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)

TODO

c) (i)

TODO

(ii)

TODO

(iii)

TODO

(iv)

TODO

3. a)

TODO

b)

TODO

c)

TODO

d)

TODO

e)

TODO

f)
TODO

4. a)
TODO

b)
TODO

c)
TODO