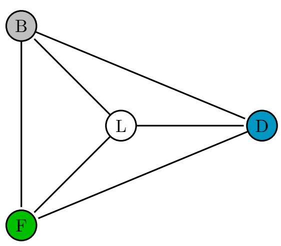
(i) Der Graph der Hamburger Bezirke sieht so aus:



(ii) Die eingezeichnete 3-Färbung ist $c_3(M) = 1$, $c_3(A) = c_3(N) = c_3(H) = c_3(B) = 2$, $c_3(E) = c_3(W) = 3$.

(iii) $\ell = 4$ bezeichnet die minimale *allgemeingültige* Anzahl Farben, nach welcher *jede* Landkarte 4-färbbar ist. Natürlich können *einzelne* Probleminstanzen darüber hinaus auch mit weniger Farben auskommen.

(iv) Zum Beispiel die Landkarte bestehend aus **L**uxemburg, **B**elgien, **F**rankreich und **D**eutschland, wie durch folgenden Graphen dargestellt:



Lösungen zu Aufgabe 3

(a) $G_1 : 1, 3, 4, 5, 2, 8, 6, 7$ und $G_2 : 1, 3, 5, 6, 4, 7, 2$

(b) $G_1 : 4, 3, 7, 6, 8, 2, 5, 1$ und $G_2 : 4, 6, 5, 3, 2, 7, 1$

(c) $G_1 : 1, 3, 5, 4, 2, 7, 8, 6$ und $G_2 : 1, 3, 4, 7, 5, 2, 6$

(d) Für G_1 existiert keine topologische Sortierung, da G_1 Zyklen enthält. In G_2 ist zum Beispiel $1, 7, 2, 3, 5, 6, 4$ eine topologische Sortierung.

(e) Die topologische Sortierung für G_2 ist gemäß Proposition 5 aus der Vorlesung eindeutig, da G_2 einen Hamilton-Pfad enthält (nämlich eben $1, 7, 2, 3, 5, 6, 4$), welcher jede andere Permutation verhindert.

(f) $G_1 : \{1, 2, 5, 6, 7, 8\}, \{3\}, \{4\}$ und $G_2 : \{1\}, \{2\}, \{3\}, \{4\}, \{5\}, \{6\}, \{7\}$.

Lösungen zu Aufgabe 4

(a) Bezeichne G den gemäß Aufgabenstellung konstruierten Graphen.

1: **function** ATTACKMCP(G)
2: $(V_1, \dots, V_N) \leftarrow SCC(G)$ ▷ Finde alle starken Zusammenhangskomponenten von G.
3: $\mathcal{E} \leftarrow \emptyset$ ▷ Initialisiere eine leere Kantenmenge für "Meta-Kanten" zwischen den V_i's.
4: **for** $i, j = 1 \dots N$ **do**
5: **if** $\exists u \in V_i, \exists v \in V_j : u \to v$ **then** ▷ Wenn irgendeine Kante aus V_i nach V_j führt.
6: $\mathcal{E} \leftarrow \mathcal{E} \cup \{V_i \to V_j\}$ ▷ Füge $\mathcal{E}$ eine Meta-Kante von V_i nach V_j hinzu.
7: **end if**
8: **end for**
9: $I \leftarrow \emptyset$ ▷ (minimale) Menge zu infiltrierender Module.
10: **for** $i = 1 \dots N$ **do**
11: **if** $InDegree(\mathcal{E}, V_i) = 0$ **then** ▷ Wenn keine Meta-Kante nach V_i führt.
12: $I \leftarrow I \cup$ CHOOSEANY(V_i) ▷ Füge I irgendeinen Knoten aus V_i hinzu.
13: **end if**
14: **end for**
15: INFILTRATEANDELIMINATE(I) ▷ Infiltriere die Module aus I und eliminiere MCP.
16: **end function**

Die Zeilen 1-8 konstruieren einen neuen "Meta-Graphen" $\mathcal{G} = (\{V_1, \dots, V_n\}, \mathcal{E})$, dessen Knoten die starken Zusammenhangskomponenten von G sind. Eine Kante $V_i \to V_j$ wird erstellt, wenn irgendein Knoten in V_j direkt abhängig von irgendeinem Knoten in V_i ist (und somit *jeder* Knoten in V_j direkt oder indirekt abhängig von *jedem* Knoten in V_i ist). Die Zeilen 9-14 bestimmen nun all diejenigen Zusammenhangskomponenten, die unabhängig von allen anderen sind, und wählt aus diesen jeweils ein beliebiges Modul aus. Zeile 15 führt schließlich die Infiltrierung aller Module aus I aus, sowie die abschließende Eliminierung des MCP.

(b) Jedes Modul $m \in V$ liegt in irgendeiner Zusammenhangskomponente V_i. Falls V_i in $\mathcal{G}$ keine eingehende Kante hat, so wird in Zeilen 11-12 irgendein Modul aus V_i zur Infiltrierung ausgewählt, wodurch aufgrund des starken Zusammenhangs in V_i auch m direkt oder indirekt eliminiert wird. Falls hingegen $V_k \to V_i$ existiert, so wird m direkt oder indirekt von allen Modulen aus V_k heraus erreicht. Induktiv lässt sich dies bis auf eine unabhängige Zusammenhangskomponente zurückführen, welche wieder dem ersten Fall genügt und infiltriert ist. Also wird letztlich *jedes* Modul aus G eliminiert.

(c) Um alle Module auszuschalten, *muss* aus jeder Zusammenhangskomponente, die nicht von einer anderen abhängt, mindestens ein Modul zur Infiltrierung ausgewählt werden. Andernfalls würde keines der Module dieser Zusammenhangskomponente infiltriert, und somit das MCP nicht komplett eliminiert. Da $|I|$ der Anzahl solcher unabhängigen Zusammenhangskomponenten gleicht, ist $|I|$ minimal.

Lösungen zu Aufgabe 5

Die Aufgabe ist eine Formulierung des sogenannten *Collatz-Problems* und ist ein ungelöstes Problem der Mathematik. Wer es löst, darf statt "nur" mit 99 Bonuspunkten vielmehr mit einer sofortigen Professoren-Stelle an einer beliebigen Universität auf der Welt rechnen...