S. 103 / 4

	k1	k2	k3	k4	k5	k6
k1		9	3	~~7~~		
k2	9			4	6	
k3	3					5
k4	~~7~~	4				4
k5		6				6
k6			5	4	6	

S. 103 / 4

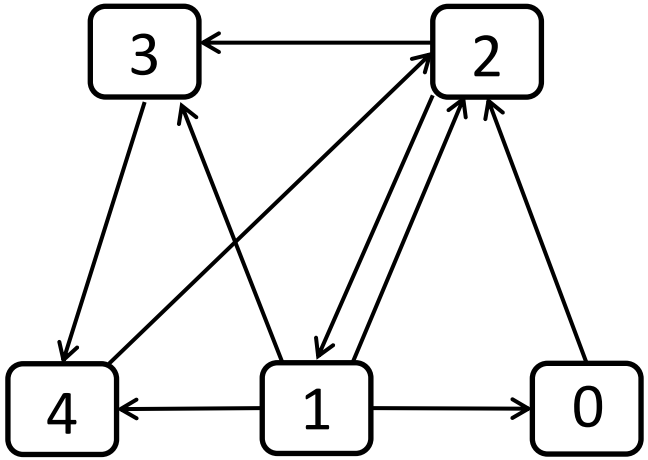
	k1	k2	k3	k4	k5	k6
k1		9	3	~~7~~		10
k2	9			4	6	
k3	3					5
k4	~~7~~	4				4
k5		6				6
k6	10		5	4	6	

S. 104 / 5

	A	B	C	D
A		X		X
B	X			X
C	X			X
D				

Permutationen der Spalten: n!

S. 104 / 6



S. 106 / 13 Sichou zhi lu

	KON	PAL	KAI	KAS	KUC	DUN	ANX	CHA	PEK	HAN
KON		1545								
PAL	1545		800?							
KAI		800?								
KAS										
KUC										
DUN										
ANX										
CHA										
PEK										
HAN										

S. 106 / 13 Sichou zhi lu

	KON	PAL	KAI	KAS	KUC	DUN	ANX	CHA	PEK	HAN
KON		1545								
PAL	1545		1100	4400						
KAI		1100								
KAS		4400			750	2200				
KUC				750			1500			
DUN				2200			200			
ANX					1500	200		1793		
CHA							1793		1125	1140
PEK								1125		
HAN								1140		

- Symmetrische Adjazenzmatrix, da ungerichteter Graph
- Gewichtete Kanten: Adjazenzmatrix int[][]
- Beim Einfügen von Kanten muss ein Wert gesetzt werden.

```
public class Stadt extends Datenelement {

    private String name;
    private String kuerzel;

    public Stadt(String n, String k){
        name = n;
        kuerzel = k;
    }

     public void datenAusgeben(){
         System.out.println(name +", " + kuerzel);
     }

     public String kuerzelGeben(){
         return kuerzel;
     }
}
```

```
public abstract class Datenelement{
    public abstract void datenAusgeben();
    public abstract String kuerzelGeben();
}

public class Knoten{

    private Datenelement inhalt;

    public Knoten(Datenelement inh){
        inhalt = inh;
    }

    public Datenelement inhaltGeben(){
        return inhalt;
    }
}
```

```
public class Graph{

   private Knoten[] knoten;
   private int[][] adjazenzmatrix;
   int maxAnzahl;
   int anzahl;

   public Graph(int m){
      knoten = new Knoten[m];
      adjazenzmatrix = new int[m][m];
      maxAnzahl = m;
      anzahl = 0;
    }
```

```
public void kanteEinfuegen(int i, int j, int wert){
    if (i<anzahl && j<anzahl) {
        adjazenzmatrix[i][j] = wert;
        adjazenzmatrix[j][i] = wert;
    }
    else {
        System.out.println("Kante kann nicht eingefuegt
                                    werden!");
    }
}
```

```
public void kanteEntfernen(int i, int j){
    if (i<anzahl && j<anzahl && adjazenzmatrix[i][j]!=0) {
        adjazenzmatrix[i][j] = 0;
        adjazenzmatrix[j][i] = 0;
    }
    else {
        System.out.println("Hier ist keine Kante
                                        vorhanden!");
    }
}
```

```
public int knotenindexSuchen (String kuerz){
    int index = -1;
    int zaehler = 0;
    while (index < 0 && zaehler < anzahl){
        if (knoten[zaehler].inhaltGeben().kuerzelGeben().
                        equals(kuerz)){
            index = zaehler;
        }
        zaehler++;
    }
    if (index<0) {System.out.println("Knoten nicht
                        vorhanden!");}
    return index;
  }
```

```
public class Seidenstraße{

  private Graph graph = new Graph(20);

  public void ablaufen(){
     //Knoten erzeugen
     graph.knotenEinfuegen(new Knoten(new Stadt("Konstantinopel", "KON")));
     graph.knotenEinfuegen(new Knoten(new Stadt("Palmyra", "PAL")));
     …

  //Kanten einfuegen
        graph.kanteEinfuegen(graph.knotenindexSuchen("KON"),
                             graph.knotenindexSuchen("PAL"), 100);
        graph.kanteEinfuegen(graph.knotenindexSuchen("KAI"),
                             graph.knotenindexSuchen("PAL"), 100);
            …
        graph.knotenlisteAusgeben();
        graph.adjazenzmatrixAusgeben();
    }
```