Prüfungsbereich „Entwicklung eines Softwaresystems“

IHK Abschlussprüfung 2012

Vorgelegt von: **Alexander Willkomm**
Prüflingsnummer: **101 20620**

Tag der Einreichung: **25.05.2012**
Programmiersprache: **C#**

AIXITEM GmbH

Gartenstraße 38
52249 Eschweiler
Tel. +49 2403 802293
Fax +49 2403 802294

Inhaltsverzeichnis

1. Eigenhändigkeitserklärung

Die praktische Arbeit habe ich selbständig erstellt und keine Hilfe in Anspruch genommen bei

- Konzepterstellung und Programmierung
- inhaltlicher Gestaltung der Dokumentation
- Auswahl und Diskussion der Testbeispiele.

Die als Arbeitshilfe benutzte Literatur ist in der Arbeit oder in dem Anhang vollständig aufgeführt.

Außerdem bestätige ich die Übereinstimmung der gedruckten mit der auf der CD befindlichen Version.

Name: Alexander Willkomm

Eschweiler, den 25.05.2012

2. Verwendete Hilfsmittel

Zur Erstellung dieser Prüfungsleistung wurde Windows 7 (Service Pack 1) mit Microsoft Visual Studio 2010 Ultimate verwendet. Die verwendete Programmiersprache ist C#.

Der verwendete Rechner hat einen Intel XEON X3420 Prozessor mit 4 GB Arbeitsspeicher. Es handelt sich um ein 64 Bit-System.

Weitere verwendete Programme sind

- Microsoft Visio
- Microsoft Word
- Microsoft Source Safe
- Sandcastle

3. Aufgabenstellung

Siehe Anhang A

4. Analyse des Auftrags

4.1. Aufgabenanalyse

Es soll ein Programm entwickelt werden, welches den kostengünstigsten und kürzesten Weg zwischen zwei Zellen in einem n x m Feld berechnet. Das Feld hat dabei maximal 20x20 Zellen und ist in einer Eingabedatei beschrieben.

Diese Datei soll zunächst entsprechend der vorgegebenen Aufgabenstellung eingelesen werden. Dabei ist die Einhaltung verschiedener Regeln (max 20x20 Zellen, angegebene Start- und Zielzelle, “alle Zellen besetzt“) zu prüfen.

Anschließend sind diese Daten zu verarbeiten. Zunächst soll eine realistische Kostengrenze für einen Weg von Start- zu Zielzelle ermittelt werden. Danach soll der kostenminimale Weg bestimmt werden, wobei bei mehreren Wegen mit minimalen Kosten der kürzeste Weg zu wählen ist.

Das Feld kann als Graph mit n mal m Knoten betrachtet werden. Die Kanten und ihre Kosten sind in der Zelle angegeben. So besteht in Beispiel 1 der Aufgabenstellung (siehe Anhang A) eine Verbindung zwischen Knoten 1,1 und 1,2 mit den Kosten 9.

Anschließend soll das Ergebnis der Verarbeitung ausgegeben werden. Dazu soll auch der Weg mit den minimalen Kosten ausgegeben werden.

Besondere Voraussetzungen:

- Zellen werden immer als Zeile, Spalte des Feldes an- bzw. ausgegeben
- Alle Zahlenwerte sind Ganzzahlen, also Integer - Werte (da sowohl die Indizes, als auch die Wegkosten als Ganzzahlen gegeben sind)
- Es bestehen keine diagonalen Verbindungen

4.2.Verbale Verfahrensbeschreibung

4.2.1. Eingabe

Zur Eingabe wird eine Datei eingelesen. In der ersten Zeile steht der Kommentar mit Beschreibung des Feldes. Diese Beschreibung ist als String zu speichern.

Anschließend folgen die Kosten der Zellen zeilenweise. Die Spalten sind mit einem Komma getrennt. Die Datei wird zeilenweise ausgelesen und die gelesenen Werte in einem 2-Dimensionalen Integer Array gespeichert. Die Startzelle und Zielzelle werden als 0 gespeichert, da dort keine Kosten entstehen. Die Speicherung erfolgt mittels den Indizes i und k. i iteriert über die Zeilen, k über die Spalten. Start- und Zielzelle sollen in gesonderten Feldern gespeichert werden, sodass diese nachher wieder zu finden sind. Dazu sind vier Integer Werte nötig (Zeile und Spalte der Start- / Zielzelle). Außerdem wird geprüft, ob Start und Ziel angegeben sind. Dazu werden vor der Iteration zwei Boolesche Werte auf falsch gesetzt. Sobald Start und Ziel gespeichert wurden, werden diese Boolesche Werte auf wahr gesetzt. Wenn am Ende der Iteration nicht beide Werte wahr sind, wird ein Fehler ausgegeben und das Programm abgebrochen.

Die erste eingelesene Zeile legt die Anzahl der Spalten fest. Hat die erste Zeile mehr als 20 Zellen wird das Programm abgebrochen und ein Fehler ausgegeben.

Werden mehr als 20 Zeilen eingelesen ist die Eingabe fehlerhaft. Ein Fehler wird ausgegeben und das Programm abgebrochen.

Wird beim Einlesen ein Fehler festgestellt, so soll in der Eingabe Funktion eine Ausnahme ausgelöst werden.

4.2.2. Verarbeitung

Kostenabschätzung

Bei der Verarbeitung soll zunächst eine Abschätzung der Kostenobergrenze ermittelt werden. Diese kann durch das eingelesene Array recht einfach erfolgen.

Die Idee ist folgende:

Von der Startzelle geht man vertikal bis zur Zeile der Zielzelle. Danach geht man von dieser Position horizontal in Richtung Zielzelle, bis man dort angekommen ist und addiert dabei jeweils den Kostenwert dieser Zellen. Die Gesamtkosten dieses Weges ist die Abschätzung der Kostenobergrenze. Es handelt sich bei dieser Lösung um den einfachsten kürzesten Weg zwischen Start und Ziel. Es ist sozusagen die triviale Lösung des Gesamtproblems. In Abbildung 1 ist ein solcher Weg eingezeichnet. Es ergibt sich folgende Kostenabschätzung:

Weg von Start zur Zeile von Ziel:	1 + 7 + 3 + 9 + 4 = 24 [KE]
Weg von 7,2 zum Ziel:	3 + 4 + 5 + 5 = 17 [KE]
Insgesamt:	**24 + 17 = 41 [KE]**

1	3	5	3	3	2	1	1
4	S	6	5	3	2	8	1
1	1	2	2	3	4	4	7
7	7	7	7	7	7	7	7
8	3	1	3	3	4	4	9
9	9	9	9	9	9	9	9
5	4	3	4	5	5	Z	3
1	1	1	1	2	2	3	3

Abbildung 1 Beispiel für einen Weg zu Kostenabschätzung

Minimale Kosten

Nachdem die Kostenabschätzung ermittelt wurde soll ein Weg mit minimalen Kosten bestimmt werden. Dazu kann das Feld als ein Graph mit Knoten und Kanten betrachtet werden. Die Zellen stellen die Knoten dar, wobei der Wert einer Zelle die Kosten von Verbindungen zu diesem Knoten angibt. Es gibt also m mal n Knoten. In Beispiel 1 ist eine solche Umwandlung dargestellt.

Zur Vereinfachung der Berechnung des minimalen Weges wird intern das Feld in eine Adjazenzmatrix umgewandelt. Eine Adjazenz-Matrix gibt Verbindungen zwischen zwei Knoten an. Eine Verbindung wird durch die entsprechenden Kosten gekennzeichnet, die Verbindung eines Knoten mit sich selbst wird als 0 notiert. Besteht keine Verbindung wird unendlich notiert (im Programm -1).

Zur Umwandlung in diesem Fall wird über das Feld aus der Eingabe iteriert und dabei die Kosten zu den jeweiligen Nachbarn in die Adjazenzmatrix übertragen. Es wird vereinbart, dass die Knoten im Bezug auf das Feld zeilenweise in die Adjazenzmatrix eingetragen sind. Die Zuordnung von Bezeichnung des Knotens in der Adjazenzmatrix zu Zeile und Spalte des Feldes bzw. umgekehrt, stellt sich demnach wie folgt dar:

$$Knoten = Zeile \cdot Breite\ Feld + Spalte$$

$$Zeile\ zu\ Knoten = Ganzzahl\ (\ \frac{Knoten}{Breite\ Feld}\ )$$

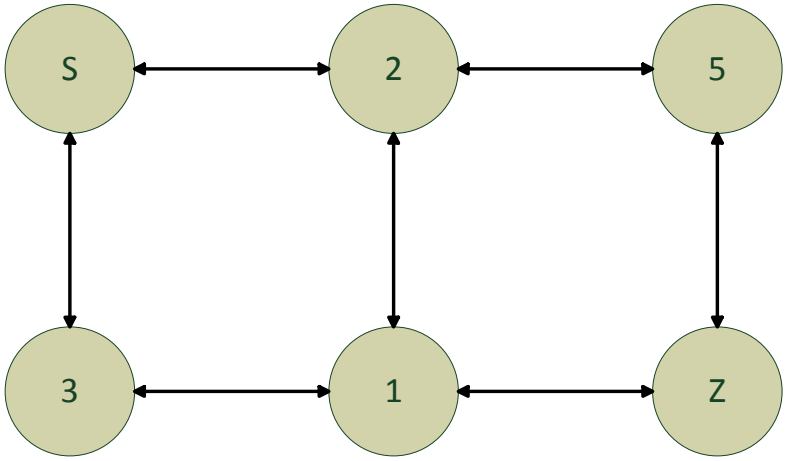
$$Spalte\ zu\ Knoten = Knoten\ modulo\ Breite\ Feld$$

Im nachfolgenden Beispiel ist eine oben beschriebene Umwandlung von Feld zu Adjazenzmatrix dargestellt:

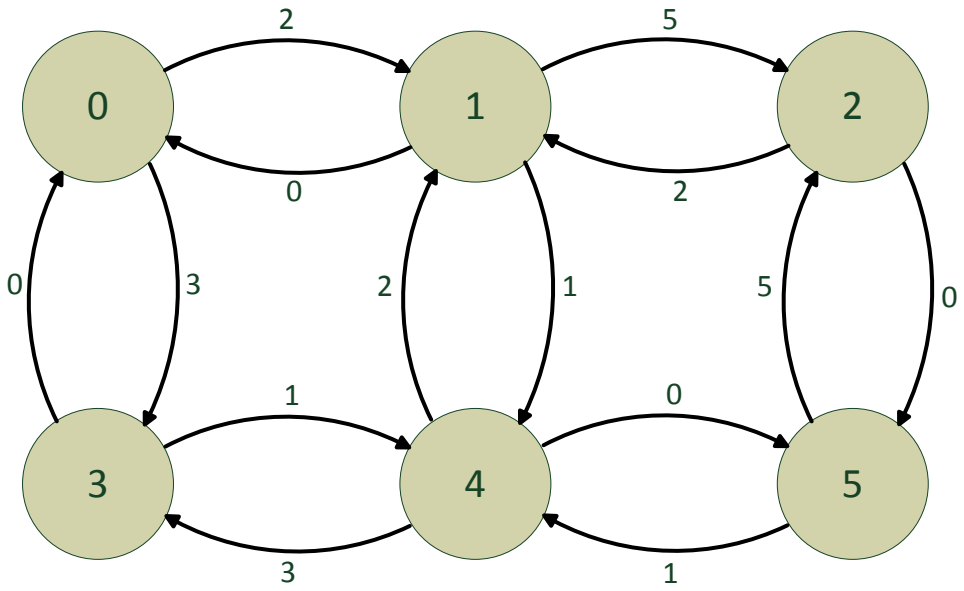
Das Ausgangsfeld (2 x 3)

S	2	5
3	1	Z

wird als Graph betrachtet



Die Kosten ergeben sich aus den Werten auf den Knoten und werden auf die Kanten übertragen. Die Kosten zu Start (Knoten 0) und Ziel (Knoten 5) sind 0:



Dieser Graph wird als Adjazenzmatrix gespeichert:

	0	1	2	3	4	5
0	0	2	∞	3	∞	∞
1	0	0	5	∞	1	∞
2	∞	2	0	∞	∞	0
3	0	∞	∞	0	1	∞
4	∞	2	∞	3	0	0
5	∞	∞	5	∞	1	0

Beispiel 1 – Vom Feld zur Adjazenzmatrix

Der Dijkstra Algorithmus ermittelt den kürzesten, bzw. kostengünstigsten Weg von einem Knoten A zu einem Knoten B. Er versucht dazu in jeder Iteration die Verbindung zu den Knoten zu verbessern und merkt sich für die Erstellung eines Weges bei Verbesserung die Vorgängerknoten. Der genaue Ablauf ist nachfolgend beschrieben.

Ablauf Dijkstra

Die Iteration beginnt mit dem Startknoten. In der ersten Iteration werden die Kosten und die Distanz vom Startknoten zu den nachfolgenden Knoten notiert. Außerdem wird bei Knoten mit Verbindung zum Startknoten, der Startknoten als Vorgänger eingetragen. Für die Knoten ohne Verbindung zum Startknoten wird unendlich markiert.

In der nächsten Iteration wird zunächst nach dem Knoten mit der bisher günstigsten Verbindung gesucht. Diese Kosten werden als derzeitige Minimalkosten gespeichert und bilden die Grundlage für die Verbesserungsversuche in dieser Iteration. Dazu wird versucht, den Weg zu einem Knoten über den aktuellen Knoten zu verbessern. Es werden sowohl die Kosten, als auch die Distanz berücksichtigt, sodass bei mehreren Wegen mit den gleichen Minimalkosten, der kürzeste Weg gefunden wird. Zur Berücksichtigung der Distanz wurde im Vergleich zum allgemeinen Dijkstra-Algorithmus die Eigenschaft Distanz eingeführt. Bei verbesserten Verbindungen wird als Vorgänger der aktuelle Knoten eingetragen. Am Ende der Iteration wird der aktuelle Knoten markiert, da dieser nicht mehr besucht werden muss, so werden Zyklen vermieden. [1]

Beim Erreichen des Zielknotens kann der Algorithmus beendet werden.

Der kürzeste Pfad lässt sich mittels einer Rückwärtssuche durch die Vorgänger ablesen.

Nachfolgend ist ein konkretes Zahlenbeispiel (mit dem Graphen aus Beispiel 1) aufgeführt:

	Kosten						Distanz						Vorgänger					
	0	1	2	3	4	5	0	1	2	3	4	5	0	1	2	3	4	5
0	0	2	∞	3	∞	∞	0	1	∞	1	∞	∞		0		0		
1	0	2	7	3	3	∞	0	1	2	1	2	∞		0	1	0	1	
3	0	2	7	3	3	∞	0	1	2	1	2	∞		0	1	0	1	
4	0	2	7	3	3	3	0	1	2	1	2	3		0	1	0	1	4
5	0	2	7	3	3	3	0	1	2	1	2	3		0	1	0	1	4

Beispiel 2 – Zahlenbeispiel des verwendeten Algorithmus

4.2.3. Ausgabe

Abschließend werden die Ergebnisse ausgegeben. Die Ergebnisse werden sowohl auf dem Bildschirm als auch in einer Datei ausgegeben. Dazu wird zunächst der String mit dem Kommentar und anschließend die gefundenen Start- und Zielzellen aus dem Feld Array ausgegeben. Dabei ist darauf zu achten, dass der Index der Ausgabe bei 1 beginnt, wobei ein Array mit Index 0 beginnt.

Die ermittelte Abschätzung der Kostenobergrenze, sowie die durch den Dijkstra Algorithmus ermittelten Minimalkosten und der optimale Weg werden ausgegeben. Für den optimalen Weg müssen die Bezeichnungen der Knoten in der Adjazenzmatrix wieder in die Koordinatenform des Feldes übertragen werden.

5. Umsetzung des Programmsystems

In diesem Kapitel wird auf die Implementierung des oben beschriebenen Lösungsansatzes eingegangen. Dazu wird Visual C# 2010 Konsolenanwendung verwendet.

5.1. Programmkonzeption (Klassen, Methoden)

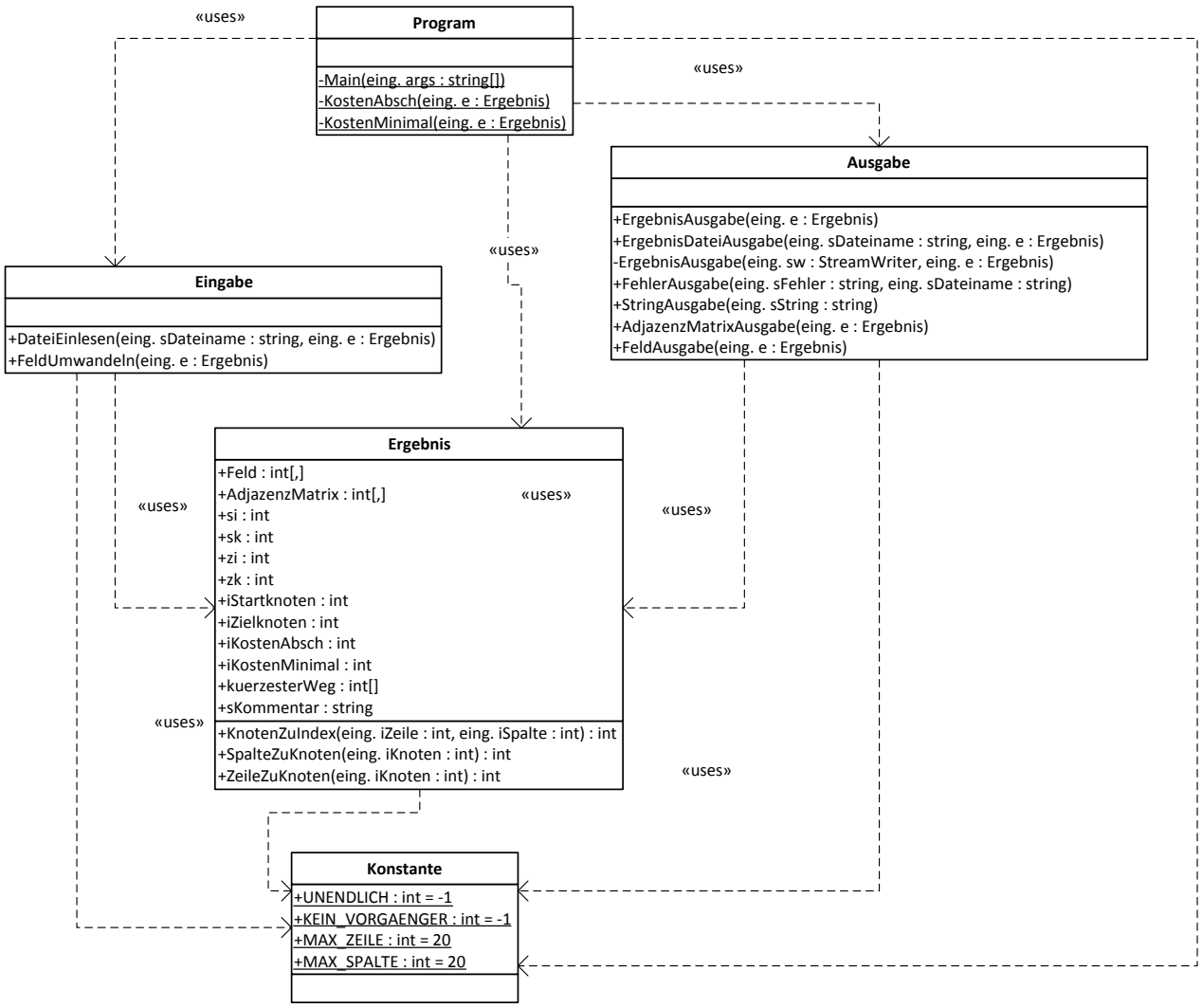


Abbildung 2 – Das Klassendiagramm

Das Programm ist in die Bereiche Eingabe, Ausgabe sowie Verarbeitung (Program) eingeteilt. Die Klasse Ergebnis dient zur Speicherung der eingegeben und errechneten Daten. In der Klasse Konstanten werden zur besseren Lesbarkeit der Algorithmen programmweite Konstanten eingetragen.

5.1.1. Klasse Eingabe

Klasse zur Eingabe und Umwandlung von Daten.

Konstruktor
Initialisiert ein neues Eingabe-Objekt.

Methoden

public DateiEinlesen (string)	Liest eine Datei ein und speichert die Daten in ein Ergebnis-Objekt Exception: FileNotFoundException, FileLoadException, ArithmeticException
public FeldUmwandeln (Ergebnis)	wandelt ein Feld in eine Adjazenzmatrix um

5.1.2. Klasse Ausgabe

Klasse zur Ausgabe von Ergebnissen, Fehlern und Informationen. Außerdem können Datenstrukturen zu Testzwecken ausgegeben werden.

Konstruktor
Initialisiert ein neues Ausgabe-Objekt.

Methoden

public AdjazenzMatrixAusgabe (Ergebnis)	Gibt eine Adjazenzmatrix des Ergebnis-Objekts auf Bildschirm zu Testzwecken aus.
public ErgebnisAusgabe(Ergebnis)	Gibt das Ergebnis in der vorgegebenen Form auf dem Bildschirm aus
public ErgebnisAusgabe(StreamWriter, Ergebnis)	Gibt das Ergebnis in der vorgegebenen Form in einem StreamWriter aus
public ErgebnisDateiAusgabe(string, Ergebnis)	Gibt das Ergebnis in der vorgegebenen Form in einer Datei aus Exception: FieldAccessException
public FehlerAusgabe(string, string)	Gibt einen Fehler auf Bildschirm und in einer Datei aus
public FeldAusgabe(Ergebnis)	Gibt ein Feld des Ergebnis Objekts auf dem Bildschirm zu Testzwecken aus
public StringAusgabe(string)	Gibt einen String auf dem Bildschirm aus

5.1.3. Klasse Program

Klasse zur Steuerung des Programms und zur Berechnung der Ergebnisse

Konstruktor
Initialisiert ein neues Program-Objekt.

Methoden

public static main(string[])	Die Main-Methode, steuert den Programmablauf
public KostenAbsch(Ergebnis)	Errechnet eine Kostenabschätzung Exception: Exception (Feld = null)
public KostenMinimal(Ergebnis)	Errechnet einen optimalen Weg mit Kostenminimum Exception: Exception (AdjazenzMatrix = null)

5.1.4. Klasse Ergebnis

Klasse zur Speicherung der eingelesenen Daten und der Ergebnisse

Konstruktor
Initialisiert ein neues Ergebnis-Objekt.

Methoden

public KnotenZuIndex(int, int)	Gibt die Zahl des Knotens in der Adjazenzmatrix für eine Zeile und Spalte des Feldes zurück
public SpalteZuKnoten(int)	Gibt die Spalte zu einer Knotenzahl der Adjazenzmatrix
public ZeileZuKnoten(int)	Gibt die Zeile zu einer Knotenzahl der Adjazenzmatrix

Eigenschaften

public int[][]	AdjazenzMatrix	Adjazenzmatrix für Berechnung der minimalen Lösung
public int[][]	Feld	Feld wie eingelesen, S,Z als 0
public int	iKostenAbsch	Errechnete Kostenabschätzung
public int	iKostenMinimal	Errechnete Minimalkosten
public int	iStartknoten	Start als Knoten in der Adjazenzmatrix
public int	iZielknoten	Ziel als Knoten in der Adjazenzmatrix
public int[]	kuerzesterWeg	Kürzester Weg bei minimalen Kosten

public int	si	Zeile Start
public int	sk	Spalte Start
public string	sKommentar	Kommentar aus der Datei eingelesen
public int	zi	Zeile Ziel
public int	zk	Spalte Ziel

5.1.5. Klasse Konstante

Klasse zur Festlegung von programmweiten Parametern

Eigenschaften

public static int	KEIN_VORGAENGER	Markierung kein Vorgänger Knoten
public static int	MAX_SPALTE	Maximale Anzahl Spalten
public static int	MAX_ZEILE	Maximale Anzahl Zeilen
public static int	UNENDLICH	Unendlich (Kosten, Distanz, etc.)

5.2. Sequenz des Programmablaufs

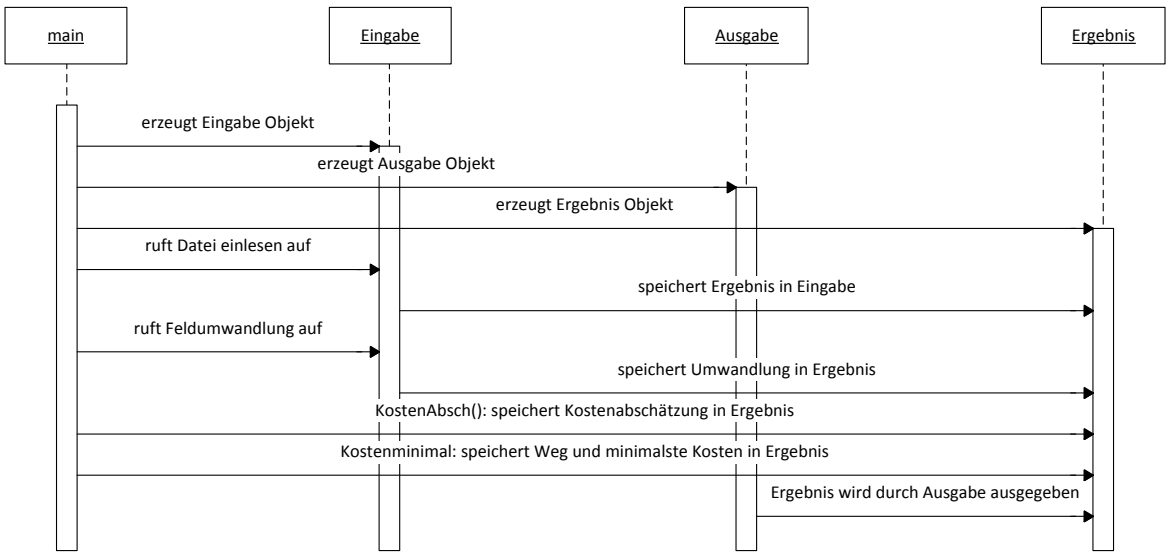


Abbildung 3 – Sequenz des Programmablaufs

5.3. Detaillierte Beschreibung der wesentlichen Methoden

Nachfolgend werden wesentliche Methoden der einzelnen Klassen in Form von Nassi-Shneiderman-Diagrammen beschrieben

5.3.1. Eingabe

DateiEinlesen

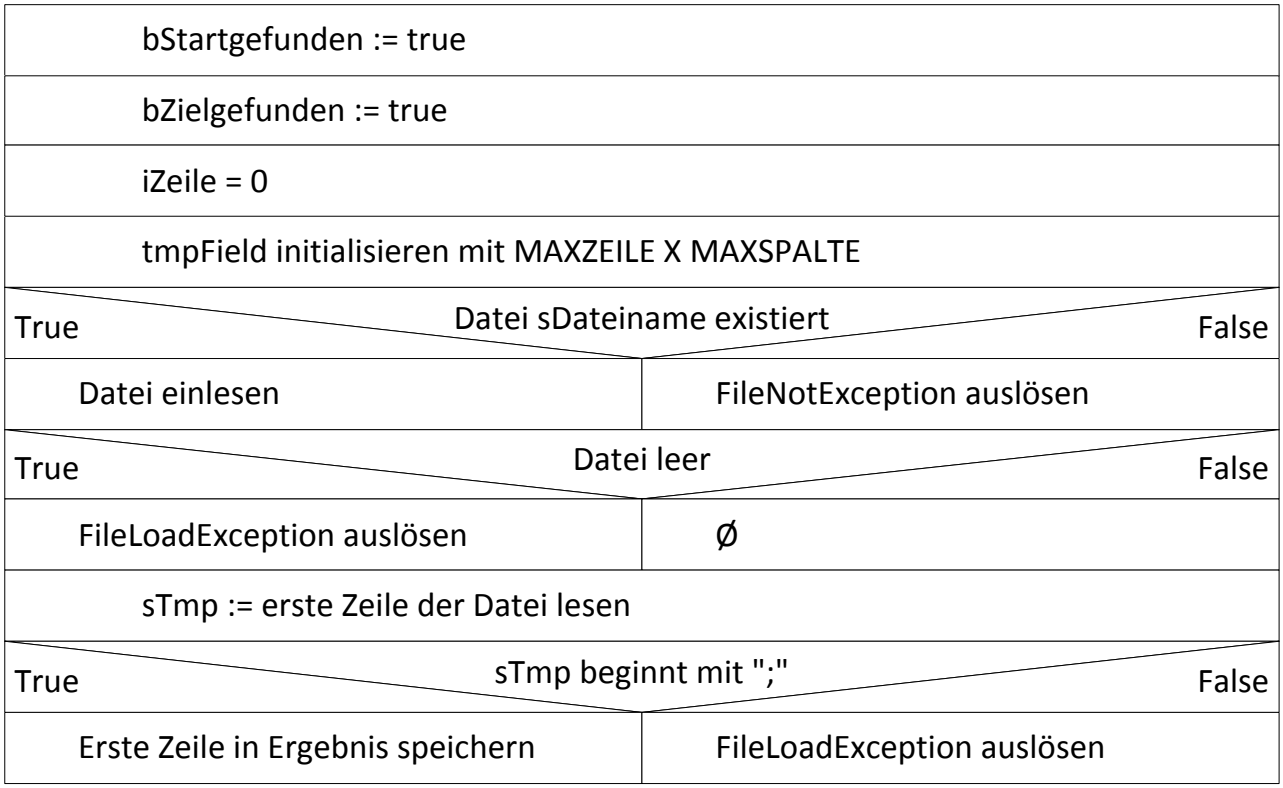


Abbildung 4 – die Methode DateiEinlesen

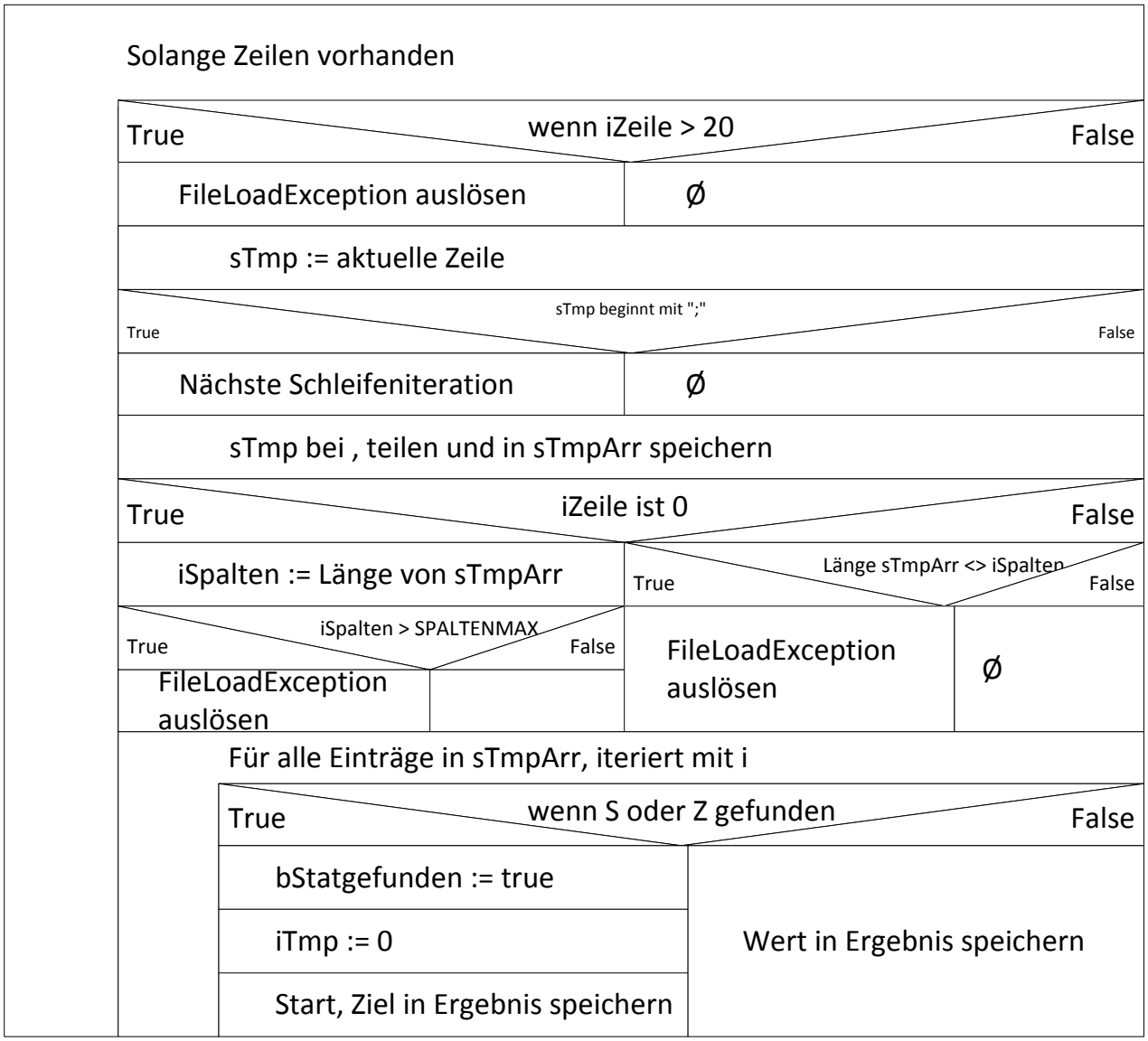


Abbildung 5 – die Methode DateiEinlesen (Fortsetzung)

Diese Funktion dient dem Einlesen von Eingabedateien. Das in Abbildung 4 und Abbildung 5 gezeigte Diagramm gibt eine vereinfachte Form der Methode an (siehe also auch mitgelieferten Code). Zunächst werden die Variablen initialisiert. Zur Speicherung des eingelesenen Feldes wird das Feld des Ergebnis-Objekts verwendet. Zu Beginn des Einlesevorgangs ist noch nicht klar, wie viele Zeilen und Spalten eingelesen werden, daher wird zunächst ein temporäres Feld mit der maximalen Höhe und Breite erstellt, um dieses dann nachher in das des Ergebnis-Objekts zu übertragen.

5.3.2. Program

KostenAbsch

Diese Methode berechnet eine Kostenabschätzung. Dazu wird zunächst bestimmt, ob die Startzelle „oberhalb“ der Zielzelle liegt und in entsprechende Richtung der Weg auf dem Feld gegangen. Es wird zunächst vertikal, danach horizontal in Richtung Zielzelle vorgegangen. Die Summe der Kosten entspricht der geforderten Abschätzung (siehe auch: 4.2.2)

Abbildung 6 zeigt das Diagramm zu dieser Methode.

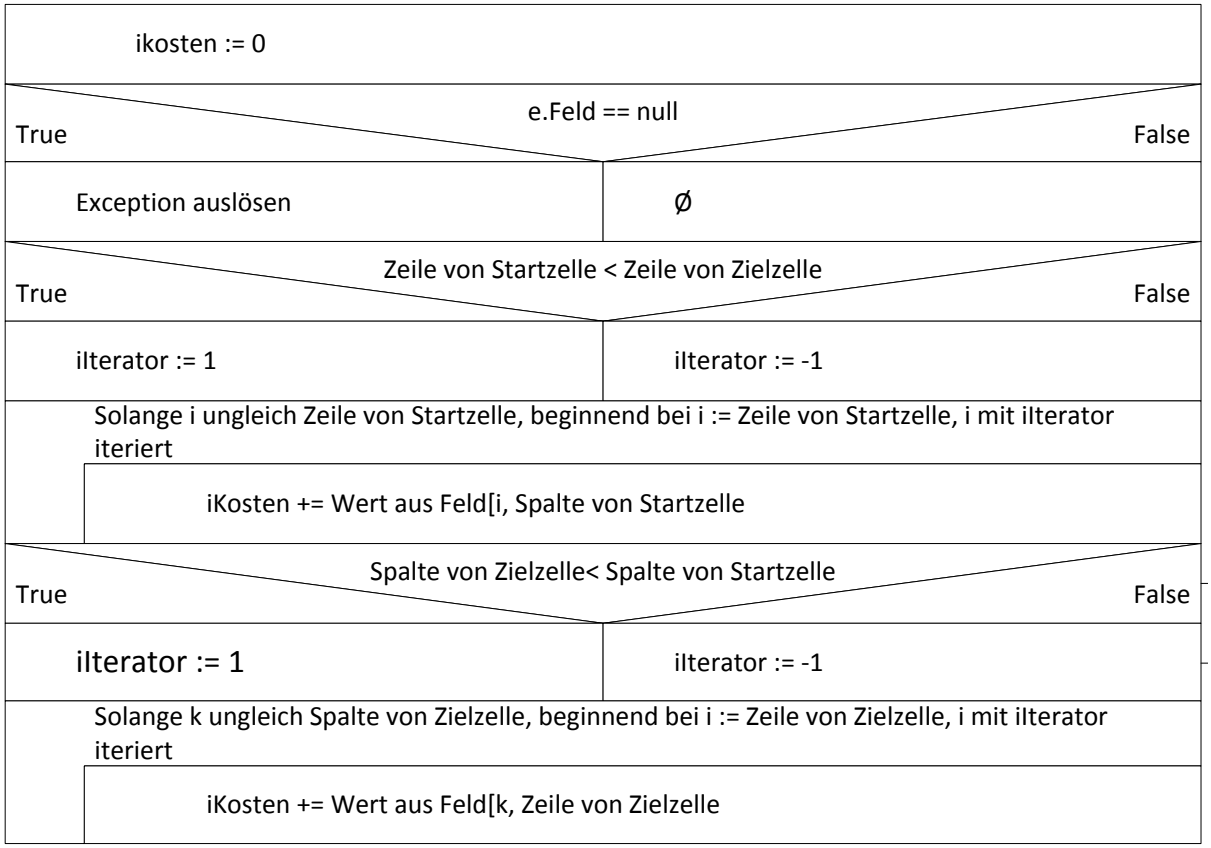


Abbildung 6 – die Methode KostenAbsch

KostenMinimal

Diese Methode errechnet nach dem in 4.2.2 beschriebenen Verfahren Wert und Weg der minimalen Kosten. Dazu werden zunächst einige Variablen und Felder initialisiert:

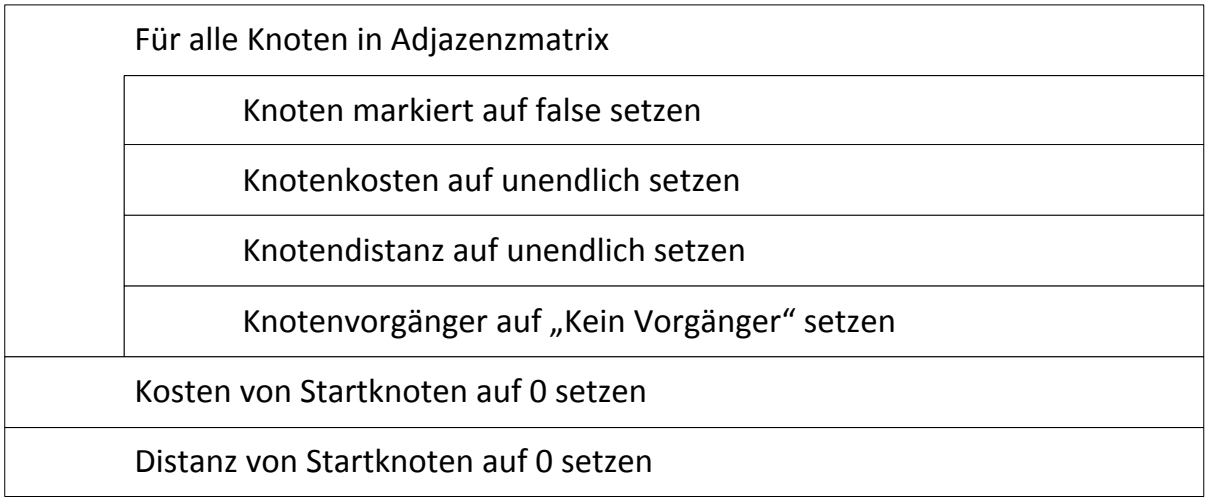


Abbildung 7 – Initialisierung der Methode KostenMinimal

Anschließend beginnt die Iteration des Verfahrens. Dazu wird zunächst das aktuelle Minimum bzw. der Knoten mit den derzeit geringsten Kosten, bestimmt (siehe Abbildung 8)

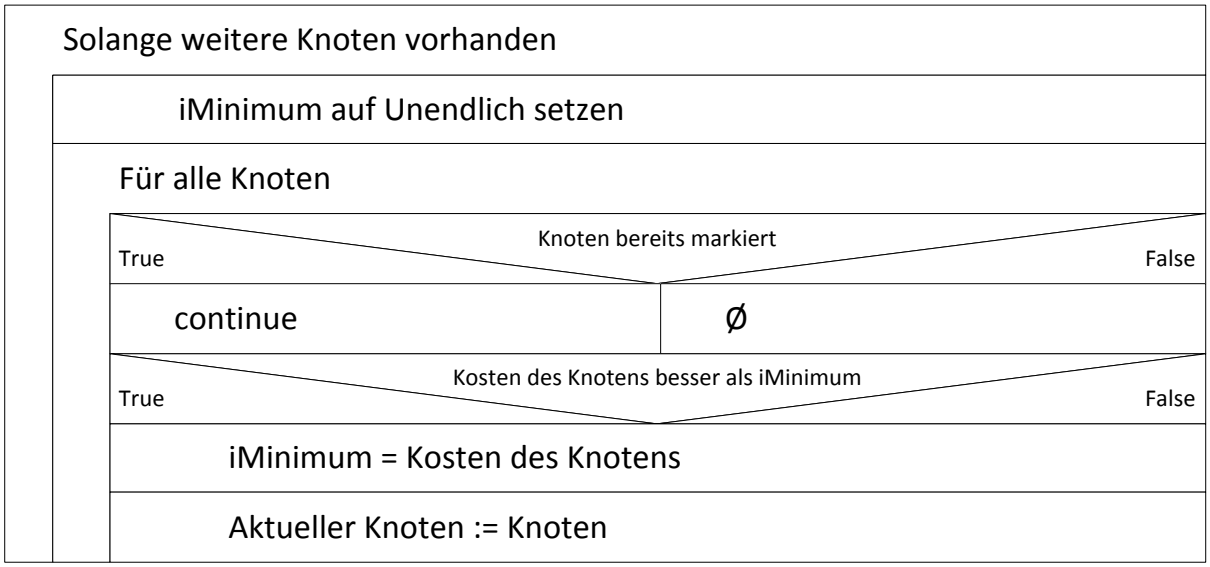


Abbildung 8 – Vorbereitung einer neuen Iteration in Methode KostenMinimal

Anschließend wird versucht, die Wege zu den noch nicht besuchten Knoten über den oben ausgewählten Knoten zu verbessern (siehe Abbildung 9).

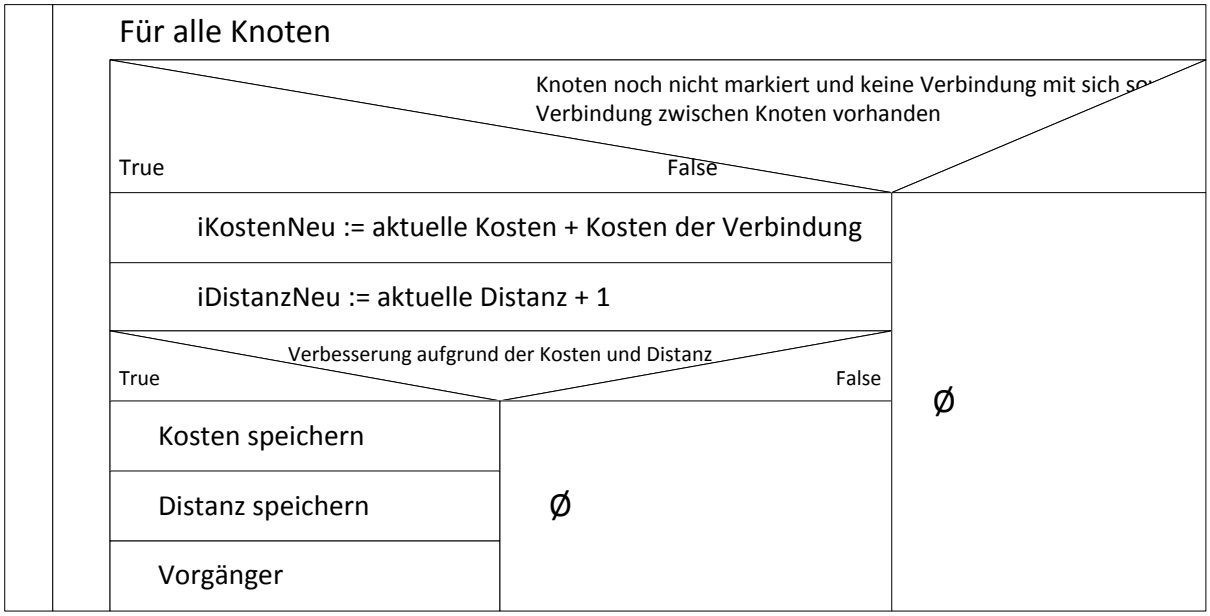


Abbildung 9 – der Verbesserungsschritt in der Methode KostenMinimal

Abschließend werden der errechnete kürzeste Weg und die dazugehörenden Minimalkosten gespeichert. Dazu wird in einer Rückwärtssuche der Weg bestimmt und die minimalen Kosten ausgelesen (siehe Abbildung 10)

	iKnoten := Zielknoten
	Solange Vorgänger von iKnoten vorhanden
	Füge Knoten zu Weg hinzu
	Setze iKnoten auf Vorgänger von Knoten
	Minimale Kosten := Kosten des Zielknotens

Abbildung 10 – Ergebnis Auswertung der KostenMinimal Methode

5.4. Abweichungen zum ursprünglichen Konzept

- Der Ansatz zur Berechnung der minimalen Kosten wurde um die Funktionalität der Distanzmessung erweitert, sodass auch der kürzeste Weg bei gleichen Kosten gefunden wird.
- Im ursprünglichen Konzept sollten Start- und Zielzelle als -1, bzw. -2 im Feld gespeichert werden. In der Implementierung wurden diese Werte mit 0 ersetzt, um Sonderbetrachtungen zu vermeiden.
- Die Ausgabe erfolgt in der Implementierung auch in eine Datei.
- Es wurden zur Klasse Ergebnis noch Methoden hinzugefügt, welche die Umrechnung von Koordinaten in Knoten der Adjazenzmatrix vereinfachen.

6. Benutzeranleitung

6.1. Systemvoraussetzungen

Betriebssystem: Windows 7 mit Service Pack 1

.Net Die Anwendung wurde für .Net Framework 4 kompiliert

6.2. Struktur der Programm CD

/dokumentation	Dokumentation.pdf :	Die Dokumentation in digitaler Form inkl. Benutzerhandbuch
	/api:	Die Entwicklerdokumentation als HTML Seite. Die Seite kann mit index.html geöffnet werden.
/programmcode	Der Quellcode des Programms	
/bin	Eine kompilierte Version des Programms, außerdem ein Skript zur Ausführung der Testbeispiele	
/testfaelle	Die unter 8.2 beschriebenen Testbeispiele	

6.3. Format der Eingabedatei

Als Eingabeformat wird gefordert:

1. Zeile: Kommentar, der das gewählte Beispiel beschreibt. Ein Kommentar beginnt mit einem „;“
2. Zeile: Kosteneinheiten einer Zellenzeile. Die Kosten der einzelnen Zellen sind jeweils durch ein Komma getrennt. Die Größe eines Feldes ergibt sich aus der Anzahl der Elemente in der ersten Zeile sowie der Anzahl der eingelesenen Zeilen.
3. Die Startzelle ist durch ein „S“, die Zielzelle durch ein „Z“ gekennzeichnet

6.4. Programmaufruf

Der Inhalt der CD muss in ein beliebiges Verzeichnis auf der Festplatte kopiert werden. Das Programm benötigt in diesem Verzeichnis Schreibrechte (ACHTUNG: bei Windows 7 ist im Ordner Programme Schreiben nur mit Administrator-Rechten möglich). Das im /bin Ordner der CD befindliche Programm (Wegplanung.exe) erwartet als Parameter einen Pfad zu einer Datei mit einzulesenden Daten. Wird kein Parameter angegeben, so wird automatisch die Datei gebiet.txt im Ordner „./Testfaelle“ des Programms eingelesen. Das Programm gibt die Ergebnisse der Berechnung auf dem Bildschirm und in einer Datei aus. Die Datei wird im gleichen Ordner der Eingabedatei erstellt und Eingabedatei_Ergebnisse.txt genannt. Das Programm benötigt daher auch in dem Ordner der Eingabedatei Schreibrechte.

Im /bin Ordner der CD befindet sich außerdem eine Batch Datei skript.bat. Dieses Skript führt das Programm mit allen Eingabedateien im Ordner „./Testfaelle“ aus.

7. Entwicklerdokumentation

Eine Entwicklerdokumentation ist auf der CD im Ordner /doc/api zu finden. Diese lässt sich durch die Datei index.html starten.

8. Testen und Qualitätssicherungsmaßnahmen

Nachfolgend werden die Testphasen des Programms beschrieben.

8.1. Testphasen

Die durchgeführten Tests gliedern sich in drei Phasen:

Schreibtischtest Einfach aufgebaute Eingabedaten wurde zunächst per Hand auf Papier durchgespielt und dann mit den Ergebnissen des Programms verglichen. Hierdurch wird der korrekte Ablauf des Algorithmus geprüft.

Black-Box-Test Die Klassen wurden anhand ihrer Schnittstellen und ohne Berücksichtigung ihrer Implementierung getestet. Dabei wurden neben zufälligen Normalfällen ebenfalls Sonder- und Grenzfälle betrachtet.

White-Box-Test Die Funktionalität des Programms wurde mit Kenntnis seiner Implementierung auf vorhandene Fehler getestet. Dazu wurden Fehler-, Sonder- und Normalfälle erstellt und durch Nachvollziehen des Programmablaufs getestet.

8.2. Testbeispiele

Die Testbeispiele lassen sich in zwei Gruppen einteilen:

Fehlerhafte Beispiele Die eingelesenen Eingabedateien weisen formale Fehler auf. Die Eingabefunktionen sollten diese Dateien als fehlerhaft erkennen und das Programm einen Fehler ausgeben.

Fehlerfreie Beispiele Die eingelesenen Eingabedateien sind formal korrekt aufgebaut und folgen dem in der Aufgabenstellung angegebenen Format. Es handelt sich bei den getesteten Eingaben um Normal-, Grenz und Sonderfälle.

8.2.1. Fehlerhafte Beispiele

Zunächst wurde getestet, ob das Programm falsche Eingaben erkennt. Dazu wurden die verschiedenen Dateien erstellt (siehe CD: Testfaelle/fehlerhaft01.txt bis Testfaelle/fehlerhaft15.txt).

Mehr als eine Startzelle definiert
Datei: fehlerhaft01.txt
Fehler: Fehler beim Einlesen der Datei: Es darf nur eine Startzelle definiert sein.

Mehr als eine Zielzelle definiert
Datei: fehlerhaft02.txt
Fehler: Fehler beim Einlesen der Datei: Es darf nur eine Zielzelle definiert sein.

Keine Startzelle definiert
Datei: fehlerhaft03.txt
Fehler: Fehler beim Einlesen der Datei: In der Datei muss eine Startzelle angegeben sein.

Keine Zielzelle definiert

Datei: fehlerhaft04.txt

Fehler: Fehler beim Einlesen der Datei: In der Datei muss eine Zielzelle angegeben sein.

Negative Zahlen in Feld

Datei: fehlerhaft06.txt

Fehler: Fehler beim Einlesen der Datei: Es sind nur Zahlenwerte zwischen 1 und 9 erlaubt.

Zahlen > 9 in Feld

Datei: fehlerhaft07.txt

Fehler: Fehler beim Einlesen der Datei: Es sind nur Zahlenwerte zwischen 1 und 9 erlaubt.

Mehr als 20 Zeilen in Feld

Datei: fehlerhaft08.txt

Fehler: Fehler beim Einlesen der Datei: Das Feld darf nicht mehr als 20 Zeilen haben.

Mehr als 20 Spalten in Feld

Datei: fehlerhaft09.txt

Fehler: Fehler beim Einlesen der Datei: Das Feld darf nicht mehr als 20 Spalten geben.

Unterschiedliche Zeilenbreite

Datei: fehlerhaft11.txt

Fehler: Fehler beim Einlesen der Datei: Unterschiedliche Spaltenanzahl in Datei gefunden**.**

Dezimalzahlen in Feld, unbekannte Zeichen in Feld

Datei: fehlerhaft12.txt, fehlerhaft13.txt, fehlerhaft14.txt

Fehler: Fehler beim Einlesen der Datei: Es sind nur ganzzahlige Zahlenwerte zwischen 1 und 9 und S, Z in den Zellen erlaubt.

8.2.2. Normalfälle

Zunächst wurden die in der Aufgabenstellung angegeben Beispiele getestet. Dazu wurden entsprechende Eingabedateien erstellt (siehe beispiel1.txt bis beispiel4.txt). Das Ergebnis der Berechnung wurde mit den Ergebnissen aus der Aufgabenstellung verglichen. Anschließend wurden weitere Normalfälle zunächst von Hand gerechnet und dann mit den Ergebnissen des Programms verglichen.

8.2.3. Sonderfälle

Als Sonderfälle wurden zunächst triviale Beispiele mit sehr wenigen Zellen getestet. Diese Fälle können sehr einfach, manuell, nachvollzogen werden. Diese Beispiele sind in den Dateien trivial1.txt bis trivial3.txt dokumentiert. Anschließend wurden Beispiele mit sehr großen Eingaben getestet bzw. die Normalfälle zu einem speziellen Fall abgewandelt.

Ziel liegt „oberhalb“ von Start

In den Beispielen der Aufgabenstellung liegt die Startzelle immer oberhalb der Zielzelle. In spezial1.txt wurden deshalb Start und Ziel im Vergleich zu Beispiel 1 vertauscht. Die Korrektheit

wurde dennoch, sowohl in der Berechnung der Abschätzung als auch bei den Minimalkosten festgestellt.

Gebiet mit 20 x 20 Zellen
Ein Gebiet mit 20 x 20 Zellen wurde getestet (spezial2.txt). Die Korrektheit konnte durch Vergleich mit einem Beispiel belegt werden.

Gebiet mit 20 x 20 Zellen, alle Wert 9
Ein Gebiet mit 20 x 20 Zellen ist nur mit Neunen und der Start- bzw. Zielzelle besetzt. Der Algorithmus findet hier den richtigen Wert.

Gebiet mit 20 x 20 Zellen, großer Weg
Ein Gebiet mit 20 x 20 Zellen zeigt, dass der Algorithmus bei gleichen Kosten auch den kürzesten Weg wählt. Dies kann auch durch beispiel4.txt gezeigt werden.

8.3. Bewertung des Tests

Durch die getesteten Beispiele kann die richtige Funktionsweise des Programms für die dargelegten Beispiele gezeigt werden. Durch sorgfältiges Debuggen konnten weitere Fehler gefunden werden. Für einen hinreichenden Test müsste das Programm noch von weiteren Testern (nicht nur vom Entwickler selber) überprüft werden.

9. Zusammenfassung und Ausblick

Diese Dokumentation hat die Umsetzung eines Programms zur Berechnung von Wegen auf einem Feld beschrieben. Das Programm kann in Zukunft noch durch eine benutzerfreundliche GUI erweitert werden, dabei kann auch die Ausgabe um eine graphische Darstellung des berechneten Weges erweitert werden.

Durch die Implementierung des Algorithmus auf einer Adjazenzmatrix kann das Programm sehr einfach für beliebige Graphen portiert werden. Es ist beispielsweise vorstellbar, das beim Betrieb des Programms festgestellt wird, dass bestimmte Wege auf dem Feld nicht möglich sind. Zur Änderung des Programms muss dann lediglich die Eingabemethode umgewandelt werden.

10. Literaturverzeichnis

[1] H.-J. Zimmermann, Operations Research Methoden und Modelle, Vieweg & Sohn Verlagsgesellschaft mbH, 1987.

Anhang A - Aufgabenstellung

„The middle of nowhere“ ist ein Land in Form eines Rechteckes, das ganz dünn besiedelt ist und hinsichtlich der Infrastruktur nur sehr schwach entwickelt ist. Die Regierung beschließt nun, zwei der im Land existierenden Dörfer durch einen Weg miteinander zu verbinden, um Handel zu ermöglichen. Das Land ist auch bereits vermessen. Es wurde in rechteckige, gleichgroße Teile sowohl in Horizontal- als auch in Vertikalrichtung eingeteilt. Jedem Teil ist so ein Index (x,y) zugewiesen. Die erste Zelle (links oben) hat den Index (1,1) erhalten, die Zelle rechts unten den Index (20,20). Die Oberflächenbeschaffenheit und die von ihr abhängigen Kosten für die Bebauung einer Zelle mit einer Straße sind bekannt. Die Regierung möchte bei der Erstellung des Verbindungsweges sparen und hat die Aufgabe der Ermittlung der kostengünstigsten Verbindung ausgeschrieben. Ihre Firma, die MATSE-Kalkül GmbH, möchte sich um den Berechnungsauftrag bewerben und hat Sie mit der Erstellung einer Lösung beauftragt.

In der Ausschreibung finden Sie die folgende etwas allgemeinere Aufgabenbeschreibung, mit der die Regierung erreichen will, dass die Lösung auch für beliebige rechteckige Gebiete nutzbar ist:

Ein rechteckiges, sehr heterogenes Gebiet (Wald, Seen, etc.) ist in ein Raster von (n x m) Zellen aufgeteilt worden. Es sollen maximal 20 x 20 Zellen sein. Es sollen nun Wege zwischen zwei Zellen innerhalb dieses Gebietes gebaut werden (z. B.: Die Startzelle (1,2) soll mit der Zielzelle (7,8) verbunden werden). Nun sind die Kosten für die Bebauung der Zellen unterschiedlich hoch. So ist z. B. der Bau einer Brücke über einen See teurer als das Roden von Bäumen in einem Waldstück. Die mittleren angenommenen Kosten für jede Zelle werden jeder Zelle zugeordnet. Es werden ganzzahlige Kosten zwischen 1 Kosteneinheit (KE) „leichter Bau“ und 9 KE „hohe Baukosten“ vereinbart und zugewiesen. Es wird nun ein Weg von der Startzelle zur Zielzelle mit minimierten Kosten gesucht. Hierzu soll vorab zum Vergleich eine mögliche Kostenobergrenze bestimmt werden. Danach soll die kostenminimale Lösung ermittelt werden.

Es sind immer nur horizontale oder vertikale Verbindungen in die nächste Zelle möglich. Sollte es mehrere Wege geben, welche die gleichen minimalen Kosten erfordern, ist die kürzeste Lösung zu wählen.

Als Eingabeformat wird gefordert:

1. Zeile: Kommentar, der das gewählte Beispiel beschreibt. Ein Kommentar beginnt mit einem „;“
2. Zeile: Kosteneinheiten einer Zellenzeile. Die Kosten der einzelnen Zellen sind jeweils durch ein Komma getrennt.
   Die Größe eines Feldes ergibt sich aus der Anzahl der Elemente in der ersten Zeile sowie der Anzahl der eingelesenen Zeilen.
3. Die Startzelle ist durch ein „S“, die Zielzelle durch ein „Z“ gekennzeichnet

Als Ausgabeformat wird gefordert

1. Zeile: Kommentar aus der Eingabe
2. Zeile: Anzahl der Zellen, Indizes der Startzelle und der Zielzelle
3. Zeile: Abschätzung der Kostenobergrenze
4. Zeile: Ermittelte Minimalkosten

ab 5. Zeile: Weg, der zu den Minimalkosten führt

Zele, Spalte

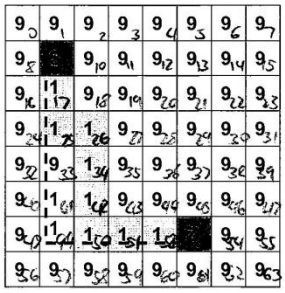
Beispiele:

Beispiel 1 mit 8 x 8 Zellen

Eingabeformat:

```
;Sumpfgebiet mit 8 x 8 Zellen
9,9,9,9,9,9,9,9
9,S,9,9,9,9,9,9
9,1,9,9,9,9,9,9
9,1,1,9,9,9,9,9
9,9,1,9,9,9,9,9
9,1,1,9,9,9,9,9
9,1,1,1,1,Z,9,9
9,9,9,9,9,9,9,9
```

Dies entspricht dem folgenden Gebiet mit der eingezeichneten Lösung:



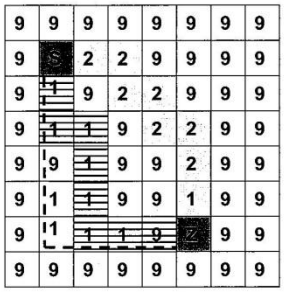
Die Zellen, die zur Bildung der Kostenobergrenze genommen wurden, sind hier durch die gestrichelte Strecke gekennzeichnet.

Ausgabe:
Gebiet mit 8 x 8 Zellen
Startzelle: 2,2, Zielzelle: 7,6
Abschätzung der Kostenobergrenze: 16 KE
Minimalkosten: 8 KE
Weg: S; 3,2; 4,2; 4,3; 5,3; 6,3; 7,3; 7,4; 7,5; Z

Beispiel 2 mit 8 x 8 Zellen

Eingabeformat:

```
; Gebiet mit 8 x 8 Zellen
9,9,9,9,9,9,9,9
9,S,2,2,9,9,9,9
9,1,9,2,2,9,9,9
9,1,1,9,2,2,9,9
9,9,1,9,9,2,9,9
9,1,1,9,9,1,9,9
9,1,1,1,1,Z,9,9
9,9,9,9,9,9,9,9
```



Startzelle ist die 2,2, Zielzelle die 7,6

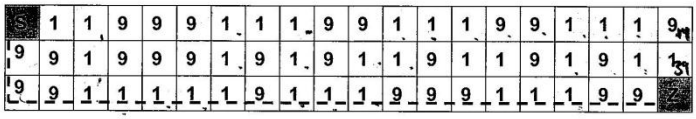
Der Weg mit den geringsten Kosten ist dunkel hinterlegt. Die Kosten betragen 15 KE. Der schraffierte Weg hat Kosten von 16 KE und ist zu verwerfen.

Ausgabe:
Gebiet mit 8 x 8 Zellen
Startzelle: 2,2, Zielzelle: 7,6
Abschätzung der Kostenobergrenze: 24 KE
Minimalkosten: 15 KE
Weg: S; 2,3; 2,4; 3,4 ; 3,5; 4,5; 4,6; 5,6; 6,6; Z

Beispiel 3 mit 20 x 3 Zellen

Eingabeformat:

```
;Gebiet mit 20 x 3 Zellen
S,1,1,9,9,9,1,1,1,9,9,1,1,1,9,9,1,1,1,9
9,9,1,9,9,9,1,9,1,9,1,1,9,1,1,9,1,9,1,1
9,9,1,1,1,1,1,9,1,1,1,9,9,9,1,1,1,9,9,Z
```



Startzelle ist die 1,1, Zielzelle die 20,3

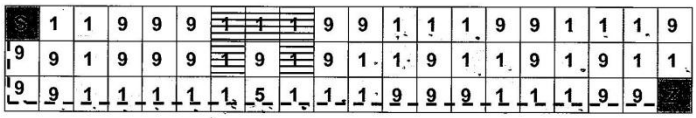
Der Weg mit den geringsten Kosten ist dunkel hinterlegt.

Ausgabe:
Gebiet mit 20 x 3 Zellen
Startzelle: 1,1, Zielzelle: 20,3
Abschätzung der Kostenobergrenze: 92 KE
Minimalkosten: 32 KE
Weg: S; 2,1; 3,1; 3,2 ; 3,3; 4,3; 5,3; 6,3; 7,3; 7,2; 7,1; 8,1; 9,1; 9,2; 9,3; 10,3; 11,3; 11,2; 12,2; 12,1; 13,1; 14,1; 14,2; 15,2; 15,3; 16,3; 17,3; 17,2; 17,1; 18,1; 19,1; 19,2; 20,2; Z

Beispiel 4 mit 20 x 3 Zellen mit Abkürzung

Eingabeformat:

```
;Gebiet mit 20 x 3 Zellen und einer Abkürzung
S,1,1,9,9,9,1,1,1,9,9,1,1,1,9,9,1,1,1,9
9,9,1,9,9,9,1,9,1,9,1,1,9,1,1,9,1,9,1,1
9,9,1,1,1,1,1,5,1,1,1,9,9,9,1,1,1,9,9,Z
```



Startzelle ist die 1,1, Zielzelle die 20,3

Der Weg mit den geringsten Kosten ist dunkel hinterlegt. Der schraffierte Weg hat die gleichen Kosten wie die Abkürzung über 8,3 ist aber länger und wird daher nicht genommen.

Ausgabe:
Gebiet mit 20 x 3 Zellen
Startzelle: 1,1, Zielzelle: 20,3
Abschätzung der Kostenobergrenze: 88 KE
Minimalkosten: 32 KE
Weg: S; 2,1; 3,1; 3,2 ; 3,3; 4,3; 5,3; 6,3; 7,3; 8,3; 9,3; 10,3; 11,3; 11,2; 12,2; 12,1; 13,1; 14,1; 14,2; 15,2; 15,3; 16,3; 17,3; 17,2; 17,1; 18,1; 19,1; 19,2; 20,2; Z

Aufgabenstellung

Abschätzen einer Kostenobergrenze
Ermitteln Sie für eine gegebene Start- und Zielzelle eine realistische Obergrenze für die zu erwartenden Baukosten und begründen Sie diese.

Verbesserung
Entwickeln Sie einen Algorithmus (Programm) zur Minimierung der Baukosten.

Hinweis zu den Beispielen und dem Programm
Die grafische Aufarbeitung der Daten ist nicht Bestandteil der Simulation.

Im Rahmen der schriftlichen Aufgabe sind am ersten Tag abzugeben:
- Aufgabenanalyse und verbale Verfahrensbeschreibung (Algorithmus)
- Programmkonzeption unter Berücksichtigung der funktionalen Trennung
  - Klassen, Methoden und Datenstrukturen in Form von UML-Diagrammen
  - UML-Sequenzdiagramme für die wesentlichen Abläufe
  - Detaillierte Beschreibung der wesentlichen Methoden in Form von Nassi-Shneiderman-Diagrammen

Im Rahmen des Prüfprodukts sind abzugeben:
- Verbale Beschreibung des realisierten Verfahrens.
- Programmsystem (bestehend aus Klassen, Methoden) als Quellcode und in ausführbarer Form
- Quellcode gedruckt und als Datei
- Entwicklerdokumentation
- Benutzeranleitung
- Beschreibung, Begründung und Diskussion
  - der angegebenen Beispiele und
  - einer ausreichenden Zahl von Beispielen, die sowohl die Normalfälle als auch auftretende Spezial- und Fehlerfälle abdecken

Ein- und Ausgabe der Beispiele gedruckt und als Dateien, bei mehr als zwei Seiten Output nur in elektronischer Form
- Ein Skript zur automatischen Ausführung aller von Ihnen angegebenen Beispiele in elektronischer Form
- Zusammenfassung und Ausblick (z. B. Erweiterungsmöglichkeiten)
- Juristische Erklärung (Eigenhändigkeitserklärung)

Im Rahmen des auftragsbezogenen Fachgesprächs sind die Aufgabenanalyse und der Lösungsentwurf zu begründen und das Prüfungsprodukt zu erläutern.

Anhang B – Testfälle

Beispiele:

beispiel1.txt

```
; Sumpfgebiet
9,9,9,9,9,9,9,9
9,S,9,9,9,9,9,9
9,1,9,9,9,9,9,9
9,1,1,9,9,9,9,9
9,9,1,9,9,9,9,9
9,1,1,9,9,9,9,9
9,1,1,1,1,Z,9,9
9,9,9,9,9,9,9,9
```

beispiel1_Ergebnis.txt

```
Sumpfgebiet
Startzelle: 2, 2, Zielzelle: 7, 6
Abschätzung der Kostenobergrenze: 16 KE
Minimalkosten: 8 KE
Weg: S; 3,2; 4,2; 4,3; 5,3; 6,3; 7,3; 7,4; 7,5; Z
```

beispiel2.txt

```
; Gebiet mit 8 x 8 Zellen
9,9,9,9,9,9,9,9
9,S,2,2,9,9,9,9
9,1,9,2,2,9,9,9
9,1,1,9,2,2,9,9
9,9,1,9,9,2,9,9
9,1,1,9,9,1,9,9
9,1,1,1,9,Z,9,9
9,9,9,9,9,9,9,9
```

beispiel2_Ergebnis.txt

```
Gebiet mit 8 x 8 Zellen
Startzelle: 2, 2, Zielzelle: 7, 6
Abschätzung der Kostenobergrenze: 24 KE
Minimalkosten: 15 KE
Weg: S; 2,3; 2,4; 3,4; 3,5; 4,5; 4,6; 5,6; 6,6; Z
```

beispiel3.txt

```
; Sumpfgebiet
S,1,1,9,9,9,1,1,1,9,9,1,1,1,9,9,1,1,1,9
9,9,1,9,9,9,1,9,1,9,1,1,9,1,1,9,1,9,1,1
9,9,1,1,1,1,1,9,1,1,1,9,9,9,1,1,1,9,9,Z
```

beispiel3_Ergebnis.txt

```
Sumpfgebiet
Startzelle: 1, 1, Zielzelle: 3, 20
Abschätzung der Kostenobergrenze: 92 KE
Minimalkosten: 32 KE
Weg: S; 1,2; 1,3; 2,3; 3,3; 3,4; 3,5; 3,6; 3,7; 2,7; 1,7; 1,8; 1,9;
2,9; 3,9; 3,10; 3,11; 2,11; 2,12; 1,12; 1,13; 1,14; 2,14; 2,15;
3,15; 3,16; 3,17; 2,17; 1,17; 1,18; 1,19; 2,19; 2,20; Z
```

beispiel4.txt

```
;Sumpfgebiet
S,1,1,9,9,9,1,1,1,9,9,1,1,1,9,9,1,1,1,9
9,9,1,9,9,9,1,9,1,9,1,1,9,1,1,9,1,9,1,1
9,9,1,1,1,1,1,5,1,1,1,9,9,9,1,1,1,9,9,Z
```

beispiel4_Ergebnis.txt

```
Sumpfgebiet
Startzelle: 1, 1, Zielzelle: 3, 20
Abschätzung der Kostenobergrenze: 88 KE
Minimalkosten: 32 KE
Weg: S; 1,2; 1,3; 2,3; 3,3; 3,4; 3,5; 3,6; 3,7; 3,8; 3,9; 3,10;
3,11; 2,11; 2,12; 1,12; 1,13; 1,14; 2,14; 2,15; 3,15; 3,16; 3,17;
2,17; 1,17; 1,18; 1,19; 2,19; 2,20; Z
```

fehlerhaft01.txt

```
; Sumpfgebiet
9,9,9,9,9,9,9,9
9,S,9,9,9,9,9,9
9,1,9,9,9,9,9,9
9,1,1,9,9,9,9,9
9,9,1,9,9,9,9,9
9,1,1,9,9,9,9,9
9,1,1,1,1,S,9,9
9,9,9,9,9,9,9,9
```

fehlerhaft01_Ergebnis.txt

```
Fehler beim Einlesen der Datei: Es darf nur eine Startzelle
definiert sein.
```

fehlerhaft02.txt

```
; Sumpfgebiet
9,9,9,9,9,9,9,9
9,Z,9,9,9,9,9,9
9,1,9,9,9,9,9,9
9,1,1,9,9,9,9,9
9,9,1,9,9,9,9,9
9,1,1,9,9,9,9,9
9,1,1,1,1,Z,9,9
9,9,9,9,9,9,9,9
```

fehlerhaft02_Ergebnis.txt

```
Fehler beim Einlesen der Datei: Es darf nur eine Zielzelle definiert
sein.
```

fehlerhaft03.txt

```
;keine Startzelle
9,9,9,9,9,9,9,9
9,9,9,9,9,9,9,9
9,1,9,9,9,9,9,9
9,1,1,9,9,9,9,9
9,9,1,9,9,9,9,9
9,1,1,9,9,9,9,9
9,1,1,1,1,Z,9,9
9,9,9,9,9,9,9,9
```

fehlerhaft03_Ergebnis.txt

```
Fehler beim Einlesen der Datei: In der Datei muss eine Startzelle
angegeben sein.
```

fehlerhaft04.txt

```
;keine Zielzelle
9,9,9,9,9,9,9,9
9,S,9,9,9,9,9,9
9,1,9,9,9,9,9,9
9,1,1,9,9,9,9,9
9,9,1,9,9,9,9,9
9,1,1,9,9,9,9,9
9,1,1,1,1,9,9,9
9,9,9,9,9,9,9,9
```

fehlerhaft04_Ergebnis.txt

```
Fehler beim Einlesen der Datei: In der Datei muss eine Zielzelle
angegeben sein.
```

fehlerhaft05.txt

```
;keine Zielzelle, keine Startzelle
9,9,9,9,9,9,9,9
9,4,9,9,9,9,9,9
9,1,9,9,9,9,9,9
9,1,1,9,9,9,9,9
9,9,1,9,9,9,9,9
9,1,1,9,9,9,9,9
9,1,1,1,1,9,9,9
9,9,9,9,9,9,9,9
```

fehlerhaft05_Ergebnis.txt

```
Fehler beim Einlesen der Datei: In der Datei muss eine Startzelle
angegeben sein.
```

fehlerhaft06.txt

```
;Fehlerhafte Werte in Zelle (-1)
9,9,9,9,9,9,9,9
9,S,9,9,9,9,9,9
9,1,9,9,9,9,9,9
9,1,1,9,9,9,9,9
9,9,1,9,9,9,9,9
9,1,1,9,-1,9,9,9
9,1,1,1,1,Z,9,9
9,9,9,9,9,9,9,9
```

fehlerhaft06_Ergebnis.txt

```
Fehler beim Einlesen der Datei: Es sind nur Zahlenwerte zwischen 1
und 9 erlaubt.
```

fehlerhaft07.txt

```
;Fehlerhafte Werte in Zelle (10)
9,9,9,9,9,9,9,9
9,S,9,9,9,9,9,9
9,1,9,9,9,9,9,9
9,1,1,9,9,9,9,9
9,9,1,9,9,9,9,9
9,1,1,9,10,9,9,9
9,1,1,1,1,Z,9,9
9,9,9,9,9,9,9,9
```

fehlerhaft07_Ergebnis.txt

```
Fehler beim Einlesen der Datei: Es sind nur Zahlenwerte zwischen 1
und 9 erlaubt.
```

fehlerhaft08.txt

```
;mehr als 20 Zeilen
9,9,9,9,9,9,9,9
9,S,9,9,9,9,9,9
9,1,9,9,9,9,9,9
9,1,1,9,9,9,9,9
9,9,1,9,9,9,9,9
9,1,1,9,9,9,9,9
9,1,1,1,1,Z,9,9
9,9,9,9,9,9,9,9
9,9,9,9,9,9,9,9
9,6,9,9,9,9,9,9
9,1,9,9,9,9,9,9
9,1,1,9,9,9,9,9
9,9,1,9,9,9,9,9
9,1,1,9,9,9,9,9
9,1,1,1,1,1,9,9
9,9,9,9,9,9,9,9
9,1,1,1,1,1,9,9
9,9,9,9,9,9,9,9
9,1,1,1,1,1,9,9
9,9,9,9,9,9,9,9
9,1,1,1,1,1,9,9
9,9,9,9,9,9,9,9
```

fehlerhaft08_Ergebnis.txt

```
Fehler beim Einlesen der Datei: Das Feld darf nicht mehr als 20
Zeilen haben.
```

fehlerhaft09.txt

```
;mehr als 20 Spalten
9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9
9,S,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9
9,1,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9
9,1,1,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9
9,9,1,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9
9,1,1,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9
9,1,1,1,1,Z,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9
9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9
9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9
9,6,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9
9,1,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9
9,1,1,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9
9,9,1,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9
9,1,1,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9
9,1,1,1,1,1,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9
9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9
9,1,1,1,1,1,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9
9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9
```

fehlerhaft09_Ergebnis.txt

```
Fehler beim Einlesen der Datei: Es darf nicht mehr als 20 Spalten
geben.
```

fehlerhaft10.txt

```
;mehr als 20 Zeilen, mehr als 20 Spalten
9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9
9,S,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9
9,1,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9
9,1,1,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9
9,9,1,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9
9,1,1,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9
9,1,1,1,1,Z,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9
9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9
9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9
9,6,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9
9,1,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9
9,1,1,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9
9,9,1,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9
9,1,1,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9
9,1,1,1,1,1,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9
9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9
9,1,1,1,1,1,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9
9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9
9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9
9,1,1,1,1,1,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9
9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9
9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9
9,1,1,1,1,1,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9
```

fehlerhaft10_Ergebnis.txt

```
Fehler beim Einlesen der Datei: Es darf nicht mehr als 20 Spalten
geben.
```

fehlerhaft11.txt

```
;Unterschiedliche Zeilenlänge
9,9,9,9,9,9,9
9,S,9,9,9,9,9,9,9,9
9,1,9,9,9,9,9,9
9,1,1,9,9,9,9,9,9,9,9
9,9,1,9,9,9,9,9,9,9
9,1,1,9,9,9,9,9,9,9,9
9,1,1,1,1,Z,9,9,9,9,9,9,9,9,9,9,9
```

fehlerhaft11_Ergebnis.txt

```
Fehler beim Einlesen der Datei: Unterschiedliche Spaltenanzahl in
Datei gefunden.
```

fehlehrhaft12.txt

```
;Dezimalwerte
9,9,9,9,9,9,9
9,S,9,9,9,9,9
9,1,9,9.4,9,9,9
9,1,1,9,9,9,9
9,9,1,9,9,9,9
9,1,1,9,9,9,9
9,1,1,1,1,Z,9
```

fehlerhaft12_Ergebnis.txt

```
Fehler beim Einlesen der Datei: Es sind nur ganzzahlige Zahlenwerte
zwiwschen 1 und 9 und S,Z in den Zellen erlaubt.
```

fehlerhaft13.txt

```
;Zahlenwert doppelt in Zelle
9,9,9,9,9,9,9
9,S,9,9,9,9,9
9,1,9,9 8,9,9,9
9,1,1,9,9,9,9
9,9,1,9,9,9,9
9,1,1,9,9,9,9
9,1,1,1,1,Z,9
```

fehlerhaft13_Ergebnis.txt

```
Fehler beim Einlesen der Datei: Es sind nur ganzzahlige Zahlenwerte
zwischen 1 und 9 und S,Z in den Zellen erlaubt.
```

fehlerhaft14.txt

```
;nicht definiertes Zeichen in Zelle
9,9,9,9,9,9,9
9,S,9,9,9,9,9
9,1,9,X,9,9,9
9,1,1,9,9,9,9
9,9,1,9,9,9,9
9,1,1,9,9,9,9
9,1,1,1,1,Z,9
```

fehlerhaft14_Ergebnis.txt

```
Fehler beim Einlesen der Datei: Es sind nur ganzzahlige Zahlenwerte
zwischen 1 und 9 und S,Z in den Zellen erlaubt.
```

spezial1.txt

```
;Beispiel 1: s.z vertauscht
9,9,9,9,9,9,9,9
9,Z,9,9,9,9,9,9
9,1,9,9,9,9,9,9
9,1,1,9,9,9,9,9
9,9,1,9,9,9,9,9
9,1,1,9,9,9,9,9
9,1,1,1,1,S,9,9
9,9,9,9,9,9,9,9
```

spezial1_Ergebnis.txt

```
Beispiel 1: s.z vertauscht
Startzelle: 7, 6, Zielzelle: 2, 2
Abschätzung der Kostenobergrenze: 72 KE
Minimalkosten: 8 KE
Weg: S; 7,5; 7,4; 7,3; 6,3; 5,3; 4,3; 4,2; 3,2; Z
```

spezial2.txt

```
;Gebiet mit 20 x 20 Zellen
7,1,9,1,3,2,9,6,5,6,3,1,2,1,9,7,7,6,9,3
3,4,9,3,7,6,1,6,1,4,7,3,9,Z,4,7,2,4,1,5
6,3,6,8,6,2,4,3,8,2,6,6,9,2,7,8,2,1,5,1
8,6,5,6,7,4,4,8,4,8,8,6,8,8,9,7,7,4,8,3
6,8,7,4,4,5,6,3,4,1,5,5,1,7,1,9,8,6,2,1
9,6,5,6,5,6,3,2,9,8,8,2,5,7,9,9,1,1,4,2
7,2,7,5,5,4,1,7,8,9,4,9,6,1,2,5,9,8,1,7
9,4,8,6,8,6,7,6,3,3,9,3,5,3,4,7,3,9,9,9
7,7,1,9,2,3,2,5,2,1,7,2,4,2,4,7,8,8,5,3
9,7,9,9,5,4,7,3,6,7,1,8,8,9,5,6,2,3,1,1
6,8,7,3,9,9,3,2,8,6,9,8,3,3,9,9,5,4,3,6
1,2,7,8,8,1,1,9,9,5,2,5,6,6,7,2,8,6,4,4
6,2,8,9,1,3,4,8,1,2,6,3,9,4,5,5,5,5,6,2
9,4,1,8,2,3,4,7,9,5,2,8,4,7,8,4,1,9,8,2
1,7,4,6,9,6,3,7,9,4,5,2,2,8,7,4,4,3,8,1
2,S,8,1,1,5,5,9,4,9,7,8,1,2,3,5,2,5,4,8
9,4,9,3,8,5,7,4,7,4,5,8,9,2,2,8,6,4,4,9
6,3,1,3,3,9,1,1,9,2,3,5,7,6,3,4,4,7,4,2
6,8,2,5,7,4,4,9,3,9,2,7,5,7,3,6,2,6,4,1
1,2,9,6,7,6,7,2,6,2,9,9,1,2,5,8,7,7,2,3
```

spezial2_Ergebnis.txt

```
Gebiet mit 20 x 20 Zellen
Startzelle: 16, 2, Zielzelle: 2, 14
Abschätzung der Kostenobergrenze: 126 KE
Minimalkosten: 86 KE
Weg: S; 16,3; 16,4; 16,5; 15,5; 14,5; 13,5; 13,6; 12,6; 12,7; 11,7;
11,8; 10,8; 9,8; 9,9; 9,10; 9,11; 9,12; 9,13; 9,14; 8,14; 7,14;
6,14; 5,14; 4,14; 3,14; Z
```

spezial3.txt

```
;Gebiet mit 20 x 20 Zellen, alle gleicher Wert
S,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9
9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9
9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9
9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9
9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9
9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9
9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9
9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9
9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9
9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9
9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9
9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9
9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9
9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9
9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9
9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9
9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9
9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9
9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9
9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,Z
```

spezial3_Ergebnis.txt

```
Gebiet mit 20 x 20 Zellen, alle gleicher Wert
Startzelle: 1, 1, Zielzelle: 20, 20
Abschätzung der Kostenobergrenze: 333 KE
Minimalkosten: 333 KE
Weg: S; 1,2; 1,3; 1,4; 1,5; 1,6; 1,7; 1,8; 1,9; 1,10; 1,11; 1,12;
1,13; 1,14; 1,15; 1,16; 1,17; 1,18; 1,19; 1,20; 2,20; 3,20; 4,20;
5,20; 6,20; 7,20; 8,20; 9,20; 10,20; 11,20; 12,20; 13,20; 14,20;
15,20; 16,20; 17,20; 18,20; 19,20; Z
```

spezial4.txt

```
;Gebiet mit 20 x 20 Zellen, großer Weg
S,9,9,9,9,9,1,1,1,1,1,9,9,9,9,9,9,9,9,9
1,9,9,9,9,9,1,1,9,9,1,9,9,9,9,9,9,9,9,9
1,9,9,9,9,9,9,1,9,9,1,9,9,9,9,9,9,9,9,9
1,9,9,9,9,9,9,1,9,9,1,9,9,9,9,9,1,9,9,9
1,9,9,9,1,1,1,1,9,9,1,9,9,9,9,9,1,9,9,9
1,9,9,9,1,9,9,9,9,9,1,9,9,9,9,9,1,9,9,9
1,9,9,9,1,9,9,9,9,9,1,9,9,9,9,9,1,1,1,1
1,9,9,9,1,1,9,9,9,9,1,9,9,9,9,9,1,9,9,1
1,9,9,9,9,1,9,9,9,9,1,1,9,9,9,9,1,9,9,1
1,9,9,9,9,1,9,9,9,9,9,1,9,9,9,9,1,9,9,1
1,9,9,9,9,1,9,9,9,9,9,1,9,9,9,9,1,9,9,1
1,1,1,9,9,1,9,9,9,9,1,1,9,9,9,9,1,9,9,1
9,9,1,9,9,1,9,9,9,9,1,9,9,9,9,9,1,9,9,1
9,9,1,9,9,1,9,9,9,9,1,9,9,9,9,9,1,9,9,1
9,9,1,9,9,1,9,9,9,9,1,9,9,9,9,9,1,9,9,1
9,9,1,9,9,1,9,9,9,9,1,1,1,1,9,9,1,9,9,1
9,9,1,9,9,1,9,9,9,9,9,9,9,1,9,9,1,9,9,1
9,9,1,1,1,1,9,9,9,9,9,9,9,1,9,9,1,9,9,1
9,9,9,9,9,9,9,9,9,9,9,9,9,1,1,1,1,9,9,1
9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,Z
```

spezial4_Ergebnis.txt

```
Gebiet mit 20 x 20 Zellen, großer Weg
Startzelle: 1, 1, Zielzelle: 20, 20
Abschätzung der Kostenobergrenze: 245 KE
Minimalkosten: 87 KE
Weg: S; 2,1; 3,1; 4,1; 5,1; 5,2; 5,3; 5,4; 5,5; 5,6; 5,7; 5,8; 4,8;
3,8; 2,8; 1,8; 1,9; 1,10; 1,11; 2,11; 3,11; 4,11; 5,11; 6,11; 7,11;
8,11; 9,11; 9,12; 10,12; 11,12; 12,12; 12,11; 13,11; 14,11; 15,11;
16,11; 16,12; 16,13; 16,14; 17,14; 18,14; 19,14; 19,15; 19,16;
19,17; 19,18; 19,19; 19,20; Z
```

trivial1.txt

```
;2x2 Feld
S,5
5,Z
```

trivial1_Ergebnis.txt

```
2x2 Feld
Startzelle: 1, 1, Zielzelle: 2, 2
Abschätzung der Kostenobergrenze: 5 KE
Minimalkosten: 5 KE
Weg: S; 1,2; Z
```

trivial2.txt

```
;1x2 Feld
S,Z
```

trivial2_Ergebnis.txt

```
1x2 Feld
Startzelle: 1, 1, Zielzelle: 1, 2
Abschätzung der Kostenobergrenze: 0 KE
Minimalkosten: 0 KE
Weg: S; Z
```

trivial3.txt

```
;1x2 Feld
S
Z
```

trivial3_Ergebnis.txt

```
1x2 Feld
Startzelle: 1, 1, Zielzelle: 2, 1
Abschätzung der Kostenobergrenze: 0 KE
Minimalkosten: 0 KE
Weg: S; Z
```

Anhang C - Programmcode

Eingabe.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.IO;

namespace Wegplanung
{
   /// <summary>
   /// Klasse zur Eingabe und Umwandlung von Daten.
   /// </summary>
   /// <remarks>
   /// Historie:  22.05.2012  (awi) Erstellt
   /// </remarks>
   class Eingabe
   {

      /// <summary>
      /// Liest eine Datei ein ein und speichert in Ergebnis Objekt
      /// </summary>
      /// <remarks>
      /// Hinweise:  Löst verschiedene Ausnahmen aus, wenn Datei nicht geladen werden kann oder Datei
      ///            fehlerhaft
      ///            S, Z werden als 0 im Feld markiert
      /// Historie:  22.05.2012  (awi) Erstellt
      /// </remarks>
      /// <param name="sDateiname">Ein String mit einem Dateinamen</param>
      /// <param name="e">Ein Ergebnis Objekt</param>
      public void DateiEinlesen(String sDateiname, Ergebnis e)
      {
         bool bStartgefunden = false;
         bool bZielgefunden = false;
         int iSpalten = 0;
         int iZeile = 0;
         String sTmp, sZelle, sTmpAusn;
         String[] sTmpArr;
         int iTmp;

         StreamReader sr;

         int[,] tmpFeld = new int[Konstante.MAX_ZEILE, Konstante.MAX_SPALTE];

         if (File.Exists(sDateiname))
         {
            // Einlesen mit ISO-8859-1 (Westeuropäisch), wird wegen Umlauten in Kommentar verwendet
            sr = new StreamReader(File.Open(sDateiname, FileMode.Open), Encoding.GetEncoding(28591));
         }
         else
         {
            throw new FileNotFoundException("Die angegebene Datei wurde nicht gefunden");
         }

         if (sr.Peek() == -1)
         {
            throw new FileLoadException("Die angegebene Datei ist leer");
         }

         // erste Zeile lesen, erwartet Kommentar
         sTmp = sr.ReadLine();
         if (sTmp.StartsWith(";"))
         {
            // Kommentar in Ergebnis Objekt speichern
            e.sKommentar = sTmp.Substring(1).Trim();
         }
         else
         {
            throw new FileLoadException("Die angegebene Datei erhält eine fehlerhafte 1. Zeile");
         }

         // weitere Zeilen durchlaufen
```

```
        while (sr.Peek() >= 0)
        {
           if (iZeile >= Konstante.MAX_ZEILE )
           {
              sTmpAusn = String.Format("Das Feld darf nicht mehr als {0} Zeilen haben.",
                 Konstante.MAX_ZEILE);
              throw new FileLoadException(sTmpAusn);
           }

           sTmp = sr.ReadLine();
           if(sTmp.StartsWith(";"))
           {
              // Kommentar gefunden
              continue;
           }

           // Zeile aufgrund von ',' trennen
           sTmpArr = sTmp.Split(',');
           if (iZeile == 0)
           {
              // erste Zeile --> Spaltenzahl festlegen
              iSpalten = sTmpArr.GetLength(0);
              if (iSpalten > Konstante.MAX_SPALTE)
              {
                 sTmpAusn = String.Format("Es darf nicht mehr als {0} Spalten geben.",
                    Konstante.MAX_SPALTE);
                 throw new FileLoadException(sTmpAusn);
              }
           }
           else
           {
              // prüfen ob einheitliche Spaltenlänge
              if (sTmpArr.GetLength(0) != iSpalten)
              {
                 throw new FileLoadException("Unterschiedliche Spaltenanzahl in Datei gefunden.");
              }
           }

           // über alle Einträge im getrennten Zeilenstring
           for (int k=0;k<sTmpArr.GetLength(0);k++)
           {
              // Leerzeichen vorne und hinten abschneiden / Groß- / Kleinschreibung nicht beachten
              sZelle = sTmpArr[k].Trim().ToUpper();

              if (sZelle.Equals("S"))
              {
                 // Startzelle gefunden
                 if (bStartgefunden)
                 {
                    // Startzelle wurde bereits gefunden
                    throw new FileLoadException("Es darf nur eine Startzelle definiert sein.");
                 }

                 // Startzelle wird in Feld als 0 gespeichert
                 iTmp = 0;
                 bStartgefunden = true;

                 //Startknoten festlegen
                 e.si = iZeile;
                 e.sk = k;
                 e.iStartknoten = iZeile * iSpalten + k;
              }
              else if (sZelle.Equals("Z"))
              {
                 // Zielzelle gefunden
                 if (bZielgefunden)
                 {
                    // Ziel bereits gefunden
                    throw new FileLoadException("Es darf nur eine Zielzelle definiert sein.");
                 }

                 // Zielzelle wird in Feld als 0 gespeichert
                 iTmp = 0;
                 bZielgefunden = true;
```

```
                //Zielknoten festlegen
                e.zi = iZeile;
                e.zk = k;
                e.iZielknoten = iZeile * iSpalten + k;
            }
            else
            {
                // wenn nicht S oder Z muss ein Zahlenwert vorliegen:
                try
                {
                    iTmp = Int32.Parse(sZelle);
                }
                catch
                {
                    sTmpAusn = "Es sind nur ganzzahlige Zahlenwerte "
                        + "zwiwschen 1 und 9 und S,Z in den Zellen erlaubt.";
                    throw new ArithmeticException(sTmp);
                }

                // liegt Zahlenwert im definierten Bereich?
                if (iTmp < 1 || iTmp > 9)
                {
                    throw new FileLoadException("Es sind nur Zahlenwerte zwischen 1 und 9 erlaubt.");
                }
            }

            // gelesenen Wert in temporären Feld speichern
            tmpFeld[iZeile, k] = iTmp;
        }

        iZeile++;   // nächste Zeile
    }
    sr.Close();

    // sind Start / Ziel gefunden?
    if (!bStartgefunden)
    {
        throw new FileLoadException("In der Datei muss eine Startzelle angegeben sein.");
    }

    if (!bZielgefunden)
    {
        throw new FileLoadException("In der Datei muss eine Zielzelle angegeben sein.");
    }

    //tmpFeld in Ergebnis - Feld übertragen
    e.Feld = new int[iZeile, iSpalten];
    for (int i = 0; i < iZeile; i++)
    {
        for (int k = 0; k < iSpalten; k++)
        {
            e.Feld[i, k] = tmpFeld[i, k];
        }
    }
}

/// <summary>
/// wandelt ein Feld in eine Adjazenzmatrix um
/// </summary>
/// <remarks>
/// Hinweise:  Die Knoten werden Zeilenweise eingelesen, nummeriert. Also erst Zeile 0,
///            dann Zeile 1, usw...
/// Historie:  22.05.2012  (awi) Erstellt
/// </remarks>
/// <param name="e">Ein Ergebnis Objekt</param>
public void FeldUmwandeln(Ergebnis e)
{
    //AdjazenzMatrix erzeugen, ein Feld der Größe m*n x m*n
    e.AdjazenzMatrix = new int[e.Feld.GetLength(0) * e.Feld.GetLength(1),
                               e.Feld.GetLength(0) * e.Feld.GetLength(1)];

    int iZeile, iSpalte, kZeile, kSpalte;
    int iBreiteFeld = e.Feld.GetLength(1);
```

```
            // über alle Zeilen der Adjazenmatrix
            for (int i = 0; i < e.AdjazenzMatrix.GetLength(1); i++)
            {
               // Spalte und Zeile im Feld des Zeilen - Knotens berechnen
               iZeile = (int)(i / iBreiteFeld);
               iSpalte = i % iBreiteFeld;

               // über alle Spalten der Adjazenzmatrix
               for (int k = 0; k < e.AdjazenzMatrix.GetLength(0); k++)
               {
                  // Spalte und Zeile im Feld des Spalten - Knotens berechnen
                  kZeile = (int)(k / iBreiteFeld);
                  kSpalte = k % iBreiteFeld;

                  if (i == k)
                  {
                     // Weg auf sich selbst ist 0
                     e.AdjazenzMatrix[i, k] = 0;
                  }
                  else
                  {
                     // alle anderen Verbindungen, prüfen ob Knoten benachbart
                     if ((iZeile == kZeile && Math.Abs(iSpalte - kSpalte) == 1) ||
                         (iSpalte == kSpalte && Math.Abs(iZeile - kZeile) == 1))
                     {
                        e.AdjazenzMatrix[i, k] = e.Feld[kZeile, kSpalte];
                     }
                     else
                     {
                        // nicht benachbart, unendlich zuweisen
                        e.AdjazenzMatrix[i, k] = Konstante.UNENDLICH;
                     }
                  }
               }
            }
         }
      }
   }
```

Ergebnis.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace Wegplanung
{
   /// <summary>
   /// Klasse zur Speicherung der eingelesenen Daten und der Ergebnisse
   /// </summary>
   /// <remarks>
   /// Historie:  22.05.2012  (awi) Erstellt
   /// </remarks>
   class Ergebnis
   {
      /// <summary>
      /// Feld wie eingelesen, S,Z als 0
      /// </summary>
      public int[,] Feld;

      /// <summary>
      /// Adjazenzmatrix für Berechnung der minimalen Lösung
      /// </summary>
      public int[,] AdjazenzMatrix;

      /// <summary>
      /// Zeile Start
      /// </summary>
      public int si;

      /// <summary>
      /// Spalte Start
      /// </summary>
      public int sk;

      /// <summary>
      /// Zeile Ziel
      /// </summary>
      public int zi;

      /// <summary>
      /// Spalte Ziel
      /// </summary>
      public int zk;

      /// <summary>
      /// Start als Knoten in der Adjazenzmatrix
      /// </summary>
      public int iStartknoten;

      /// <summary>
      /// Ziel als Knoten in der Adjazenzmatrix
      /// </summary>
      public int iZielknoten;

      /// <summary>
      /// Errechnete Kostenabschätzung
      /// </summary>
      public int iKostenAbsch;

      /// <summary>
      /// errechnete Minimalkosten
      /// </summary>
      public int iKostenMinimal;

      /// <summary>
      /// errechneter kürzester Weg
      /// </summary>
      public int[] kuerzesterWeg;
```

```
        /// <summary>
        /// Kommentar aus der Datei eingelesen
        /// </summary>
        public String sKommentar;

        /// <summary>
        /// Gibt die Zahl des Knotens in der Adjazenzmatrix für eine Zeile und Spalte des Feldes zurück
        /// </summary>
        /// <remarks>
        /// Historie:  22.05.2012  (awi) Erstellt
        /// </remarks>
        /// <param name="iZeile">Zeile im Feld des Objekts</param>
        /// <param name="iSpalte">Spalte im Feld des Objekts</param>
        /// <returns>die Zahl des Knotens als Integer</returns>
        public int KnotenZuIndex(int iZeile, int iSpalte)
        {
            // ist Adjazenzmatrix bereits erstellt?
            if(AdjazenzMatrix != null)
            {
                return iZeile * Feld.GetLength(1) + iSpalte;
            }
            else
            {
                throw new NullReferenceException("Es wurde noch keine Adjazenzmatrix angelegt");
            }
        }

        /// <summary>
        /// Gibt die Spalte zu einer Knotenzahl der Adjazenzmatrix
        /// </summary>
        /// <remarks>
        /// Historie:  22.05.2012  (awi) Erstellt
        /// </remarks>
        /// <param name="iKnoten">Der Knoten in der Adjazenzmatrix</param>
        /// <returns>die Spalte des Knotens im Feld als Integer</returns>
        public int SpalteZuKnoten(int iKnoten)
        {
            if (AdjazenzMatrix != null)
            {
                return iKnoten % Feld.GetLength(1);
            }
            else
            {
                throw new NullReferenceException("Es wurde noch keine Adjazenzmatrix angelegt");
            }
        }

        /// <summary>
        /// Gibt die Zeile zu einer Knotenzahl der Adjazenzmatrix
        /// </summary>
        /// <remarks>
        /// Historie:  22.05.2012  (awi) Erstellt
        /// </remarks>
        /// <param name="iKnoten">Der Knoten in der Adjazenzmatrix</param>
        /// <returns>die Zeile des Knotens im Feld als Integer</returns>
        public int ZeileZuKnoten(int iKnoten)
        {
            if (AdjazenzMatrix != null)
            {
                return (int)(iKnoten / Feld.GetLength(1));
            }
            else
            {
                throw new NullReferenceException("Es wurde noch keine Adjazenzmatrix angelegt");
            }
        }
    }
}
```

Ausgabe.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.IO;

namespace Wegplanung
{
   /// <summary>
   /// Klasse zur Ausgabe von Ergebnissen, Fehlern und Informationen. Außerdem können Datenstrukturen
zu Testzwecken ausgegeben werden.
   /// </summary>
   /// <remarks>
   /// Historie:  22.05.2012  (awi) Erstellt
   /// </remarks>
   class Ausgabe
   {

      /// <summary>
      /// Gibt das Ergebnis in der vorgegebenen Form auf dem Bildschirm aus
      /// </summary>
      /// <remarks>
      /// Historie:  22.05.2012  (awi) Erstellt
      ///            23.05.2012  (awi) Ausgabe in gesonderte Methode gekapselt
      /// </remarks>
      /// <param name="e">Ein Ergebnis Objekt</param>
      public void ErgebnisAusgabe(Ergebnis e)
      {
         // Console als SW übergeben
         StreamWriter sw = new StreamWriter(Console.OpenStandardOutput(), Console.OutputEncoding);
         sw.AutoFlush = true;
         ErgebnisAusgabe(sw, e);
         sw.Close();
      }

      /// <summary>
      /// Gibt das Ergebnis in der vorgegebenen Form in einer Datei aus
      /// </summary>
      /// <remarks>
      /// Historie:  22.05.2012  (awi) Erstellt
      ///            23.05.2012  (awi) Ausgabe in gesonderte Methode gekapselt
      /// </remarks>
      /// <param name="sDateiname">Ein String mit einem Dateinamen</param>
      /// <param name="e">Ein Ergebnis Objekt</param>
      public void ErgebnisDateiAusgabe(String sDateiname, Ergebnis e)
      {
         // Datei als SW öffnen
         StreamWriter sw;
         try
         {
            sw = new StreamWriter(sDateiname, false, Encoding.UTF8);
         }
         catch
         {
            throw new FieldAccessException("Datei (" + sDateiname + ") konnte nicht zum Schreiben
geöffnet werden.");
         }

         try
         {
            ErgebnisAusgabe(sw, e);
         }
         catch
         {
            Console.WriteLine("Ergebnis konnte nicht in Datei (" + sDateiname + ") geschrieben
werden.");
         }
```

```
            sw.Close();
        }

        /// <summary>
        /// Gibt das Ergebnis in der vorgegebenen Form in einem StreamWriter aus
        /// </summary>
        /// <remarks>
        /// Historie:  23.05.2012  (awi) Erstellt
        /// </remarks>
        /// <param name="sw">Ein StreamWriter zur Ausgabe</param>
        /// <param name="e">Ein Ergebnis Objekt</param>
        private void ErgebnisAusgabe(StreamWriter sw, Ergebnis e)
        {
            if (sw != null)
            {
                sw.WriteLine(e.sKommentar);
                sw.WriteLine("Startzelle: {0}, {1}, Zielzelle: {2}, {3}", e.si + 1, e.sk + 1, e.zi + 1,
e.zk + 1);
                sw.WriteLine("Abschätzung der Kostenobergrenze: {0} KE", e.iKostenAbsch);
                sw.WriteLine("Minimalkosten: {0} KE", e.iKostenMinimal);
                sw.Write("Weg: ");
                for (int iKnoten = 0; iKnoten < e.kuerzesterWeg.GetLength(0); iKnoten++)
                {
                    if (iKnoten == 0)
                    {
                        sw.Write("S; ");
                    }
                    else if (iKnoten == e.kuerzesterWeg.GetLength(0) - 1)
                    {
                        sw.Write("Z\n");
                    }
                    else
                    {
                        sw.Write("{0},", e.ZeileZuKnoten(e.kuerzesterWeg[iKnoten]) + 1);
                        sw.Write("{0}; ", e.SpalteZuKnoten(e.kuerzesterWeg[iKnoten]) + 1);
                    }
                }
            }
        }

        /// <summary>
        /// Gibt einen Fehler auf Bildschirm und in einer Datei aus
        /// </summary>
        /// <remarks>
        /// Historie:  22.05.2012  (awi) Erstellt
        /// </remarks>
        /// <param name="sFehler">Ein String mit einem Fehlertext</param>
        /// <param name="sDateiname">Ein String mit dem Dateinamen zur Fehlerausgabe</param>
        public void FehlerAusgabe(String sFehler, String sDateiname)
        {
            Console.WriteLine(sFehler);

            // Fehler in Ausgabedatei schreiben
            StreamWriter sr;
            try
            {
                sr = new StreamWriter(sDateiname, false, Encoding.UTF8);
            }
            catch
            {
                Console.WriteLine("Datei (" + sDateiname + ") konnte nicht zum Schreiben geöffnet
werden");
                return;
            }

            sr.WriteLine(sFehler);

            sr.Close();
        }

        /// <summary>
        /// Gibt einen String auf dem Bildschirm aus
        /// </summary>
        /// <remarks>
```

```
        /// Historie:  22.05.2012  (awi) Erstellt
        /// </remarks>
        /// <param name="sString">Ein String mit einem Infotext</param>
        public void StringAusgabe(String sString)
        {
            Console.WriteLine(sString);
        }

        /// <summary>
        /// Gibt eine Adjazenzmatrix des Ergebnis Objekts auf dem Bildschirm zu Testzwecken aus
        /// Hinweise:  Unendlich wird als INF (infinity) dargestellt
        /// Historie:  22.05.2012  (awi) Erstellt
        /// </summary>
        /// <param name="e">Ein Ergebnis Objekt</param>
        public void AdjazenzMatrixAusgabe(Ergebnis e)
        {
            String s;

            // über alle Zeilen
            for (int i = 0; i < e.AdjazenzMatrix.GetLength(0); i++)
            {
                s = "";  // String für die Zeilenweise Ausgabe
                // über alle Spalten
                for (int k = 0; k < e.AdjazenzMatrix.GetLength(1); k++)
                {
                    if (e.AdjazenzMatrix[i, k] == Konstante.UNENDLICH)
                    {
                        // Unendlich wird als INF (infinity) ausgegeben
                        s = s + "INF , ";
                    }
                    else
                    {
                        // Wert ausgeben
                        s = s + String.Format("{0,3}", e.AdjazenzMatrix[i, k]) + " , ";
                    }
                }
                Console.WriteLine(s);
            }
        }

        /// <summary>
        /// Gibt ein Feld des Ergebnis Objekts auf dem Bildschirm zu Testzwecken aus
        /// </summary>
        /// <remarks>
        /// Historie:  22.05.2012  (awi) Erstellt
        /// </remarks>
        /// <param name="e">Ein Ergebnis Objekt</param>
        public void FeldAusgabe(Ergebnis e)
        {
            String s;

            // über alle Zeilen
            for (int i = 0; i < e.Feld.GetLength(0); i++)
            {
                s = ""; // String zur zeilenweisen Ausgabe
                // über alle Spalten
                for (int k = 0; k < e.Feld.GetLength(1); k++)
                {
                    s = s + e.Feld[i, k] + " , ";
                }
                Console.WriteLine(s);
            }
        }

    }
}
```

Program.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.IO;
using System.Diagnostics;

namespace Wegplanung
{
   /// <summary>
   /// Klasse zur Festlegung von Programmweiten Parametern
   /// </summary>
   /// <remarks>
   /// Historie:  23.05.2012  (awi) Erstellt
   /// </remarks>
   class Konstante
   {
      /// <summary>
      /// Unendlich (Kosten, Distanz, etc.)
      /// </summary>
      public const int UNENDLICH = -1;

      /// <summary>
      /// Markierung kein Vorgänger Knoten
      /// </summary>
      public const int KEIN_VORGAENGER = -1;

      /// <summary>
      /// Maximale Anzahl Zeilen
      /// </summary>
      public const int MAX_ZEILE = 20;

      /// <summary>
      /// Maximale Anzahl Spalten
      /// </summary>
      public const int MAX_SPALTE = 20;
   }

   /// <summary>
   /// Klasse zur Steuerung des Programms und zur Berechnung der Ergebnisse
   /// </summary>
   /// <remarks>
   /// Historie:  22.05.2012  (awi) Erstellt
   /// </remarks>
   class Program
   {
      /// <summary>
      /// Die Main-Methode, steuert den Programmablauf
      /// </summary>
      /// <remarks>
      /// Historie:  22.05.2012  (awi) Erstellt
      /// </remarks>
      /// <param name="args">Die Programmparameter. Es kann ein Dateiname zum Einlesen angegeben
werden</param>
      static void Main(string[] args)
      {
         // Zeitmessung zur Performance-Verbesserung
         Stopwatch watch = new Stopwatch();
         watch.Start();

         // Initialisierung
         Ergebnis erg = new Ergebnis();
         Ausgabe ausg = new Ausgabe();
         Eingabe eing = new Eingabe();

         String sDateinameEingabe = "";
         String sDateinameAusgabe = "";

         if (args.Length > 0)
         {
            // Parameter zum einlesen der Datei
```

```
            sDateinameEingabe = args[0];
         }
         else
         {
            // Standard Input verwenden
            sDateinameEingabe = @"Testfaelle/spezial4.txt";
         }

         // Ausgabe-Dateinamen festlegen (NameEingabedatei_Ergebnis.txt)
         sDateinameAusgabe = Path.GetDirectoryName(sDateinameEingabe)
            + @"\" + Path.GetFileNameWithoutExtension(sDateinameEingabe) + "_Ergebnis.txt";

         try
         {
            // Datei einlesen
            eing.DateiEinlesen(sDateinameEingabe, erg);
         }
         catch(Exception e)
         {
            ausg.FehlerAusgabe("Fehler beim Einlesen der Datei: " + e.Message, sDateinameAusgabe);
            return;
         }

         // Feld in Adjazenz Matrix umwandeln
         eing.FeldUmwandeln(erg);

         // Kosten abschätzen
         KostenAbsch(erg);

         // Minimale Kosten + Weg bestimmen
         KostenMinimal(erg);

         watch.Stop();
         ausg.StringAusgabe("Ausführungszeit: " + watch.Elapsed);

         // Ergebnis ausgeben
         ausg.ErgebnisAusgabe(erg);

         // Ergebnis in Datei ausgeben
         ausg.ErgebnisDateiAusgabe(sDateinameAusgabe, erg);

         // Programm beenden
         Console.ReadKey();
      }

      /// <summary>
      /// Errechnet eine Kostenabschätzung
      /// </summary>
      /// <remarks>
      /// Hinweise:  -  errechnet eine triviale Lösung des Problems.
      ///               Läuft im eingelesenen Feld zunächst vom Start auf
      ///               die Höhe des Ziels und von dort zum Ziel.
      ///               Die summierten Kosten dieses Weges ist die
      ///               Kostenabschätzung
      ///            -  Feld im Ergebnis Objekt muss bereits angelegt sein
      /// Historie:  22.05.2012  (awi) Erstellt
      /// </remarks>
      /// <param name="e">Ein Ergebnis Objekt</param>
      static void KostenAbsch(Ergebnis e)
      {
         int iKosten = 0;
         int iIterator;

         if (e.Feld == null)
         {
            throw new Exception("Feld im Ergebnis Objekt nicht angelegt");
         }

         //vertikale Richtung
         if (e.si < e.zi)
         {
            // Ziel liegt "unterhalb" von Start
            iIterator = 1;
         }
```

```
            else
            {
               // Ziel liegt "oberhalb" von Start
               iIterator = -1;
            }

            // Weg gehen und Kosten summieren
            for (int i = e.si; i != e.zi; i = i + iIterator)
            {
               iKosten += e.Feld[i,e.sk];
            }

            //horizontale Richtung
            if (e.sk < e.zk)
            {
               // Ziel liegt "rechts" von Start
               iIterator = 1;
            }
            else
            {
               // Ziel liegt "links" von Start
               iIterator = -1;
            }

            // Weg gehen und Kosten summieren
            for (int k = e.sk; k != e.zk; k = k + iIterator)
            {
               iKosten += e.Feld[e.zi, k];
            }

            //Kosten speichern
            e.iKostenAbsch = iKosten;
         }

         /// <summary>
         /// Errechnet einen optimalen Weg mit Kostenminimum
         /// </summary>
         /// <remarks>
         /// Hinweise:  -  errechnet eine optimale Lösung des Problems inklusive dem Weg.
         ///               Es wird ein angepasster Dijkstra-Algorithmus verwendet
         ///            -  die Adjazenzmatrix des Ergebnis Objekts muss bereits erstellt
         ///               sein
         /// Historie:  22.05.2012  (awi) Erstellt
         /// </remarks>
         /// <param name="e">Ein Ergebnis Objekt</param>
         static void KostenMinimal(Ergebnis e)
         {
            if (e.AdjazenzMatrix == null)
            {
               throw new Exception("Adjazenzmatrix im Ergebnis Objekt muss angelegt sein");
            }

            int iMinimum;
            int iKnoten;

            //Initialisierung
            int iAnzahlKnoten = e.AdjazenzMatrix.GetLength(0);

            int[] kosten = new int[iAnzahlKnoten];             // Array für Kosten
            int[] distanz = new int[iAnzahlKnoten];            // Array für Entfernung
            int[] vorgaenger = new int[iAnzahlKnoten];         // Array für Vorgängerknoten

            int iKostenNeu, iDistanzNeu;

            bool[] markiert = new bool[iAnzahlKnoten];

            for (int i = 0; i < iAnzahlKnoten; ++i)
            {
               // kein Knoten markiert
               markiert[i] = false;

               // Kosten auf unendlich setzen
               kosten[i] = Konstante.UNENDLICH;
```

```
            // Distanz auf unendlich setzen
            distanz[i] = Konstante.UNENDLICH;

            // keine Vorgänger
            vorgaenger[i] = Konstante.KEIN_VORGAENGER;
         }

         // Startknoten mit Kosten/Distanz 0
         kosten[e.iStartknoten] = 0;
         distanz[e.iStartknoten] = 0;

         bool bWeitereKnoten = true;

         while (bWeitereKnoten)
         {
            // minimale Kosten initialisieren
            iMinimum = Int32.MaxValue;

            // zugehöriger (minimaler) Knoten
            iKnoten = 0;

            // Minimale Distanz ermitteln
            for (int j = 0; j < iAnzahlKnoten; j++)
            {
               // Knoten schon markiert -> überspringen (Vermeidung von Zyklen!)
               if (markiert[j])
               {
                  continue;
               }

               // Distanz kleiner als Minimum -> neues Minimum gefunden
               if (kosten[j] != Konstante.UNENDLICH && kosten[j] < iMinimum)
               {
                  iMinimum = kosten[j];
                  iKnoten = j;
               }
            }

            // Distanz aktualisieren, wenn Zielknoten über den gefundenen
            // Minimumknoten billiger erreichbar
            for (int j = 0; j < iAnzahlKnoten; j++)
            {
               // wenn nicht markiert und nicht Verbindung gleicher Knoten und Verbindung vorhanden
               if (  !markiert[j] &&
                     iKnoten != j &&
                     e.AdjazenzMatrix[iKnoten, j] != Konstante.UNENDLICH)
               {
                  // Neuberechnung von Distanz und Kosten
                  iKostenNeu = kosten[iKnoten] + e.AdjazenzMatrix[iKnoten, j];
                  iDistanzNeu = distanz[iKnoten] + 1;

                  // besteht Verbesserung? Unendlich ist immer zu verbessern,
                  // ansonsten vergleichen, ob neuer Wert besser als alter Wert
                  // oder, Kosten gleich, kürzeren Weg wählen
                  if (kosten[j] == Konstante.UNENDLICH ||
                     kosten[j] > iKostenNeu ||
                     (kosten[j] == iKostenNeu && distanz[j] > iDistanzNeu))
                  {
                     // Verbesserung gefunden
                     kosten[j] = iKostenNeu;
                     distanz[j] = iDistanzNeu;
                     vorgaenger[j] = iKnoten;
                  }
               }
            }

            // gerade bestimmten Minimumknoten markieren
            markiert[iKnoten] = true;

            // sind noch Knoten nicht markiert -> weiter, sonst Ende
            bWeitereKnoten = false;
            for (int j = 0; j < iAnzahlKnoten && !bWeitereKnoten; j++)
            {
               bWeitereKnoten = !markiert[j];
```

```
            }
         }

         // kuerzester Weg speichern

         // Länge des kürzesten Weges
         e.kuerzesterWeg = new int[distanz[e.iZielknoten] + 1];

         // Vorgänger verfolgen bis zum Ziel ("Rückwärtssuche")

         // Rückwärtssuche startet beim Zielknoten
         iKnoten = e.iZielknoten;

         // Ergebnis Array von hinten befüllen
         int kuerzesterWegIterator = distanz[e.iZielknoten];
         while (vorgaenger[iKnoten] != Konstante.KEIN_VORGAENGER)
         {
            // Knoten speichern
            e.kuerzesterWeg[kuerzesterWegIterator] = iKnoten;
            // Vorgänger festlegen
            iKnoten = vorgaenger[iKnoten];
            kuerzesterWegIterator--;
         }

         // Startknoten speichern
         e.kuerzesterWeg[0] = e.iStartknoten;

         // Minimalkosten speichern
         e.iKostenMinimal = kosten[e.iZielknoten];
      }
   }
}
```