FAU, Informatik 2, AUD-Team
aud@i2.cs.fau.de

12. Übung

Abgabe bis 08.02.2017, 10:00 Uhr

Einzelaufgabe 12.1: Radix-Sort — 20 EP

In dieser Aufgabe sollen Sie das aus der Vorlesung bekannte Radix-Sort-Verfahren verwenden, um eine Liste von Namen zu sortieren. Laden Sie sich hierzu die Vorgabe `NameSorter.java` sowie die Containerklasse `NameEntry.java` herunter. In `NameEntry` können von einer Person sowohl Nachname (`surname`) also auch Vorname (`firstName`) gespeichert werden. Sie können davon ausgehen, dass die zu sortierenden Namen nur aus den Buchstaben A-Z und a-z bestehen.

a) Für das Radix-Sort-Verfahren ist es eigentlich notwendig, dass alle zu sortierenden Elemente die gleiche Länge haben. Da dies bei Vor- und Nachnamen nicht der Fall ist, muss das Auslesen eines Zeichens in der Methode `getCharOrDefault()` so abstrahiert werden, dass bei ausreichender Länge das gewünschte Zeichen zurückgegeben wird, andernfalls aber kein Fehler auftritt, sondern ein „Standardzeichen“ verwendet wird. Dieses soll so gewählt werden, dass `String`s, die Präfix eines längeren `String` sind, vor dem Längeren stehen (also z.B. „Tim“ vor „Timo“).

b) Implementieren Sie dann die Methode `radixOneStep()`, mit der ein Sortierschritt des Radix-Sort-Verfahrens ausgeführt wird. Als Parameter kann angegeben werden, ob für den aktuellen Schritt der Vor- oder Nachname verwendet werden soll und welches Zeichen zur Einordnung verwendet wird. Zurückgegeben wird dann eine Liste, mit den nach diesen Kriterien sortierten Eingabewerten. Achten Sie auf stabile Sortierung!

c) Die vollständige Sortierung soll in der Methode `sortEntries()` erfolgen. Implementieren Sie hier die Radix-Sortierung durch verschiedene `radixOneStep()`-Aufrufe. Die Sortierung soll nach dem Schema „Nachname, Vorname“ aufsteigend alphabetisch vorgenommen werden, wobei zunächst nur nach dem Nachnamen und bei gleichen Nachnamen nach dem Vornamen sortiert wird.

Geben Sie `NameSorter.java` im EST ab.

Einzelaufgabe 12.2: Spannbäume — 17 EP

Hinweis: 12.2 und 12.3 können erst mit dem Stoff der Vorlesung vom 30. Januar vollständig gelöst werden.
Gegeben sei folgender gewichteter (aber ungerichtete!) Graph in Mengendarstellung: $\{(A, 2, B)$ $(A, 1, D)$ $(A, 5, E)$ $(B, 7, C)$ $(B, 12, D)$ $(B, 13, G)$ $(C, 1, D)$ $(C, 13, F)$ $(D, 9, G)$ $(E, 13, F)$ $(E, 10, G)\}$. Alle Kanten sind lexikographisch sortiert anzugeben (also BC statt CB).

a) Zeichnen Sie den gegebenen Graphen

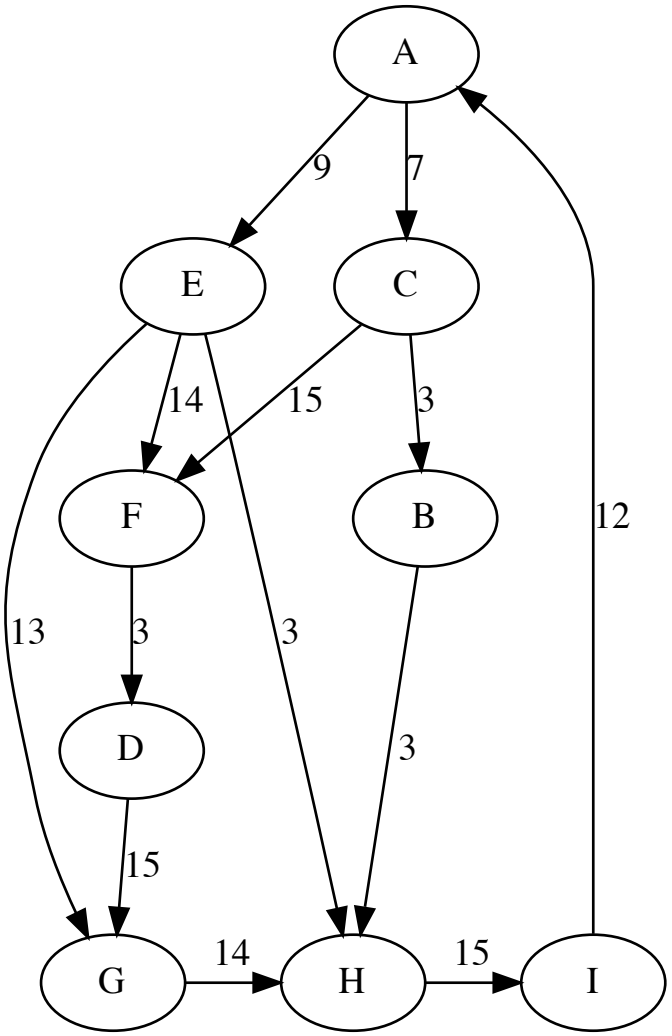
b) Bestimmen Sie den minimalen Spannbaum mit Hilfe des Algorithmus von Prim. Verwenden Sie dabei **F** als Startknoten und geben sie die Kanten in der Reihenfolge an, in der Sie diese in die Ergebnisliste aufnehmen. Wie groß ist die Kantensumme im finalen Spannbaum?

c) Bestimmen Sie nun den minimalen Spannbaum mit Hilfe des Algorithmus von Kruskal. Geben Sie auch hier die Kanten in der Reihenfolge an, in der Sie diese in die Ergebnisliste übernehmen. Wie groß ist die Kantensumme in diesem Spannbaum? Wie unterscheidet sich das Ergebnis der beiden Durchläufe?

Geben Sie Ihre Lösung als `spanningtree.pdf` ab.

Einzelaufgabe 12.3: Dijkstra **9 EP**

a) Ermitteln Sie mit Hilfe des Dijkstra-Algorithmus die jeweils kürzesten Pfade ausgehend von Knoten **C** zu allen anderen Knoten im folgenden Graphen:

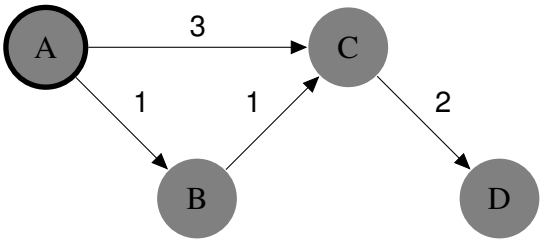


Der Algorithmus soll auf dem Papier ausgeführt werden und wie folgt dokumentiert werden.

Auf der X-Achse sind die Knoten aufgetragen, auf der Y-Achse werden die Einzelschritte des Algorithmus aufgetragen. Markieren Sie in jeder Zeile den Knoten, der im nächsten Schritt als Ausgangsknoten verwendet wird durch ein Rechteck. Wenn es mehrere mögliche Startknoten für den nächsten Schritt gibt, muss der Knoten gewählt werden, der im Alphabet zuerst kommt. Zellen, in denen es zu keiner Veränderung zum vorherigen Schritt gekommen ist, müssen nicht erneut ausgefüllt werden.

In der letzten Zeile wird die Länge des jeweils kürzesten Pfades von Knoten **C** aus als Ergebnis eingetragen.

Halten Sie sich an folgendes Beispiel und verwenden Sie das gleiche Schema für Ihre Tabelle:



	A	B	C	D
Schritt 1	0	-	-	-
Schritt 2		1	3	
Schritt 3			2	
Schritt 4				4
Ergebnis	0	1	2	4

b) Geben Sie für jeden Knoten den soeben berechnete, kürzesten Pfad – ausgehend von Knoten **C** – in folgender Notation an:

$$Z \leftarrow Y \leftarrow X$$

Das Beispiel besagt, dass man den Knoten Z mit den geringsten Kosten erreicht, wenn man beim Startknoten X startet und dann über Y zu Z geht.

c) Erklären Sie kurz, wie ein Algorithmus den Pfad mit den geringsten Kosten aus der Tabelle ablesen kann. Reichen die Informationen, die unsere Tabelle enthält, damit ein Algorithmus ohne weitere Berechnungen den kürzesten Pfad ermitteln kann?

Geben Sie Ihre Lösung als `dijkstra.pdf` ab.

Gruppenaufgabe 12.4: Web Of Trust — 24 GP

Das Web of Trust ist eine Methode zur Identitätsermittlung im Internet. Die Grundannahme dabei ist wie folgt: Wenn Alice Bob kennt und Bob identifiziert Carol, dann kann sich auch Alice auf die Identität von Carol verlassen. Natürlicherweise ist Alice fester von Bobs Identität (den sie direkt kennt) überzeugt als von Carols und jeder weitere Schritt fügt etwas Ungewissheit hinzu. Daher interessiert sich Alice für die Länge eines solchen "Vertrauenspfades" und möchte natürlich auch wissen, wem sie hier (indirekt) vertraut.

a) Implementieren Sie eine Methode `trusts(WebOfTrustNode other)`. Dabei soll `alice.trusts(bob)` bedeuten, dass Alice Bob kennt. Es handelt sich dabei also um eine gerichtete Kante von `alice` zu `bob` im Graphen. Für die Aufgabe soll die Darstellung als Adjanzenzliste gewählt werden.

b) Zum Auffinden eines Pfades zwischen zwei Benutzern soll Breitensuche verwendet werden. Implementieren Sie `ArrayList<WebOfTrustNode> findPath(WebOfTrustNode)`. Die Methode gibt den Pfad zurück. Falls kein Pfad existiert wird `null` zurückgegeben.

Beachten Sie, dass die hier betrachteten Graphen (fast) alle zyklisch sind (wenn Alice Bob kennt, dann kennt Bob oft auch Alice)! Betrachten Sie jeden Knoten nur genau ein mal.

c) Eine spannende Frage in einem solchigen Graphen ist "Welche der Teilnehmer kann Alice (indirekt) identifizieren". Implementieren Sie eine Methode `calculateReachableSubset()`, die eine Menge von allen Knoten ausgibt, die vom Ursprungsknoten aus erreichbar sind. Auch hier gelten die selben Regeln zur zyklischen Natur des Graphen!

d) In der letzten Teilaufgabe haben wir alle Knoten identifiziert, die Alice identifizieren kann. Aber nicht alle diese Personen können auch Alice identifizieren. Wir wollen nun basierend auf der letzten Teilaufgabe die starke Zusammenhangskomponente bestimmen, in der Alice enthalten ist. Implementieren Sie dazu die Methode `calculateStrongSubset()`, die alle Teilnehmer findet, die sowohl von Alice identifiziert werden können, als auch Alice identifizieren können.

Berechnen Sie dazu mit der Methode `calculateReachableSubset()` die Menge von Kandidaten und überprüfen Sie für alle Kandidaten, ob ein Weg zurück zu Alice existiert.

Geben Sie Ihre Lösung als `WebOfTrustNode.java` über EST ab.
Hinweis: Identität bei der Verwendung von OpenPGP funktioniert fast genau so!

46 EP + 24 GP = 70 Punkte