Image Analysis Lab
Jun.-Prof. Dr.
Olaf Ronneberger
Claudius Korzen

Algorithmen und Datenstrukturen
(ESE + IEMS) WS 2013 / 2014

http://lmb.informatik.uni-freiburg.de/lectures/

UNI FREIBURG

Musterlösung Klausur

Freitag, 28. Februar 2014, 16.00 Uhr - 18:00 Uhr, Gebäude 082, Raum 00-006 (Kinohörsaal)

Aufgabe 1 (Heaps, 15 Punkte)

Aufgabe 2 (Hashing & Sortieren, 20 Punkte)

2.1 (4 Punkte)

$$
\begin{aligned}
0 &\mapsto \{7\} \\
1 &\mapsto \varnothing \\
2 &\mapsto \{3,\ 10\} \\
3 &\mapsto \{1,\ 8,\ 15\} \\
4 &\mapsto \varnothing \\
5 &\mapsto \varnothing \\
6 &\mapsto \{2,\ 16\}
\end{aligned}
$$

2.2 (8 Punkte)

H ist $\frac{7}{6}$-universell wenn für alle $x, y \in U$ mit $x \neq y$ gilt:

$$\frac{|\{h \in H : h(x) = h(y)\}|}{|H|} \leq \frac{7}{6} \cdot \frac{1}{m}$$

$$\Leftrightarrow \frac{|\{h \in H : h(x) = h(y)\}|}{6} \leq \frac{1}{6}$$

$$\Leftrightarrow |\{h \in H : h(x) = h(y)\}| \leq 1$$

Jedes $h \in H$ bildet x und $y = x + 7i$ ($i \in \mathbb{N}$) auf den selben Wert ab. In diesem Fall gilt: $|\{h \in H : h(x) = h(y)\}| = 6$. Das verletzt aber die Bedingung an die $\frac{7}{6}$-Universalität. Da dieser Fall bereits für $x = 1$ und $y = 8$ ($8 = 1 + 7 \cdot 1$) auftreten kann, muss demzufolge $n \leq 7$ sein.

2.3 (8 Punkte)

```
int[] reduceAndSort(int[] numbers) {
  HashSet<Integer> hash = new HashSet<Integer>();
  List<Integer> reduced = new ArrayList<Integer>();
  // Reduce the list of numbers.
  for (int i = 0; i < numbers.length; i++) {
   int number = numbers[i];
   if (!hash.contains(number)) {
     reduced.add(number);
     hash.add(number);
   }
  }
  // Sort the numbers numerically.
  Collections.sort(reduced);
  return reduced.toArray();
}
```

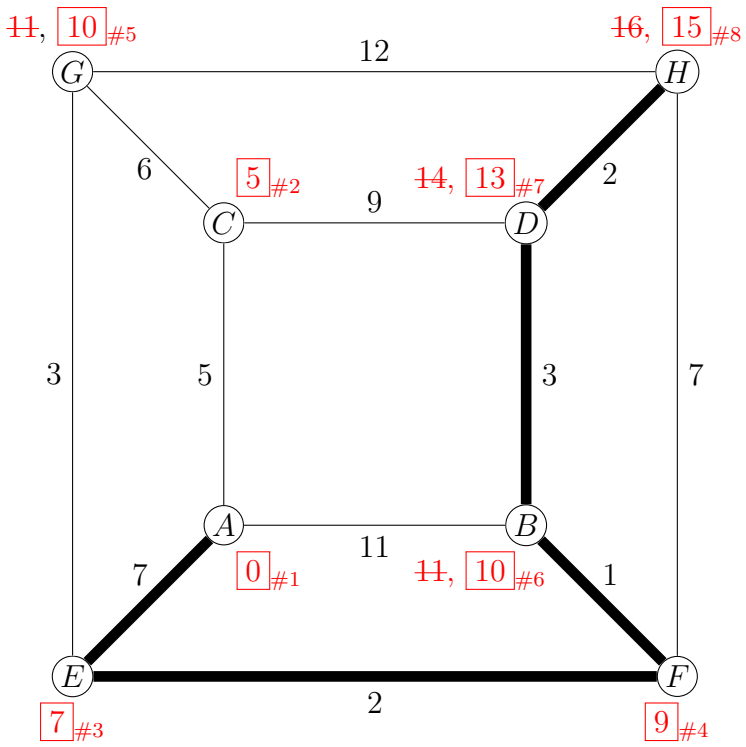
Anmerkung: Statt der HashSet kann auch eine HashMap verwendet werden (wie in der Aufgabenstellung vorgeschlagen), in der z.B. jede Nummer auf sich selbst abgebildet wird oder auf die Anzahl der Vorkommen, etc.

Einfügen und Lookup in HashSet/HashMap kostet $\mathcal{O}(1)$. Das Reduzieren der Liste mit n Elementen kostet also insgesamt $\mathcal{O}(n)$. Die reduzierte Liste enthält nur noch k Elemente; das Sortieren kostet also $\mathcal{O}(k \cdot \log k)$. (Laufzeit von `reduced.toArray()`: $\mathcal{O}(k)$).

Die Laufzeit des gesamten Programms ist somit: $\mathcal{O}(n + k \log k)$.

Aufgabe 3 (Graphen, 20 Punkte)

3.1 (5 Punkte)



X $_{\#i}$ = gelöst in Runde i.

3.2 (5 Punkte)

Die Behauptung stimmt nicht, da es zwischen dem billigsten und dem zweitbilligsten Knoten einen Pfad über mehrere Kanten geben kann, über den der zweitbilligste Knoten mit geringeren Kosten erreicht werden könnte.

3.3 (10 Punkte)

```
int computeLocation(Graph G, int i) {
  int minAvgTime = Integer.MAX_VALUE;
  int index = -1;
  for (int i = 0; i < G.numNodes(); i++) {
    int sum = 0;
    // Compute travel times between i and all other nodes.
    int[] distances = computeDijkstra(G, i);
    // Compute the average travel time.
    for (int j = 0; j < distances.length; j++) {
      sum += distances[j];
    }
    int avgTime = sum / distances.length;
    // Take the avgTime, if it is smaller than minAvgTime.
    if (avgTime < minAvgTime) {
      minAvgTime = avgTime;
      index = i;
    }
  }
  return index;
}
```

Anmerkung: Statt die durchschnittliche Anfahrtszeit `avgTime` zu berechnen, reicht es auch, nur die Summe der Anfahrtszeiten `sum` zu berechnen. Da `distances.length` konstant ist, ändert das nichts am Endergebnis.

Aufgabe 4 ($\mathcal{O}$-Notation, Vollständige Induktion, 25 Punkte)