

Klausur „Algorithmen und Datenstrukturen (AuD) II“ (Wiederholung Sommer 2010)

28.03.2011

Aufgabe 1 [Gesamtpunktzahl: 10]

a) Geben Sie zum Graphen aus Abb. 1 die Adjazenzlisten an. Die Knoten sollen dabei in alphabetischer Reihenfolge angegeben werden.

b) Führen Sie im Graphen aus Abb. 1 von Hand schrittweise den Algorithmus von *Kruskal* zur Bestimmung eines minimalen aufspannenden Baumes (engl. minimum spanning tree, kurz auch MST) aus.

Geben Sie bei jedem Schritt an, welcher Knoten jeweils in den bereits vorliegenden Teilbaum aufgenommen wird.

c) Nach welchem Entwurfsprinzip arbeitet der Algorithmus von *Dijkstra*? Führen Sie diesen Algorithmus für den Graphen aus Abb. 1 ausgehend von Knoten H aus. Geben Sie alle Schritte nachvollziehbar an.

d) Begründen Sie, warum der Algorithmus von Dijkstra für Graphen mit negativen Kantengewichten nicht korrekt arbeitet.

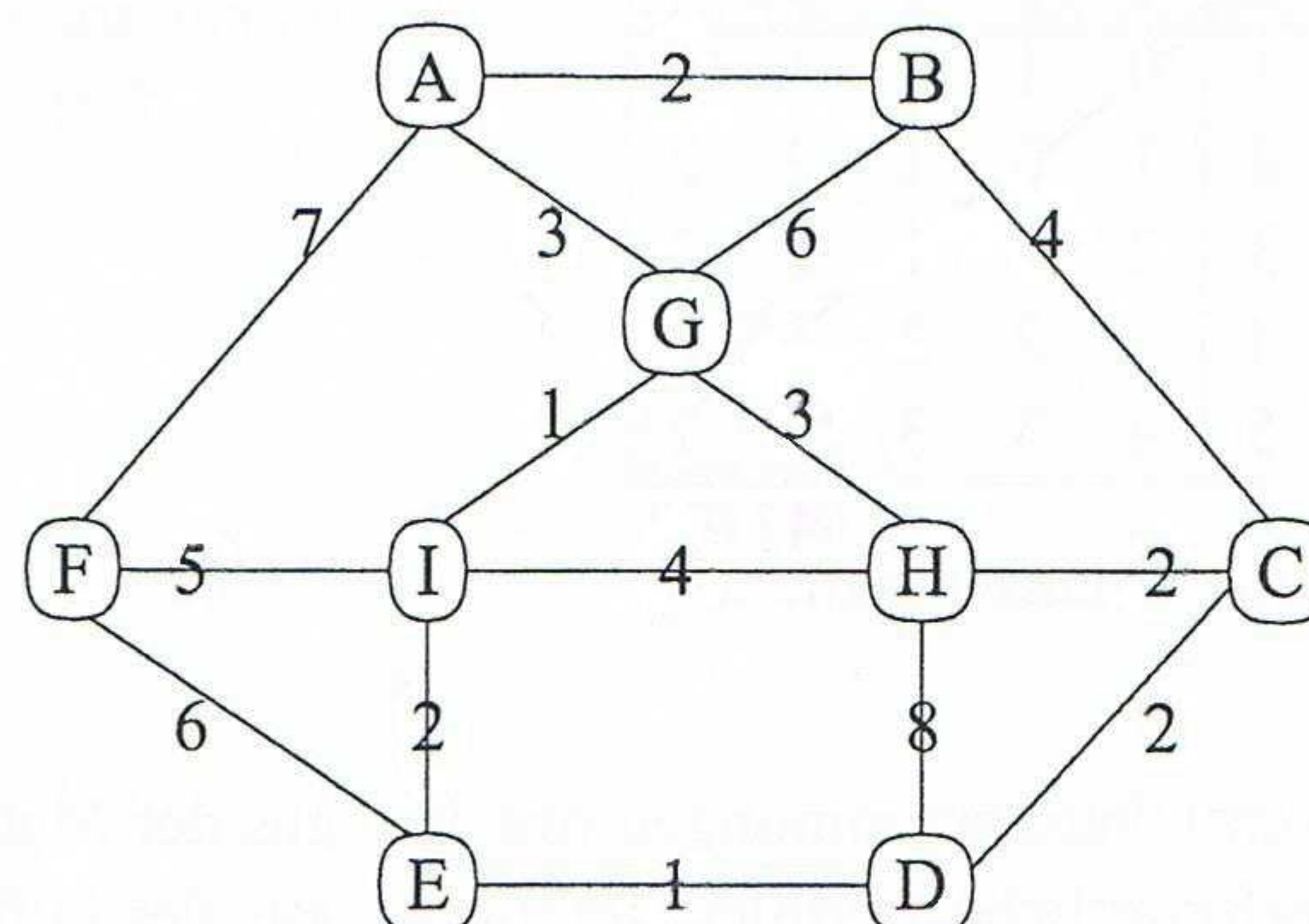


Abbildung 1: Ein ungerichteter, gewichteter Graph

Aufgabe 2 [Gesamtpunktzahl: 10]

Sei ein Muster P ein String der Länge m aus Buchstaben und mindestens einem Vorkommen eines Pluszeichens '+'. Das Plus wirkt als sog. 'wild-card' oder Metazeichen; es kann eine beliebige Folge von **einem oder mehr** Zeichen abdecken ('matchen').

Ein Beispiel:

- Wenn das Muster $P = \text{"Dat+ruk+r"}$
- und der Text $T = \text{"Datenstruktur"}$ vorliegen,
- dann gibt es eine Übereinstimmung (Match) zwischen Muster und Text beginnend mit dem 'D' und endend mit dem letzten 'r'; dabei deckt das erste Pluszeichen den Teilstring "enst" und das zweite Pluszeichen den Teilstring "tu" ab.
- Bei $P = \text{"Dat+ruk+r"}$ und $T = \text{"Datrukr"}$ liegt hingegen **kein** Match vor.

a) Konzipieren Sie einen Algorithmus, um einen Match für ein Muster P (der Länge m) mit mehreren Vorkommen des Pluszeichens in einem Text T (der Länge n) zu finden, falls ein solcher existiert, und beschreiben Sie diesen Algorithmus zunächst in Worten.

b) Unter welchen Bedingungen ergeben sich schlechteste Fälle für Ihren Algorithmus? Wie ist dann der Aufwand in Abhängigkeit von der Länge des Textes n und der Länge des Musters m ($\mathcal{O}$ -Notation)?

c) Setzen Sie Ihren Algorithmus nach eigener Wahl **entweder** in Haskell **oder** in Java als Funktion bzw. Methode namens **mww** (für match with wild-card) um. Die Funktion/Methode soll einen boolschen Wert zurückgeben (Match liegt vor oder nicht).

d) Gegeben ist die Differenzmatrix aus Tabelle 1.

		n	i	c	e	l
	0	0	0	0	0	0
n	1	0	1	1	1	1
i	2	1	0	1	2	2
z	3	2	1	1	2	3
e	4	3	2	2	1	2
r	5	4	3	3	2	2

Tabelle 1: Differenzmatrix

Lesen Sie für **alle** k -approximativen Übereinstimmungen mit dem aus der Matrix ablesbaren *kleinsten* k die jeweils erforderlichen Editieroperationen aus der Differenzmatrix aus und geben sie diese in nachvollziehbarer Weise an.

Aufgabe 3 [Gesamtpunktzahl: 10]

Der Graph G sei als Adjazenzmatrix gegeben (Abb. 2).

	1	2	3	4	5
1	0	1	0	0	0
2	1	0	1	1	0
3	0	1	0	0	1
4	0	1	0	0	1
5	0	0	1	1	0

Abbildung 2: Adjazenzmatrix des Graphen G

a) Geben Sie die bildliche Darstellung des Graphen an.

- Zeigen Sie, welchen Weg die *Tiefensuche* nimmt, um die Knoten in diesem Graphen zu besuchen. Der Startknoten sei dabei beliebig.
- Zeigen Sie, in welcher Reihenfolge die Knoten bei der *Breitensuche* besucht werden.

b) Vergleichen Sie die beiden durch die unterschiedlichen Suchstrategien entstandenen Suchbäume bezüglich ihrer Höhe. Leiten Sie daraus eine allgemeingültige Aussage ab.

c) Die Erreichbarkeit von Knoten in 2 Schritten kann durch Multiplikation der Adjazenzmatrix mit sich selbst bestimmt werden. Prüfen Sie dies nach, indem Sie die Adjazenzmatrix des abgebildeten Graphen mit sich selbst multiplizieren. Was sagen die verschiedenen Werte in der Matrix aus?

d) Schreiben Sie ein Programm in Java zur Multiplikation von Matrizen. Welche Komplexität ($\mathcal{O}$ -Notation) in Abhängigkeit von der Größe der Matrix hat Ihr Algorithmus?

e) Welche Aussage kann man machen, wenn die Adjazenzmatrix n -mal ($n > 2$) mit sich selbst multipliziert wird?

Aufgabe 4 [Gesamtpunktzahl: 10]

Das Oldenburger Milchkannenproblem:

Jemand betritt einen Milchladen mit einer Kanne K , die genau die Menge V_K fasst, und möchte die (ganzzahlige) Menge $Soll$ kaufen (mit $0 < Soll \leq V_K$). Der Milchverkäufer hat nur einen Eimer E , der es erlaubt, die Menge V_E (mit $V_E \geq V_K$) abzumessen.

Man kann mit diesen beiden Gefäßen durch Eingießen, Umschütten und Zurückgießen aus einer bzw. in eine großen Milchwanne auf verschiedenen Wegen die gewünschte Menge abfüllen.

Mit Hilfe einer *Suche mit Backtracking* können alle Lösungen für dieses Problem – d.h. das Abmessen der Sollmenge auf unterschiedlichen Wegen ohne wiederholte Zustände – gefunden werden.

a) Entwickeln und skizzieren Sie zunächst *von Hand* den Suchraum für $V_K = 7l$, $V_E = 10l$ und $Soll = 4l$ bis zu einer Tiefe von mindestens drei durchgeführten Zügen.

b) Konzipieren Sie dann Datenstrukturen in Haskell, um die *Situationen* (d.h. die Aufteilungen der Flüssigkeit auf die zwei Gefäße) und die Zustände (Knoten) des Suchraums darzustellen. Wie sieht der Startzustand, wie sehen mögliche Zielzustände für das Beispiel aus a) aus?

c) Für die Verwendung in der Haskell-Funktion `searchDfs` (s.u.) ist eine Nachfolgerfunktion `succ` erforderlich, die einem *Knoten* im Suchraum die Liste der Folgeknoten zuordnet.

Um `succ` zu definieren, soll eine Hilfsfunktion `possibleMoves` verwendet werden, die einer *Situation* (s.o.) die Liste der möglichen Folgesituationen zuordnet. Diese sei definiert als

```
possibleMoves sit = fromMug1 sit ++ fromMug2 sit
```

Konzipieren Sie im folgenden die Funktion `fromMug1`, die für eine Situation – wie in b) definiert – bestimmt, welche Folgesituationen sich *aus Gefäß1* gewinnen lassen. (Die Funktion `fromMug2` arbeitet analog und braucht in der Klausur nicht definiert zu werden).

Hinweis: Vergessen Sie nicht, dass man die leeren Gefäße mit Milch aus der Milchwanne vollständig füllen und sowie gefüllte Gefäße vollständig ausleeren darf.

c1) Beschreiben Sie Ihren Ansatz zunächst detailliert mit Worten.

c2) Definieren Sie dann die Funktion `fromMug1` (und gegebenenfalls erforderliche Hilfsfunktionen) in Haskell.

Zur Erinnerung:

```
searchDfs :: (Eq node) => (node -> [node]) -> (node -> Bool)
                                         -> node -> [node]

searchDfs succ goal x = search' (push x emptyStack)
  where
    search' s
      | stackEmpty s = []
      | goal (top s) = top s : search' (pop s)
      | otherwise    = let x = top s
                      in search' (foldr push (pop s) (succ x))
```