

Übungsblatt 13 - Abgabe 31.01.05

Aufgabe 1 (Breitensuche und Mengen)

Sei $G = (V, E)$ ein ungerichteter Graph mit Kanten E und Knoten V . Die Entfernung zweier Knoten u und v aus V , ist die kleinste Anzahl von Kanten in einem Pfad von u nach v . Pfade mit dieser minimalen Entfernung bezeichnet man als u - v -kürzesten Pfad. Wir betrachten nun die Mengen an Knoten $I(u, v)$ für die gilt, dass ein Knoten genau dann in $I(u, v)$ ist, wenn er auf einem kürzesten Pfad von u nach v liegt. Für jede Teilmenge S aus V definieren wir

$$S \subseteq V, I(S) = \bigcup_{u,v \in S} I(u, v)$$

und bezeichnen eine solche Menge S als *geodätisch*, wenn $I(S) = S$ ist.

Eure Aufgabe ist es jetzt für einen gegebenen Graphen G ein Programm zu schreiben, das für eine Teilmenge aus V entscheidet, ob diese geodätisch ist oder nicht.

	<p>Einige Beispiele: $I(2,5) = \{2,3,4,5\}$, da sowohl 2,3,5 als auch 2,4,5 kürzeste Pfade zwischen 2 und 5 sind. Die Menge $\{1,2,3,4,5\}$ ist intuitiv geodätisch für G, aber auch $\{1,2,5\}$, $\{1,3,4\}$ und $\{1,4,5\}$ sind geodätisch. $\{3,4,5\}$ ist nicht geodätisch, da 1 nicht in $I(\{3,4,5\}) = \{2,3,4,5\}$ enthalten ist.</p>
--	--

Die Eingabe besteht aus einem Graphen und mehreren Testfällen. Die erste Zeile der Eingabe gibt die Anzahl der Knoten des Graphen n an und auf den n nächsten Zeilen, werden die Nachbarn des jeweils i -ten Knoten mit Leerzeichen getrennt angegeben. Dann folgt eine Zeile mit Integer m , welche die Anzahl der folgenden Testfälle beschreibt. Auf den nächsten m Zeilen, stehen dann die zu überprüfenden Mengen als Folge von Knotennummern.

Auszugeben ist für jeden Testfall JA oder NEIN.

<p>Beispieleingabe:</p> <pre>5 2 3 1 3 4 1 2 5 2 5 3 4 6</pre>	<p>Beispielausgabe:</p> <pre>yes yes no yes yes no</pre>
--	--

1 2 3 4 5	
1 2 5	
2 4	
1 3 4	
1 4 5	
3 4 5	

Hinweise:

- Beginnt mit einer Berechnung der Entfernung zwischen paarweise allen Knoten.
- Da man im Gegensatz zu der Traversierung von Bäumen darauf achten muss, ob man in einem Graphen bereits in einem Knoten war, sollte man sich eine Datenstruktur hinzunehmen, in welcher man die besuchten Knoten speichert (ein boolsches Feld genügt).
- Ihr habt freie Sprachwahl (d.h. Haskell oder Java).

Aufgabe 2 (Hashing)

Wie nehmen an, dass im Rahmen einer linguistischen Untersuchung die Worthäufigkeiten in vorgegebenen Texten bestimmt werden sollen. Dafür soll eine Klasse „Histogramm“ entwickelt werden, die mit einem geschlossenen Streuspeicherverfahren mit Doppel-Hashing arbeitet (vgl. 7.1.4.2).

```
interface IHistogram<T> {
    // model: h: T -> int
    // inv: h(t) >= 0

    // init()
    // pre: true
    // post: h' == \i -> 0

    public void enter(T t);
    // pre: true
    // post: all(\i -> h'(i)==h(i)) ([T] \ \ {t}) &&
    //         h'(t) == h(t)+1

    public int count(T t);
    // pre: true
    // post: result == h(t), h' == h
}

```

Natürlich ist für endliche Funktionen g und h $g \circ h == h$ entscheidbar.

Für eure konkrete Implementierung braucht ihr dann folgenden Konstruktor:

```
Histogram(int initialSize, float loadfactor, Function<T,Integer> h,
Function<T,Integer> g);
// pre: size >= 0, 0 < loadfactor <= 1.0

```

Implementieren Sie die Schnittstelle (ohne Java's Hashklassen natürlich), suchen Sie zwei brauchbare HashFunktionen g und h und machen Sie ein paar

Tests mit einem Text von <http://www.gutenberg.org/>. Geben Sie ihre Abstraktionsfunktion an.

Achten Sie darauf, dass ihr Histogramm auch dynamisch mitwachsen kann, indem Sie die Datenstruktur beim Überschreiten der Beladungsgrenze in einen größeren Behälter umsortieren.

Hier als Einstieg die Main-Methode:

```
public static void main(String[] args) throws IOException {

    BufferedReader in = new BufferedReader(new FileReader("input.in"));

    Function<String, Integer> g = new Function<String, Integer>() {
        ...
    };

    Function<String, Integer> h = new Function<String, Integer>() {
        ...
    };

    Histogram<String> histo = new Histogram<String>(128, 0.75f, h, g);

    // Use a regular expression for word matching
    Pattern word = Pattern.compile("\\w+");
    String line;

    while ((line = in.readLine()) != null) {
        Matcher m = word.matcher(line);
        while (m.find())
            histo.enter(m.group(0));
    }
    in.close();

    System.out.println(histo.count("the"));
    System.out.println(histo.count("a"));
    System.out.println(histo.count("happy"));
    System.out.println(histo.count("restrictions"));
}
```