Institut 186 für Computergraphik	Algorithmen und Datenstrukturen 2			Bsp.-Test 29. Januar 1999 Gruppe A
Matrikelnummer	Beilagen‡	1 (10)	2 (25)	3 (15)
NACHNAME, Vorname				Σ (50)

‡ Geben Sie an, wieviele Zusatzbl. Sie abgeben (jedes mit Name & Matr.-Nr. beschriftet!).

Zur Erinnerung:
Alle Aufgaben beziehen sich auf die 10 Übungsbeispiele im WS '98/'99.

(10) **Aufgabe 1 - zum Beispiel 9: Travelling Salesman**

Generell gilt: Lösungen, aus denen nicht hervorgeht, wie sie erarbeitet wurden, werden nicht anerkannt. Schreiben Sie die Zwischenergebnisse auf.

1. Erstellen Sie aus der folgenden Angebotsdatei die Kostenmatrix. 2. Geben Sie zudem auch noch die reduzierte Kostenmatrix an. Reduzieren Sie zuerst die Zeilen, dann die Spalten. Wenden Sie nur Subtraktionen an. 3. Um wieviel ist die billigste Rundreise bezüglich der reduzierten Kostenmatrix günstiger gegenüber der ursprünglichen Kostenmatrix?

```
Wien > Salzbg.      = 80
Wien > Linz         = 36
Wien > Graz         = 56
Salzbg. > Villach   = 16
Salzbg. > Graz      = 40
Salzbg. > Linz      = 70
Graz > Linz         = 90
Graz > Wien         = 58
Graz > Salzbg.      = 40
Villach > Linz      = 22
Villach > Salzbg    = 20
Villach > Wien      = 70
Linz > Villach      = 32
Linz > Wien         = 34
Linz > Graz         = 96
```

1.	W.	S.	G.	V.	L.
Wien >	—				
Salzbg. >		—			
Graz >			—		
Villach >				—	
Linz >					—

2.	W.	S.	G.	V.	L.
Wien >	—				
Salzbg. >		—			
Graz >			—		
Villach >				—	
Linz >					—

3.

(25) **Aufgabe 2 - zum Beispiel 7: Linked Trie**

Gegeben sei folgende Deklaration eines Linked Trie:

```
CONST M = 2147483647;
TYPE
  NodePtr = ^Node;
  Node = RECORD
           letter       : CHAR;
           end          : BOOLEAN;
           nextletter   : NodePtr; { naechster Buchstabe im Wort }
           otherletter  : NodePtr; { anderer Buchstabe an der }
                                   { selben Stelle im Wort    }
  END;
```

Sie sollen nun Suffix Compression für den Linked Trie implementieren, wobei die Endungen selbst als Linked Trie gespeichert werden. Dazu stehen ihnen folgende Hilfsfunktionen zur Verfügung:

Die `PROCEDURE insert(trie:NodePtr; hashvalue:INTEGER);` trägt den übergebenen `trie` in einer globalen Hashtabelle mit dem gegebenen `hashvalue` ein.

Die `FUNKTION search(hashvalue:INTEGER; trie:NodePtr): NodePtr;` sucht nach einem trie mit vorgegebenem `hashvalue` in einer globalen Hashtabelle. Wenn der übergebene Trie gleich `nil` ist, so wird der erste Trie mit diesem `hashvalue` zurückgegeben, bei weiteren Aufrufen mit dem selben `hashvalue`, erhält man bei Übergabe des ersten gefundenen Tries, den nächsten mit dem selben `hashvalue` usw. solange bis man `nil` erhält, wenn kein Trie mit dem gegebenen `hashvalue` mehr in der Tabelle ist.

(7) a) Schreiben Sie nun zuerst die *rekursive, effiziente*

```
FUNCTION same(a, b : NodePtr) : BOOLEAN;
```

die genau dann `TRUE` zurück liefert, wenn die beiden übergebenen Linked Tries exakt die gleichen Worte enthalten. (Dazu beötigen Sie die Hashtabelle noch nicht.)

PRÜFUNGSORDNER - ein Service Deiner Fachschaft Informatik!
LVA: Algdat 2 - UE
Preis: -

(18) b) Schreiben Sie nun die

```
PROCUDURE compress(VAR t : NodePtr; VAR hashvalue : INTEGER);
```

die die eigentliche Suffix Compression durchführen soll. Verwenden Sie dazu `same`, `insert`, und `search`.

Zunächst soll diese Prozedur rekursiv die Hashwerte der Nodes im Trie berechnen, und über den `VAR` Parameter zurückliefern. Die Formel für die Hashfunktion ist folgende:

$$h(node) = \left(\sum_{i=1}^{26} (29 \cdot ord(letter_i) + (h(subnode_i) \; div \; 13)) \right) mod \, M$$

wobei die Summierung sich auf alle Buchstaben an der selben Stelle im Wort bezieht, $h(nil) = 0$ ist, und Buchstaben die nicht verwendet werden, nicht in die Summierung eingehen.

Sobald ein Hashwert berechnet wurde, kann in der Hashtabelle nach dem Trie gesucht werden, und wenn die entsprechende Endung gefunden wird, soll sie sofort ersetzt werden (auf korrektes Löschen des ersetzten Tries können sie verzichten). Wenn die Endung nicht gefunden wurde, so kann der Trie in die Hashtabelle eingetragen werden.

(15) **Aufgabe 3 - zum Beispiel 3: Rot-Schwarz Baum**

Gegeben sei folgende Deklaration eines Rot-Schwarz Baumes.

```
TYPE Tree =     OBJECT
                  pNode: ^Node;
                  FUNCTION red: BOOLEAN;
                  FUNCTION isNil: BOOLEAN;
                END;
     Node =     RECORD
                  left, right: Tree;
                  rededge: BOOLEAN;
                  k: KEY;
                  val: VALUE;
                END;
```

Die Methode `red` liefert genau dann `TRUE` zurück wenn es sich bei der Kante, die auf den Knoten zeigt, um eine rote Kante handelt. `red` von der Wurzel eines Rot-Schwarz Baumes gibt immer `FALSE`. Die Methode `isNil` liefert genau dann `TRUE` zurück wenn `pNode` gleich `NIL` ist.

(13) a) Schreiben Sie eine *rekursive* Methode

```
PROCEDURE Tree.statistics(...);
```

die folgende Statistiken durch einmaliges Abarbeiten des Rot-Schwarz Baumes berechnet:

- Anzahl der Datensätze
- Anzahl der Knoten des äquivalenten 2-3-4 Baumes
- Tiefe des Baums

Verwenden Sie dabei KEINE GLOBALEN VARIABLEN. Geben Sie an, welche Übergabeparameter Ihre Methode verwendet.

(2) b) Geben Sie außerdem an, wie Ihre Methode vom Hauptprogramm aufgerufen wird.

Institut 186 für Computergraphik	Algorithmen und Datenstrukturen 2		Bsp.-Test 29. Januar 1999 Gruppe B	
Matrikelnummer	Beilagen‡	1 (10)	2 (15)	3 (25)
NACHNAME, Vorname				Σ (50)

‡ Geben Sie an, wieviele Zusatzbl. Sie abgeben (**jedes mit Name & Matr.-Nr. beschriftet!**).

Zur Erinnerung:
Alle Aufgaben beziehen sich auf die 10 Übungsbeispiele im WS '98/'99.

(10) Aufgabe 1 – zum Beispiel 9: Travelling Salesman

Generell gilt: Lösungen, aus denen nicht hervorgeht, wie sie erarbeitet wurden, werden nicht anerkannt. Schreiben Sie die Zwischenergebnisse auf.

1. Erstellen Sie aus der folgenden Angebotsdatei die Kostenmatrix. 2. Geben Sie zudem auch noch die reduzierte Kostenmatrix an. Reduzieren Sie zuerst die Zeilen, dann die Spalten. Wenden Sie nur Subtraktionen an. 3. Um wieviel ist die billigste Rundreise bezüglich der reduzierten Kostenmatrix günstiger gegenüber der ursprünglichen Kostenmatrix?

```
Wien > Salzbg.      = 40
Wien > Linz         = 18
Wien > Graz         = 28
Salzbg. > Graz      = 20
Salzbg. > Bregenz   =  8
Salzbg. > Linz      = 35
Graz > Linz         = 45
Graz > Wien         = 29
Graz > Salzbg.      = 20
Bregenz > Linz      = 11
Bregenz > Salzbg.   = 10
Bregenz > Wien      = 35
Linz > Bregenz      = 16
Linz > Wien         = 17
Linz > Graz         = 48
```

1.

	W.	S.	G.	B.	L.
Wien >	—				
Salzbg. >		—			
Graz >			—		
Bregenz >				—	
Linz >					—

2.

	W.	S.	G.	B.	L.
Wien >	—				
Salzbg. >		—			
Graz >			—		
Bregenz >				—	
Linz >					—

3.

(15) Aufgabe 2 – zum Beispiel 3: Rot-Schwarz Baum

Gegeben sei folgende Deklaration eines Rot-Schwarz Baumes:

```
TYPE Baum =    OBJECT
                 pKnoten: ^Knoten;
                 FUNCTION schwarz: BOOLEAN;
                 FUNCTION istNIL: BOOLEAN;
               END;
     Knoten =  RECORD
                 links, rechts: Baum;
                 schwarzekante: BOOLEAN;
                 s: SCHLUESSEL;
                 w: WERT;
               END;
```

Die Methode `schwarz` liefert genau dann `TRUE` zurück wenn es sich bei der Kante, die auf den Knoten zeigt, um eine schwarze Kante handelt. `schwarz` von der Wurzel eines Rot-Schwarz Baumes gibt immer `TRUE`. Die Methode `istNIL` liefert genau dann `TRUE` zurück wenn `pKnoten` gleich `NIL` ist.

(13) **a)** Schreiben Sie eine *rekursive* Methode

```
PROCEDURE Baum.statistik(...);
```

die folgende Statistiken durch einmaliges Abarbeiten des Rot-Schwarz Baumes berechnet:

- Anzahl der Datensätze
- Anzahl der Knoten des äquivalenten 2-3-4 Baumes
- Tiefe des Baums

Verwenden Sie dabei KEINE GLOBALEN VARIABLEN. Geben Sie an, welche Übergabeparameter Ihre Methode verwendet.

(2) **b)** Geben Sie außerdem an, wie Ihre Methode vom Hauptprogramm aufgerufen wird.

(25) **Aufgabe 3 – zum Beispiel 7: Linked Trie**

Gegeben sei folgende Deklaration eines Linked Trie:

```
CONST M = 2147483647;
TYPE
  KnotenPtr = ^Node;
  Node = RECORD
           buchst        : CHAR;
           end           : BOOLEAN;
           nextbuchst    : KnotenPtr; { naechster Buchstabe im Wort }
           andererbuchst : KnotenPtr; { anderer Buchstabe an der }
                                      { selben Stelle im Wort    }
  END;
```

Sie sollen nun Suffix Compression für den Linked Trie implementieren, wobei die Endungen selbst als Linked Trie gespeichert werden. Dazu stehen ihnen folgende Hilfsfunktionen zur Verfügung:

Die `PROCEDURE einfuege(trie:KnotenPtr; hashwert:INTEGER);` trägt den übergebenen `trie` in einer globalen Hashtabelle mit dem gegebenen `hashwert` ein.

Die `FUNKTION suche(hashwert:INTEGER; trie:KnotenPtr): KnotenPtr;` sucht nach einem trie mit vorgegebenem `hashwert` in einer globalen Hashtabelle. Wenn der übergebene Trie gleich `nil` ist, so wird der erste Trie mit diesem `hashwert` zurückgegeben, bei weiteren Aufrufen mit dem selben `hashwert`, erhält man bei Übergabe des ersten gefundenen Tries, den nächsten mit dem selben `hashwert` usw. solange bis man `nil` erhält, wenn kein Trie mit dem gegebenen `hashwert` mehr in der Tabelle ist.

(7) a) Schreiben Sie nun zuerst die *rekursive, effiziente*

```
FUNCTION gleich(a, b : KnotenPtr) : BOOLEAN;
```

die genau dann `TRUE` zurück liefert, wenn die beiden übergebenen Linked Tries exakt die gleichen Worte enthalten. (Dazu beötigen Sie die Hashtabelle noch nicht.)

(18) b) Schreiben Sie nun die

```
PROCUDURE packe(VAR t : KnotenPtr; VAR hashwert : INTEGER);
```

die die eigentliche Suffix Compression durchführen soll. Verwenden Sie dazu `gleich`, `einfuege`, und `loesche`.

Zunächst soll diese Prozedur rekursiv die Hashwerte der Nodes im Trie berechnen, und über den `VAR` Parameter zurückliefern. Die Formel für die Hashfunktion ist folgende:

$$h(knoten) = \left(\sum_{i=1}^{26} (29 \cdot ord(buchst_i) + (h(subknoten_i) \; div \; 13)) \right) mod \; M$$

wobei die Summierung sich auf alle Buchstaben an der selben Stelle im Wort bezieht, $h(nil) = 0$ ist, und Buchstaben die nicht verwendet werden, nicht in die Summierung eingehen.

Sobald ein Hashwert berechnet wurde, kann in der Hashtabelle nach dem Trie gesucht werden, und wenn die entsprechende Endung gefunden wird, soll sie sofort ersetzt werden (auf korrektes Löschen des ersetzten Tries können sie verzichten). Wenn die Endung nicht gefunden wurde, so kann der Trie in die Hashtabelle eingetragen werden.

Algorithmen und Datenstrukturen 2

2. Übungstest
15. Jänner 1999

Gruppe **A**

1
2
3
4
Σ

Name Vorname Kennzahl Mat.-Nr. Beil.

Beispiel 1 (15 Punkte) - Genetische Algorithmen

Folgender Genotyp ist gegeben: `Genotyp = ARRAY[1..8] OF 'A'..'D'`

a) Geben Sie vier unterschiedliche Schemata an, die jeweils 64 Strings enthalten.

b) Geben Sie zu den folgenden Schemata an, welche Ordnung bzw. welche definierende Länge sie haben:

	Ordnung	definierende Länge
CCA*CACA		
*DD*****		
*CA***D*		
AB*A**D*		

c) Wieviele Schemata gibt es insgesamt für den gegebenen Genotyp?
Wieviele Schemata gibt es, die 'ADCCDCAD' enthalten?
Wieviele Schemata gibt es, die 'ADC**CDA' enthalten?

d) Schema S1='**ACA***' soll mit S2='*AD*****' mittels 1-Point-Crossover gekreuzt werden. Wie groß ist die Wahrscheinlich keit, daß S1 *und* S2 zerstört werden? Wie groß ist die Wahrscheinlichkeit, daß S1 *und* S2 erhalten bleiben?

e) Geben Sie alle Schemata an, die weder 'A' noch 'C' enthalten und durch 1-Point-Crossover nicht zerstört werden können.

Beispiel 2 (15 Punkte) - Rucksackproblem

Todd ißt gerne und viel in der Mensa. Zur Kostenersparnis werden dort Wertmarken angeboten, deren *Gegenwert* beim Einlösen an der Mensakasse höher ist als der *Einkaufspreis*. Diese Wertmarken werden in verschiedenen Kombinationen von Einkaufspreis/Gegenwert angeboten:

	A	B	C	D	E
Einkaufspreis	21	23	43	22	24
Gegenwert	39	33	51	27	35

Todd besitzt je eine Wertmarke vom Typ A, B, C, D und E und muß an der Mensakasse eine Rechnung von 69,- bezahlen. Dieses Problem löst er (im Kopf) als *Rucksackproblem*.

a) Geben Sie die Formel an, mit der Sie den *relativen Wert* eines Rucksackproblem-Gegenstandes aus Einkaufspreis und Gegenwert errechnen können.

b) Berechnen Sie die Auswahl an Wertmarken, um die maximale Ersparnis für das gegebene Essen zu erzielen. Verwenden Sie dazu das kombinierte Verfahren. Verwenden Sie die zeilenweise Notation, wie sie im Skriptum vorgestellt wurde. Markieren Sie die ausgestrichenen Knoten mit einem 'N', wenn sie nicht zulässig sind und mit einem 'S', wenn die Schranke zu klein ist. Berechnen Sie alle Werte auf genau 2 Dezimalstellen. Verwenden Sie folgende Darstellung für die Knoten:

Inhalt (z. B. K, L)	Schranke (z. B. 32.67)
Gewicht / Wert (z. B. 14/23)	mind. Erreichbar (z. B. 30)

Beispiel 3 (20 Punkte) - Probabilistische Algorithmen

Die Firma BILLO hat einen *Bananenmeister*, dessen Aufgabe es ist, die ankommenden Lieferungen von Bananen auf ihre Qualität zu überprüfen. Jede Bananenlieferung besteht aus 8000 Paletten zu je 20000 Bananen. Um zu prüfen, ob eine Lieferung von *guter* Qualität ist, wendet der Bananenmeister folgende *goldene Regeln* an, die er in langjähriger Erfahrung erworben hat:

1. Bei einer Lieferung von *guter* Qualität enthalten die ersten 100 Bananen, die aus einer Palette entnommen werden, nie mehr als 20 verfaulte Bananen.
2. Die Wahrscheinlichkeit, daß eine Lieferung von *schlechter* Qualität innerhalb der ersten 100 Bananen, die aus einer Palette entnommen werden, mehr als 20 verfaulte Bananen enthält, ist 40%.

Sie sollen nun die Überprüfung der Bananen auf EDV umstellen. Dazu stehen Ihnen folgende Hilfsmittel zur Verfügung: Eine Funktion

```
FUNCTION verfault(palettenNr: INTEGER; bananenNr: INTEGER): BOOLEAN;
```

liefert genau dann TRUE, wenn die angegebene Banane in der angegebenen Palette verfault ist. Weiters steht Ihnen ein geeigneter Zufallszahlengenerator

```
FUNCTION random(max: INTEGER): INTEGER;
```

welcher ganzzahlige Zufallszahlen im Intervall [0; max[erzeugt.

Schreiben Sie eine effiziente (nicht rekursive) Funktion

```
FUNCTION guteLieferung(proz: REAL): BOOLEAN;
```

welche mit mindestens `proz` % Sicherheit ermittelt, ob die vorliegende Lieferung von guter Qualität gemäß den goldenen Regeln des Bananenmeisters ist.

Algorithmen und Datenstrukturen 2

1 |
2 |
3 |
4 |

2. Übungstest
15. Jänner 1999

Gruppe **B**

Name Vorname Kennzahl Mat.-Nr. Beil.

Beispiel 1 (15 Punkte) - Rucksackproblem

Todd ißt gerne und viel in der Mensa. Zur Kostenersparnis werden dort Wertmarken angeboten, deren *Gegenwert* beim Einlösen an der Mensakasse höher ist als der *Einkaufspreis*. Diese Wertmarken werden in verschiedenen Kombinationen von Einkaufspreis/Gegenwert angeboten:

	A	B	C	D	E
Einkaufspreis	22	24	43	23	21
Gegenwert	27	35	51	33	39

Todd besitzt je eine Wertmarke vom Typ A, B, C, D und E und muß an der Mensakasse eine Rechnung von 69,- bezahlen. Dieses Problem löst er (im Kopf) als *Rucksackproblem*.

a) Geben Sie die Formel an, mit der Sie den *relativen Wert* eines Rucksackproblem-Gegenstandes aus Einkaufspreis und Gegenwert errechnen können.

b) Berechnen Sie die Auswahl an Wertmarken, um die maximale Ersparnis für das gegebene Essen zu erzielen. Verwenden Sie dazu das kombinierte Verfahren. Verwenden Sie die zeilenweise Notation, wie sie im Skriptum vorgestellt wurde. Markieren Sie die ausgestrichenen Knoten mit einem 'N', wenn sie nicht zulässig sind und mit einem 'S', wenn die Schranke zu klein ist. Berechnen Sie alle Werte auf genau 2 Dezimalstellen. Verwenden Sie folgende Darstellung für die Knoten:

Inhalt (z. B. K, L)	Schranke (z. B. 32.67)
Gewicht / Wert (z. B. 14/23)	mind. Erreichbar (z. B. 30)

Beispiel 2 (20 Punkte) - Probabilistische Algorithmen

Die Firma BILLO hat einen *Bananenmeister*, dessen Aufgabe es ist, die ankommenden Lieferungen von Bananen auf ihre Qualität zu überprüfen. Jede Bananenlieferung besteht aus 5000 Paletten zu je 15000 Bananen. Um zu prüfen, ob eine Lieferung von *guter* Qualität ist, wendet der Bananenmeister folgende *goldene Regeln* an, die er in langjähriger Erfahrung erworben hat:

1. Bei einer Lieferung von *guter* Qualität enthalten die ersten 200 Bananen, die aus einer Palette entnommen werden, nie mehr als 25 verfaulte Bananen.
2. Die Wahrscheinlichkeit, daß eine Lieferung von *schlechter* Qualität innerhalb der ersten 200 Bananen, die aus einer Palette entnommen werden, mehr als 25 verfaulte Bananen enthält, ist 20%.

Sie sollen nun die Überprüfung der Bananen auf EDV umstellen. Dazu stehen Ihnen folgende Hilfsmittel zur Verfügung: Eine Funktion

```
FUNCTION rotten(paletteNo: INTEGER; bananaNo: INTEGER): BOOLEAN;
```

liefert genau dann TRUE, wenn die angegebene Banane in der angegebenen Palette verfault ist. Weiters steht Ihnen ein geeigneter Zufallszahlengenerator

```
FUNCTION rnd(max: INTEGER): INTEGER;
```

welcher ganzzahlige Zufallszahlen im Intervall [0; max[erzeugt.

Schreiben Sie eine effiziente (nicht rekursive) Funktion

```
FUNCTION goodFreight(p: REAL): BOOLEAN;
```

welche mit mindestens p % Sicherheit ermittelt, ob die vorliegende Lieferung von guter Qualität gemäß den goldenen Regeln des Bananenmeisters ist.

Beispiel 3 (15 Punkte) - Genetische Algorithmen

Folgender Genotyp ist gegeben: `Genotyp = ARRAY[1..8] OF 'C'..'F'`

a) Geben Sie vier unterschiedliche Schemata an, die jeweils 64 Strings enthalten.

b) Geben Sie zu den folgenden Schemata an, welche Ordnung bzw. welche definierende Länge sie haben:

	Ordnung	definierende Länge
CCF*CFCF		
*DD*****		
*CF***D*		
FE*F**D*		

c) Wieviele Schemata gibt es insgesamt für den gegebenen Genotyp?
Wieviele Schemata gibt es, die 'EDCCDCED' enthalten?
Wieviele Schemata gibt es, die 'EDC**CDE' enthalten?

d) Schema S1='**DCD***' soll mit S2='*EC*****' mittels 1-Point-Crossover gekreuzt werden. Wie groß ist die Wahrscheinlich keit, daß S1 *und* S2 zerstört werden? Wie groß ist die Wahrscheinlichkeit, daß S1 *und* S2 erhalten bleiben?

e) Geben Sie alle Schemata an, die weder 'C' noch 'D' enthalten und durch 1-Point-Crossover nicht zerstört werden können.

Institut 186 für Computergraphik **Algorithmen und Datenstrukturen 2**		1. Test 27. Nov. 1998	**A**
Kennz.	**Matrikelnummer**	**Beilagen**	1 2 3
Nachname, Vorname			Σ

Beispiel 1 (18 Punkte)

Eine Telefonzentrale speichert alle Telefonnummern von Kunden in einem Indexed-Trie von folgendem Typ:

```
TYPE PDaten = ^Daten;
     TelNr = OBJECT
               FUNCTION gerade(tiefe: INTEGER): BOOLEAN;
               datenPtr: PDaten;
               FUNCTION leer: BOOLEAN;
             END;
     Daten = ARRAY[0..9] OF RECORD
               nachfolger: TelNr;
               endeFlag: BOOLEAN;
             END;
```

Implementieren Sie eine effiziente rekursive Methode

```
FUNCTION TelNr.gerade(tiefe: INTEGER): BOOLEAN;
```

die genau dann TRUE liefert, wenn für jede gespeicherte Telefonnummer gilt: Die Länge der Telefonnummer ist genau 8, 10 oder 12.

Geben Sie auch den Aufruf der Methode vom Hauptprogramm aus an.

Beispiel 2 (16 Punkte)

Ein PC-Versandhaus verwaltet seine Produkte mittels einer auf Extendible Hash basierenden Datenbank. Die Poduktnummern werden als Schlüssel verwendet, wobei die Hashfunktion gleich der *Ziffernsumme der Produktnummer* ist. Eine Datenseite faßt genau 2 Datensätze. Beim Splitten kommen die Datensätze mit dem größeren Hashwert auf die Datenseite mit der größeren Nummer.

(a)
Tragen Sie nacheinander zunächst "Monitor" (Prod.-Nr. 8984) und dann "Laserdrucker" (Prod.-Nr. 1100) in die folgende Datei ein. Zeichnen Sie jeweils nach jedem Eintragen die Datei neu.

```
Adressdatei
0xxxx | 1
1xxxx | 2
```

```
Datenseite 1
1416 | Scanner
0001 | Netzwerkkarte

Datenseite 2
4660 | Modem
9990 | Soundkarte
```

(b)
Tragen Sie nun in die nachfolgende Datei "ZipDrive" (Prod.-Nr. 7112) ein. Zeichnen Sie auch hier die Datei nach dem Eintragen des Datensatzes.

```
Adressdatei
xxxxx | 1
```

```
Datenseite 1
1416 | Scanner
1417 | Graphikkarte
```

Achtung - Fehler im Skriptum!
Die für dieses Beispiel relevaten Abb. 4.9 im Skriptum vom WS98/99 (S. 53) enthält einen Fehler. Der rechte Teil der Abbildung (Datenseiten, nachher) muss so aussehen (die Unterstreichungen weisen auf den Fehler hin):

```
3 - 000X xx
4 - 010X xx
5 - 100X xx
6 - 001X xx
```

Beispiel 3 (16 Punkte): Parsing durch mittelbare Rekursion

Folgendes Programm überprüft eine Beschreibungssprache für geometrische Objekte auf syntaktische Korrektheit. Zeichnen Sie dazu das Syntaxdiagramm. Verwenden Sie dieselbe Notation wie im Skriptum (Skriptum vom WS98/99: S. 72).

Erläuterungen: In der globalen Variable `ch` steht jeweils das aktuelle Zeichen. Die Funktion `NextChar` liest das nächste zu parsende Zeichen ein.

```
FUNCTION Keyword(s:STRING):BOOLEAN;
VAR i: INTEGER;
BEGIN
  i := 1;
  WHILE (i<=Length(s))AND(ch=s[i]) DO
  BEGIN
    NextChar; Inc(i)
  END;
  Keyword := i>Length(s);
END

FUNCTION Digit: BOOLEAN;
BEGIN
  IF ch IN '0'..'9' THEN BEGIN
    NextChar;
    Digit := TRUE
  END ELSE Digit := FALSE
END;

FUNCTION Number: BOOLEAN;
BEGIN
  Number := TRUE
  IF ch = '-' THEN BEGIN
    NextChar;
    IF NOT Digit THEN Number := FALSE
    ELSE WHILE Digit DO (*nothing*);
  END ELSE IF Digit THEN BEGIN
    WHILE Digit DO (* nothing *);
    IF ch = '.' THEN NextChar;
    WHILE Digit DO (* nothing *);
  END ELSE Number := FALSE;
END;

FUNCTION Tuple: BOOLEAN;
VAR v: BOOLEAN;
BEGIN
  v := Number;
  IF v THEN BEGIN
    v := ch = ',';
    IF v THEN BEGIN
      NextChar;
      v := Number;
      IF v THEN BEGIN
        v := ch = ',';
        IF v THEN BEGIN
          NextChar;
          v := Number
        END
      END
    END
  END;
  Tuple := v;
END;
```

```
FUNCTION ListOfTuples: BOOLEAN;
VAR v, flag: BOOLEAN;
BEGIN
  v := Tuple;
  WHILE v AND ch = ';' DO BEGIN
    NextChar;
    v := Tuple
  END
  ListOfTuples := v
END;

FUNCTION Node: BOOLEAN;
VAR v: BOOLEAN;
BEGIN
  Node := TRUE;
  IF Keyword("POINTSET") THEN
  BEGIN Node := ListOfTuples; END
  ELSE IF Keyword("BOX")THEN
  BEGIN Node := Tuple; END
  ELSE IF Keyword("CYLINDER")THEN
  BEGIN
    v := Number;
    IF v THEN BEGIN
      v := ch = ';';
      IF v THEN BEGIN
        v := Number;
      END
    END;
    Node := v;
  END
  ELSE
  IF Keyword("GROUP") THEN
  BEGIN
    v := ch = '(';
    IF v THEN BEGIN
      v := Node;
      WHILE v AND ch = '|' DO BEGIN
        NextChar;
        v := Node;
      END;
      IF ch = ')' THEN BEGIN
        NextChar;
        v := TRUE;
      END;
    END;
    Node := v;
  END ELSE Node := FALSE;
END;
```

Institut 186 für Computergraphik **Algorithmen und Datenstrukturen 2**		1. Test 27. Nov. 1998	**B**
Kennz.	**Matrikelnummer**	**Beilagen**	1 2 3
Nachname, Vorname			Σ

Beispiel 1 (16 Punkte)

Ein PC-Versandhaus verwaltet seine Produkte mittels einer auf Extendible Hash basierenden Datenbank. Die Poduktnummern werden als Schlüssel verwendet, wobei die Hashfunktion gleich der *Ziffernsumme der Produktnummer* ist. Eine Datenseite faßt genau 2 Datensätze. Beim Splitten kommen die Datensätze mit dem größeren Hashwert auf die Datenseite mit der größeren Nummer.

(a)
Tragen Sie nacheinander zunächst "Graphikkarte" (Prod.-Nr. 4997) und dann "DAT" (Prod.-Nr. 2000) in die folgende Datei ein. Zeichnen Sie jeweils nach jedem Eintragen die Datei neu.

```
Adressdatei
0xxxx | 1
1xxxx | 2
```

```
Datenseite 1
2361 | Modem
1000 | Streamer
```

```
Datenseite 2
2554 | Scanner
6498 | Netzwerkkarte
```

(b)
Tragen Sie nun in die nachfolgende Datei "Drucker" (Prod.-Nr. 6221) ein. Zeichnen Sie auch hier die Datei nach dem Eintragen des Datensatzes.

```
Adressdatei
xxxxx | 1
```

```
Datenseite 1
2361 | Modem
2371 | JazDrive
```

Achtung - Fehler im Skriptum!
Die für dieses Beispiel relevaten Abb. 4.9 im Skriptum vom WS98/99 (S. 53) enthält einen Fehler. Der rechte Teil der Abbildung (Datenseiten, nachher) muss so aussehen (die Unterstreichungen weisen auf den Fehler hin):

```
3 - 000X xx
4 - 010X xx
5 - 100X xx
6 - 001X xx
```

Beispiel 2 (16 Punkte): Parsing durch mittelbare Rekursion

Folgendes Programm überprüft eine Beschreibungssprache für geometrische Objekte auf syntaktische Korrektheit. Zeichnen Sie dazu das Syntaxdiagramm. Verwenden Sie dieselbe Notation wie im Skriptum (Skriptum vom WS98/99; S. 72).

Erläuterungen: In der globalen Variable ch steht jeweils das aktuelle Zeichen. Die Funktion LiesChar liest das nächste zu parsende Zeichen ein.

```
FUNCTION
Schluessel(s:STRING):BOOLEAN;
VAR i: INTEGER;
BEGIN
  i := 1;
  WHILE (i<=Length(s))AND(ch=s[i]) DO
  BEGIN
    LiesChar; Inc(i)
  END;
  Schluessel := i>Length(s);
END

FUNCTION Ziffer: BOOLEAN;
BEGIN
  IF ch IN '0'..'9' THEN BEGIN
    LiesChar;
    Ziffer := TRUE
  END ELSE Ziffer := FALSE
END;

FUNCTION Zahl: BOOLEAN;
BEGIN
  Zahl := TRUE
  IF ch = '.' THEN BEGIN
    LiesChar;
    IF NOT Ziffer THEN Zahl := FALSE
    ELSE WHILE Ziffer DO (*nothing*);
  END ELSE IF Ziffer THEN BEGIN
    WHILE Ziffer DO (* nothing *);
    IF ch = '.' THEN LiesChar;
    WHILE Ziffer DO (* nothing *);
  END ELSE Zahl := FALSE;
END;

FUNCTION Vektor: BOOLEAN;
VAR v: BOOLEAN;
BEGIN
  v := Zahl;
  IF v THEN BEGIN
    v := ch = ',';
    IF v THEN BEGIN
      LiesChar;
      v := Zahl;
      IF v THEN BEGIN
        v := ch = ',';
        IF v THEN BEGIN
          LiesChar;
          v := Zahl
        END
      END
    END
  END;
  Vektor := v;
END;
```

```
FUNCTION VektorListe: BOOLEAN;
VAR v, flag: BOOLEAN;
BEGIN
  v := Vektor;
  WHILE v AND ch = ';' DO BEGIN
    LiesChar;
    v := Vektor
  END
  VektorListe := v
END;

FUNCTION Knoten: BOOLEAN;
VAR v: BOOLEAN;
BEGIN
  Knoten := TRUE;
  IF Schluessel("PUNKTE") THEN
  BEGIN Knoten := VektorListe; END
  ELSE IF Schluessel("WUERFEL")THEN
  BEGIN Knoten := Vektor; END
  ELSE IF Schluessel("ZYLINDER")THEN
  BEGIN
    v := Zahl;
    IF v THEN BEGIN
      v := ch = ';';
      IF v THEN BEGIN
        v := Zahl;
      END
    END;
    Knoten := v;
  END
  ELSE
  IF Schluessel("GRUPPE") THEN
  BEGIN
    v := ch = '(';
    IF v THEN BEGIN
      v := Knoten;
      WHILE v AND ch = '|' DO BEGIN
        LiesChar;
        v := Knoten;
      END;
      IF ch = ')' THEN BEGIN
        LiesChar;
        v := TRUE;
      END;
    END;
    Knoten := v;
  END ELSE Knoten := FALSE;
END;
```

Beispiel 3 (18 Punkte)

Eine Telefonzentrale speichert alle Telefonnummern von Kunden in einem Indexed-Trie von folgendem Typ:

```
TYPE PNode = ^Node;
     Telephone = OBJECT
                   FUNCTION isOdd(depth: INTEGER): BOOLEAN;
                   NodePtr: PNode;
                   FUNCTION empty: BOOLEAN;
                 END;
     Node = ARRAY[0..9] OF RECORD
                   next: Telephone;
                   endFlag: BOOLEAN;
                 END;
```

Implementieren Sie eine effiziente rekursive Methode

```
FUNCTION Telephone.isOdd(depth: INTEGER): BOOLEAN;
```

die genau dann TRUE liefert, wenn für jede gespeicherte Telefonnummer gilt: Die Länge der Telefonnummer ist genau 7, 9 oder 11.

Geben Sie auch den Aufruf der Methode vom Hauptprogramm aus an.

Institut 186 für Computergraphik	**Algorithmen und Datenstrukturen 2**			2. Test 16. Jan. 1998 Gruppe A
Matrikelnummer	**Beilagen**‡	1 (18)	2 (16)	3 (16)
NACHNAME, Vorname				Σ (50)

‡ Geben Sie an, wieviele Zusatzbl. Sie abgeben (**jedes mit Name & Matr.-Nr. beschriftet!**).

Bemerkung: Lesen Sie zu jeder Aufgabe (im eigenen Interesse) die Angabe zuerst einmal *ganz* durch und beginnen Sie erst dann mit dem Lösen!

(18) Aufgabe 1 – Probabilistischer Algorithmus

Gegeben ist (1.) eine `FUNCTION zufZahl: INTEGER`, die (angeblich) `INTEGER`-Zufallszahlen erzeugt. Mit jedem Aufruf wird eine neue (Zufalls-)Zahl generiert, sodaß durch wiederholten Aufruf dieser Funktion eine ganze Reihe von (Zufalls-)Zahlen erzeugt werden kann. Gegeben ist (2.) eine `PROCEDURE zufInit(wert: INTEGER)`, die eine Initialisierung dieses Prozesses durchführt. Abhängig von `wert` (beim Aufruf von `zufInit`) erzeugt `zufZahl` immer dieselbe Sequenz von (Zufalls-)Zahlen. Weiters definieren wir für diese Aufgabe folgendes:

1. Die ersten 1024 Zahlen, die *ein guter Zufallszahlengenerator* erzeugt, enthalten *nie* weniger als 256 gerade Zahlen.
2. Die Wahrscheinlichkeit, daß *ein schlechter Zufallszahlengenerator* innerhalb der ersten 1024 Zahlen weniger als 256 gerade Zahlen generiert, ist 40%.

$\Longrightarrow$ Schreiben Sie nun analog zum probabilistischen Ansatz von Rabin eine effiziente (aber nicht rekursive)

```
FUNCTION istGut(prozent: REAL): BOOLEAN,
```

welche mit mindesten `prozent`% Sicherheit ermittelt, ob `zufZahl` zusammen mit `zufInit` gemäß den oben definierten Kriterien ein guter Zufallszahlengenerator ist. Mit der `FUNCTION rand: INTEGER` steht Ihnen dafür ein wirklich guter Zufallszahlengenerator zur Verfügung.

(16) Aufgabe 2 – Rekursive Funktionen

Zur Berechnung einer rekursiven Funktion $f : \mathrm{N_0}^2 \to \mathrm{N}$ (vgl. Ackermann-Funktion, Fibonacci-Zahlen, etc.) steht ein Objekt `eff` zur Verfügung, welches mittels einer Hash-Tabelle (Linear Probing) die Mehrfach-Auswertung von f bei gleichen Parametern verhindert. Konstruktor `eff.init` löscht alle Einträge in der Tabelle. Methode `eff.f(n,m)` berechnet $f(\mathtt{n},\mathtt{m})$ und legt alle dabei berechneten Werte von $f(i,j)$ in der internen Hash-Tabelle ab. Mit Methode `eff.plot` werden alle in der Hash-Tabelle abgespeicherten Werte (jeweils `i`, `ht[i].n`, `ht[i].m` und `ht[i].f`) ausgegeben.

```
TYPE TEin = RECORD       n,m,f: INTEGER                          END;
     TEff = OBJECT         ht: ARRAY [0..40] OF TEin;
              CONSTRUCTOR init;
                PROCEDURE plot;
                 FUNCTION    f(n,m: INTEGER): INTEGER;          END;

CONSTRUCTOR TEff.init; VAR i: INTEGER;
BEGIN FOR i:= 0 TO 40 DO ht[i].n:= -1 ;                         END;

PROCEDURE TEff.plot; VAR i: INTEGER;
BEGIN FOR i:=0 TO 40 DO IF ht[i].f<>0 THEN WITH ht[i] DO
  writeLn("ht[",i,"]:   n=",n,"   m=",m,"   f=",f)              END;

FUNCTION TEff.f(n,m: INTEGER): INTEGER;
VAR i: INTEGER;
BEGIN i:= (6*n+m) MOD 41;                    {...Hashwert berechnen}
  WHILE ht[i].n>-1 AND (ht[i].n<>n OR ht[i].m<>m) DO
    i:= (i+1) MOD 41;                              {...Linear Probing}
  IF (ht[i].n<0) {...f(n,m) noch nicht in ht...} THEN
  BEGIN ht[i].n:= n; ht[i].m:= m;
    IF      m=0 THEN ht[i].f:=          n+1
    ELSE IF m=1 THEN ht[i].f:= f(n+1,m-1)+2
    ELSE             ht[i].f:= f(n+2,m-2)+f(n+1,m-1) END;
  f:= ht[i].f {...f(n,m) ist (jetzt) in ht...}                  END;

VAR eff: TEff; erg: INTEGER;
```

$\Longrightarrow$ Welche Ausgabe erzeut die folgende Befehlszeile (im Hauptprogramm)? Geben Sie als Begründung für Ihre Lösung den (rekursiven) Berechnungsweg von `eff.f(1,5)` an.

```
eff.init; erg:= eff.f(1,5); eff.plot
```

(16) Aufgabe 3 – Rucksack-Problem

Ein Kaufmann mit Boot steht vor folgendem Problem: Er hat fünf Container ('A', 'B', 'C', 'D' und 'E') mit wertvollen Waren, die aber gemeinsam zu schwer sind, um sie gleichzeitig auf sein Boot zu laden. Das Boot kann maximal 170 Tonnen transportieren. Der Kaufmann muß nun entscheiden, welche Container er mitnimmt. Natürlich ist er daran interessiert, daß der Gesamtwert der mitgenommenen Ware maximal ist, also handelt es sich um ein klassisches Rucksackproblem. Gewichte und Werte der einzelnen Container sind in folgender Tabelle (ungeordnet) aufgelistet:

Kennbuchstabe	'A'	'B'	'C'	'D'	'E'
Gewicht	100t	90t	75t	80t	65t
Wert	900$	800$	500$	750$	600$

$\Longrightarrow$ Zeichnen Sie nun den Entscheidungsbaum, der sich für dieses Problem ergibt. Geben Sie für jeden Knoten folgende vier Attribute an:

- alle geladenen Container (deren Kenn-Buchstaben)
- das Gesamtgewicht (in t)
- den Gesamtwert (in $)
- und die Schranke (in $)

Ein Knoten könnte, zum Beispiel, folgendermaßen aussehen:

XY, 340t, 1500$, 1912.13$

Verwenden Sie für die Konstruktion des Entscheidungsbaums Backtracking

- mit Abschneiden von unzulässigen Knoten (Ⓤ) und
- mit Abschneiden von Knoten mit zu kleiner Schranke (Ⓢ).

Geben Sie jedesmal, wenn Sie einen Knoten abschneiden, deutlich an, warum dieser Knoten abgeschnitten werden kann, also bei Ⓤ das (zu große) Gewicht und bei Ⓢ die (zu kleine) Schranke. Knoten, die Sie abschneiden, sollen nicht weiterverfolgt werden, d.h., Sie müssen nicht den maximalen Entscheidungsbaum zeichnen! Vergessen Sie nicht, `bisherbeste` entsprechend verwenden.

Institut 186 für Computergraphik	**Algorithmen und Datenstrukturen 2**			2. Test 16. Jan. 1998 Gruppe B
Matrikelnummer	**Beilagen**‡	1 (18)	2 (16)	3 (16)
NACHNAME, Vorname				Σ (50)

‡ Geben Sie an, wieviele Zusatzbl. Sie abgeben (**jedes mit Name & Matr.-Nr. beschriftet!**).

Bemerkung: Lesen Sie zu jeder Aufgabe (im eigenen Interesse) die Angabe zuerst einmal *ganz* durch und beginnen Sie erst dann mit dem Lösen!

(18) Aufgabe 1 – Probabilistischer Algorithmus

Gegeben ist (1.) eine `FUNCTION zufZahl: INTEGER`, die (angeblich) `INTEGER`-Zufallszahlen erzeugt. Mit jedem Aufruf wird eine neue (Zufalls-)Zahl generiert, sodaß durch wiederholten Aufruf dieser Funktion eine ganze Reihe von (Zufalls-)Zahlen erzeugt werden kann. Gegeben ist (2.) eine `PROCEDURE zufInit(wert: INTEGER)`, die eine Initialisierung dieses Prozesses durchführt. Abhängig von `wert` (beim Aufruf von `zufInit`) erzeugt `zufZahl` immer dieselbe Sequenz von (Zufalls-)Zahlen. Weiters definieren wir für diese Aufgabe folgendes:

1. Die ersten 1024 Zahlen, die *ein guter Zufallszahlengenerator* erzeugt, enthalten *nie* weniger als 256 gerade Zahlen.
2. Die Wahrscheinlichkeit, daß *ein schlechter Zufallszahlengenerator* innerhalb der ersten 1024 Zahlen weniger als 256 gerade Zahlen generiert, ist 40%.

$\Longrightarrow$ Schreiben Sie nun analog zum probabilistischen Ansatz von Rabin eine effiziente (aber nicht rekursive)

```
FUNCTION istGut(prozent: REAL): BOOLEAN,
```

welche mit mindesten `prozent`% Sicherheit ermittelt, ob `zufZahl` zusammen mit `zufInit` gemäß den oben definierten Kriterien ein guter Zufallszahlengenerator ist. Mit der `FUNCTION rand: INTEGER` steht Ihnen dafür ein wirklich guter Zufallszahlengenerator zur Verfügung.

(16) Aufgabe 2 – Rucksack-Problem

Ein Kaufmann mit Boot steht vor folgendem Problem: Er hat fünf Container ('A', 'B', 'C', 'D' und 'E') mit wertvollen Waren, die aber gemeinsam zu schwer sind, um sie gleichzeitig auf sein Boot zu laden. Das Boot kann maximal 340 Tonnen transportieren. Der Kaufmann muß nun entscheiden, welche Container er mitnimmt. Natürlich ist er daran interessiert, daß der Gesamtwert der mitgenommenen Ware maximal ist, also handelt es sich um ein klassisches Rucksackproblem. Gewichte und Werte der einzelnen Container sind in folgender Tabelle (ungeordnet) aufgelistet:

Kennbuchstabe	'A'	'B'	'C'	'D'	'E'
Gewicht	200t	180t	160t	150t	130t
Wert	900$	800$	750$	500$	600$

$\Longrightarrow$ Zeichnen Sie nun den Entscheidungsbaum, der sich für dieses Problem ergibt. Geben Sie für jeden Knoten folgende vier Attribute an:

- alle geladenen Container (deren Kenn-Buchstaben)
- das Gesamtgewicht (in t)
- den Gesamtwert (in $)
- und die Schranke (in $)

Ein Knoten könnte, zum Beispiel, folgendermaßen aussehen:

XY, 170t, 1500$, 1912.13$

Verwenden Sie für die Konstruktion des Entscheidungsbaums Backtracking

- mit Abschneiden von unzulässigen Knoten (Ⓤ) und
- mit Abschneiden von Knoten mit zu kleiner Schranke (Ⓢ).

Geben Sie jedesmal, wenn Sie einen Knoten abschneiden, deutlich an, warum dieser Knoten abgeschnitten werden kann, also bei Ⓤ das (zu große) Gewicht und bei Ⓢ die (zu kleine) Schranke. Knoten, die Sie abschneiden, sollen nicht weiterverfolgt werden, d.h., Sie müssen nicht den maximalen Entscheidungsbaum zeichnen! Vergessen Sie nicht, `bisherbeste` entsprechend verwenden.

(16) Aufgabe 3 – Rekursive Funktionen

Zur Berechnung einer rekursiven Funktion $f : \mathbf{N_0}^2 \to \mathbf{N}$ (vgl. Ackermann-Funktion, Fibonacci-Zahlen, etc.) steht ein Objekt `eff` zur Verfügung, welches mittels einer Hash-Tabelle (Linear Probing) die Mehrfach-Auswertung von f bei gleichen Parametern verhindert. Konstruktor `eff.init` löscht alle Einträge in der Tabelle. Methode `eff.f(n,m)` berechnet f(n,m) und legt alle dabei berechneten Werte von $f(i,j)$ in der internen Hash-Tabelle ab. Mit Methode `eff.plot` werden alle in der Hash-Tabelle abgespeicherten Werte (jeweils `i`, `ht[i].n`, `ht[i].m` und `ht[i].f`) ausgegeben.

```
TYPE TEin = RECORD       n,m,f: INTEGER                          END;
     TEff = OBJECT         ht: ARRAY [0..40] OF TEin;
              CONSTRUCTOR init;
                PROCEDURE plot;
                 FUNCTION    f(n,m: INTEGER): INTEGER;          END;

CONSTRUCTOR TEff.init; VAR i: INTEGER;
BEGIN FOR i:= 0 TO 40 DO ht[i].n:= -1 ;                         END;

PROCEDURE TEff.plot; VAR i: INTEGER;
BEGIN FOR i:=0 TO 40 DO IF ht[i].f<>0 THEN WITH ht[i] DO
  writeLn("ht[",i,"]:   n=",n,"   m=",m,"   f=",f)              END;

FUNCTION TEff.f(n,m: INTEGER): INTEGER;
VAR i: INTEGER;
BEGIN i:= (n+7*m) MOD 41;                  {...Hashwert berechnen}
  WHILE ht[i].n>-1 AND (ht[i].n<>n OR ht[i].m<>m) DO
    i:= (i+1) MOD 41;                          {...Linear Probing}
  IF (ht[i].n<0) {...f(n,m) noch nicht in ht...} THEN
  BEGIN ht[i].n:= n; ht[i].m:= m;
    IF      n=0 THEN ht[i].f:=          1+m
    ELSE IF n=1 THEN ht[i].f:=          2+f(n-1,m+1)
    ELSE             ht[i].f:= f(n-2,m+2)+f(n-1,m+1) END;
  f:= ht[i].f {...f(n,m) ist (jetzt) in ht...}                  END;

VAR eff: TEff; erg: INTEGER;
```

$\Longrightarrow$ Welche Ausgabe erzeut die folgende Befehlszeile (im Hauptprogramm)? Geben Sie als Begründung für Ihre Lösung den (rekursiven) Berechnungsweg von `eff.f(5,0)` an.

```
eff.init; erg:= eff.f(5,0); eff.plot
```

Institut 186 für Computergraphik	Algorithmen und Datenstrukturen 2			1. Test 28. Nov. 1997 Gruppe B
Matrikelnummer	Beilagen‡	1 (12)	2 (20)	3 (18)
NACHNAME, Vorname				Σ (50)

‡ Geben Sie an, wieviele Zusatzbl. Sie abgeben (jedes mit Name & Matr.-Nr. beschriftet!).

Bemerkung: Lesen Sie zu jeder Aufgabe (im eigenen Interesse) die Angabe zuerst einmal *ganz* durch und beginnen Sie erst dann mit dem Lösen!

Aufgabe 1 – m-Wege-Mischen

Auf einem Band ist folgende Buchstabenfolge gespeichert:

Band 1: • I • D • K • B • C • J • E • G • A • L • M • F • H •

Sortieren Sie die angegebene Folge alphabetisch mittels m-Wege-Mischen. Nehmen Sie m = 3 sowie einen Hauptspeicher mit 4 Blöcken an (1 Buchstabe pro Block!). Geben Sie dabei *nach jedem* Schritt, d.h. nach dem Aufteilen, nach dem ersten Mischen, usw., das folgende Formular ausgefüllt an. Markieren Sie dabei sortierte Teilfolgen wie angegeben mit '•'.

Band 1: Buchstabenfolge auf Band 1 ...
Band 2: ...
usw. (überlegen Sie, wie viele Bänder Sie brauchen!) ...
⋮

Bemerkungen: • Verwenden Sie den im Skriptum primär vorgestellten Algorithmus - die Optimierungen (extra-lange Anfangsteilfolgen beim Aufteilen, Mehrphasen-Sortieren) sind *nicht* zu berücksichtigen. • Für das Sortieren steht zusätzlich zu den 4 Blöcken Hauptspeicher noch eine Zahl zusätzlich zur Verfügung.

Nehmen Sie nun folgende Voraussetzungen für das m-Wege-Mischen an: 8 verfügbare Bänder, 10 Blöcke Haupspeicher (1 Block ↔ 100 Datensätze). Wie oft muß gemischt werden, damit 17634 Datensätze, die anfangs unsortiert auf einem Band vorliegen, sortiert werden können. Begründen Sie Ihr Ergebnis!

(20) Aufgabe 3 – Linked-Trie

Ein Linked-Trie zum Speichern von Zahlen ist wie folgt gegeben:

```
TYPE KnoPtr = ^Kno;
     LTrie =      OBJECT  pRoot: KnoPtr;
             CONSTRUCTOR   init;
              DESTRUCTOR   done;
                FUNCTION  isNIL: BOOLEAN;
                FUNCTION getKno: KnoPtr;
                PROCEDURE setKno(p: KnoPtr); END;
       Kno =      RECORD   ziff: '0'..'9';
                           ende: BOOLEAN;
                           nach: LTrie;
                          ander: LTrie;     END;

CONSTRUCTOR LTrie.init;   BEGIN pRoot:= NIL;                END;
DESTRUCTOR  LTrie.done;   BEGIN {loescht ganzen Trie...} END;
FUNCTION    LTrie.isNil;  BEGIN isNil:= (pRoot=NIL);      END;
FUNCTION    LTrie.getKno; BEGIN getKno:= pRoot;           END;
PROCEDURE   LTrie.setKno(p: KnoPtr); BEGIN pRoot:= p;     END;
```

Bemerkung zu `Kno`: `nach` ist wie `naechsterbuchstabe` (im Skriptum) zu verstehen und `ander` wie `andererbuchstabe`.

(15) a) Schreiben Sie in OOP-Pascal eine *rekursive* Methode

```
PROCEDURE LTrie.einf(str:STRING; von,bis:INTEGER),
```

welche die Zahl, deren Ziffern in den Buchstaben `str[von]`...`str[bis]` gespeichert sind, in den Linked-Trie einträgt. Gehen Sie davon aus, daß beim ersten Aufruf von `einf` die Argumente sinnvoll belegt sind, zum Beispiel also `von`>0 und `bis`≤`length(str)` sicher erfüllt ist. Verwenden Sie `init`, `isNIL`, `getKno` und `setKno` wie angegeben.

(5) b) 1. Wieoft wird Methode `einf` bei nachfolgender Befehlszeile aufgerufen?
`s:='12345'; lt.einf(s,2,2); lt.einf(s,2,5); lt.einf(s,3,5);`
2. Wieviele Knoten (`Kno`) hat `lt` nach obiger Befehlszeile?

Achtung: • Geben Sie jeweils auf den Berechnungsweg für die Lösung an, eine richtige Lösung ohne Erklärung wird nicht gewertet! • Gehen Sie für beide Fragen davon aus, daß `lt` vor der angegebenen Befehlszeile leer ist.

PRÜFUNGSORDNER - ein Service Deiner Fachschaft Informatik! 26,-
LVA: ALGODAT 2 - UE
Preis:

(18) **Aufgabe 2 – Hashtabelle**

Gegeben ist die folgende Hashtabelle:

```
VAR ht: ARRAY [0..3799] OF HTEintrag;
```

Generelle Bemerkung: Geben Sie im Weiteren *zu jeder Frage* eine Antwort *mit Begründung* (Rechenweg, verwendete Formel, etc.) an. Selbst richtige Lösungen, aus denen nicht hervorgeht, wie es zu ihnen kommt, können nicht gewertet werden!

(6) 1. Geben Sie für ht eine geeignete Hashfunktion an. Nehmen sie an, daß die Schlüssel folgenden Typ haben:

```
TYPE SchlTyp = ARRAY [1..14] OF 'A'..'Z'
```

2. Geben Sie weiters eine geeignete Kollisionsfunktion an.

(5) Gehen Sie zunächst davon aus, daß bereits 1745 Datensätze in ht eingetragen wurden. Wieviele Schritte sind nun 1. mindestens, 2. im Mittel bzw. 3. maximal notwendig, um nach einem gespeicherten Datensatz zu suchen?

(7) 1. Wieviele Datensätze müssen in ht gespeichert sein, damit das Suchen nach einem *nicht* vorhandenen Datensatz (im Mittel) 2.5 Schritte braucht?

2. Ab welchem Füllgrad α ist ht (im Mittel) langsamer bei der Suche nach einem *nicht* vorhandenen Datensatz als ein AVL-Baum?

Hinweise: • Gehen Sie vom denkbar schlechtesten AVL-Baum aus und vergleichen Sie beide Datenstrukturen auf der Basis von Suchschritten.
• Das Ableiten einer analytischen Formel ist hier nicht gefragt, α muß nur *auf ganze Prozent* genau angegeben werden!

Institut 186 für Computergraphik	**Algorithmen und Datenstrukturen 2**			1. Test 28. Nov. 1997 Gruppe A
Matrikelnummer	**Beilagen**‡	1 (12)	2 (18)	3 (20)
NACHNAME, Vorname				Σ (50)

‡ Geben Sie an, wieviele Zusatzbl. Sie abgeben (jedes mit Name & Matr.-Nr. beschriftet!).

Bemerkung: Lesen Sie zu jeder Aufgabe (im eigenen Interesse) die Angabe zuerst einmal *ganz* durch und beginnen Sie erst dann mit dem Lösen!

(12) **Aufgabe 1 – m-Wege-Mischen**

Auf einem Band ist folgende Buchstabenfolge gespeichert:

Band 1: • V • Y • T • W • U • N • Z • O • R • Q • X • P • S •

(8) a) Sortieren Sie die angegebene Folge alphabetisch mittels m-Wege-Mischen. Nehmen Sie m = 3 sowie einen Hauptspeicher mit 4 Blöcken an (1 Buchstabe pro Block!). Geben Sie dabei *nach jedem* Schritt, d.h. nach dem Aufteilen, nach dem ersten Mischen, usw., das folgende Formular ausgefüllt an. Markieren Sie dabei sortierte Teilfolgen wie angegeben mit '•'.

Band 1: Buchstabenfolge auf Band 1 ...
Band 2: ...
usw. (überlegen Sie, wie viele Bänder Sie brauchen!) ...
⋮

Bemerkungen: • Verwenden Sie den im Skriptum primär vorgestellten Algorithmus - die Optimierungen (extra-lange Anfangsteilfolgen beim Aufteilen, Mehrphasen-Sortieren) sind *nicht* zu berücksichtigen. • Für das Sortieren steht zusätzlich zu den 4 Blöcken Hauptspeicher noch eine Zahl zusätzlich zur Verfügung.

(4) b) Nehmen Sie nun folgende Voraussetzungen für das m-Wege-Mischen an: 10 verfügbare Bänder, 24 Blöcke Haupspeicher (1 Block ↔ 50 Datensätze). Wie oft muß gemischt werden, damit 21345 Datensätze, die anfangs unsortiert auf einem Band vorliegen, sortiert werden können. Begründen Sie Ihr Ergebnis!

(18) **Aufgabe 3 – Hashtabelle**

Gegeben ist die folgende Hashtabelle:

```
VAR ht: ARRAY [0..8199] OF HTEintrag;
```

Generelle Bemerkung: Geben Sie im Weiteren *zu jeder Frage* eine Antwort *mit Begründung* (Rechenweg, verwendete Formel, etc.) an. Selbst richtige Lösungen, aus denen nicht hervorgeht, wie es zu ihnen kommt, können nicht gewertet werden!

5) Gehen Sie zunächst davon aus, daß bereits 3745 Datensätze in ht eingetragen wurden. Wieviele Schritte sind nun [1.] mindestens, [2.] im Mittel bzw. [3.] maximal notwendig, um nach einem gespeicherten Datensatz zu suchen?

7) [1.] Wieviele Datensätze müssen in ht gespeichert sein, damit das Suchen nach einem *nicht* vorhandenen Datensatz (im Mittel) 2.5 Schritte braucht?

[2.] Ab welchem Füllgrad α ist ht (im Mittel) langsamer bei der Suche nach einem *nicht* vorhandenen Datensatz als ein AVL-Baum?

Hinweise: • Gehen Sie vom denkbar schlechtesten AVL-Baum aus und vergleichen Sie beide Datenstrukturen auf der Basis von Suchschritten.
• Das Ableiten einer analytischen Formel ist hier nicht gefragt, α muß nur *auf ganze Prozent* genau angegeben werden!

9) [1.] Geben Sie für ht eine geeignete Hashfunktion an. Nehmen sie an, daß die Schlüssel folgenden Typ haben:

```
TYPE SchlTyp = ARRAY [1..42] OF 'a'..'z'
```

[2.] Geben Sie weiters eine geeignete Kollisionsfunktion an.

(20) **Aufgabe 2 – Linked-Trie**

Ein Linked-Trie zum Speichern von Zahlen ist wie folgt gegeben:

```
TYPE  NodeP = ^Node;
      LTrie =       OBJECT  pRoot: NodeP;
              CONSTRUCTOR   init;
               DESTRUCTOR   done;
                 FUNCTION  isNIL: BOOLEAN;
                 FUNCTION getNod: NodeP;
                PROCEDURE setNod(p: NodeP); END;
       Node =       RECORD   lett: '0'..'9';
                             ende: BOOLEAN;
                             next: LTrie;
                            other: LTrie;     END;

CONSTRUCTOR LTrie.init;   BEGIN pRoot:= NIL;                 END;
DESTRUCTOR  LTrie.done;   BEGIN {loescht ganzen Trie...} END;
FUNCTION    LTrie.isNil;  BEGIN isNil:= (pRoot=NIL);       END;
FUNCTION    LTrie.getNod; BEGIN getNod:= pRoot;            END;
PROCEDURE   LTrie.setNod(p: NodeP); BEGIN pRoot:= p;     END;
```

Bemerkung zu Node: next ist wie naechsterbuchstabe (im Skriptum) zu verstehen und other wie andererbuchstabe.

(15) a) Schreiben Sie in OOP-Pascal eine *rekursive* Methode

```
PROCEDURE LTrie.ins(str:STRING; from,to:INTEGER),
```

welche die Zahl, deren Ziffern in den Buchstaben str[from]...str[to] gespeichert sind, in den Linked-Trie einträgt. Gehen Sie davon aus, daß beim ersten Aufruf von ins die Argumente sinnvoll belegt sind, zum Beispiel also from>0 und to≤length(str) sicher erfüllt ist. Verwenden Sie init, isNIL, getNod und setNod wie angegeben.

(5) b) [1.] Wieoft wird Methode ins bei nachfolgender Befehlszeile aufgerufen?
s:='98765'; lt.ins(s,1,1); lt.ins(s,1,4); lt.ins(s,3,5);
[2.] Wieviele Knoten (Node) hat lt nach obiger Befehlszeile?

Achtung: • Geben Sie jeweils auf den Berechnungsweg für die Lösung an, eine richtige Lösung ohne Erklärung wird nicht gewertet! • Gehen Sie für beide Fragen davon aus, daß lt vor der angegebenen Befehlszeile leer ist.

Institut 186 für Computergraphik	Algorithmen und Datenstrukturen 2		Bsp.-Test 31. Januar 1997 Gruppe B	
Matrikelnummer	**Beilagen**‡	1 (18)	2 (21)	3 (11)
NACHNAME, Vorname				Σ (50)

‡ Geben Sie an, wieviele Zusatzbl. Sie abgeben (jedes mit Name & Matr.-Nr. beschriftet!).

Zur Erinnerung:
Alle Aufgaben beziehen sich auf die 10 Übungsbeispiele im WS '96/'97.

(18) Aufgabe 1 – zu den Beispielen 1 und 7

Generell gilt: Lösungen, aus denen nicht hervorgeht, wie sie erarbeitet wurden, werden nicht anerkannt. Schreiben Sie die Zwischenergebnisse auf.

(6) a) **zu Beispiel 1:** [1.] Wenn nachfolgende Kontrolldatei gegeben ist, welche zusammenfassenden Informationen (`summary insert ...`, 3 Zeilen) sind vom beschriebenen Programm zu erwarten? [2.] Zeichnen Sie außerdem beide Bäume, wie sie nach dem Abarbeiten der Kontrolldatei aussehen.

```
i: 1111111, 881, Eins
i: 2222222, 881, Zwei
i: 3333333, 881, Drei
i: 4444444, 881, Vier
i: 6666666, 881, Sechs
i: 7777777, 881, Sieben
i: 5555555, 881, Fuenf
```

(21) Aufgabe 2 – zum Beispiel 5: B-Baum

Gegeben sei folgende Deklaration eines B-Baums:

```
TYPE
  BNodePtr = ^BNode;
  BTree = OBJECT pRoot: BNodePtr;      {Zeiger auf Wurzel}
                 t: INTEGER;           {Tiefe des Baumes}
        FUNCTION depth: INTEGER;
         FUNCTION full: BOOLEAN;
       PROCEDURE setIt(p: BNodePtr);
  END;
  BNode = RECORD  reg: ARRAY [1..M] OF Registration;
                 next: ARRAY [0..M] OF BTree;
                  num: 1..M;
  END;

FUNCTION BTree.depth: INTEGER; BEGIN depth:=t; END;
```

Bemerkungen: → `M` ist eine vorgegebene, *ungerade* Konstante. → `Registration` ist ein `RECORD` mit Schlüssel `mat` (Typ `INTEGER`). → `b.full` ergibt genau dann `TRUE`, wenn der Wurzelknoten von `b` voll ist. → `b.setIt(p)` initialisiert den B-Baum `b` mit Wurzel `p`. → Gehen Sie davon aus, daß `pRoot` und `t` private Variable sind, verwenden Sie die Zugriffsmethoden (`full`, usw.)! → Die Tiefe des leeren B-Baums ist 0;

(7) a) Schreiben Sie die

```
FUNCTION BTree.split(VAR a: Registration): BNodePtr,
```

die vom Wurzelknoten des B-Baums jene Hälfte abspaltet, welche die Anmeldungen mit den größeren Matrikelnummern enthält und in `a` den mittleren Datensatz retourniert. Die abgespaltene Hälfte der Wurzel soll als `BNode` mittels dem Funktionsergebnis zurückgegeben werden werden. Gehen Sie davon aus, daß die Wurzel des B-Baums voll ist.

(14) b) Schreiben Sie (unter Zuhilfenahme von `split`) die

```
PROCEDURE BTree.insert2(a: Registration),
```

welche eine Anmeldung `a` in den B-Baum einträgt. Gehen Sie (bei Aufruf `b.insert2(a)`) von folgenden Voraussetzungen aus: `b.full=FALSE` und `b.depth>1`, d.h., Sie müssen nur das Eintragen in einen Zwischenknoten programmieren! **Wichtig**: Splitten Sie schon beim Suchen (wie im Skriptum beschrieben), wenn Sie auf einen vollen Teilbaum stoßen.

(12) b) **zu Beispiel 7:** [1.] Erstellen Sie aus der folgenden Angebotsdatei die Kostenmatrix. [2.] Geben Sie zudem auch noch die reduzierte Kostenmatrix an. Reduzieren Sie zuerst die Zeilen, dann die Spalten. Wenden Sie nur Subtraktionen an. [3.] Um wieviel ist die billigste Rundreise bezüglich der reduzierten Kostenmatrix günstiger gegenüber der ursprünglichen Kostenmatrix?

```
Wien > Paris     = 82
Wien > Berlin    = 31
Wien > Rom       = 52
Paris > London   = 14
Paris > Rom      = 43
Paris > Berlin   = 66
Rom > Berlin     = 93
Rom > Wien       = 55
Rom > Paris      = 42
London > Berlin  = 24
London > Paris   = 18
London > Wien    = 70
Berlin > London  = 26
Berlin > Wien    = 32
Berlin > Rom     = 98
```

[1.]	W.	P.	R.	L.	B.
Wien >	—				
Paris >		—			
Rom >			—		
London >				—	
Berlin >					—

[2.]	W.	P.	R.	L.	B.
Wien >	—				
Paris >		—			
Rom >			—		
London >				—	
Berlin >					—

[3.]

(11) **Aufgabe 3 – zum Beispiel 9: Nearest Pair**

Auf einem Planeten in Torusform (Schwimmreifen) werden Orte in Listenform gespeichert. Es gibt 36*12 Listen (`alleOrte`), in denen alle Orte auf dem Planeten verzeichnet sind:

```
TYPE
  TOrtPtr = ^TOrt;                    {Zeiger => Liste}
  TOrt = RECORD name:    STRING;      {eindeutiger Name}
                rho,phi: REAL;        {Koordinaten (Winkel!)}
                next:    TOrtPrt;     {naechster Ort}
  END;
VAR
  alleOrte: ARRAY [0..35,0..11] OF TOrtPtr;  {alle Orte}
```

Bemerkung: Für alle Orte in `alleOrte[i,j]` gilt, daß $i \cdot 10° \leq$ `rho` $< (i+1) \cdot 10°$ und $j \cdot 30° \leq$ `phi` $< (j+1) \cdot 30°$, d.h., daß es für 36*12 regelmäßig angeordneten Torusstücke je eine Liste gibt, die alle Orte in diesem Torusstück speichert.

Es ist weiters die folgende Funktion np2 gegeben, die in den übergebenen Ort-Listen (l1 und l2) das Nearest Pair findet, also n1 und n2 auf die Namen der (nähesten) Orte setzt und deren Entfernung als Ergebnis zurückliefert:

```
FUNCTION np2(l1,l2: TOrtPtr; VAR n1,n2: STRING): REAL;
BEGIN
  ...
END;
```

Bemerkung: np2 findet *immer* zwei *unterschiedliche* Orte (n1 $\neq$ n2) als Nearest Pair, auch wenn l1 gleich l2 ist. Wenn in l1 $\cup$ l2 weniger als zwei Orte sind, setzt np2 sowohl n1 als auch n2 auf `'NN'` und gibt ∞ zurück.

(11) Schreiben Sie eine *effiziente*

```
FUNCTION np(VAR n1,n2: STRING): REAL,
```

die in `alleOrte` das Nearest Pair findet. Die Namen der beiden Orte sollen in n1 bzw. n2 zurückgegeben werden und deren Distanz als Funktionswert (wie bei np2). Gehen Sie davon aus, daß es zumindest ein Paar (i, j) gibt, sodaß in `alleOrte[i,j]` mehrere Orte gespeichert sind.

Institut 186 für Computergraphik	Algorithmen und Datenstrukturen 2		Bsp.-Test 31. Januar 1997 Gruppe A	
Matrikelnummer	**Beilagen‡**	1 (18)	2 (21)	3 (11)
NACHNAME, Vorname				Σ (50)

‡ Geben Sie an, wieviele Zusatzbl. Sie abgeben (**jedes mit Name & Matr.-Nr. beschriftet!**).

Zur Erinnerung:
Alle Aufgaben beziehen sich auf die 10 Übungsbeispiele im WS '96/'97.

(18) Aufgabe 1 – zu den Beispielen 1 und 7

Generell gilt: Lösungen, aus denen nicht hervorgeht, wie sie erarbeitet wurden, werden nicht anerkannt. Schreiben Sie die Zwischenergebnisse auf.

(6) a) **zu Beispiel 1:** 1. Wenn nachfolgende Kontrolldatei gegeben ist, welche zusammenfassenden Informationen (`summary insert ...`, 3 Zeilen) sind vom beschriebenen Programm zu erwarten? 2. Zeichnen Sie außerdem beide Bäume, wie sie nach dem Abarbeiten der Kontrolldatei aussehen.

```
i: 9000000, 881, Neun
i: 8000000, 881, Acht
i: 7000000, 881, Sieben
i: 6000000, 881, Sechs
i: 4000000, 881, Vier
i: 3000000, 881, Drei
i: 5000000, 881, Fuenf
```

(21) Aufgabe 2 – zum Beispiel 5: B-Baum

Gegeben sei folgende Deklaration eines B-Baums:

```
TYPE
  BKnotenPtr = ^BKnoten;
  BBaum = OBJECT pWurz: BKnotenPtr;      {Zeiger auf Wurzel}
                 t: INTEGER;             {Tiefe des Baumes}
      FUNCTION tiefe: INTEGER;
       FUNCTION voll: BOOLEAN;
     PROCEDURE setze(p: BKnotenPtr);
  END;
  BKnoten = RECORD data: ARRAY [1..M] OF Anmeldung;
                   sub: ARRAY [0..M] OF BBaum;
                   anz: 1..M;
  END;

FUNCTION BBaum.tiefe: INTEGER; BEGIN tiefe:=t; END;
```

Bemerkungen: → `M` ist eine vorgegebene, *ungerade* Konstante. → `Anmeldung` ist ein `RECORD` mit Schlüssel `mat` (Typ `INTEGER`). → `b.voll` ergibt genau dann `TRUE`, wenn der Wurzelknoten von `b` voll ist. → `b.setze(p)` initialisiert den B-Baum `b` mit Wurzel `p`. → Gehen Sie davon aus, daß `pWurz` und `t` private Variable sind, verwenden Sie die Zugriffsmethoden (`voll`, usw.)! → Die Tiefe des leeren B-Baums ist 0;

(7) a) Schreiben Sie die

```
FUNCTION BBaum.split(VAR a: Anmeldung): BKnotenPtr,
```

die vom Wurzelknoten des B-Baums jene Hälfte abspaltet, welche die Anmeldungen mit den größeren Matrikelnummern enthält und in `a` den mittleren Datensatz retourniert. Die abgespaltene Hälfte der Wurzel soll als `BKnoten` mittels dem Funktionsergebnis zurückgegeben werden werden. Gehen Sie davon aus, daß die Wurzel des B-Baums voll ist.

(14) b) Schreiben Sie (unter Zuhilfenahme von `split`) die

```
PROCEDURE BBaum.eintrZW(a: Anmeldung),
```

welche eine Anmeldung `a` in den B-Baum einträgt. Gehen Sie (bei Aufruf `b.eintrZW(a)`) von folgenden Voraussetzungen aus: `b.voll=FALSE` und `b.tiefe>1`, d.h., Sie müssen nur das Eintragen in einen Zwischenknoten programmieren! **Wichtig:** Splitten Sie schon beim Suchen (wie im Skriptum beschrieben), wenn Sie auf einen vollen Teilbaum stoßen.

(12) **b) zu Beispiel 7:** 1. Erstellen Sie aus der folgenden Angebotsdatei die Kostenmatrix. 2. Geben Sie zudem auch noch die reduzierte Kostenmatrix an. Reduzieren Sie zuerst die Spalten, dann die Zeilen. Wenden Sie nur Subtraktionen an. 3. Um wieviel ist die billigste Rundreise bezüglich der reduzierten Kostenmatrix günstiger gegenüber der ursprünglichen Kostenmatrix?

```
Wien > Paris    = 82
Wien > Berlin   = 33
Wien > Rom      = 52
Paris > London  = 12
Paris > Rom     = 43
Paris > Berlin  = 66
Rom > Berlin    = 92
Rom > Wien      = 55
Rom > Paris     = 42
London > Berlin = 24
London > Paris  = 18
London > Wien   = 74
Berlin > London = 28
Berlin > Wien   = 36
Berlin > Rom    = 98
```

1.	W.	P.	R.	L.	B.
Wien >	—				
Paris >		—			
Rom >			—		
London >				—	
Berlin >					—

2.	W.	P.	R.	L.	B.
Wien >	—				
Paris >		—			
Rom >			—		
London >				—	
Berlin >					—

3.

(11) **Aufgabe 3 – zum Beispiel 9: Nearest Pair**

Auf einem Planeten in Torusform (Schwimmreifen) werden Orte in Listenform gespeichert. Es gibt 30*15 Listen (`allLocs`), in denen alle Orte auf dem Planeten verzeichnet sind:

```
TYPE
  TLocPtr = ^TLoc;                    {Zeiger => Liste}
  TLoc = RECORD id:       INTEGER;    {eindeutige ID}
                rho,phi:  REAL;       {Koordinaten (Winkel!)}
                next:     TLocPrt;    {naechster Ort}
  END;
VAR
  allLocs: ARRAY [0..29,0..14] OF TLocPtr;  {alle Orte}
```

Bemerkung: Für alle Orte in `allLocs[i,j]` gilt, daß $i \cdot 12° \leq$ `rho` $< (i+1) \cdot 12°$ und $j \cdot 24° \leq$ `phi` $< (j+1) \cdot 24°$, d.h., daß es für 30*15 regelmäßig angeordneten Torusstücke je eine Liste gibt, die alle Orte in diesem Torusstück speichert.

Es ist weiters die folgende Funktion `np1` gegeben, die in den übergebenen Ort-Listen (`l1` und `l2`) das Nearest Pair findet, also `id1` und `id2` auf die IDs der (nähesten) Orte setzt und deren Entfernung als Ergebnis zurückliefert:

```
FUNCTION np1(l1,l2: TLocPtr; VAR id1,id2: INTEGER): REAL;
BEGIN
  ...
END;
```

Bemerkung: `np1` findet *immer* zwei *unterschiedliche* Orte (`id1` $\neq$ `id2`) als Nearest Pair, auch wenn `l1` gleich `l2` ist. Wenn in `l1` $\cup$ `l2` weniger als zwei Orte sind, setzt `np1` sowohl `id1` als auch `id2` auf `-1` und gibt ∞ zurück.

(11) Schreiben Sie eine *effiziente*

```
FUNCTION np(VAR id1,id2: INTEGER): REAL,
```

die in `allLocs` das Nearest Pair findet. Die IDs der beiden Orte sollen in `id1` bzw. `id2` zurückgegeben werden und deren Distanz als Funktionswert (wie bei `np1`). Gehen Sie davon aus, daß es zumindest ein Paar (`i`, `j`) gibt, sodaß in `allLocs[i,j]` mehrere Orte gespeichert sind.

Institut 186 für Computergraphik	**Algorithmen und Datenstrukturen 2**			2. Übungstest 10. Jän. 1997 Gruppe A
Matrikelnr.	**Beil.**†	1 (18)	2 (16)	3 (16)
NACHNAME, Vorname				Σ (50)

† Geben Sie an, wieviele Zusatzbl. Sie abgeben (**jedes mit Name & Matr.-Nr. beschriftet!**).

Achtung: Geben Sie bei allen Berechnungen immer den vollen Berechnungsweg mit Begründung Ihrer Antwort an! Ein richtiges Resultat wird **nicht gewertet**, wenn aus Ihrer Abgabe nicht hervorgeht, wie das Ergebnis zustande gekommen ist.

(16) Aufgabe 1 – Monte-Carlo Methode

Das zweidimensionale Integral der Funktion

```
FUNCTION fun(x,y: REAL): REAL
```

soll mit Hilfe der Monte-Carlo Methode berechnet werden. Dazu steht Ihnen folgende Hilfsfunktion zur Verfügung:

```
FUNCTION zufallsZahl: REAL
```

die bei jedem Aufruf eine neue, gleichverteilte Zufallszahl im Intervall [0,1] zurückliefert.

a) Schreiben sie die Funktion

```
FUNCTION monteCarlo1(a,b,c,d: REAL; n:INTEGER): REAL
```

die das Integral $\int_a^b \int_c^d fun(x,y)dxdy$ mit Hilfe der 1. im Skriptum angegebenen Methode berechnet, wobei n zufällige Punkte zur Berechnung verwendet werden sollen. Nehmen Sie an, daß sich die Funktionswerte der Funktion `fun` immer positiv sind, und das die globale Variable `limit` den größten Funktionswert der Funktion im übergebenen Intervall enthält.

b) Schreiben sie die Funktion

```
FUNCTION monteCarlo2(a,b,c,d: REAL; n:INTEGER): REAL
```

die das Integral $\int_a^b \int_c^d fun(x,y)dxdy$ mit Hilfe der 2. im Skriptum angegebenen Methode berechnet, wobei n zufällige Punkte zur Berechnung verwendet werden sollen.

(16) Aufgabe 2 – Optimalbaum

Für die Datensätze D_1 bis D_5 soll ein Optimalbaum konstruiert werden. i sei o.B.d.A. der Schlüssel von Datensatz D_i und w_i dessen Zugriffshäufigkeit:

i	1	2	3	4	5
w_i	0.22	0.10	0.42	0.18	0.08

a) Berechnen Sie $w[i,j]$, $Q[i,j]$ und $r[i,j]$ für alle $i \leq j$. Tip: Beachten Sie dabei den Satz von Knuth, Sie sparen sich eine Menge Rechnerei. Sie brauchen den Optimalbaum der sich daraus ergibt nicht zu zeichnen!

j	1	2	3	4	5
$w[1,j]$	0.22				
$w[2,j]$	–	0.10			
$w[3,j]$	–	–	0.42		
$w[4,j]$	–	–	–	0.18	
$w[5,j]$	–	–	–	–	0.08

j	1	2	3	4	5
$Q[1,j]$	0.22				
$Q[2,j]$	–	0.10			
$Q[3,j]$	–	–	0.42		
$Q[4,j]$	–	–	–	0.18	
$Q[5,j]$	–	–	–	–	0.08

j	1	2	3	4	5
$r[1,j]$	1				
$r[2,j]$	–	2			
$r[3,j]$	–	–	3		
$r[4,j]$	–	–	–	4	
$r[5,j]$	–	–	–	–	5

b) Schreiben Sie in die folgende Tabelle an die Stelle $Q[i,j]$ die Liste all diejeniger r, für die Sie sich in *a)* durch den Satz von Knuth die Berechnung des jeweiligen $Q_{i,j}(r)$ erspart haben.

j	1	2	3	4	5
$Q[1,j]$					
$Q[2,j]$	–				
$Q[3,j]$	–	–			
$Q[4,j]$	–	–	–		
$Q[5,j]$	–	–	–	–	

c) Zeichnen Sie für die folgende $r[i,j]$ Tabelle eines Optimalbaums den zugehörigen Baum (dies ist ein anderer Baum als der aus *a)* und *b)*!):

j	1	2	3	4	5
$r[1,j]$	1	2	2	3	3
$r[2,j]$	–	2	2	3	5
$r[3,j]$	–	–	3	3	5
$r[4,j]$	–	–	–	4	5
$r[5,j]$	–	–	–	–	5

(16) **Aufgabe 3 – Genetische Algorithmen**
Eine Population sei aus Strings der Länge 5 über dem Alphabet V={a,c,g,t} aufgebaut.

a) Geben Sie all diejenigen Schemata der *definierenden* Länge zwei an, die nur das Symbol a enthalten. Wieviele Schemata mit der definierenden Länge zwei gibt es insgesamt?

b) Geben Sie all diejenigen Schemata der *Ordnung* drei an, die nur das Symbol a enthalten. Wieviele Schemata mit der Ordnung drei gibt es insgesamt?

c) Die beiden Schemata `**a*t` und `c**g*` werden mittels 1-Point cross-over gekreuzt. Geben Sie alle Schemata an die sich daraus ergeben. Geben Sie bei jedem der sich ergebenden Schemata an, welche der beiden ursprünglichen Schemata noch erhalten sind.

d) Wieviele verschiedene Möglichkeiten des 2-Point cross-over gibt es? Wie groß ist die Chance, daß dabei ein Schema der definierenden Länge eins zerstört wird. Nehmen Sie an, daß die Grenzen des cross-over Intervalls nicht an den Stringenden liegen dürfen.

Institut 186 für Computergraphik	Algorithmen und Datenstrukturen 2			2. Übungstest 10. Jän. 1997 Gruppe B
Matrikelnr.	**Beil.†**	1 (18)	2 (16)	3 (16)
NACHNAME, Vorname				Σ (50)

† Geben Sie an, wieviele Zusatzbl. Sie abgeben (**jedes mit Name & Matr.-Nr. beschriftet!**).

Achtung: Geben Sie bei allen Berechnungen immer den vollen Berechnungsweg mit Begründung Ihrer Antwort an! Ein richtiges Resultat wird **nicht gewertet**, wenn aus Ihrer Abgabe nicht hervorgeht, wie das Ergebnis zustande gekommen ist.

(18) **Aufgabe 1 – Monte-Carlo Methode**
Das zweidimensionale Integral der Funktion

```
FUNCTION fun(u,v: REAL): REAL
```

soll mit Hilfe der Monte-Carlo Methode berechnet werden. Dazu steht Ihnen folgende Hilfsfunktion zur Verfügung:

```
FUNCTION randomNumber: REAL
```

die bei jedem Aufruf eine neue, gleichverteilte Zufallszahl im Intervall [0,1] zurückliefert.

a) Schreiben sie die Funktion

```
FUNCTION monteCarlo1(e,f,g,h: REAL; n:INTEGER): REAL
```

die das Integral $\int_e^f \int_g^h fun(u,v)dudv$ mit Hilfe der 1. im Skriptum angegebenen Methode berechnet, wobei n zufällige Punkte zur Berechnung verwendet werden sollen. Nehmen Sie an, daß die Funktionswerte der Funktion `fun` immer positiv sind, und das die globale Variable `limit` den größten Funktionswert der Funktion im übergebenen Intervall enthält.

b) Schreiben sie die Funktion

```
FUNCTION monteCarlo2(e,f,g,h: REAL; n:INTEGER): REAL
```

die das Integral $\int_e^f \int_g^h fun(u,v)dudv$ mit Hilfe der 2. im Skriptum angegebenen Methode berechnet, wobei n zufällige Punkte zur Berechnung verwendet werden sollen.

(16) Aufgabe 2 – Optimalbaum

Für die Datensätze D_1 bis D_5 soll ein Optimalbaum konstruiert werden. i sei o.B.d.A. der Schlüssel von Datensatz D_i und w_i dessen Zugriffshäufigkeit:

i	1	2	3	4	5
w_i	0.07	0.19	0.42	0.09	0.23

a) Berechnen Sie $w[i,j]$, $Q[i,j]$ und $r[i,j]$ für alle $i \leq j$. Tip: Beachten Sie dabei den Satz von Knuth, Sie sparen sich eine Menge Rechnerei. Sie brauchen den Optimalbaum der sich daraus ergibt nicht zu zeichnen!

j	1	2	3	4	5
$w[1,j]$	0.07				
$w[2,j]$	-	0.19			
$w[3,j]$	-	-	0.42		
$w[4,j]$	-	-	-	0.09	
$w[5,j]$	-	-	-	-	0.23

j	1	2	3	4	5
$Q[1,j]$	0.07				
$Q[2,j]$	-	0.19			
$Q[3,j]$	-	-	0.42		
$Q[4,j]$	-	-	-	0.09	
$Q[5,j]$	-	-	-	-	0.23

j	1	2	3	4	5
$r[1,j]$	1				
$r[2,j]$	-	2			
$r[3,j]$	-	-	3		
$r[4,j]$	-	-	-	4	
$r[5,j]$	-	-	-	-	5

b) Schreiben Sie in die folgende Tabelle an die Stelle $Q[i,j]$ die Liste all diejeniger r, für die Sie sich in *a)* durch den Satz von Knuth die Berechnung des jeweiligen $Q_{i,j}(r)$ erspart haben.

j	1	2	3	4	5
$Q[1,j]$					
$Q[2,j]$					
$Q[3,j]$	-				
$Q[4,j]$	-	-			
$Q[5,j]$	-	-	-		

c) Zeichnen Sie für die folgende $r[i,j]$ Tabelle eines Optimalbaums den zugehörigen Baum (dies ist ein anderer Baum als der aus *a)* und *b)*!).

j	1	2	3	4	5
$r[1,j]$	1	1	2	3	3
$r[2,j]$	-	2	2	3	4
$r[3,j]$	-	-	3	3	4
$r[4,j]$	-	-	-	4	4
$r[5,j]$	-	-	-	-	5

(16) Aufgabe 3 – Genetische Algorithmen

Eine Population sei aus Strings der Länge 5 über dem Alphabet V={a,c,g,t} aufgebaut.

a) Geben Sie all diejenigen Schemata der *definierenden Länge* drei an, die nur das Symbol g enthalten. Wieviele Schemata mit der definierenden Länge drei gibt es insgesamt?

b) Geben Sie all diejenigen Schemata der *Ordnung* zwei an, die nur das Symbol g enthalten. Wieviele Schemata mit der Ordnung zwei gibt es insgesamt?

c) Die beiden Schemata `a*c**` und `*g**t` werden mittels 1-Point cross-over gekreuzt. Geben Sie alle Schemata an die sich daraus ergeben. Geben Sie bei jedem der sich ergebenden Schemata an, welche der beiden ursprünglichen Schemata noch erhalten sind.

d) Wieviele verschiedene Möglichkeiten des 2-Point cross-over gibt es? Wie groß ist die Chance, daß dabei ein Schema der definierenden Länge eins zerstört wird. Nehmen Sie an, daß die Grenzen des cross-over Intervalls nicht an den Stringenden liegen dürfen.

Institut 186 für Computergraphik	**Algorithmen und Datenstrukturen 2**			1. Übungstest 21. Nov. 1996 Gruppe A	
Kennz.†	**Matrikelnr.**	**Beil.**‡	1 (16)	2 (16)	3 (18)
NACHNAME, Vorname					Σ (50)

† Geben Sie jene Kennzahl an, auf die das Zeugnis ausgestellt werden soll.
‡ Geben Sie an, wieviele Zusatzbl. Sie abgeben (**jedes mit Name & Matr.-Nr. beschriftet!**).

(16) Aufgabe 1 – Hashtabelle

In eine Hashtabelle mit $M = 8192$ Plätzen wurden $N = 2048$ Datensätze eingetragen. Über die Hashfunktion und die Kollisionsbehandlung ist nichts bekannt. Sie können aber annehmen, daß es sich um eine "gute" Hashfunktion handelt, und auch die Kollisionsbehandlung ordentlich implementiert wurde.

a) Wieviele Suchschritte sind im Mittel erforderlich, wenn ein *nicht* vorhandener DS gesucht wird? (D.h. wieviele Schritte braucht man im Mittel, um den Mißerfolg festzustellen?

b) In die Tabelle werden weitere 2048 DS eingetragen. Wieviele Suchschritte sind jetzt im Mittel für das Suchen eines nicht vorhandenen DS erforderlich?

c) Wie groß müßte die Hashtabelle sein, damit bei weiteren 2048 eingetragenen DS (also insgesamt 4096) das Suchen eines nicht vorhandenen DS im Mittel nicht mehr Schritte erfordert als in *a)*?

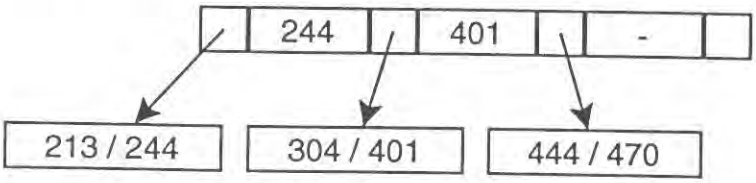
d) Nehmen Sie an, die Schlüssel der DS seien gewöhnliche `integer`-Zahlen. Geben Sie eine geeignete Hashfunktion zur *Kollisionsbehandlung* an. Welche Besonderheiten muß diese Funktion aufweisen? (Beachten Sie, daß M keine Primzahl ist.)

Achtung: Geben Sie immer den vollen Berechnungsweg mit allen verwendeten Formeln und einer Begründung Ihrer Antwort an! Ein richtiges Resultat wird **nicht gewertet**, wenn aus Ihrer Abgabe nicht hervorgeht, wie das Ergebnis zustande gekommen ist.

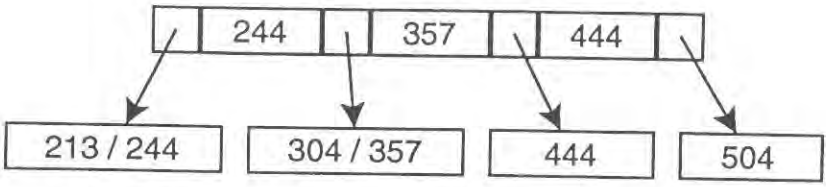
(16) Aufgabe 2 – Top-Down B*-Baum

Die folgenden Beispiele beziehen sich auf einen B*-Baum mit $M_S = 4$ und $M_D = 2$. Die Schlüssel in den Zwischenknoten entsprechen jeweils den größten Schlüsseln des linken Unterbaums.

a) Tragen Sie in den folgenden B*-Baum nacheinander die DS mit den Schlüsseln 320 und 364 ein! Zeichenen Sie den B*-Baum nach jeder der beiden Operationen. *WICHTIG:* Verwenden Sie beim Eintragen den Top-Down-Algorithmus!



b) Löschen Sie aus dem folgenden B*-Baum nacheinander die DS mit den Schlüsseln 444 und 357! Zeichenen Sie den B*-Baum nach jeder der beiden Operationen.

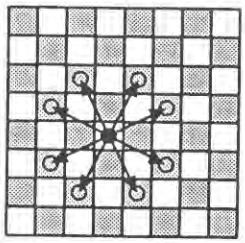


c) Ein wie oben definierter B*-Baum ($M_S = 4$ und $M_D = 2$) enthalte 512 DS. Wieviele Knoten (Zwischenknoten und Blattknoten) müssen mindestens, wieviele maximal besucht werden um einen DS mit gegebenem Schlüssel zu finden?

d) Wieviele Zwischen- bzw. Blattknoten hat ein wie oben definierter B*-Baum ($M_S = 4$ und $M_D = 2$) mindestens, wieviele maximal wenn er 512 DS enthaelt?

(18) Aufgabe 3 – Springerwege

Ein Springer auf einem Schachbrett hat die folgenden Zugmöglichkeiten:



Die folgenden Datenstrukturen stehen Ihnen zur Verfügung:

```
TYPE Feld =   OBJECT
                linie: 0..8;    (* 0 markiert ein    *)
                reihe: 0..8;    (* ungueltiges Feld *)
                FUNCTION gueltig: BOOLEAN;
                PROCEDURE drucke;
              END
     Brett =  ARRAY[1..8][1..8] OF BOOLEAN;
```

Schreiben Sie eine *effiziente, rekursive* Prozedur

```
PROCEDURE SpringerWege(start, ziel:  Feld)
```

die alle möglichen Wege des Springers von `start` nach `ziel` ausgibt. Beachten Sie dabei, daß der Springer bei einem Weg von `start` nach `ziel` kein Feld zweimal betreten darf. Deklarieren Sie außerdem alle notwendigen globalen Variablen, und geben Sie an, mit welchen Werten sie initialisiert werden müssen. Um die Aufgabe zu vereinfachen, stehen Ihnen folgende Prozeduren zur Verfügung:

- `FUNCTION springerZug(von:  Feld; num:  INTEGER): Feld;`
  Diese Funktion liefert ihnen für jeden Parameter `num` von 1 bis 8 einen der acht gültigen Springerzüge (fertig konstruiert) ausgehenden vom Feld `von`. Da der Springer nicht über den Brettrand springen darf, kann es sein, daß das zurückgelieferte Feld ungültig ist.
- Die Methode `FUNKTION gueltig :  BOOLEAN` gibt an, ob das jeweilige Feld auch wirklich gültig ist.
- Zur Ausgabe eines Feldes können Sie noch die Methode `PROCEDURE drucke;` verwenden.

Institut 186 für Computergraphik	**Algorithmen und Datenstrukturen 2**			1. Übungstest 21. Nov. 1996 Gruppe B	
Kennz.†	**Matrikelnr.**	**Beil.**‡	1 (16)	2 (16)	3 (18)
NACHNAME, **Vorname**					Σ (50)

† Geben Sie jene Kennzahl an, auf die das Zeugnis ausgestellt werden soll.
‡ Geben Sie an, wieviele Zusatzbl. Sie abgeben (**jedes mit Name & Matr.-Nr. beschriftet!**).

(16) Aufgabe 1 – Hashtabelle

In eine Hashtabelle mit $M = 16384$ Plätzen wurden $N = 2048$ Datensätze eingetragen. Über die Hashfunktion und die Kollisionsbehandlung ist nichts bekannt. Sie können aber annehmen, daß es sich um eine "gute" Hashfunktion handelt, und auch die Kollisionsbehandlung ordentlich implementiert wurde.

a) Wieviele Suchschritte sind im Mittel erforderlich, wenn ein *nicht* vorhandener DS gesucht wird? (D.h. wieviele Schritte braucht man im Mittel, um den Mißerfolg festzustellen?

b) In die Tabelle werden weitere 2048 DS eingetragen. Wieviele Suchschritte sind jetzt im Mittel für das Suchen eines nicht vorhandenen DS erforderlich?

c) Wie groß müßte die Hashtabelle sein, damit bei weiteren 2048 eingetragenen DS (also insgesamt 4096) das Suchen eines nicht vorhandenen DS im Mittel nicht mehr Schritte erfordert als in *a)*?

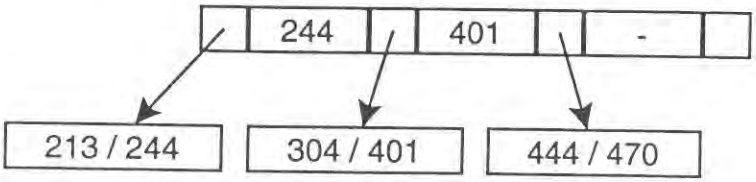
d) Nehmen Sie an, die Schlüssel der DS seien gewöhnliche `integer`-Zahlen. Geben Sie eine geeignete Hashfunktion zur *Kollisionsbehandlung* an. Welche Besonderheiten muß diese Funktion aufweisen? (Beachten Sie, daß M keine Primzahl ist.)

Achtung: Geben Sie immer den vollen Berechnungsweg mit allen verwendeten Formeln und einer Begründung Ihrer Antwort an! Ein richtiges Resultat wird **nicht gewertet**, wenn aus Ihrer Abgabe nicht hervorgeht, wie das Ergebnis zustande gekommen ist.

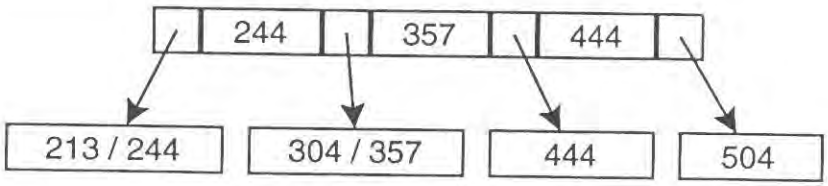
(16) Aufgabe 2 – Top-Down B*-Baum

Die folgenden Beispiele beziehen sich auf einen B*-Baum mit $M_S = 4$ und $M_D = 2$. Die Schlüssel in den Zwischenknoten entsprechen jeweils den größten Schlüsseln des linken Unterbaums.

a) Tragen Sie in den folgenden B*-Baum nacheinander die DS mit den Schlüsseln 220 und 235 ein! Zeichenen Sie den B*-Baum nach jeder der beiden Operationen. *WICHTIG:* Verwenden Sie beim Eintragen den Top-Down-Algorithmus!



b) Löschen Sie aus dem folgenden B*-Baum nacheinander die DS mit den Schlüsseln 244 und 213! Zeichenen Sie den B*-Baum nach jeder der beiden Operationen.



c) Ein wie oben definierter B*-Baum ($M_S = 4$ und $M_D = 2$) enthalte 2048 DS. Wieviele Knoten (Zwischenknoten und Blattknoten) müssen mindestens, wieviele maximal besucht werden um einen DS mit gegebenem Schlüssel zu finden?

d) Wieviele Zwischen- bzw. Blattknoten hat ein wie oben definierter B*-Baum ($M_S = 4$ und $M_D = 2$) mindestens, wieviele maximal wenn er 2048 DS enthaelt?