Technische Universität München
Fakultät für Informatik
Prof. Dr. M. Broy

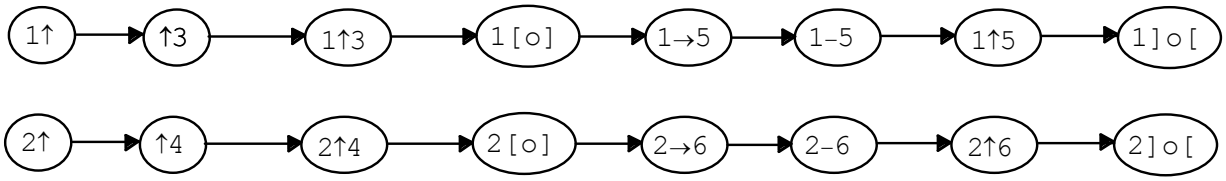
WS 2002
Lösungsblatt 1
25. Oktober 2002

Übungen zur Einführung in die Informatik III

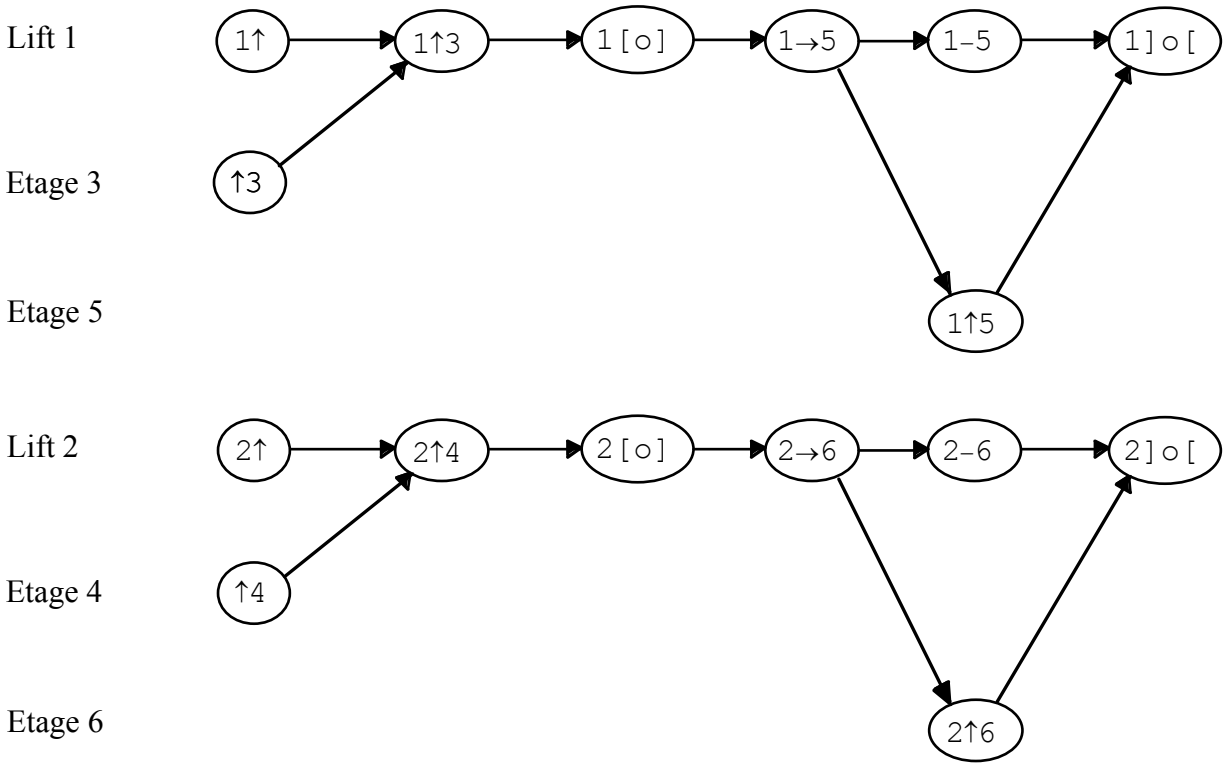
Aufgabe 1 (T) Fahrstuhl

a)

i) Es gibt viele mögliche Aktionsdiagramme, die jeweils Prozesse beschreiben, in denen Lift 1 eine Person von Etage 3 nach Etage 5 und Lift 2 eine zweite von Etage 4 nach 6 befördert. Ein einfaches Beispiel ist:



Ein allgemeineres Diagramm, aus dem sich obiges durch unvollständige Sequentialisierung

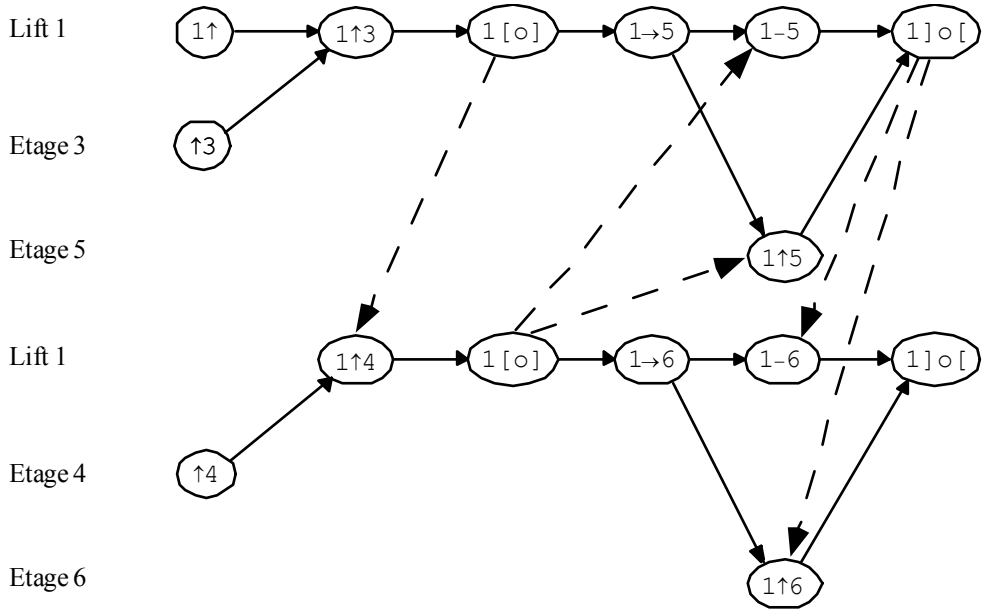


ergibt ist:

ii) Auch hier gibt es mehrere mögliche Diagramme, man erhält sie im Wesentlichen aus denen der Teilaufgabe i), wobei

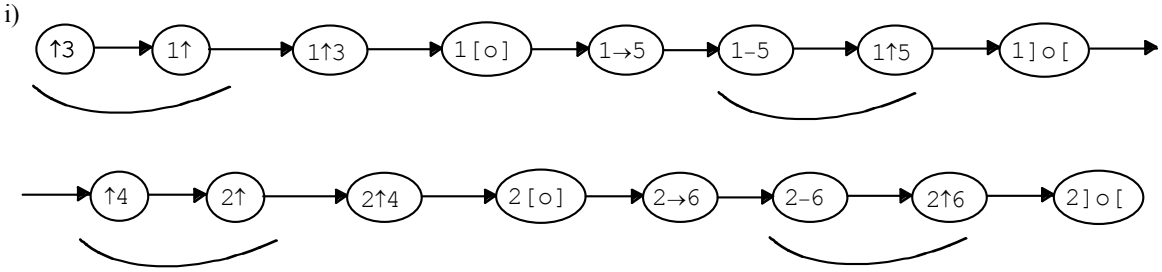
- Lift 2 in Lift 1 umbenannt wird,
- das initiale 

Ereignis nur einmal auftritt und
- die gestrichelten Linien zur Synchronisation ergänzt werden.



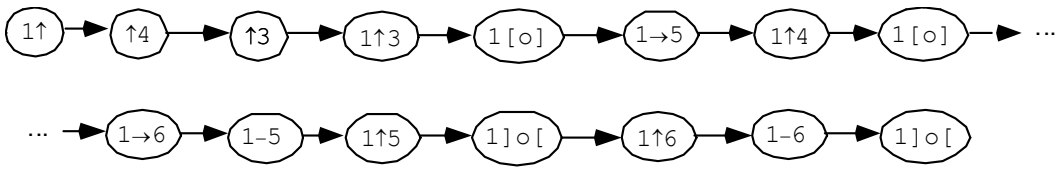
b)

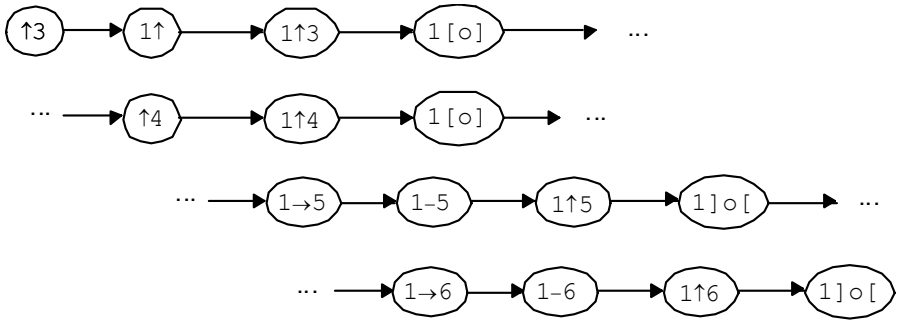
i)



Die geklammerten Ereignisse können jeweils beliebig vertauscht werden.

ii) Mögliche Sequentialisierungen sind u.a.:





Aufgabe 2 Kleinstes Präfix

Ausgehend von der Ereignismenge $E = \{e_i : i \in \mathbb{N}_0\}$ und der Aktionenmenge $A = \{a_0, a_1, a_2, a_3\}$ sei der Prozess $P = (E, \leq, \alpha)$ gegeben durch

$\alpha : E \to A$ def. durch $\alpha(e_i) = a_{i \bmod 4}$ für $i \in \mathbb{N}_0$

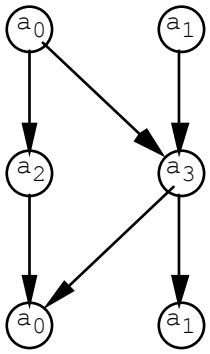
$\leq \subseteq E \times E$ def. als die reflexiv transitive Hülle der Menge

$\{(e_i, e_{i+2}), (e_{4i}, e_{4i+3}), (e_{4i+3}, e_{4i+4}) : i \in \mathbb{N}_0\}$

a) Ein Präfix von P ist ein Prozess $(E', \leq', \alpha')$ mit

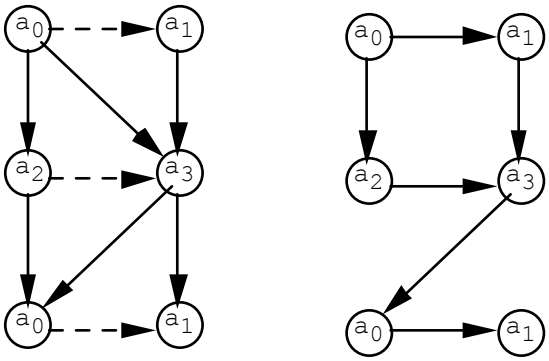
$E' \subseteq E$, $\leq' = \leq|_{E' \times E'}$ $\alpha' = \alpha|_{E'}$ und $\forall e \in E, e' \in E' : e \leq e' \Rightarrow e \in E'$.

Das kleinste Präfix $P_{4,5}$ von P, das die Ereignisse e_4 und e_5 enthält:



$P_{4,5}$

b) Eine unvollständige Sequentialisierung von $P_{4,5}$ ergibt sich z. B. durch Hinzufügen der drei gestrichelten Kanten und Weglassen jeder Kante, zwischen deren Endpunkten es noch einen anderen Pfad gibt:



Lässt man eine der drei gestrichelten Kanten aus, ist die Sequentialisierung natürlich auch unvollständig.

Eine vollständige Sequentialisierung erhält man durch *topologisches Sortieren*. Dabei sind die Knoten derart in einer Linie anzuordnen, dass alle Kanten in die gleiche Richtung zeigen. (Das lässt sich leicht bewerkstelligen, indem man mit einem Knoten beginnt, auf den keine Kante zeigt, diesen aus dem Graphen samt seinen ausgehenden Kanten entfernt und so fortfährt, bis die Knotenmenge erschöpft ist. Zyklenfreiheit ist bei Prozessen gegeben.) Die lineare Ordnung, die sich daraus in derselben Richtung zwischen den Knoten ergibt, erweitert die ursprüngliche Relation. Das Ergebnis einer vollständigen Sequentialisierung ist ein sequentieller Prozess.

c) Die Spuren (die Aktionsströme der vollständigen Sequentialisierungen) von $P_{4,5}$:

$<a_0, a_1, a_2, a_3, a_0, a_1>$	$<a_1, a_0, a_2, a_3, a_0, a_1>$
$<a_0, a_1, a_2, a_3, a_1, a_0>$	$<a_1, a_0, a_2, a_3, a_1, a_0>$
$<a_0, a_1, a_3, a_2, a_0, a_1>$	$<a_1, a_0, a_3, a_2, a_0, a_1>$
$<a_0, a_1, a_3, a_2, a_1, a_0>$	$<a_1, a_0, a_3, a_2, a_1, a_0>$
$<a_0, a_1, a_3, a_1, a_2, a_0>$	$<a_1, a_0, a_3, a_1, a_2, a_0>$
$<a_0, a_2, a_1, a_3, a_0, a_1>$	
$<a_0, a_2, a_1, a_3, a_1, a_0>$	

<u>Aufgabe 3</u> (H) Aktionsstruktur (Prozess)

a) Zur Definition der Aktionsstruktur $(E_0, \leq_0, \alpha)$ über einer Menge E von Ereignissen und einer Menge A von Aktionen gehört die Forderung, dass die Kausalitätsrelation $\leq_0$ endlich fundiert ist.

(1) Die Forderung drückt aus, dass jedes Ereignis (auch in unendlichen Prozessen) nur endlich viele kausale Vorgänger besitzt. Darauf beschränkt sich unsere Theorie, in der Annahme ein breites Spektrum von Anwendungen zu erfassen.

(1) Eine partielle Ordnung heißt Noethersch, wenn es keine unendlich absteigende Kette gibt. Beispiel einer Noetherschen aber nicht endlich fundierten partiellen Ordnung: Auf den natürlichen Zahlen stehe jede gerade Zahl zu allen ungeraden Zahlen in Relation, und sonst stehe nur jede Zahl zu sich selbst in Relation.

b) Sei $P_1=(E_1, \leq_1, \alpha_1)$ der Prozess

$$(\{e_1, e_2\}, \{(e_1, e_1), (e_2, e_2)\}, \{e_1 \mapsto a_1, e_2 \mapsto a_2\})$$

und $P_2=(E_2, \leq_2, \alpha_2)$ der Prozess

$$(\{e_1, e_2\}, \{(e_1, e_1), (e_2, e_2)\}, \{e_1 \mapsto a_2, e_2 \mapsto a_1\})$$

(1) P_1 ist <u>kein</u> Teilprozess von P_2,

(2) P_1 ist <u>keine</u> Sequentialisierung von P_2,

(3) P_1 ist aber isomorph zu P_2.

Begründung zu (1) und (2): Die Abbildungen $\alpha_1|E_2$ und α_2 sind verschieden.

Begründung zu (3): Die Abbildung $\varphi: E_1 \to E_2$, die die beiden Elemente vertauscht, ist surjektiv. Die Kausalitätsrelation bleibt erhalten, d.h.

$e_i \leq_1 e_k$ gilt genau dann, wenn $\varphi(e_i) \leq_2 \varphi(e_k)$ für i=1, 2 und k=1, 2

und für die Aktionszuordnungen gilt

$\alpha_1(e_i) = \alpha_2(\varphi(e_i))$ für i=1, 2

Aufgabe 4 (P) Aktionsstruktur, Darstellung

Es ist zu beachten, dass ein echter Graph in der Implementierung zwar viele Gemeinsamkeiten mit einem Baum aufweist, aber algorithmisch doch schwerer zu behandeln ist. Als Unterschiede fallen hier ins Gewicht: Auf einen Knoten kann mehr als ein Verweis gerichtet sein und der Graph kann mehr als eine Wurzel haben.

Zur Bearbeitung sind zwei gängige Hilfskonstruktionen benutzbar. Zum einen werden neben den Aktionen auch Ereignisnummern im Knoten gespeichert. Zum anderen werden Operationen auf dem Graphen durch ein Array, in dem Verweise auf sämtliche Knoten des Graphen gespeichert sind, beschleunigt.

Im Hauptprogramm main wird zunächst ein Eingabestrom erzeugt und dann an den Konstruktor für die Klasse AcGraph übergeben. Dort wird eine Aktionsstruktur eingelesen und aufgebaut, die zuletzt ausgegeben wird.

Der Aufbau der Aktionsstruktur erfolgt durch initiale Eingabe der Aktionen als Zeichenfolge. Damit werden die Ereignisknoten erzeugt und es sind für jedes Ereignis die Menge der Nachfolger einzugeben.

Die Ereignisse sind als Elemente der Klasse AcNode implementiert. Sie enthalten Ereignisnummer, Aktion und einen Vektor von Nachfolgern. Es können Knoten erzeugt werden und Nachfolger anschließend hinzugefügt werden.

Der entsprechende Quelltext ist im Folgenden aufgelistet. Er entspricht weitgehend gängigen Java-Codingstandards, und sind deshalb etwas länglich.

```
// Die Klasse AcNode repraesentiert einen Knoten des Aktionsgraphen.
import java.util.*;

public class AcNode {
    private int    _eventNumber;         // Ereignisnummer
    private char   _action;              // Aktion
    private Vector _successor;           // Nachfolger

    /**
     * Der Standardkonstruktor benoetigt die Aktionsbeschreibung (char) und
     * eine Aktionsnummer um den Knoten eindeutig zu identifizieren
     */
    public AcNode(char action, int eventNumber) {
      _eventNumber = eventNumber;
      _action      = action;
      _successor   = new Vector();
    }

    public int getEventNumber() {
      return _eventNumber;
    }

    public char getAction() {
      return _action;
    }

    public int getNumberOfSuccessors() {
```

```
      return _successor.size();
    }

    /**
     * @return Referenz auf den n-ten Nachfolger
     */
    public AcNode getSuccessor( int n ) {
      return (AcNode)_successor.elementAt( n );
    }

    /**
     * Nachfolger hinzufuegen
     */
    public void addSuccessor(AcNode next) {
      _successor.addElement( next );
    }

    /**
     * @return Stringrepraesentation des Aktionsknotens
     */
    public String toString () {
      return "(" + _eventNumber + "," + _action + ")";
    }

    /**
     * @return Stringrepraesentation des Aktionsknotens
     */
    public String toStringWithSuccessors () {
        String returnValue = "Knoten " + this + " -> ";
        for ( int i=0; i<getNumberOfSuccessors(); i++ ) {
          returnValue += " " + getSuccessor(i);
        }
        return returnValue;
    }
}
//---------------------------------------------------------------------------
/**
 * Die Klasse AcGraph verwaltet einen Aktionsgraphen unter Verwendung der Klasse
 * AcNode. Ueber ihre Methoden readNodes und readRelations bietet sie ein textuelles
 * Benutzerinterface. Desweiteren enthaelt die Klasse eine main-Methode, die bei
 * Verwendung dieser Klasse als Applikationshauptklasse eine Instanz von AcStruct
 * aufbaut und mit der Eingabekonsole verknuepft. Bei Dateiein- und ausgabe koennen
 * Exceptions auftreten, deren Behandlung noch einzubauen ist.
*/
import java.util.*;
import java.io.*;

public class AcGraph {

    /**
     * Kreiere einen I/O-Streams und lese Datenstruktur.
     */
    public static void main(String [] args) throws Exception {
      BufferedReader input = new BufferedReader(
        new InputStreamReader( System.in )
      );
      AcGraph a = new AcGraph();
```

```
  a.readNodes( input , System.out );
  a.readRelations( input , System.out );

  System.out.print( a.toString() );
}

protected Vector _nodes;          // Liste der Ereignisknoten

/**
 * Der Standardkonstruktor initialisiert die zu verwaltende Liste der
 * Knoten.
 */
public AcGraph() {
  _nodes = new Vector();
}

/**
 * Liste der Knoten
 */
public Vector getNodes() { return _nodes; }

/**
 * Fuegt einen neuen Knoten hinzu
 */
public void addNode( char action, int eventNumber ) {
  _nodes.add( new AcNode( action, eventNumber ));
}

/**
 * Fuegt eine neue Abhängigkeit hinzu
 * @return Stringrepraesentation der neuen Relation
 */
public String addDependency( int startIndex, int endIndex ) {
  AcNode startNode   = (AcNode)_nodes.elementAt( startIndex );
  AcNode endNode     = (AcNode)_nodes.elementAt( endIndex   );
  startNode.addSuccessor( endNode );
  return startNode.toString()  + " -> " + endNode.toString();
}

/**
 * Diese Methode liest aus der Eingabe Knoten aus
 */
public void readNodes( BufferedReader ins, PrintStream out) throws Exception {
  // Einlesen der Daten
  out.println(
    "Zunaechst jedem Ereignis-Index eine Aktion in Form eines \n" +
    "einzelnen Zeichens zuordnen. Ende durch leere Eingabe.\n\n" +
    "\tIndex\tEingabe: <aktion>"
  );
  for ( int i = 0; i > -1; i++ ) {
    out.print( "\t   " + i + "\t   " );

    String currentLine = ins.readLine();
    if ( currentLine.length() == 0 ) {
```

```
          return;
        }

        addNode( currentLine.charAt(0), i );
      };
    }

    /**
     * Diese Methode liest aus der Eingabe Abhaengigkeiten aus
     */
    public void readRelations(BufferedReader ins, PrintStream out) throws Exception
{
      out.println(
        "\n\nAbhaengigkeiten werden je durch die Eingabe zweier\n" +
        "Ereignis-Indizes hergestellt. Das Format ist \n" +
        "dabei 'startindex'-'endindex', also z.B. 0-1.\n" +
        "Ende durch leere Eingabe." );
      for ( int i = 1; i > -1; i++ ) {
        out.print( i + ". Abhaengigkeit: " );

        String currentLine = ins.readLine();
        if ( currentLine.length() == 0 ) {
          return;
        }

        int  startIndex = Integer.parseInt(
          currentLine.substring( 0, currentLine.indexOf( "-" ))
        );
        int  endIndex   = Integer.parseInt(
          currentLine.substring( currentLine.indexOf( "-" )+ 1 )
        );

        String message = addDependency( startIndex, endIndex );
        out.println( "Abhaengigkeit " + message + " wurde hergestellt!" );
      };
    }

    /**
     * @return Repraesentation der verwalteten Liste von Knoten
     */
    public String toString () {
      String returnValue = "Aktionsstruktur mit " + _nodes.size() + " Knoten: \n";

      for(int i=0; i<_nodes.size(); i++) {
        returnValue += ((AcNode)_nodes.elementAt(i)).toStringWithSuccessors();
        returnValue += "\n";
      }

      return returnValue;
    }
}
```

Bemerkungen:

Nachfolgend ist eine Testklasse angegeben, die das JUnit-Framework nutzt. Damit wurde die Musterlösung auf Plausibilität geprüft. Junit erlaubt die einfache Definition von automatisierten Tests. Mehr Information über Junit ist in www.junit.org zu finden. Der Aufruf erfolgt mit:

javac -classpath junit3.8.1/junit.jar AcGraph.java AcGraphTest.java AcNode.java

java -classpath ".;junit3.8.1/junit.jar" junit.swingui.TestRunner AcGraphTest

Weil im obigen Programm Ein- und Ausgabe über explizite Parameter erfolgen, können in den Tests beide simuliert werden. Das ist für automatisierte Tests wichtig.

```
// Klasse AcGraphTest.java
// Drei einfache Tests für den Aufbau der ACGraphen
import junit.framework.*;
import java.io.*;

public class AcGraphTest extends TestCase {

  public AcGraphTest(String name) { super(name); }

  // Sammlung von Tests
  public static Test suite()
  {
    TestSuite suite= new TestSuite();
    suite.addTest(new AcGraphTest("testEmptyAcGraph"));
    suite.addTest(new AcGraphTest("testDoubleAcGraph"));
    suite.addTest(new AcGraphTest("testAcGraphLong"));
    return suite;
  }

  // Ausgabe wird hierher umgeleitet
  PrintStream out = new PrintStream( new ByteArrayOutputStream() );

  // Test fuer einen leeren AcGraphen
  public void testEmptyAcGraph() throws Exception {
    BufferedReader input = new BufferedReader(
      new StringReader("\n"+  "\n"));
    AcGraph a = new AcGraph();
    a.readNodes( input , out );
    a.readRelations( input , out );

    assertEquals(a.getNodes().size(), 0);
  }

  // Test fuer einen leeren AcGraphen
  public void testDoubleAcGraph() throws Exception {
    BufferedReader input = new BufferedReader(
      new StringReader("a\nb\n\n"+  "0-1\n\n"));
    AcGraph a = new AcGraph();
    a.readNodes( input , out );
    a.readRelations( input , out );

    assertEquals(a.getNodes().size(), 2);
    assertEquals(((AcNode)a.getNodes().get(1)).getAction(), 'b');
    assertEquals(((AcNode)a.getNodes().get(0)).getNumberOfSuccessors(), 1);
```

```
  }

  // Test fuer einen leeren AcGraphen
  public void testAcGraphLong() throws Exception {
    BufferedReader input = new BufferedReader(
      new StringReader("a\nb\nc\nd\ne\nf\ng\n\n"+
                       "0-1\n0-2\n1-3\n1-4\n2-4\n3-5\n4-5\n4-6\n\n"));
    AcGraph a = new AcGraph();
    a.readNodes( input , out );
    a.readRelations( input , out );

    assertEquals(a.getNodes().size(), 7);
    assertEquals(((AcNode)a.getNodes().get(6)).getAction(), 'g');
    assertEquals(((AcNode)a.getNodes().get(0)).getSuccessor(1).getEventNumber(), 2);
  }
}
```