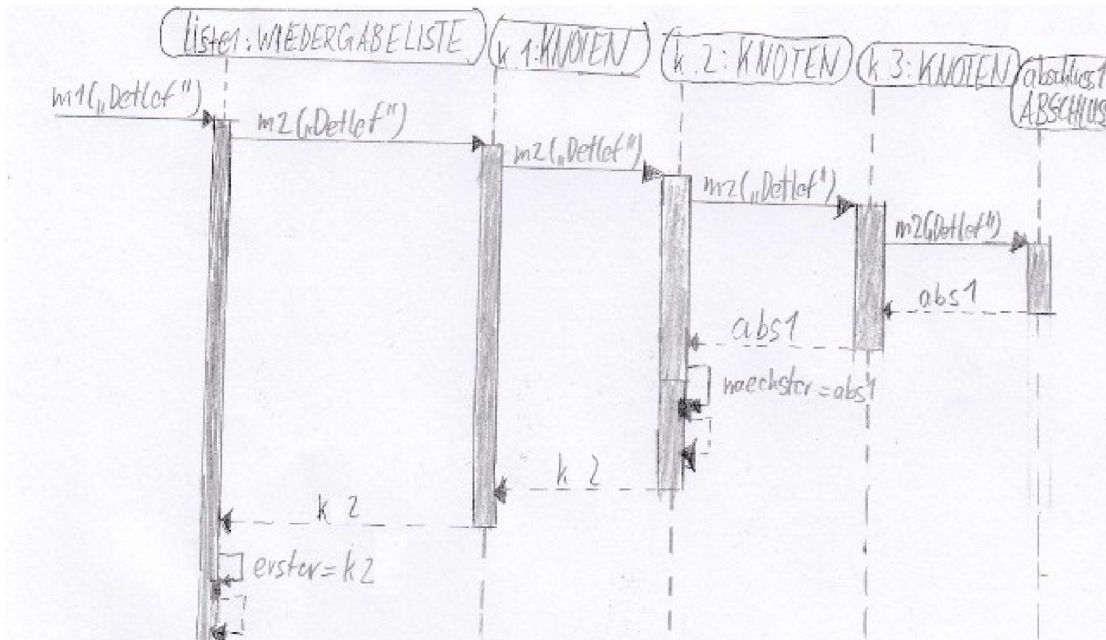
Informatik Abitur Bayern 2017 / II - Lösung

Autoren:
Wolf (1)
Wagner (2)
Scharnagl (3-5)

1a 5

1b Diese Methode vergleicht den Namen des Interpreten eines jeden Elements der Liste mit dem gegebenen Namen. Stimmen diese überein wird das Element mit dem folgenden Element der Liste überschrieben. Insgesamt löscht diese Methode dadurch alle Einträge aus der Liste bei denen der Name des Interpreten "Detlef" ist. 9



1c In der Klasse LISTENELEMENT: 7

```
//Methode zur Berechnung der Gesamtdauer wird in den Unterklassen
deklariert.
abstract int gesamtdauerGeben();
```

In der Klasse KNOTEN:

```
int gesamtdauerGeben(){
/*rekursive Methode, die immer wieder die schon aufsummierte Gesamtdauer
* plus die Dauer des Lieds des aktuellen Knotens wiedergibt
*/
      return naechster.gesamtdauerGeben() + inhalt.dauerGeben();
}
```


Dieses Werk ist lizenziert unter einer Creative Commons Namensnennung - Nicht-kommerziell - Weitergabe unter gleichen Bedingungen 4.0 International Lizenz.

In der Klasse ABSCHLUSS:

```
int gesamtdauerGeben(){
//Gibt für den Abschluss die Dauer 0 zurueck
      return 0;
}
```

In der Klasse WIEDERGABELISTE:

```
int gesamtdauerGeben(){
//ruft für das erste Listenelement die Methode gesamtdauerGeben() auf
      return erster.gesamtdauerGeben();
}
```

2a *Grundidee:* 15

Jeder Knoten kann für seinen Nachfolger überprüfen, ob der neue Song häufiger gespielt wurde als der Nachfolger. Wenn das der Fall ist, ruft er die Methode sortiertEinfuegen(song) *seines Nachfolgers auf (Rekursion). Falls nicht, wird ein neuer Knoten erzeugt, der aktuelle Nachfolger als dessen Nachfolger und schließlich der neue Knoten als Nachfolger des aktuellen Knotens gesetzt. Falls der neue Knoten vor keinem der vorhandenen Knoten gesetzt werden kann, wird die Methode* sortiertEinfuegen(song) *der Klasse* ABSCHLUSS *aufgerufen und der Knoten wird der neue letzte Knoten.*

```
Klasse SONG:
public boolean istHaeufigerAls(SONG andererSong) {
   if (gespielt > andererSong.gespielt) {
         return true;
      } else {
         return false;
      }
}
```

Kurzversion:

```
public boolean istHaeufigerAls(SONG andererSong) {
   return (gespielt > andererSong.gespielt);
}
```

Folgende Methode soll zur Klasse *LISTENELEMENT* hinzugefügt werden:

```
public abstract void sortiertEinfuegen (SONG song);
```

Folgender Methoden soll in der Klasse *KNOTEN* implementiert werden:

```
public KNOTEN(SONG s) {
   inhalt = s;
}
public void naechstenSetzen(KNOTEN k) {
   naechster = k;
}
```


Dieses Werk ist lizenziert unter einer Creative Commons Namensnennung - Nicht-kommerziell - Weitergabe unter gleichen Bedingungen 4.0 International Lizenz.

```
public void sortiertEinfuegen (SONG song) {
   if (naechster.istHaeufigerAls(song)) {
         naechster.sortiertEinfuegen(song);
       } else {
          KNOTEN k = new KNOTEN(song);
          k.naechstenSetzen(naechster);
          naechster = k;
       }
}
```

Folgende Methode soll in die Klasse *WIEDERGABELISTE* eingefügt werden:

```
public void sortiertEinfuegen(SONG song) {
   if (erster.istHaeufigerAls(song)) {
      erster.sortiertEinfuegen (song);
   } else {
      KNOTEN k = new KNOTEN(song);
      k.naechstenSetzen(erster);
      erster = k;
       }
}
```

2b 8

```
wenn (liste.istEnthalten(song)) {liste.entfernen(song);};
liste.sortiertEinfuegen(song);
solange(liste.anzahlGeben()>20) {liste.entfernen(20);};
```

Zuerst wird mithilfe von istEnthalten(song) geprüft, ob der übergebene Song song bereits in der Wiedergabeliste enthalten ist. Wenn das der Fall ist, wird sein Attribut gespielt um eins erhöht und der Methodenaufruf ist beendet. Andernfalls wird zunächst anzahlGeben() aufgerufen, um die aktuelle Anzahl der Songs in der Wiedergabeliste zu bestimmen. Falls sich bereits 20 Songs in der Wiedergabeliste befinden, wird der letzte Song mithilfe von entfernen(19) von der Wiedergabeliste entfernt. Anschließend wird der übergebene Song song mittels sortiertEinfuegen(song) an die korrekte Position in die Liste eingefügt und der Methodenaufruf ist beendet.

Es ist auch möglich, den neuen Song sortiert einzufügen und anschließend den 21. Knoten mittels entfernen(20) (die Zählung beginn bei 0) zu löschen. Wichtig ist hierbei: Kein Song darf doppelt vorkommen und die Wiedergabeliste darf nicht mehr als 20 Songs beinhalten.

2c Pro: Die Implementierung mithilfe eines Feldes ist simpler und daher schneller zu realisieren. 4
Contra: Das Einfügen eines neuen Songs innerhalb der Wiedergabeliste ist sehr aufwendig.

weitere Argumente:

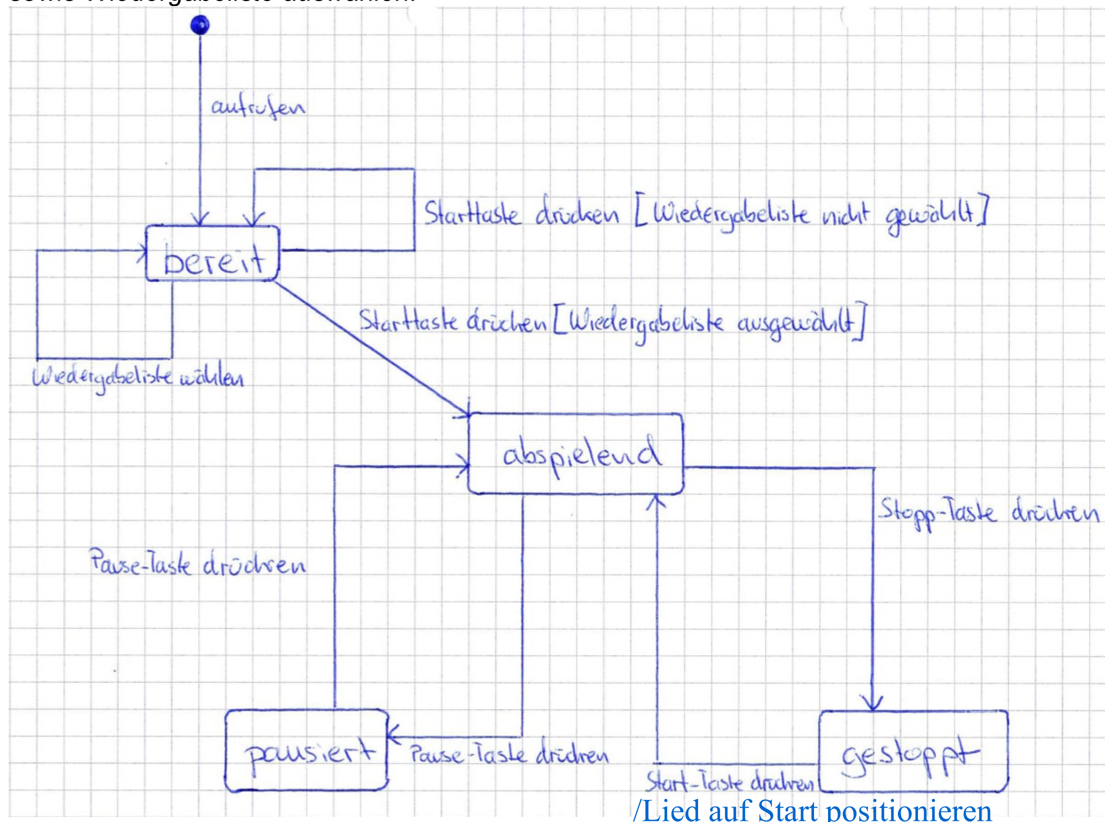
Das Löschen eines Songs innerhalb der Wiedergabeliste ist sehr aufwendig (alle anderen müssen nachrücken).
Eine Erweiterung der Wiedergabeliste auf mehr Plätze ist je nach Programmiersprache sehr aufwendig.

Dieses Werk ist lizenziert unter einer Creative Commons Namensnennung - Nicht-kommerziell - Weitergabe unter gleichen Bedingungen 4.0 International Lizenz.

Hintergrundinformation:

Die Geschwindigkeit von Erstell-, Einfüge-, Such- und Löschoperationen auf Datenstrukturen ist abhängig von der verwendeten Programmiersprache und im Falle von Java auch von der Implementierung der virtuellen Maschine, auf der das Java-Programm ausgeführt wird. Felder sind oft ein wenig schneller, was diese Operationen angeht, jedoch sind sie (aus oben genannten Gründen) sehr unflexibel.

3 *Vorüberlegung: Aus der Aufgabenstellung werden die vier Zustände bereit*, *abspielend*, *pausiert* und *gestoppt* herausgelesen. Dazu sollte man noch einen Startzustand einfügen, von dem die App in den Zustand *bereit* übergeht. Als Aktionen, die Übergänge zwischen den Zuständen auslösen können, dient einerseits das Drücken der vorhandenen Tasten: *Start-Taste*, *Stop-Taste* und *Pause-Taste*. Zusätzlich gibt es noch die Aktionen *aufrufen* (der App), sowie *Wiedergabeliste auswählen.* 8

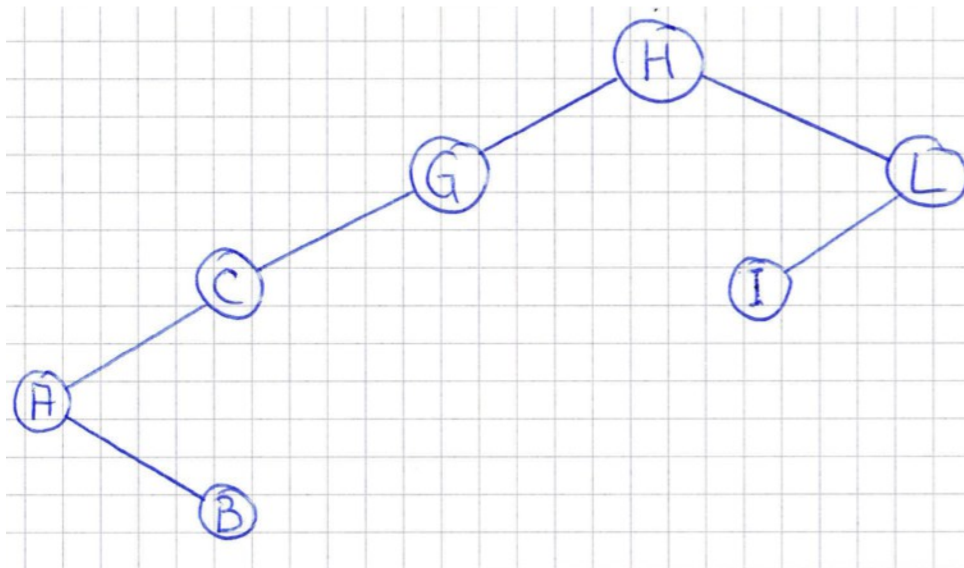


4a Da der Binärbaum lexikographisch geordnet ist, müssen die Songs nach dem Alphabet eingeordnet werden: 3

- Elemente, die dem Alphabet nach weiter hinten sind, sind im rechten Teilbaum.
- Elemente, die dem Alphabet nach weiter vorne sind, sind im linken Teilbaum.

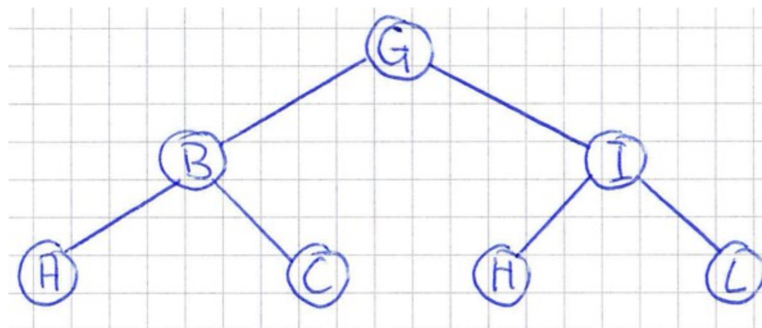
4b 6

Dieses Werk ist lizenziert unter einer Creative Commons Namensnennung - Nicht-kommerziell - Weitergabe unter gleichen Bedingungen 4.0 International Lizenz.



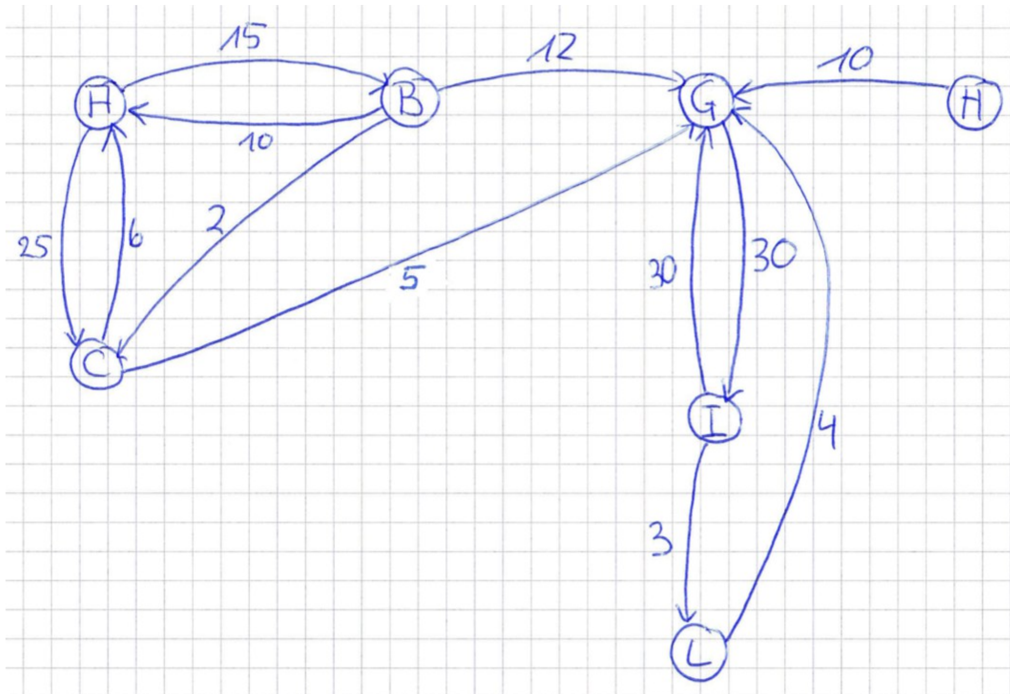
Der so entstandene Baum ist hinsichtlich der Suche nicht optimal, da er nicht balanciert ist. Ein balancierter Baum hätte immer die gleiche Höhe (maximal 1 Unterschied, wenn die unterste Ebene nicht komplett gefüllt ist) von jedem Blatt zur Wurzel.

Optimierter Baum:



5a Für einen ungerichteten Graphen müsste die Adjazenzmatrix an der Hauptdiagonalen (von links oben nach rechts unten) gespiegelt sein. 6

Dieses Werk ist lizenziert unter einer Creative Commons Namensnennung - Nicht-kommerziell - Weitergabe unter gleichen Bedingungen 4.0 International Lizenz.



5b

```
Algo(KNOTEN aktuellerKnoten){
   markiere aktuellerKnoten;
   WIEDERHOLE für alle ausgehenden Kanten{
         WENN (Gewicht >=10 UND führt zu unmarkiertem Knoten){
               gehe zu diesem Knoten k;
               speichere Knoten in Liste;
               rufe Algo(k) auf;
               }
         }
}
```

9

- A->B->G->I->C (oder A->C->B->G->I)

Dieses Werk ist lizenziert unter einer Creative Commons Namensnennung - Nicht-kommerziell - Weitergabe unter gleichen Bedingungen 4.0 International Lizenz.