Fachbereich 7 – Mathematik
Prof. Dr. Andreas Frommer

Algorithmen und Datenstrukturen

WS 2002/2003

Klausur

Datum: 1. April 2003
Beginn: 10.00 Uhr s.t.
Bearbeitungszeit: 120 Minuten

Name: .... Matr.-Nr.: ....

Inhalt: – Aufgaben 1 – 5 (8 Seiten)
– Anlage zu Aufgabe 4 (3 Seiten)
– Anlage zu Aufgabe 5 (1 Seite)

Aufgabe 1	max. 20 Punkte	
Aufgabe 2	max. 20 Punkte	
Aufgabe 3	max. 20 Punkte	
Aufgabe 4	max. 20 Punkte	
Aufgabe 5	max. 20 Punkte	
Bonus	max. 25 Punkte	
Summe	max. 125 Punkte	

Note:

Aufgabe 1

a) In eine Liste L wird eine Folge von Zeichen eingetragen, indem folgende Operationen durchgeführt werden:

create(L)

reset(z,L)

insert(‘p’,L,z)

reset(z,L)

insert(‘o’,L,z)

reset(z,L)

insert(‘s’,L,z)

reset(z,L)

advance(z,L)

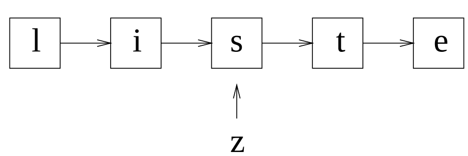
insert(‘t’,L,z)

Skizzieren Sie Liste L nach jeder insert-Operation.

Eine Liste L kann mit Hilfe zweier Stacks $S1, S2$ realisiert werden. Zum Beispiel hat man dann

advance(z,L) : falls nicht empty($S2$)
pop($S2$,e)
push($S1$,e)

b) Skizzieren Sie $S1$ und $S2$ für die Liste



— **Fortsetzung der Aufgabe auf der nächsten Seite** —

Fortsetzung zu Aufgabe 1 (Seite 2)

c) Geben Sie an, wie man entsprechend die übrigen Listenoperationen und -funktionen

create(L)

reset(z,L)

insert(e,L,z)

delete(L,z)

empty(L)

out_of_list(z,L)

durch Stackoperationen und -funktionen mit $S1$, $S2$ realisieren kann.

Aufgabe 2

Man nennt eine Folge $a_0, \ldots, a_{n-1}$ h-sortiert, wenn $a_i \leq a_{i+h}$ für alle $0 \leq i \leq n-1-h$ gilt.

a) Ist die Folge

$$3 \quad 12 \quad 7 \quad 5 \quad 20 \quad 9 \quad 9 \quad 21 \quad 10$$

3-sortiert? Ist sie 2-sortiert? Begründen Sie Ihre Antworten.

b) Inspizieren Sie folgende C-Funktion `ShellSort`, die als Eingabe eine Folge a von n Datensätzen und eine Folge h von Inkrementen erhält. Dabei ist h eine abnehmende mit 1 endende Folge h von hn natürlichen Zahlen.

```
void ShellSort (datensatz a[], int n, int h[], int hn)
/* h[hn-1] = 1; h[i] > h[j]  für  0 ≤ i < j < hn */
{
 int i,j,l;
 datensatz t;
 for (l = 0; l < hn; l++)
 {
   for (i = h[l]; i < n; i++)
   {
     j = i;
     t = a[i];
     while ((j >= h[l]) && groesser(a[j-h[l]].schluessel, t.schluessel) )
     {
       a[j] = a[j-h[l]];
       j = j-h[l];
     }
     a[j] = t;
   }
   /* Die Folge a ist nun h[l]-sortiert */
 }
}
```

— Fortsetzung der Aufgabe auf der nächsten Seite —

Fortsetzung zu Aufgabe 2 (Seite 2)

Welche Vertauschungen werden durch `ShellSort` vorgenommen bei Eingabe der Folge a mit den Schlüsseln

$$15 \quad 2 \quad 43 \quad 4 \quad 8 \quad 47$$

und den Inkrementen $h = \{4, 2, 1\}$ (also `h[0]=4, h[1]=2, h[2]=1`)?

c) Welchem in der Vorlesung vorgestellten Sortierverfahren entspricht `Shellsort` bei Eingabe von $h = \{1\}$?

d) Zeigen Sie mit einem Beispiel, dass `ShellSort` nicht stabil ist.

Aufgabe 3

Gegeben sind n Briefmarkenwerte $w_1, w_2, \ldots, w_n \in N$ mit der Eigenschaft $w_i > w_{i+1}$ und $w_n = 1$. Gesucht ist die *kleinstmögliche* Anzahl von Briefmarken $minanz \in N$, die ausreicht, einen Brief mit dem Porto $Zielwert \in N$ zu frankieren. Jeder Briefmarkenwert w_i aus der oben angegebenen Menge darf dabei mehrfach verwendet werden.

Beispiel: Mit Hilfe der Briefmarkenwerte $20, 19, 12, 7, 2, 1$ soll ein Brief mit dem Zielwert 37 frankiert werden. In diesem Fall gilt $minanz = 4$, denn:

$$37 = \mathbf{3} * 12 + \mathbf{1} * 1$$

Für alle anderen Kombinationen, die den Zielwert ergeben, werden mehr Briefmarken benötigt.

Es soll nun ein Verfahren entwickelt werden, welches das Problem allgemein löst:

a) Wie groß kann der Wert fr $minanz$ höchstens sein? Begründen Sie Ihre Antwort!

b) Eine Strategie, das Problem zu lösen, besteht darin, den Brief schrittweise zu frankieren und eventuelle Fehlentscheidungen zurückzunehmen. Im Folgenden ist

- $bisher\_verwendet \in N$ die bisher verwendete Anzahl von Briefmarken
- $summe \in N$ die bisherige Summe der zugehörigen Werte
- $aktuell \in \{1, \ldots, n\}$ der Index des momentan verwendeten Briefmarkenwertes w_{aktuell}

Formulieren Sie mit Hilfe dieser Bezeichnungen geeignete Kriterien, um festzustellen, welche der folgenden Situationen vorliegt:

α) Eine getroffene Entscheidung ist *erkennbar* falsch.

β) Eine getroffene Entscheidung ist zulässig, aber es liegt *noch* keine Lösung vor.

γ) Eine getroffene Entscheidung ist zulässig *und* es liegt eine neue (vorläufige) Lösung des Problems vor.

a) Entwerfen Sie einen Backtracking-Algorithmus, der zu einer gegebenen Briefmarkenmenge $\{w_1, \ldots, w_n\}$ und einem gegebenen Porto $Zielwert$ die minimal benötigte Anzahl von Briefmarken $minanz$ ermittelt. Formulieren Sie Ihr Verfahren in Form einer rekursiven Prozedur

`frankieren(stufe, bisher_verwendet, summe)` .

— Fortsetzung der Aufgabe auf der nächsten Seite —

Fortsetzung zu Aufgabe 3 (Seite 2)

Hinweise:

- Mit Hilfe des Parameters $stufe$ wird der Prozedur `frankieren` mitgeteilt, mit welchem Index fr den zu verwendenden Briefmarkenwert gestartet werden soll.
- Verwenden Sie innerhalb der Prozedur `frankieren` fr den Index der momentan verwendeten Briefmarke $aktuell$ eine Schleife, die von $stufe$ bis n läuft.
- Es sollte versucht werden, so lange wie möglich mit dem gleichen Briefmarkenwert zu frankieren.
- Initialisieren Sie in einem „Hauptprogramm“ die globale Variable $minanz$ geeignet, bevor die Rekursion mit dem Aufruf

  ```
  frankieren(1,0,0)
  ```

  gestartet wird.

Aufgabe 4

(Programmierung: Huffman-Dekodierung)

Ergänzen Sie im beiliegenden Programm die Funktion `print_decoded`, die zu einem gegebenen Baum `b` und Wörtern `woerter` den Code `c` der Länge `codelen` dekodiert und ausgibt.

Hinweise:

- Der Code `c` soll mehrere kodierte Wörter hintereinander enthalten.
- Sie dürfen davon ausgehen, dass der Code `c` korrekt, d. h. fehlerfrei dekodierbar ist.
- Sie benötigen hierfür keine Rekursion.
- Sie müssen den Baum mehrfach von der Wurzel aus durchlaufen und jedesmal, wenn Sie bei einem Blatt angekommen sind, das entsprechende Wort ausgeben.
- Das beiliegende Programm entspricht dem aus der Vorlesung, bis auf den Kopf der neuen Funktion und das erweiterte Hauptprogramm.

Aufgabe 5

Gegeben ist ein durch den beigefügten `C`-Quellcode charakterisierter Graphenalgorithmus.

Die Konstante `MAXREAL` bezeichnet eine implementationsabhängige Konstante, die die größte positive Gleitkommazahl repräsentiert.

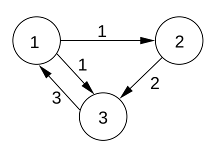
Die Funktion `ReadData(...)` liest die Daten eines gewichteten Graphen $G = (V, E)$ ein und belegt die folgenden Variablen:

- `n` : Anzahl der Knoten
- `el` : hierbei gilt `el[i][j]=1` falls $(v_i, v_j) \in E$, sonst `el[i][j]=0`
- `value` : hierbei wird `value[i][j]` mit dem Kantengewicht bewertet, falls $(v_i, v_j) \in E$, sonst `value[i][j]=0`

a) Welchen Algorithmus repräsentiert die Funktion `DoSomething`?

b) Geben Sie den Aufwand der Funktion `DoSomething` an. Begründen Sie Ihre Antwort ($\mathcal{O}$-Analyse).

c) Skizzieren Sie die Belegung der Variablen `n`, `el` und `value` nach dem Einlesen der Informationen für folgenden kantengewichteten Graphen.



Skizzieren Sie außerdem die Belegung der Variablen `distance` nach Durchlaufen des ersten Schleifenblockes in der Funktion `DoSomething`.

d) Gegeben ist der unten skizzierte Graph. Wenden Sie den Algorithmus zur Berechnung eines kritischen Pfades auf die Knoten 0 und 13 an. Erstellen Sie eine Tabelle, in der in der Waagerechten die Knoten in der topologischen, und in der Senkrechten die Knoten in der natürlichen Sortierung stehen. Tragen Sie dann für $v \in V$ in die Tabelle die jeweils aktualisierten Abstände $\delta_v^g(0)$ ein. Geben Sie den kritischen Pfad und sein Gewicht an.