berlin

Fakultät IV

Klausur MPGI 1
19.02.2008

Prof. Dr. Glesner
Dipl.-Ing. Beyer, Dipl.-Inf. Göthel, Dipl.-Sys. Hahne

Name: .....

Vorname: .....

Matr.-Nr.: .....

Bearbeitungszeit: 120 Minuten

- Als Hilfsmittel ist nur ein doppelseitig und **in eigener Handschrift** beschriebenes DIN-A4-Blatt zugelassen.
- Benutzen Sie für die Lösung der Aufgaben nur das mit diesem Deckblatt ausgeteilte Papier. **Lösungen, die auf anderem Papier geschrieben werden, können nicht gewertet werden!**
- Schreiben Sie Ihre Lösungen auf das Aufgabenblatt der jeweiligen Aufgabe. Verwenden Sie auch die Rückseiten. **Schreiben Sie keine Lösungen einer Aufgabe auf ein Blatt, das nicht zu dieser Aufgabe gehört!** Wenn Sie zusätzliche, von uns ausgegebene Blätter verwenden, geben Sie unbedingt an, zu welcher Aufgabe die Lösung gehört!
- Schreiben Sie deutlich! Doppelte, unleserliche oder mehrdeutige Lösungen werden nicht gewertet! Streichen Sie gegebenenfalls eine Lösung durch!
- Schreiben Sie nur in **blau** oder **schwarz**. Lösungen, die mit Bleistift geschrieben sind, werden nicht gewertet!
- Erscheint Ihnen eine Aufgabe mehrdeutig, wenden Sie sich an die Betreuer.
- Sollten Sie im Softwareteil eine Teilaufgabe nicht lösen können, so dürfen Sie die dort geforderte Funktion in anderen Teilaufgaben verwenden.
- Tragen Sie zu Beginn der Bearbeitungszeit auf *allen* Blättern Ihren Namen und Ihre Matrikelnummer ein. **Blätter ohne Namen werden nicht gewertet.**
- Für alle Multiple-Choice Aufgaben gilt, dass für jede richtige Antwort 0.5 Punkte gegeben, für jede falsche 0.5 Punkte abgezogen werden. In jedem Fall erhält man in der jeweiligen Teilaufgabe nicht weniger als 0 Punkte.

Aufgabe	Punkte	erreicht
1	14	
2	8	
3	9	
4	5	
5	9	
6	10	
7	5	

Hinweise zur Bearbeitung der OPAL-Aufgaben

Bei der Bearbeitung der OPAL-Aufgaben kann auf `IMPORT`-, `SIGNATURE`- und `IMPLEMENTATION`-Deklarationen verzichtet werden. Es können alle Funktionen aus der OPAL-Standardbibliothek verwendet werden. Eine **nicht vollständige** Liste der Strukturen und Funktionen aus der Bibliothek ist im Folgenden angegeben.

```
SIGNATURE Bool
-- -----------
FUN true false :                 bool
FUN not        : bool         -> bool
FUN and or     : bool ** bool -> bool
FUN =          : bool ** bool -> bool

SIGNATURE Denotation
-- -----------------
FUN ++      : denotation ** denotation -> denotation
FUN #       : denotation -> nat
FUN < > =   : denotation ** denotation -> bool

SIGNATURE Option
-- -------------
SORT data
TYPE option == nil
               avail( cont: data )

SIGNATURE Seq[data]
-- ----------------
SORT data
TYPE seq == <>
            ::(ft : data, rt : seq)
FUN #       : seq -> nat
FUN ++      : seq ** seq -> seq
FUN =       : (data ** data -> bool) -> seq ** seq -> bool
FUN ft last : seq -> data
FUN split   : nat ** seq -> seq ** seq
FUN +%      : seq ** data -> seq
FUN ..      : nat ** nat -> (nat -> data) -> seq

SIGNATURE SeqMap[from,to]
-- ----------------------
SORT from to
FUN map : (from -> to) -> seq[from] -> seq[to]

SIGNATURE SeqFilter[data]
-- ----------------------
SORT data
FUN filter : (data -> bool) -> seq[data] -> seq[data]

SIGNATURE SeqReduce[from,to]
-- -------------------------
SORT from to
FUN reduce : (from ** to -> to) ** to -> seq[from] -> to

SIGNATURE SeqZip[from1,from2,to]
-- -----------------------------
SORT from1 from2 to
FUN zip : (from1 ** from2 -> to) -> seq[from1] ** seq[from2] -> seq[to]
```

1. Aufgabe (14 Punkte): Soziale Netzwerke

Soziale Netzwerke wie studiVZ und facebook werden immer populärer. Dort können Benutzer Informationen über sich selbst ablegen sowie eine Liste ihrer Freunde oder Bekannten führen.

1.1. Induzierte Signatur (2 Punkte) Im Folgenden ist eine Datenstruktur für einen Benutzer eines solchen Netzwerks angegeben. Jeder Benutzer erhält eine eindeutige Identifikationsnummer (`id`), einen Benutzernamen, ein Alter sowie eine Freundesliste (Liste von Identifikationsnummern). Geben Sie für diesen Datentypen die **induzierte Signatur** an und benennen Sie die einzelnen Teile.

```
TYPE Benutzer == einfacherBenutzer(id:nat,name: denotation,alter:nat,freunde:seq[nat])
```

1.2. Datentyp (1 Punkt) Geben Sie an, um welche Art von Datentyp es sich handelt. Begründen Sie Ihre Antwort in einem Satz.

1.3. Erweiterung des Datentyps Benutzer (2 Punkte) Erweitern Sie den Datentyp `Benutzer` um einen besonderen Benutzer `premiumBenutzer`. Dieser **kann** zusätzlich eine Lieblingsfarbe angeben, braucht es aber nicht. Deklarieren Sie dazu auch einen Aufzählungstypen `Farbe`, den Sie für die Lieblingsfarbe verwenden. Dieser Datentyp soll mindestens die Farben `rot`, `grün` und `blau` enthalten.
Welche Art von Datentyp ist Benutzer nun?

1.4. Hinzufügen von Freunden (2 Punkte) Deklarieren und definieren Sie eine Funktion `neuerFreund`, die die Freundesliste eines Benutzers um einen weiteren Freund (das heißt um dessen „id“) ergänzt. Stellen Sie sicher, dass man sich nicht als Freund von sich selbst eintragen kann und dass keine Freunde doppelt eingetragen werden. Für die Prüfung auf Vorhandensein des Freundes in der Freundesliste ist die Hilfsfunktion `exists?` angegeben. Geben Sie außerdem die Rekursionsart der Funktion `exists?` an.

```
FUN exists? : nat ** seq[nat] -> bool
DEF exists?(_,<>) == false
DEF exists?(f,l::L) == IF f = l THEN true ELSE exists?(f,L) FI
```

1.5. Existenz von Freunden (1 Punkt) Definieren Sie die Hilfsfunktion `exists?` neu, diesmal unter Verwendung der **Listenfunktionale** `map`, `reduce`, `filter` und `zip`. Es müssen nicht alle der angegebenen Funktionale verwendet werden.

1.6. Suchen eines Freundes (2 Punkte) Definieren Sie eine Funktion `findeFreund`, die eine Liste von Benutzern und die „id“ des gesuchten Freundes übergeben bekommt und dann den entsprechenden Benutzer zurückgibt. Damit die Funktion auch für den Fall, dass es den gesuchten Benutzer nicht gibt, einen Wert zurückgibt, soll die Struktur `option` aus der Bibliotheca Opalica verwendet werden. Als Hilfestellung ist die Deklaration vorgegeben:

```
FUN findeFreund : nat ** seq[Benutzer] -> option[Benutzer]
```

1.7. Ausgabe der Freunde (2 Punkte) Deklarieren und definieren Sie eine Funktion `zeigeFreunde`, die die Namen aller Freunde eines gegebenen Benutzers ermittelt und diese als `denotation` zeilenweise zurückgibt. Die Funktion erhält dazu einen Benutzer sowie eine Liste aller Benutzer des Netzwerks.

Hinweis: Sie können die Funktion `findeFreund` verwenden. Weiterhin dient die Zeichenkette `"\n"` als Zeilenumbruch.

1.8. Sortieren der Freundesliste (2 Punkte) Geben Sie die Deklarationen der gegebenen Funktionen `sortiereFreunde, sort` und `help` an. Diese sortieren die Freundesliste eines „einfachen“ Benutzers. Um welche Rekursionsart handelt es sich bei den Funktionen `sort` und `help`?

```
DEF sortiereFreunde(einfacherBenutzer(_,_,_,F)) == einfacherBenutzer(_,_,_,sort(F))
```

```
DEF sort(<>) == <>
DEF sort(a::A) == help(a,sort(A))
```

```
DEF help(x,<>) == x :: <>
DEF help(x,a::A) == IF x <= a THEN x::(a::A)
                              ELSE a::(help(x,A))
                    FI
```

2. Aufgabe (8 Punkte): Vektoren und Matrizen

In dieser Aufgabe sollen Funktionen auf Vektoren und Matrizen implementiert werden.

Die Vektoren werden als Sequenzen von ganzen Zahlen implementiert. Eine $n \times m$ Matrix setzt sich aus n Zeilenvektoren mit je m Elementen zusammen.

Beispiel:

```
FUN mat : seq[seq[int]]
DEF mat == (3 :: -(2) :: 1 :: 3 :: <>) :: (9 :: 4 :: -(6) :: 2 :: <>) ::
           (-(5) :: 8 :: 3 :: -(1) :: <>) :: <>
```

Die Konstante `mat` entspricht folgender Matrix:

	1	2	3	4
1	3	-2	1	3
2	9	4	-6	2
3	-5	8	3	-1

2.1. N-tes Element einer Sequenz (1 Punkt) Mit Hilfe der nachfolgenden Funktion `getElem`, kann auf das n-te Element einer Sequenz zugegriffen werden. In dem gegebenen Programmauszug ist `alpha` eine Typvariable.

```
FUN getElem :  seq[alpha] ** nat -> alpha
DEF getElem(l::L, succ(0)) == l
DEF getElem(l::L, n) == getElem(L, n-1)
```

Welches Ergebnis liefert der folgende Oasys-Aufruf, wenn `alpha` mit `nat` instantiiert wird?

```
> e getElem(5 :: 3 :: 2 :: <>, succ(succ(0)))
```

Um welche Rekursionsart handelt es sich?

2.2. Zeile einer Matrix (1 Punkt) Deklarieren und definieren Sie eine Funktion `getRow`, die eine Matrix und eine Zeilennummer übergeben bekommt und die entsprechende Zeile als Vektor zurückliefert. Gehen Sie davon aus, dass die Zeilennummer gültig ist, das heißt zwischen 1 und n liegt.

Sie können zur Lösung der Aufgabe die oben gegebene Funktion `getElem` verwenden.

Beispiel:

```
> e getRow(mat, 2)
<9,4,-6,2>
```

2.3. Spalte einer Matrix (2 Punkte) Deklarieren und definieren Sie eine Funktion `getCol`, die eine Matrix und eine Spaltennummer übergeben bekommt und die entsprechende Spalte als Vektor zurückliefert. Gehen Sie davon aus, dass die Spaltennummer gültig ist, das heißt zwischen 1 und m liegt.

Sie können zur Lösung der Aufgabe die oben gegebene Funktion `getElem` verwenden. Schreiben Sie **rekursive Funktionen**. Benutzen Sie **keine** Listenfunktionale.

Beispiel:

```
>e getCol(mat, 2)
<-2,4,8>
```

2.4. Summe aller Elemente einer Matrix (2 Punkte) Deklarieren und definieren Sie eine Funktion `getSum`, die eine Matrix übergeben bekommt und die Summe aller Elemente zurückliefert.

Benutzen Sie **Listenfunktionale**. Schreiben Sie **keine** rekursiven Funktionen.

Beispiel:

```
>e getSum(mat)
19
```

2.5. Typinferenz - Determinante einer Matrix (2 Punkte) Gegeben sei der folgende OPAL-Code zur Berechnung der Determinante einer Matrix:

```
FUN determ : seq[seq[int]] -> int
DEF determ((m :: <>) :: <>) == m
DEF determ(m) == reduce(+,0)((1..(#(m)))(raetsel(m, 1, <>)))



DEF raetsel(<>  , a, A) (_) == a * determ(A)
DEF raetsel(m::M, a, A) (succ(0)) == raetsel(M, a * ft(m), A) (0)
DEF raetsel(m::M, a, A) (0) == raetsel(M, a, A +% rt(m)) (0)
DEF raetsel(m::M, a, A) (r) == raetsel(M, (-(1)) * a, A +% rt(m)) (r-1)
```

Geben Sie die fehlende Deklaration der Funktion `raetsel` an.

3. Aufgabe (9 Punkte): Lambda-Kalkül

3.1. Induktiver Aufbau (2 Punkte) Es sei eine Variablenmenge V gegeben. Dann ist die Menge der Lambda-Terme Λ wie folgt induktiv definiert:

1. $x \in V \Longrightarrow x \in \Lambda$
2. $x \in V\ ,\ E \in \Lambda \Longrightarrow (\lambda x.\ E) \in \Lambda$
3. $E_1\ ,\ E_2 \in \Lambda \Longrightarrow (E_1\ E_2) \in \Lambda$

Es gelte $\{x, y, z\} \subseteq V$. Welche der folgenden Terme sind korrekt nach obigen Regeln gebildet und welche nicht? Geben Sie jeweils eine kurze Begründung an.

a) $(\lambda x.(\lambda y.(y\ x)))$
b) $((x\ y)(y\ z))(\lambda x.x)$
c) $(\lambda(\lambda x.(x\ x)).z)$
d) $(x\ (\lambda x.x))$

3.2. Substitution (2 Punkte) Substituieren Sie in dem Term $((\lambda x.x\ y)\ (\lambda y.y))$ schrittweise die Variable y durch $(x\ y)$. Führen Sie, wenn nötig, α-Konversionen durch.

$((\lambda x.x\ y)(\lambda y.y))[(x\ y)/y] = \ldots$

3.3. Tupel im Lambda-Kalkül (2 Punkte) Tupel von Lambda-Termen $[A, B]$ lassen sich im Lambda-Kalkül wie folgt darstellen:

$$[A, B] \equiv \lambda x.((x\ A)\ B)$$

Weiterhin seien die Church-Booleans gegeben mit:

$$\mathit{true} \equiv \lambda x.\lambda y.x$$
$$\mathit{false} \equiv \lambda x.\lambda y.y$$

Zeigen Sie, dass *true* als Selektor der ersten Komponente von Tupeln dient, zeigen Sie also

$$[A, B]\ \mathit{true} \equiv_\beta A$$

für beliebige λ-Terme A und B.

3.4. Allgemeine Fragen (3 Punkte) Beantworten Sie die folgenden Fragen stichpunktartig.

a) Erklären Sie kurz die Begriffe α-Konversion, β-Reduktion und η-Konversion (höchstens 3 Sätze!)

b) Welche Auswertungsstrategien von λ-Termen sind Ihnen aus der Vorlesung bekannt und nach welcher wertet Opal am ehesten aus?

c) Was ist eine Funktion höherer Ordnung?

Welche der folgenden Aussagen sind richtig?

Ja	**Nein**	
◯	◯	Im λ-Kalkül ist es möglich, natürliche Zahlen darzustellen.
◯	◯	Jeder λ-Term E kann zu einem Term E' reduziert werden ($E \rightarrow_\beta^* E'$), so dass E' nicht weiter reduziert werden kann.
◯	◯	Eine λ-Abstraktion kann immer als eine Funktion höherer Ordnung aufgefasst werden.

4. Aufgabe (5 Punkte): Aufwand

4.1. $\mathcal{O}$-Kalkül (2 Punkte) Es seien Funktionen f,g : $\mathbb{N} \Rightarrow \mathbb{R}^+$ gegeben mit $f(n) = 14n^2 - n$ und $g(n) = 8n^2 + 4n$. Welche der folgenden Aussagen gelten: $f \in \mathcal{O}(g)$, $f \in \Theta(g)$, $f \in \Omega(g)$? Beweisen Sie Ihre Antworten.

Hinweis: Es kann hilfreich sein, eine entsprechende Grenzwertbetrachtung durchzuführen.

4.2. Rekurrenzgleichungen (3 Punkte) Bestimmen Sie den Aufwandsterm der Opal-Funktion `f` in Abhängigkeit der Länge der übergebenen Liste und leiten Sie daraus die Aufwandsklasse ab. Zur korrekten Lösung gehört die Angabe der verwendeten Zeile in der Tabelle sowie die Berechnung der entsprechenden Wertebelegung und die Angabe der daraus resultierenden Aufwandsklasse.

	Rekurrenzrelation		$A \in$
1	$A(n) = A(n-1) + bn^k$		$\mathcal{O}(n^{k+1})$
2	$A(n) = cA(n-1) + bn^k$	mit $c > 1$	$\mathcal{O}(c^n)$
3	$A(n) = cA(n/d) + bn^k$	mit $c > d^k$	$\mathcal{O}(n^{\log_d c})$
4	$A(n) = cA(n/d) + bn^k$	mit $c < d^k$	$\mathcal{O}(n^k)$
5	$A(n) = cA(n/d) + bn^k$	mit $c = d^k$	$\mathcal{O}(n^k \log_d n)$

```
FUN f: seq[nat] -> seq[nat]
DEF f(<>) == <>
DEF f(a::A) ==
        LET D == square(A)
        IN a :: f(D)

FUN square: seq[nat] -> seq[nat]
DEF square(<>) == <>
DEF square(a::A) == (a*a) :: square(A)
```

5. Aufgabe (9 Punkte): Bäume

5.1. Allgemeine Fragen (3 Punkte) Welche der folgenden Aussagen sind richtig?

Ja	Nein	
○	○	In jedem Binärbaum gilt für alle Knoten, dass die Höhe des linken und des rechten Teilbaums sich höchstens um 1 unterscheiden.
○	○	In jedem Suchbaum ist der Wert in einem Knoten mindestens so groß wie die Werte in allen seinen Nachfolgern.
○	○	Ein Blatt in einem Baum ist ein Knoten, der keine Nachfolger hat.
○	○	Die Höhe eines Baums ist gleich der Länge eines Wegs von der Wurzel bis zu einem beliebigen Blatt.
○	○	Es sei T ein Suchbaum. Besitzen der rechte und linke Teilbaum der Wurzel die gleiche Höhe, so ist T stets ein AVL-Baum.
○	○	Das sogenannte n-Damen Problem kann mit Hilfe einer Backtracking-Suche gelöst werden.

5.2. AVL-Bäume (3 Punkte) Welche der gegebenen Bäume sind AVL-Bäume? Begründen sie jeweils kurz Ihre Entscheidung.

Antwort für Baum A:

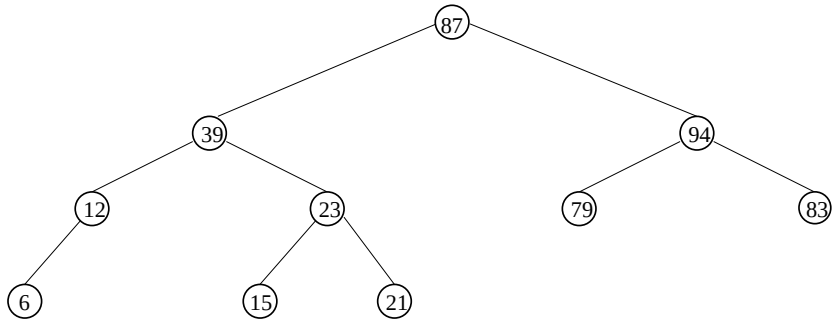


Abbildung 1: Baum A

Antwort für Baum B:

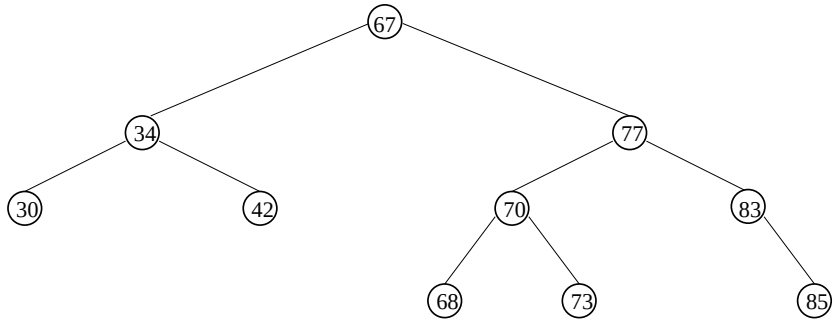


Abbildung 2: Baum B

Antwort für Baum C:

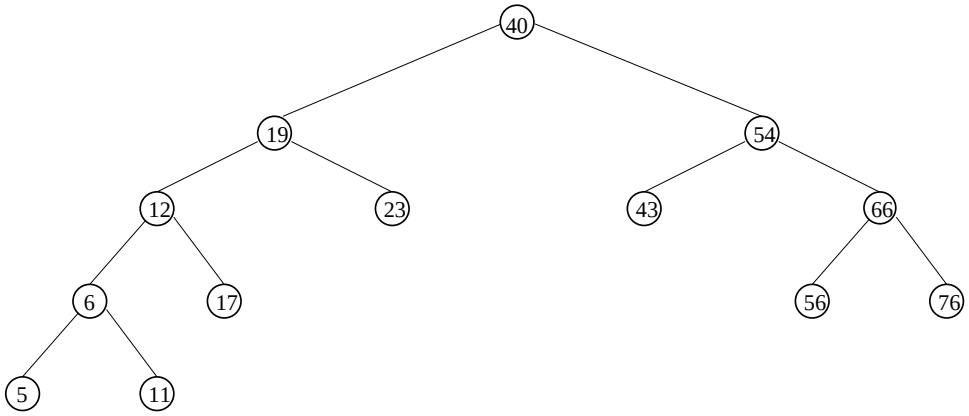
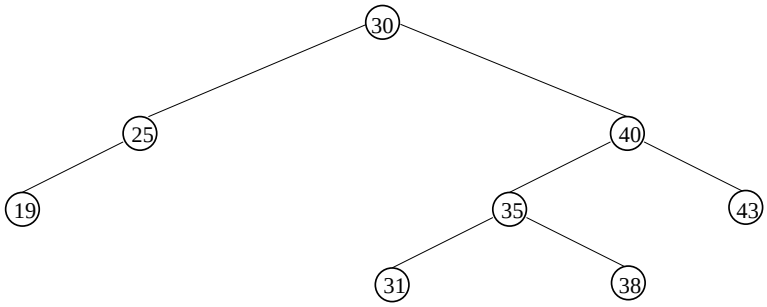


Abbildung 3: Baum C

5.3. Löschen in AVL-Bäumen (3 Punkte) Löschen Sie den Koten 25 aus dem gegebenen AVL-Baum und führen Sie anschließend die entsprechenden Rotationen durch, um die AVL-Eigenschaft wieder herzustellen. Erläutern Sie Ihre Schritte.



6. Aufgabe (10 Punkte): Graphen

6.1. Allgemeine Fragen (3 Punkte) Welche der folgenden Aussagen sind richtig?

Ja	Nein	
◯	◯	Sei G ein Graph. Dann ist G der einzige aufspannende Untergraph von G.
◯	◯	Jeder Teilgraph ist ein Untergraph.
◯	◯	Ein zusammenhängender gewichteter Graph kann mehrere minimale Spannbäume besitzen.
◯	◯	Kruskals Algorithmus bestimmt den kürzesten Weg zwischen zwei Knoten s und t eines Graphen.
◯	◯	Dijkstras Algorithmus berechnet kürzeste Wege ausgehend von einem Startknoten.
◯	◯	Der Algorithmus von Ford-Fulkerson bestimmt den minimalen Fluss in einem Netzwerk.

6.2. Darstellung von Graphen (2 Punkte) In Abbildung 4 ist die grafische Repräsentation eines **gerichteten** Graphen $G = (V, E)$ zu sehen. Geben Sie die Mengen V und E für diesen Graphen in Form der mathematischen Mengen- und Tupelschreibweise an.

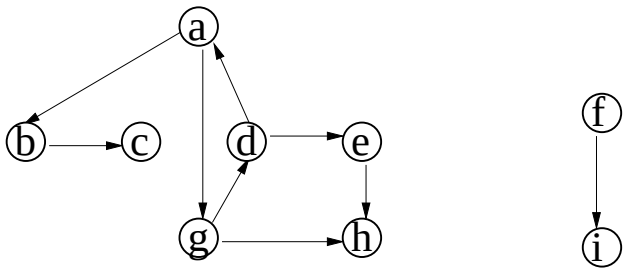


Abbildung 4: Graph G

6.3. Darstellung von Graphen (2 Punkte) Welche der folgenden Aussagen sind richtig?
Der Graph G in Abbildung 4 ist:

Ja	Nein	
◯	◯	schleifenfrei
◯	◯	zyklenfrei
◯	◯	zusammenhängend
◯	◯	vollständig

6.4. Breitensuche (3 Punkte) Führen Sie auf dem Graphen G aus Abbildung 4 eine Breitensuche beginnend bei Knoten a durch. Verwenden Sie zur Darstellung der einzelnen Schritte des Algorithmus die folgende Tabelle. Wenn Sie mehrere Elemente in einem Abarbeitungsschritt in die Queue einfügen müssen, tun Sie dies in alphabetischer Reihenfolge.

t	AK	Queue

Dabei bezeichne

- t den Abarbeitungsschritt
- AK den aktuellen Knoten
- *Queue* den aktuellen Zustand der Queue

7. Aufgabe (5 Punkte): Sortierverfahren

7.1. (1 Punkt) Abgebildet ist die Sortierung einer Sequenz von natürlichen Zahlen mit einem aus der Vorlesung bekannten Sortierverfahren. Geben Sie für den unten abgebildeten Ablauf an, mit welchem Sortierverfahren die Sortierung erfolgt ist.

```
< >   < 7 , 6 , 2 , 8 , 3 >
< 2 >     < 6 , 7 , 8 , 3 >
< 2 , 3 >     < 7 , 8 , 6 >
< 2 , 3 , 6 >     < 8 , 7 >
< 2 , 3 , 6 , 7 >     < 8 >
< 2 , 3 , 6 , 7 , 8 >   < >
```

7.2. (1 Punkt) Welches Sortierverfahren wird durch die Funktion `sort` implementiert?

```
FUN sort : seq[nat] -> seq[nat]
DEF sort(<>) == <>
DEF sort(l :: <>) == l :: <>
DEF sort(L) == LET (A,B) == split(#(L)/2,L)
               IN helpfun(sort(A),sort(B))

FUN helpfun : seq[nat] ** seq[nat] -> seq[nat]
DEF helpfun(<>, <>) == <>
DEF helpfun(A,<>) == A
DEF helpfun(<>,B) == B
DEF helpfun(a::A, b::B) == IF a<b THEN a::helpfun(A,b::B)
                                  ELSE b::helpfun(a::A,B)
                           FI
```

7.3. (3 Punkte) Zeigen Sie schrittweise die Vorgehenweise von Quicksort (Handsimulation) an der Beispielsequenz $< 7, 4, 9, 8, 3, 6, 2 >$.

Beachten Sie dabei folgende Regeln:

1. Pivotelement ist immer das erste Element in der (Teil-)liste.
2. Achten Sie darauf, dass alle Teilsequenzen, die beim Zerlegen und Zusammensetzen entstehen, dargestellt sind.
3. Das Aufschreiben des OPAL-Codes oder gar eine schrittweise Funktionsauswertung ist nicht notwendig.