AD [HA] zum 4. 12. 2013

Arne Struck, Lars Thoms

3. Dezember 2013

1. a)

Der Algorithmus funktioniert nicht mehr. Dies wird an folgendem Gegenbeispiel deutlich (gesucht wird 5, die Zeile ist abgeändert):

```
in: A[int] = [1,2,3,4,5]
low = 0, high = 4, mid = 2
A[mid] < 5

low = 2 + 1, high = 4, mid = 3
A[mid] < 5

low = 3 + 1, high = 4, mid = 4

return not_found
```

Die 5 wird nicht gefunden, da die Bedingung (while low $<$ high) nicht mehr gilt.

b)

Die Verarbeitung einer absteigend sortierten Liste lässt sich durch das Invertieren der Vergleichszeichen innerhalb der while-Schleife erreichen, wodurch die Verarbeitung des Algorithmus ”umgedreht” wird.

```
BinarySearch(A[0..N-1],value){
    low = 0;
    high = N - 1;
    while(low <= high){
        mid = (low + high) / 2;
        if(A[mid] < value){
            high = mid - 1;
        }
        else if (A[mid] > value){
             low = mid + 1;
        }
        else{
            return mid;
        }
    }
    return not_found;
}
```

c)

Korrekte Eingaben vorausgesetzt, ist es unumgänglich die while-Schleife zu betreten, hier gilt low <= mid <= high. Nun wird die Differenz von low und high pro Iteration um mindestens 1 verringert, wenn der Algorithmus noch nicht terminiert hat. Die Verringerung der Differenz resultiert aus den beiden Vergleichen. Daraus resultiert, dass nach spätestens n (n = Array- Länge) Iterationen low = mid = high gilt, womit der Wert A[mid] ausgegeben wird. Damit ist bei korrekter Eingabe Terminierung garantiert.

d)

2. a) (i)

Zu zeigen wären 2 Behauptungen:

$$E = \emptyset \Rightarrow 1 - \text{färbbar} \qquad E = \emptyset \Leftarrow 1 - \text{färbbar}$$

Da keine Kante existiert, besitzt kein Knoten einen Nachbarknoten. Also kann auch jeder Knoten in der gleichen Farbe gefärbt werden

Wenn alle Knoten in der gleichen Farbe gefärbt sind, können auch keine Nachbarknoten existieren, da diese nicht gleich eingefärbt werden dürfen. Damit ist gezeigt, dass im Fall der 1-Färbung keine Kanten existieren können.

(ii)

Zu zeigen ist also, wenn eine Abbildung $c_k : V \to \{1, .., k\}$ existiert muss auch eine Abbildung $c_k : V \to \{1, .., k, k+1\}$ existieren. Da c_k nicht surjektiv sein muss, kann jeder Graph trivialer Weise als (k+1)-färbbar angesehen werden (es müssen ja nicht alle Färbungen Anwendung finden). Wäre dem nicht so, dann wären injektive Abbildungen ein Problem bei der (k+1)-Färbung.

(iii)

Annahme: $n = |V|$ (n wird nicht genauer spezifiziert)
Für jedes $n \in V$ wird eine Farbe in c_k reserviert (wenn $k < n$ gilt, werden neue Farben hinzugefügt). Nun wird eine injektive Abbildung erstellt (bijektiv, wenn zuvor $k < n$ und jetzt $k = n$). Damit ist jedes n einer eigenen Farbe zugeordnet und somit eine n-Färbung erreicht.

b) (i)

Zu zeigen: $2 - \text{färbbar} \Rightarrow$ bipartit.
Die 2-Färbung bedeutet, dass 2 "Gruppen" von Knoten keine direkten Nachbarknoten in der gleichen "Gruppe" haben (ansonsten könnten sie nicht gleich gefärbt sein). Diese "Gruppen" kann man auch als Mengen auffassen. Damit ist die Definition von bipartiten Graphen hergestellt, da diese einen Graphen in 2 Mengen unterteilen, wobei die Elemente der Teilmengen dieser 2 Mengen nicht miteinander durch Kanten verbunden sind.

(ii)

```
2colored(V){
    Set1 = Set;
    Set2 = Set;
    color1 = randomElement(V).getcolor();
    color2 = color1;
    while(color1 = color2){
        color2 = randomElement(V).getcolor();
    }
    forAll(v in V){
        if(v.getcolor() = color1){
            Set1.add(v);
        }
        else if (v.getcolor() = color1){
            Set2.add(v);
        }
        else{
            return no_2color;
        }
    }
    return Set1,Set2;
}
```

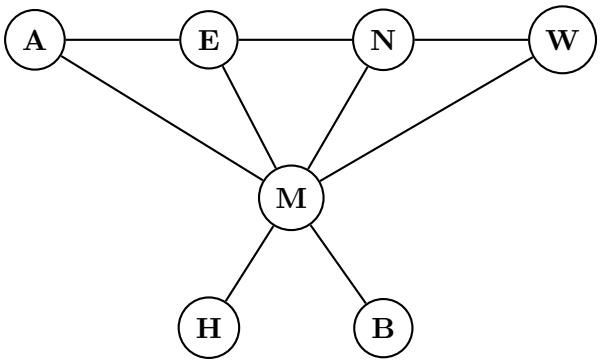
(iii)

Sofern der Graph zusammenhängend ist (und es sich um eine echte 2-Färbung handelt, also nur 2 Farben vorhanden sind) existieren 2 mögliche Färbungen (die Farben werden vertauscht).
Ansonsten hängt die Anzahl der Färbungen von der Anzahl der Zusammenhangskomponenten (n) ab. Man kann das ganze Problem als Binärbaum auffassen. Die Höhe des Baumes entspricht der Anzahl der Zusammenhangskomponenten. Jede Färbung würde dann einem Blatt entsprechen. 2^n Wäre damit die Anzahl der Blätter in der n-ten Ebene, also bei n Zusammenhangskomponenten.

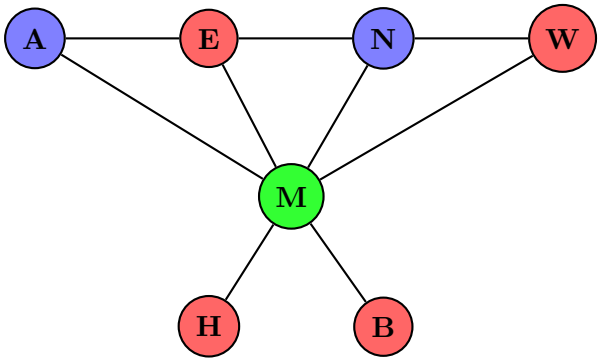
c)

Es sind laut dem Vier-Farben-Satz 4 Farben vonnöten, um eine Karte komplett einzufärben.

(i)



(ii)

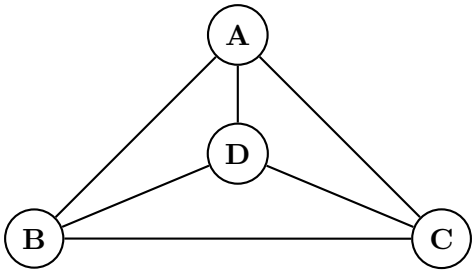


Farbe1 = {A,N}
Farbe2 = {E, W, H, B}
Farbe3 = {M}

(iii)

Der Vier-Farben-Satz sagt aus, dass 4 Farben minimal vorhanden sein müssen, um jede Karte vollständig zu färben (worst-case). Der Satz macht keine Aussage darüber, dass nicht Karten existieren, welche mit weniger auskommen.

(iv)



Ein Land ist von 3 anderen Ländern umgeben. Wenn man hier das Einfärben beginnt kann man 3 Ländern je eine Farbe zuordnen. Das letzte Land kann nicht eine der 3 Farben erhalten, da seine Nachbarländer schon je eine der 3 Farben innehaben.

3. a)

G_1 1, 3, 4, 5, 2, 8, 6, 7

G_2 1, 3, 5, 6, 4, 2, 7

b)

G_1 4, 3, 1, 7, 6, 8, 2, 5

G_2 4, 6, 5, 3, 1, 2, 7

c)

G_1 1, 3, 5, 4, 2, 7, 8, 6

G_2 1, 3, 4, 7, 5, 2, 6

d)

G_1 Für diesen Graphen kann keine topologische Sortierung existieren, da er sowohl einen reflexiven Knoten enthält (8, Verstoß gegen die Bedingung der Irreflexität) und mehrere Zyklen enthält ([1,5], [7,8], [1,5,2], ...)

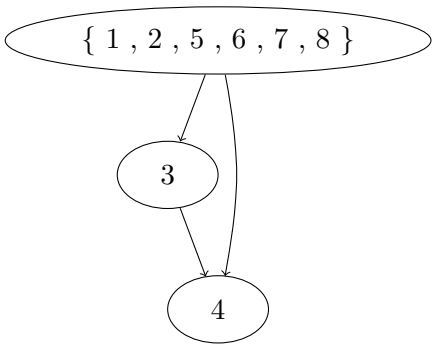
G_2 1, 7, 2, 3, 5, 6, 4

e)

Die von uns gefundene topologische Sortierung ist nicht eindeutig. Damit eine solche Sortierung eindeutig ist, muss sie einen Hamiltonkreis aufweisen. Dieser Graph kann aber keinen Hamiltonkreis beinhalten, da allein schon der Knoten "1" drei ausgehende Kanten besitzt.

f)

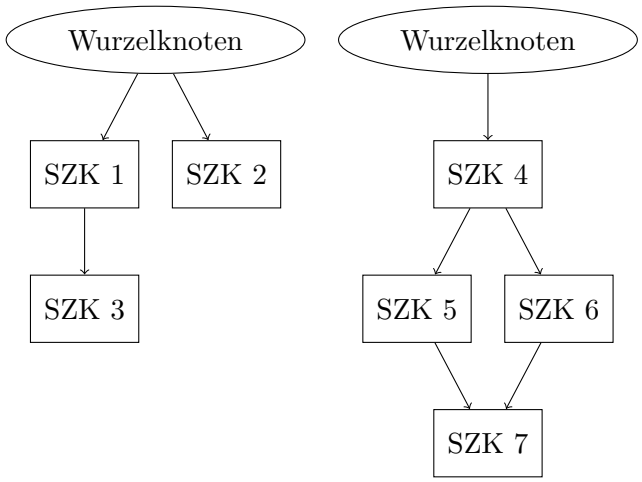
Der Graph "G_2" besitzt keine starken Zusammenhangskomponenten, da er keinerlei Zyklen aufweist.
Nur der Graph "G_1" besitzt eine starke Zusammenhangskomponente, die Knoten "3" oder "4" bilden keine starke Zusammenhangskomponente.



4. a)

Zuerst müssen wir alle starken Zusammhangskomponenten in den Graphen finden. Dafür benutzen wir den Algorithmus von Tarjan. Nachdem wir alle Komponenten gefunden haben, finden wir das Wurzelelement jeder Komponente. Duplikate werden mit Sicherheit entstehen, aber die können wir ohne Probleme eleminieren.
Danach müssen wir nur noch die restlichen Wurzelelemente infiltrieren und dann wird das komplette MCP lahmgelegt.

```
include "Tarjan's Algorithm";

List l, r;

while(nodes.flags > 0)
{
    l.append(tarjan.find());
}

foreach(l as elem)
{
    r.append(find_rootnode(elem));
}

r.remove_duplicates();

infiltrate(r);
```



b)

Dadurch, dass alle Wurzelelemente eines gerichteten Graphen infiltriert werden, muss das komplette MCP eleminiert werden. Denn sobald Wurzelelement infiltriert ist, fallen alle Module aus, die davon anhängig sind. Da es sich hier um die Wurzel handelt, fallen alle Module unter diesem Modul aus.

c)

Durch die Bildung von stark zusammhängenden Modulkomponenten wurde der Graph maximal abstrahiert. Und durch die Infiltration von Wurzelknoten ist es möglich, einen kompletten Graphen zu eleminieren. Eine weitere Minimierung ist nicht mehr möglich.