Hochschule Karlsruhe
Technik und Wirtschaft
UNIVERSITY OF APPLIED SCIENCES

Aufgaben

zur

Vorlesung Algorithmen und Datenstrukturen

Hochschule Karlsruhe – Technik und Wirtschaft
Fakultät EIT
Prof. Dr.-Ing. Gerhard Schäfer

WS 16/17

Inhalt

1. Symbolische Berechnung von Übertragungsfunktionen

1.1 Problembeschreibung

Die komplexe Übertragungsfunktion von RCL-Netzwerken kann beim Vorliegen bestimmter Strukturen ohne die Lösung von Gleichungssystemen ermittelt werden. Dazu soll ein solches Netzwerk als Graph abgebildet werden und durch sukzessive Zusammenfassung von Bauelementen entsprechend der Reihenschaltungs- bzw. der Parallelschaltungsregeln zusammengefasst werden. Blöcke, die nicht zusammengefasst werden können, sollen in einem eigenständigen Teilgraphen zusammengefasst werden, der dann als eigenständige Einheit in einem weiteren Reduzierungsschritt aufgefasst werden kann. Hauptanliegen ist jedoch die Zusammenfassung der Gesamtschaltung in Form einer rationalen Funktion.

1.2 Schnittstellenbeschreibung

1.2.1 Eingabedaten

Nets: a:IN; b:Out; c: CMN; d,e: Internal; //CMN entspricht Masse
R1:R (a, d);
C2:C (d, b);
L4:L (b, c);

1.2.2 Ausgabe

Es sollen die einzelnen Schritte der Zusammenfassungsoperationen und eine textuelle Form der Resultatsfunktion ausgegeben werden.
Zur besseren Darstellung der Transferfunktion soll auch eine Bruchrepräsentation mit der grafischen Oberfläche GDE erzeugt werden.

1.3 Hilfsmittel

Skript Algorithmen und Datenstrukturen Kapitel 7.3 Programm lexan
GDE – Software

2. Zeichnen von Graphen

2.1 Problembeschreibung

Die Abbildung von Graphenstrukturen in eine zweidimensionale Ebene ist neben der übersichtlichen Darstellung von Zusammenhängen auf den verschiedensten Gebieten auch bei dem Auffinden technischer Lösungen von großer Bedeutung. Das Entflechten von Schaltungen zur Positionierung der Bauelemente auf Platinen gehört dazu. Mathematische Kriterien von Euler [1] oder Ausschlussregeln von Kuratowski [2] bieten erste Hilfestellungen bei der Bewältigung dieser Probleme. W.T. Tutte gab 1960 einen Algorithmus an [3], der es erlaubt eine möglichst annehmbare Version eines Graphen zu zeichnen. Dieser Algorithmus soll in dieser Arbeit implementiert werden und an verschiedenen Beispielen überprüft werden. Die Ergebnisse sollen einem einfachen Algorithmus gegenübergestellt werden, der die Knoten auf der Diagonalen eines Gitters platziert und dann auf den verbliebenen Gitterlinien die Knoten verbindet.

1.2 Schnittstellenbeschreibung

1.2.1 Eingabedaten

NODE n1, n2, n3
Edge s1(n1,n2) : VMAX(50), LENGTH(100),LOAD(20)
Edge s1(n2,n3) : VMAX(50), LENGTH(100),LOAD(10)

1.2.2 Ausgabe

Grafische Ausgabe des Graphen mit Hilfe der GDE-Oberfläche

1.3 Hilfsmittel

[1] http://de.wikipedia.org/wiki/Planarer_Graph
[2] http://de.wikipedia.org/wiki/Satz_von_Kuratowski
[3] http://www.csse.monash.edu.au/~gfarr/research/slides/Eades-HowToDrawGraphRevisitedv11.pdf
[4] GDE – Software

3. Auswertung von March-Tests

3.1 Problembeschreibung

March Tests werden zum Test von RAM-Strukturen eingesetzt. Hierzu werden Vorschriften angegeben, in welcher Richtung Adressen erzeugt werden sollen und welche Werte in die Speicherzellen geschrieben bzw. gelesen werden sollten. Bestimmte Folgen werden hierbei benötigt um beispielsweise die Funktion des Decoders zu testen oder bestimmte Fehler zu überprüfen. Die Fehler werden aufgrund der charakteristischen Umgebung einer Speicherzelle eingeordnet. Im vorliegenden Fall sollen die charakteristischen Umgebung mit den Testsignalen in Beziehung gesetzt werden, und zusätzlich überprüft werden, ob die Tests ggf. doppelt durchgeführt werden. Ebenso soll die charakteristische Folge für den Decodertest erkannt werden.
Ein Durchlauf wird als March bezeichnet. Er ist durch seine Richtung (Adressen werden aufsteigend oder absteigend angelegt) und seinen Operationen gekennzeichnet (Schreiben bzw. Lesen mit Testen des richtigen Wertes).

3.2.1 Eingabedaten

March 1 (Up, W1, R1, W0); //March Nr. (Richtung, Write 1, Read 1, Write 0)
March 2 (Dn, R0, W1, R0); //Ist die Adressrichtung beliebig, wird UD angegeben

3.2.2 Ausgabe

Es sollen die ermittelten Umgebungen mit ihren Signalwechseln ausgegeben werden und bei Dubletten eine Kennzeichnung erfolgen. Eine Kennzeichnung des Decodertests ist ebenfalls vorzunehmen. Die Ausgabe soll als Text und als Grafik in der GDE - Oberfläche ausgegeben werden.

3.3 Hilfsmittel

Skript Algorithmen und Datenstrukturen, Kapitel 7.3 Programm lexan
Skript Test digitaler Schaltungen, Kapitel 8 Test spezieller Strukturen
GDE – Software

4. Verkehrsflusssteuerung

4.1 Problembeschreibung

Zur Optimierung der Ampelanlagen einer Stadt soll ein einfacher Algorithmus getestet werden. Das Straßennetz soll als Graph angegeben werden können, wobei die Knoten die Kreuzungen darstellen und die Kanten die Straßen modellieren sollen. Die Straßen werden mit Kenndaten, wie maximale Geschwindigkeit oder ihrer Länge versehen. Ein Auto soll mit einer durchschnittlichen Länge von 3.5 m angenommen werden.
In der Anfangsphase soll für jedes Fahrzeug eine optimale Route ermittelt werden. Die Startposition des Fahrzeuges liegt dann auf der ersten Straße der ermittelten Route. Erreicht das Fahrzeug seinen Zielknoten, wird es aus den weiteren Betrachtungen entfernt. Das Fließen des Verkehrs soll durch die Vorgabe eines Zeitschrittes und die Berechnung der neuen Positionen mit der erlaubten Geschwindigkeit modelliert werden. Die Ampeln sollen zunächst unabhängig in einem festen Raster geschaltet werden und dann von der Anzahl der Fahrzeuge, die auf den Straßen stehen, die direkt an einen Knoten (Kreuzung) führen, abhängig sein. Eine Neuberechnung der optimalen Route soll bei jedem Fahrzeug dann durchgeführt werden, wenn sich eine Möglichkeit ergibt, eine andere Route einzuschlagen.

4.2 Schnittstellenbeschreibung

4.2.1 Eingabedaten

NODE n1, n2, n3
STREET s1(n1,n2) : VMAX(50), LENGTH(100)
STREET s1(n2,n3) : VMAX(50), LENGTH(100)
CAR Name FROM n1 TO n2
…

4.2.2 Ausgabe

Das Straßennetz soll zusätzlich grafisch dargestellt und die berechneten Routen der Fahrzeuge angegeben werden. Der Verlauf der Simulation soll ebenfalls verfolgt werden können.

4.3 Hilfsmittel

[1] Programm lexan Kapitel 7.3 Skript Algorithmen und Datenstrukturen
[2] Kürzeste Wege in einem Graphen Kapitel 6.4.4 Skript Algorithmen und Datenstrukturen
[3] GDE – Software

5. Ameisenalgorithmus

5.1 Problembeschreibung

Das Problem des Handlungsreisenden dient als Problemstellung für den Vergleich von Optimierungsalgorithmen. Es werden hierbei Positionen von Städten vorgegeben, die durch vorgegebene Straßen verbunden sind. Der Handlungsreisende soll jetzt die Städte alle durchlaufen, ohne eine Stadt zweimal zu besuchen. Die letzte Forderung kann auch fallen gelassen werden. Hierzu gibt es bereits Algorithmen, die das Problem durch das Auffinden von sogenannten Hamiltonkreisen entweder exakt oder näherungsweise lösen [1]. Das Finden der exakten Lösung wurde als NP-schwer klassifiziert. Ein weiterer Ansatz zur annähernden Lösung des Problems ist der Ameisenalgorithmus. Hier werden Wege von den Ameisen ausprobiert und durch das Kollektiv bewertet. Es soll nun ein vorhandenes Programmpaket [2] so erweitert werden, dass die dort vorhandenen unterschiedlichen Ameisenalgorithmen gegen die bereits vorhandenen Algorithmen getestet werden können. Hierzu sind das Programmpaket zu erweitern und ein Vergleich vorzunehmen. Die Dokumentation soll entsprechend angepasst werden.

5.2 Schnittstellenbeschreibung

5.2.1 Eingabedaten

Die Eingabedaten sollen an das vorhandene Programmpaket [2] angepasst werden.

5.2.2 Ausgabe

Verwendung der Ausgabemöglichkeiten von [2]
Laufzeit und Gütedaten der vorhandenen und eingefügten Algorithmen

5.3 Hilfsmittel

[1] U. Kaiser, C/C++ Von den Grundlagen zur professionellen Programmierung, Galileo Computing, Abschnitt 17.3.4, Im EIT Downloadbereich unter ADS_Hamilton.pdf
[2] http://www.codeproject.com/Articles/644067/Applying-Ant-Colony-Optimization-Algorithms-to-Sol

6. Analyse des Spiels Rush Hour

6.1 Problembeschreibung

Bei dem Spiel Rush Hour [1] können Fahrzeuge, deren Größe durch ein unterlegtes Gitter bestimmt ist, auf dem Gitter jeweils nur in einer Richtung verschoben werden. Ziel ist es, ein besonderes Auto (RH) zum Ausgang zu bewegen in dem die anderen Autos verschoben werden und somit den Weg zum Ausgang freimachen. Für das Spiel gibt es verschiedene schwere Ausgangssituationen, die auf einem 6*6 Gitter definiert sind [1]. Es sollen jetzt auch auf anderen Gittergrößen Aufgabenstellungen erzeugt, auf ihre Lösungsfähigkeit untersucht und in ihrer Schwere charakterisiert werden können. Hierzu gibt es bereits Lösungsalgorithmen, die auf einer Tiefensuche im Lösungsraum basieren [2]. Aufgabe ist es jetzt beliebige Szenarien zu erzeugen und als Ausgangssituation abzuspeichern [3].

6.2 Schnittstellenbeschreibung

6.2.1 Eingabedaten

```
DefGrid (6,6)        //Gittergröße (XSize, YSize)
Car C1 V 3 (5, 2)    //Car Carname V|H Size (Startkoordinaten X,Y)
                     //V für Vertikal, H für Horizontal
```

Der Name CH ist für das Auto reserviert, das zum Ausgang gebracht werden soll.
Die Notation kann auch für die generierten Aufgaben verwendet werden.

6.2.2 Ausgabe

Gitterbelegung grafisch mit Hilfe der GDE.
Gitterbelegung textuell in der Konsole
Es sollen auch die Lösungsfolgen angezeigt werden

6.3 Hilfsmittel

[1] http://de.wikipedia.org/wiki/Rush_Hour_%28Spiel%29
[2] http://stackoverflow.com/questions/2877724/rush-hour-solving-the-game
[3] http://cs.ulb.ac.be/~fservais/rushhour/
[4] GDE – Software

6. Strategieoptimierung für eine modifizierte Form des Spieles „Schiffe versenken“

6.1 Problembeschreibung

Das Spiel „Schiffe versenken“ soll in seinen Regeln erweitert und dazu sollen mehrere Spielstrategien getestet werden. Eines der eigenen Schiffe kann hierbei nach einem Schuss des Gegners um ein Kästchen vorwärts oder rückwärts bewegt oder um 90 Grad gedreht werden. Der vom Gegner getroffene Rasterpunkt darf für drei weitere Schüsse von eigenen Schiffen nicht belegt werden. Sind jeweils nur noch zwei Schiffe vorhanden, dürfen diese nicht mehr bewegt werden. Es soll hierbei der Computer gegen sich selbst spielen. Zunächst sollen zufallsorientiert oder über eine Textdatei vorgegeben die Positionen der Schiffe für zwei Widersacher generiert werden. Für die gilt es dann eine Angriffssequenz zu ermitteln um möglichst schnell, viele Treffer zu erreichen. Jeder Gegner soll seine eigene Strategie in einer Folge von Spielen ermitteln können.
Ansatzpunkte für eine Strategie könnten sein:

1. Auswahl von Punkten über einen Zufallsgenerator
2. Auswahl der Punkte in Form eines parametrierbaren Gitters
3. Punktauswahl über eine Grundstruktur und deren Verzweigung
4. Folgetabelle von Punkten aus erfolgreichen Treffern aus den vorausgegangenen Spielen
5. Raumfüllende Kurven z.B. Hilbertkurve
6. Ausweichmanöver, wenn z.B. eine Rasterstrategie des Gegners entdeckt wird.

6.2 Schnittstellenbeschreibung

6.2.1 Eingabedaten

Ein Schiff muss mindestens 2 Kästchen groß sein.
Spieler1
Schifftyp (Rasterxstart, Rasterystart), (Rasterxend, Rasteryend)
…
Spieler2
Schifftyp (Rasterxstart, Rasterystart), (Rasterxend, Rasteryend)
…

6.2.2 Ausgabe

Grafische Anzeige der Spielfelder mit den Ausgangssituationen und des Verlaufs des Spiels.
Gegenüberstellung der Wirksamkeit der Strategien während eines Spiels.
Gesamtbewertung der Strategien.

6.3 Hilfsmittel

[1] Programm lexan Kapitel 7.3 Skript Algorithmen und Datenstrukturen
[2] GDE – Software

7. Strategieoptimierung für das Spiel „2048“

7.1 Problembeschreibung

Spielstrategien beim Spiel 2048 sollen gegeneinander getestet werden. Es soll der Computer das Spiel durchführen. Es sollen zwei Varianten des Spiels auf die Tauglichkeit von Spielstrategien getestet werden:

1. Tatsächlich das Erreichen der Zahl 2048
2. Erreichen einer möglichst großen Zahl

Spielstrategien können sein:

- Immer sofort zusammenfassen mit bevorzugten Bewegungsrichtungen.
- Erst möglichst viele zusammenfassbare Felder erzeugen und dann zusammenfassen.
- Es könnten auch Regeln für eine Abfolge von Zügen erstellt werden.

7.2 Schnittstellenbeschreibung

7.2.1 Eingabedaten

Feldgröße x,y
Spieltyp Benennung
Vorbelegung x1,y1, z1;…xn,yn,zn; //Um auch von Zwischenständen aus spielen zu können

7.2.2 Ausgabe

Grafische Anzeige des Spielfeldes mit den Ausgangssituationen und des Verlaufs des Spiels.
Gegenüberstellung der Wirksamkeit der Strategien während eines Spiels.
Gesamtbewertung der Strategien.

7.3 Hilfsmittel

[1] Programm lexan Kapitel 7.3 Skript Algorithmen und Datenstrukturen
[2] GDE – Software
[3] http://www.news.de/technik/855525240/2048-strategie-und-loesung-einfache-tipps-um-2048-zu-erreichen-und-das-zahlen-kultspiel-zu-knacken/1/
[4] http://www.pcgames.de/2048-Spiel-54182/Tipps/2048-Tipps-So-raeumt-ihr-spielend-hohe-Zahlen-ab-1116290/