Prüfungsteilnehmer	Prüfungstermin	Einzelprüfungsnummer

Kennzahl: _____

Kennwort: _____

Arbeitsplatz-Nr.: _____

Herbst 2009

46114

Erste Staatsprüfung für ein Lehramt an öffentlichen Schulen

— Prüfungsaufgaben —

Fach: **Informatik (Unterrichtsfach)**

Einzelprüfung: **Algorithmen/Datenstrukturen/Programmiermethoden**

Anzahl der gestellten Themen (Aufgaben): **2**

Anzahl der Druckseiten dieser Vorlage: **8**

Bitte wenden!

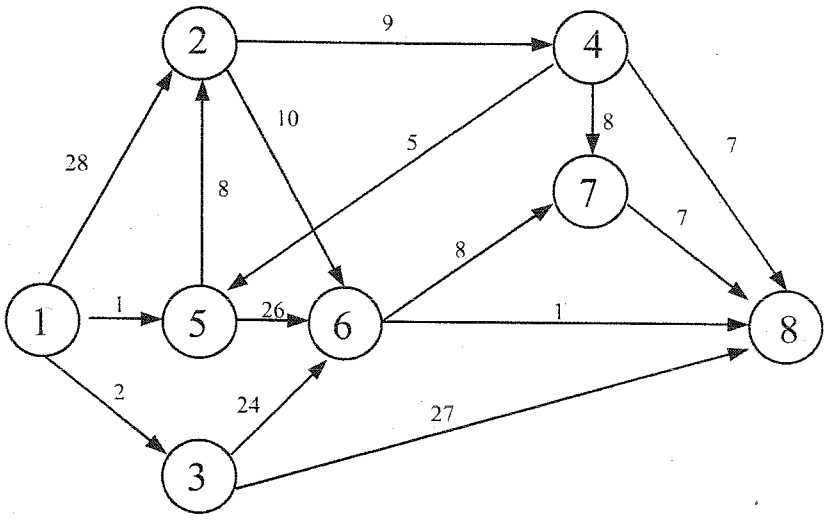
Thema Nr. 1

Teilaufgabe 1 (Listen)

Gegeben seien zwei einfach verkettete Listen $X = (x_1, x_2, ..., x_m)$ und $Y = (y_1, y_2, ..., y_n)$. Schreiben Sie eine Prozedur zum Mischen dieser Listen in der Form $Z = (x_1, y_1, x_2, y_2, ...)$, wobei eine neue Liste erzeugt werden soll. Achten Sie auf die Restglieder, falls $m \neq n$.

Teilaufgabe 2 (Kürzeste Wege)

Berechnen Sie für das folgende Netzwerk mit dem Algorithmus von Dijkstra ausgehend von der Ecke 1 die Längen der kürzesten Wege zu allen anderen Knoten. Stellen Sie die Kostenmatrix auf und geben Sie für jeden Schritt die Entfernungsangaben an.



Fortsetzung nächste Seite!

Teilaufgabe 3 (AVL-Bäume)

Gegeben seien die Schlüsselwerte 3, 5, 11, 8, 4, 1, 12, 7, 2, 6, 10, 9.

(a) Generieren Sie den binären Suchbaum, der durch sukzessives Einsetzen dieser Werte entsteht. Gehen Sie dabei von einem leeren Baum aus.

(b) Geben Sie zu den jeweiligen Knoten die zugehörige Balance ein. Ist der Baum ein AVL-Baum?

(c) Generieren Sie wie in a) erneut einen Suchbaum durch sukzessives Einsetzen der gegebenen Werte und stellen Sie nach jedem Schritt die AVL-Eigenschaft wieder her, so dass beim nächsten Einsetzen ein AVL-Baum zugrunde liegt und auch der Baum am Ende ein AVL-Baum ist.

(d) Stellen Sie jetzt für obige Schlüsselwertmenge einen B-Baum der Ordnung 3 auf. Gehen Sie schrittweise vor, so dass die Vorgehensweise nachvollziehbar ist.

Teilaufgabe 4 (Systementwurf und Programmiermethodik)

Das Newton-Verfahren ist ein nummerisches Verfahren zur näherungsweisen Bestimmung von Nullstellen einer reellen Funktion $f(x)$. Zum Startwert x_o wird der Punkt $(x_o, f(x_o))$ berechnet und die Tangente dieses Punktes bestimmt. Der Schnittpunkt der Tangente mit der x-Achse $(x_1,0)$ liefert einen neuen Punkt $(x_1, f(x_1))$. Zu diesem wird wieder die Tangente bestimmt und erneut der Schnittpunkt mit der x-Achse berechnet, etc. Allgemein lautet die Iterationsvorschrift wie folgt:

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)} \ .$$

Man bricht das Verfahren ab, wenn die Nullstelle hinreichend genau bestimmt ist oder wenn eine obere Grenze für die Anzahl der Iterationsschritte erreicht wurde. Ferner wird abgebrochen, falls $f'(x_n)$ gilt. Im letzten Fall ist die Nullstelle mit dem Verfahren in dieser Form nicht bestimmbar.

Es soll nun ein Programm zur Berechnung einer Nullstelle einer beliebigen Funktion mit Hilfe des Newton-Verfahrens erstellt werden. Verwenden Sie geeignete Abbruchkriterien unter Berücksichtigung von Rundungsfehlern und Schnittstellen (Unterprogrammaufrufe) zur Berechnung von $f(x)$ und $f'(x)$.

(a) Erstellen Sie ein Programmablaufdiagramm.

(b) Erstellen Sie ein Struktogramm und schreiben Sie das Programm in Pseudocode oder einer Programmiersprache Ihrer Wahl.

Fortsetzung nächste Seite!

Thema Nr. 2

Aufgabe 1

Aufgabe 1.1

Entwerfen Sie ein objektorientiertes Modell für gerichtete, gewichtete Graphen und beschreiben Sie Ihren Entwurf durch ein UML-Klassendiagramm. Verwenden Sie dabei die Klassen `Node`, `Edge` und `Graph` als Repräsentationen für Knoten, Kanten und den Graphen. Knoten sollen durch einen Namen identifiziert werden und Kantengewichte seien ganzzahlig (vom Typ `Integer`).
Im Diagramm sollen für alle Klassen die benötigten Attribute und Assoziationen sowie die dazugehörigen Akzessor- (Getter) und Mutator-Operationen (Setter) eingetragen werden. Zusätzlich soll die Klasse `Graph` Operationen für die folgenden Aufgaben enthalten:

- Hinzufügen von Knoten und Kanten
- Zugriff auf Knoten anhand ihres Namens
- Zugriff auf Kanten anhand ihres Start- und Endknotens
- Zugriff auf die Menge aller Knoten des Graphen
- Zugriff auf die Menge aller Nachfolger eines Knotens
- Zugriff auf die Menge aller ausgehenden Kanten eines Knotens

Aufgabe 1.2

Implementieren Sie die Klasse `Graph` aus dem in Aufgabe 1.1 erstellten Modell in einer objektorientierten Programmiersprache Ihrer Wahl. Die Klassen `Node` und `Edge` müssen nicht implementiert werden. Stattdessen können die in Aufgabe 1.1 definierten Operationen dieser Klassen als vorhanden angenommen und in der Implementierung von `Graph` verwendet werden. Ebenso können benötigte Konstruktoren mit geeigneten Parametern als gegeben betrachtet werden. Die Fehlerbehandlung kann bei der Implementierung vernachlässigt werden.

Für die Implementierung seien die folgenden Datenstruktur-Klassen gegeben:

`Hashtable`
Implementiert eine Hashtabelle, die Schlüssel und Werte mit beliebigen Typen speichert.

Operationen:

`put(key:Object, value:Object)`	Fügt das Objekt in `value` in die Hashtabelle ein und assoziiert es mit dem angegebenen Schlüssel. Sowohl `key` als auch `value` können von beliebigem Typ sein.
`get(key:Object):Object`	Gibt das Element zurück, das mit dem angegebenen Schlüssel assoziiert ist.
`values():LinkedList`	Gibt ein Objekt vom Typ `LinkedList` (siehe unten) zurück, das alle in der Hashtabelle gespeicherten Werte enthält.

Fortsetzung nächste Seite!

`LinkedList`
Implementiert eine verkettete Liste mit Elementen von beliebigem Typ.

Operationen:

`addLast(element:Object)`	Fügt das angegebene Element am Ende der Liste ein.
`addFirst(element:Object)`	Fügt das angegebene Element am Anfang der Liste ein.
`contains(element:Object):Boolean`	Gibt `true` zurück, wenn das angegebene Element in der Liste enthalten ist, sonst `false`.
`iterator():Iterator`	Gibt ein Objekt vom Typ `Iterator` (siehe unten) zurück, der eine Iteration alle Elemente der Liste ermöglicht. Nach Aufruf der Operation `iterator()` zeigt der aktuelle Wert des Iterators auf das erste Element der Liste.

`Iterator`
Ermöglicht eine Iteration über die Elemente einer Liste.

Operationen:

`hasNext():Boolean`	Gibt `true` zurück, wenn in der aktuellen Iteration noch weitere Elemente folgen, sonst `false`.
`next():Object`	Gibt das nächste Element aus der Iteration zurück.

Fortsetzung nächste Seite!

Aufgabe 1.3

Dijkstras Algorithmus zur Bestimmung der kürzesten Pfade ist ein bekannter Graphenalgorithmus, der von Edsger W. Dijkstra 1959 veröffentlicht wurde. Im Folgenden soll, aufbauend auf den Ergebnissen der Aufgaben 1.1 und 1.2, die Klasse `Dijkstra` implementiert werden, die diesen Algorithmus realisiert. Sie können dabei, wie in Aufgabe 1.2 erläutert, auf die Klassen `Hashtable`, `LinkedList` und `Iterator` zurückgreifen. Zusätzlich sollen folgende Klassen verwendet werden:

`NodeDistancePriorityQueue`
Implementiert eine Prioritätswarteschlange, die Elemente des Typs `Node` (aus Aufgabe 1.1) aufnehmen kann und diese gemäß einem Distanzwert sortiert. Der Distanzwert für einen Knoten wird dabei von einem Objekt abgerufen, das die Schnittstelle `NodeDistanceProvider` (siehe unten) implementiert und das bei der Konstruktion übergeben wird.

Operationen:

`add(node:Node)`	Fügt einen Knoten zur Prioritätswarteschlange hinzu.
`addAll(nodes:LinkedList)`	Fügt alle in der angegebenen Liste enthaltenen Knoten zur Prioritätswarteschlange hinzu.
`extractMinimum():Node`	Gibt den Knoten mit der geringsten Distanz zurück und löscht ihn aus der Liste.
`isEmpty():Boolean`	Gibt `true` zurück, falls die Prioritätswarteschlange kein Element enthält, sonst `false`.

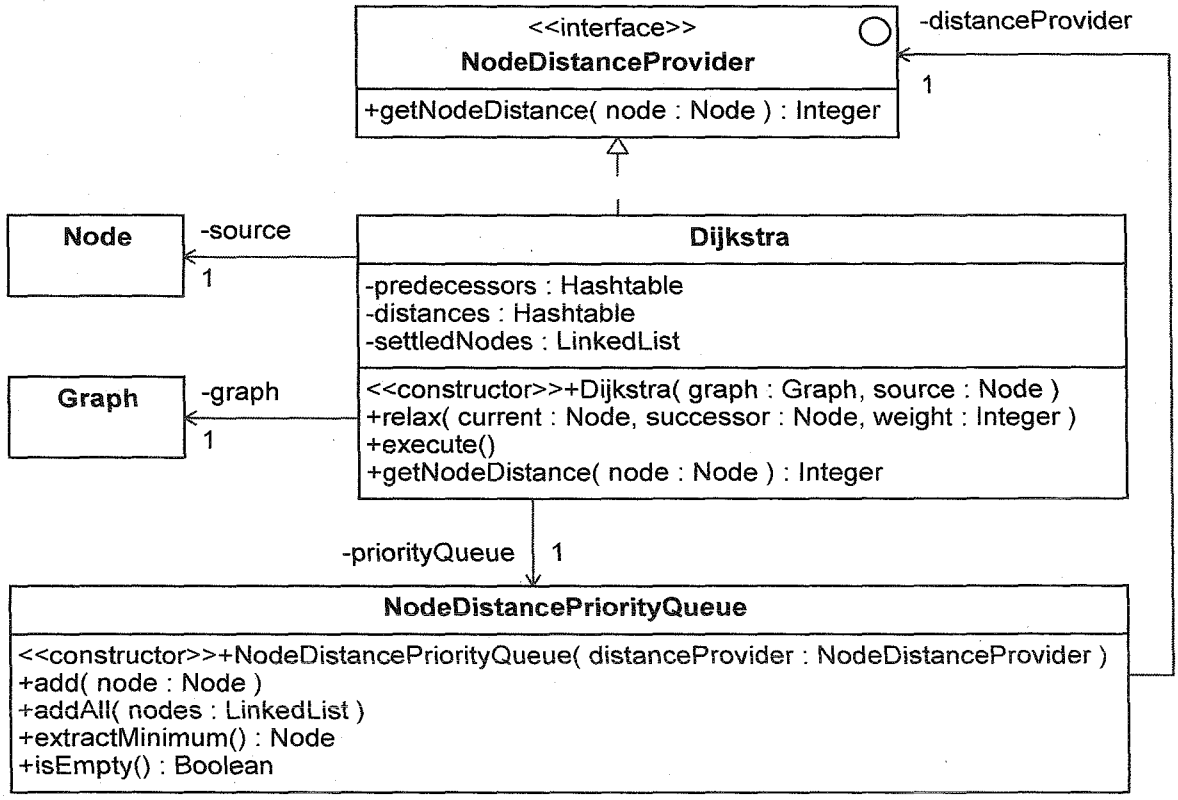
`NodeDistanceProvider`
Schnittstelle, die eine Akzessor-Operation zum Zugriff auf die Distanz eines Knotens von einer implizit gegebenen Quelle liefert.

Operationen:

`getNodeDistance(node:Node): Integer`	Gibt die Distanz des angegebenen Knotens zurück.

Fortsetzung nächste Seite!

Das folgende Klassendiagramm zeigt die zu implementierende Klasse `Dijkstra` zusammen mit ihren Attributen, Assoziationen und Operationen.



a) Geben Sie zunächst eine informelle Beschreibung des Algorithmus von Dijkstra an (textuell oder in Pseudo-Code).

b) Geben Sie ein Quelltext-Gerüst für die Klasse `Dijkstra` an, das der Spezifikation im oben gegebenen Klassendiagramm entspricht. Ihr Code-Fragment soll dabei die Deklaration aller Instanzvariablen enthalten sowie leere Rümpfe für alle Operationen und den Konstruktor der Klasse. Die Implementierung der Operationen erfolgt in den folgenden Teilaufgaben.

c) Geben Sie eine Implementierung für die Operation `getNodeDistance` an sowie für den Konstruktor, der alle Instanzvariablen gemäß den Anforderungen des Dijkstra-Algorithmus initialisiert.

d) Geben Sie eine Implementierung für die Operation `relax` an, die in einem Schritt des Dijkstra-Algorithmus die Aktualisierung des aktuellen Distanz-Schätzwerts und des Vorgängers eines Knotens durchführt.

e) Geben Sie eine Implementierung für die Operation `execute` an, die den eigentlichen Ablauf des Algorithmus realisiert.

Fortsetzung nächste Seite!

Aufgabe 1.4

Welche Zeitkomplexität hat Ihre Lösung? Begründen Sie Ihre Antwort.

Sie können annehmen, dass die Prioritätswarteschlange die Knoten in einer verketteten Liste speichert und dass alle Zugriffe auf Hashtabellen in konstanter Zeit erfolgen.