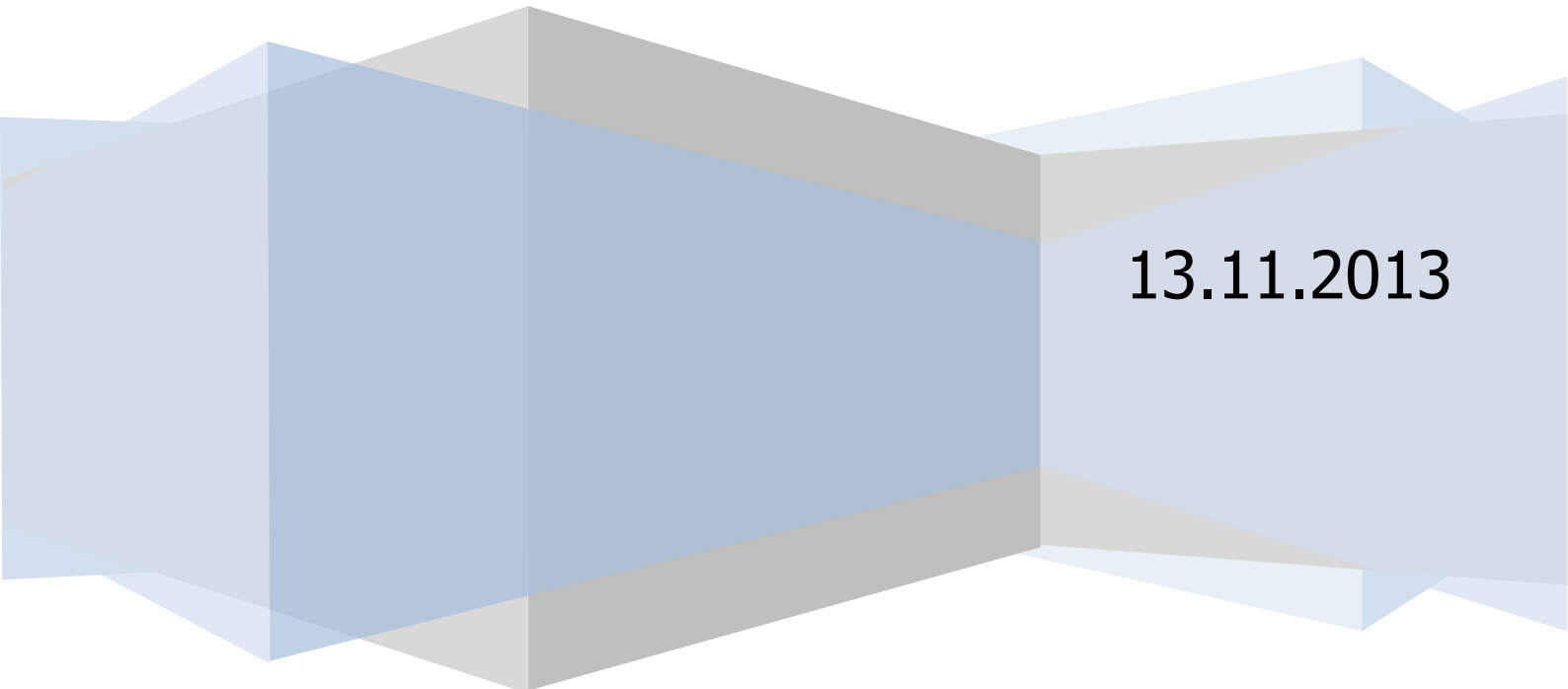


4AHIT

Sudoku

Protokoll

Ayvazyan Ari & Belinic Vennesa



13.11.2013

Inhaltsangabe

Git-Link	1
Aufgabenstellung	1
Designüberlegung	2
Arbeitsaufteilung	3
Aufwandschätzung	3
Endzeitaufteilung.....	3
Arbeitsdurchführung	5
Testbericht	7
Quellenangaben	11

Git-Link

<https://github.com/aayvazyan-tgm/Sudoku>

Aufgabenstellung

Erstellen Sie einen Algorithmus zur Lösung von Sudokurätseln. Dabei sollen nicht nur klassische Sudokus gelöst werden können, sondern zumindest ein ähnliches Rätsel (z.B. X-Sudoku oder Squiggly) [1]. Die Eingabe der Rätsel erfolgt zwingend über Dateien (csv) [2]. Optional kann es auch händisch über die Konsolenapplikation eingegeben werden (zusätzlich zum Ladevorgang aus einer Datei), wobei dann das Rätsel auch abspeicherbar sein soll.

Das Menü soll somit die gewählten zwei Rätselalgorithmen anzeigen, eine Lademöglichkeit anbieten und optional eine Eingabemöglichkeit.

Teamwork

Diese Aufgabenstellung MUSS von zwei Schülern gelöst werden, um das Trennen von Funktionen und Arbeitsteilung zu trainieren. Verwenden Sie auf jeden Fall selbsterstellte Headerdateien und lagern Sie zumindest den Berechnungsalgorithmus in eine eigene .c-Datei aus! Es muss ein Headerfile für das Kompilieren und Linken sowie Ausführen und Aufräumen bereitgestellt werden.

Abgabe

Es ist auf jeden Fall ein Protokoll über die Arbeitsschritte und die verwendete Logik zu erstellen. Dabei sind auch Testfälle zu beschreiben. Das Protokoll muss als PDF-Dokument den Sources beigegeben und als ZIP Archiv abgegeben werden.



Designüberlegung

Sudoku

Da es erlaubt ist bereits bestehenden Sourcecode zu verwenden, hatte ich vor im Internet nach brauchbarem Sourcecode zu suchen (ein Programm in C, dass Sudokurätsel löst und mit Files arbeitet). Diese Prgramm würde ich dann so anpassen, dass es der Angabe entspricht.

X-Sudoku oder Squiggly

Da sich keine frei zugängliche Lösung zu XSudoku finden lässt ist es geplant eine bestehende java/c++ anwendung zu suchen und den algorythmus zu übernehmen, diese ist dann um die csv-File funktionen zu erweitern.



Arbeitsaufteilung

Die Arbeit wird so aufgeteilt, dass jedes Gruppenmitglied, soweit wie möglich, von einander unabhängig arbeiten kann. Deshalb werden den jeweiligen Gruppenmitglieder ganze Programmteile als Arbeitspakete zugeteilt. Jedes Gruppenmitglied testes die von ihm/ihr geschriebenen Files so weit es nötig ist.

Vennesa

- Sudoku

Ari

- X-Sudoku oder Squiggly

	<i>geschätzte Zeit</i>	<i>tatsächliche Zeit</i>	<i>Kommentar</i>
Ayvazyan Ari	6h	7h	1h suche nach bestehenden Lösungen 0,5h einlesen in den bestehenden Code. 4,5h erweitern des bestehenden Codes um auch Xsudokus lösen zu können. 1h für die Dokumentation & Testing
Belinic Vennesa	3 h	6 h	1 h um den Code zu lesen und zu verstehen 3 h für das Ändern des Codes 2 h für die Dokumentation

Aufwandschätzung

Arbeitspaket	Aufwand in h
Sudoku	3
X-Sudoku oder Squiggly	6
SUMME	

Endzeitaufteilung

Arbeitspaket	Aufwand in h
Sudoku	6
X-Sudoku oder Squiggly	7



SUMME



Arbeitsdurchführung

Sudoku

Als erstes suchte ich im Internet nach bereits vorhandenen Algorithmen, um nicht "das Rad neu zu erfinden". Nach gründlicher Recherche stieß, auf einige brauchbare Beispiel. Nachdem ich mir diese ein wenig durchgelesen hatte entschloss ich mich für ein Beispiel das mit txt-Files arbeitet, da ich annahm, dass ich bei diesem Beispiel nicht allzu große Änderungen vornehmen muss.

Ich nahm einige wenige Änderung im Code vor, damit dieser mit CSV-Files funktioniert:

Das Programm arbeitet standartmäßig mit txt-Files, für die noch nicht gelösten Zahlen wurden Punkte eingesetzt. Ich änderte diesen Punkt auf 0, da bei mir der Benutzer 0 eingeben soll wenn die Zahl noch nicht gelöst ist.

```
/* etwas geändert . auf 0 */
const char value_chars[psize + 1] = "0123456789";
```

Da in CSV-File einen Seperator git, um die Werte von einander zu trennen, muss dies berücksichtigt werden, in der Methode zum Einlesen. Dies befindet sich in der puzzle_read-Funktion. Das c ist das aktuelle Zeichen, col ist die aktuelle Spalte.

```
/* Hinzugefuegt */
if (c == ',') {
    col--;
} else {
```

Hier habe ich zur puzzle_solve Methode einen Parameter stream vom Type FILE* hinzugefügt, um die Lösung in ein CSV-File schreiben zu können.

```
/* Hinzufuegen des Parameter stream FILE* */
void puzzle_solve(value_t puzzle[psize][psize], int depth, int* moves,
    int* solutions, FILE* stream) {
```

Hier führe ich dann meine eigene Funktion puzzle_write_csv aus, hier verwende ich den Parameter den ich oben hinzugefügt habe.

```
/* Hinzugefuegt um Loesung in ein csv-File schreibe */
puzzle_write_csv(puzzle, stream);
```

Inhalt der puzzle_write_csv Funktion.

```
void puzzle_write_csv(value_t puzzle[psize][psize], FILE* stream) {
    int row, col;
    for (row = 0; row < psize; ++row) {
        for (col = 0; col < psize; ++col) {
            value_t value = puzzle[row][col];
            fputc(value_chars[value], stream);
            if (col < (psize - 1))
                fputc(',', stream);
        }
        fputc('\n', stream);
    }
}
```

Letztlich trennte ich die Includes und Defines von den Funktionen und der Main Methode.



X-Sudoku oder Squiggly

Da kein bestehender XSudoku Code in C gefunden werden konnte und der Sudoku Code fertig war, musste eine andere Lösung als Implementieren des bestehenden Sourcecodes gesucht werden. Der erste Gedanke lag darin, eine bestehende Java/C++ Anwendung zu nehmen und den Sourcecode in C zu übersetzen. Nach genauerem Ansehen des Konzepts hinter XSudoku wurde schnell klar, dass es ein einfacherer Weg ist die bereits bestehende Sudoku Lösung zu erweitern.

Hier wurde in der Überprüfungsmethode angesetzt, welche rekursiv aufgerufen wird und das bestehende Sudoku auf seine Richtigkeit überprüft.

An dieser Stelle wurde Code implementiert um zu prüfen ob auch die Diagonalen des Sudokus korrekt eingegeben wurden.

Das ganze wurde noch in eine Überprüfung eingepackt, welche prüft ob gerade ein XSudoku gelöst wird. Dies kann man über die arguments der Anwendung festlegen.



Testbericht

Sudoku

Hier probiere ich die Targets des Makefiles aus:

```
schueler@debianlamp:~/workspace/sudoku$ make compile
gcc -Wall -c -o sudokualg.o sudokualg.c
schueler@debianlamp:~/workspace/sudoku$ make link
gcc -Wall -c -o sudokualg.o sudokualg.c
gcc sudoku.c -Wall sudokualg.o -o sudoku.sh
schueler@debianlamp:~/workspace/sudoku$ make clean
schueler@debianlamp:~/workspace/sudoku$ make all
gcc -Wall -c -o sudokualg.o sudokualg.c
gcc sudoku.c -Wall sudokualg.o -o sudoku.sh
schueler@debianlamp:~/workspace/sudoku$ make run
gcc -Wall -c -o sudokualg.o sudokualg.c
gcc sudoku.c -Wall sudokualg.o -o sudoku.sh
./sudoku.sh
Arguments: normal|x-sudoku inputfile.csv outputfile.csv
```

Im Makefile ist noch einen Target enthalten, dass das Programm mit fixen Werten testet, diese führe ich hier ausführen:

```
schueler@debianlamp:~/workspace/sudoku$ make run-test
gcc -Wall -c -o sudokualg.o sudokualg.c
gcc sudoku.c -Wall sudokualg.o -o sudoku.sh
./sudoku.sh normal puzzle1.csv out.csv
got initial puzzle:
```

```
000000201
001000060
800700300
560047000
910605087
000820015
007006004
090000500
406000000
```

found solution 1 in 51580 moves:

```
679384251
341952768
825761349
568147923
912635487
734829615
157296834
293418576
486573192
```

total 68769 moves, 1 solutions



Hier probiere ich das Programm mit dem Standartinput aus:

```
schueler@debianlamp:~/workspace/sudoku$ ./sudoku.sh normal  
reading from standard input:
```

```
020000000  
000600003  
074080000  
000003002  
080040010  
600500000  
000010780  
500009000  
000000040
```

```
got initial puzzle:
```

```
020000000  
000600003  
074080000  
000003002  
080040010  
600500000  
000010780  
500009000  
000000040
```

```
found solution 1 in 705624 moves:
```

```
126437958  
895621473  
374985126  
457193862  
983246517  
612578394  
269314785  
548769231  
731852649  
1,2,6,4,3,7,9,5,8  
8,9,5,6,2,1,4,7,3  
3,7,4,9,8,5,1,2,6  
4,5,7,1,9,3,8,6,2  
9,8,3,2,4,6,5,1,7  
6,1,2,5,7,8,3,9,4  
2,6,9,3,1,4,7,8,5  
5,4,8,7,6,9,2,3,1  
7,3,1,8,5,2,6,4,9
```

```
total 6046560 moves, 1 solutions
```



Falls ein fehlerhafter Wert im Sudoku steht wird dieser it 0 eingelesen:

```
schueler@debianlamp:~/workspace/sudoku$ ./sudoku.sh normal puzzlefalsch.csv out1.csv
got initial puzzle:
```

```
000000201
001000060
800700300
560047000
910605087
000820015
007006004
090000500
406000000
```

```
found solution 1 in 51580 moves:
```

```
679384251
341952768
825761349
568147923
912635487
734829615
157296834
293418576
486573192
```

```
total 68769 moves, 1 solutions
```

Inhalt von puzzlefalsch.csv

```
0,0,0,0,0,0,2,0,1
0,0,1,0,0,0,0,6,0
8,0,0,7,0,0,3,0,0
5,6,0,0,4,7,0,0,0
9,1,0,6,0,5,0,8,7
0,0,0,8,2,a,0,1,5
0,0,7,0,0,6,0,0,4
0,9,0,0,0,0,5,0,0
4,0,6,0,0,0,0,0,0
```

Über Standartinput einlesen, mit Fehlern:

```
schueler@debianlamp:~/workspace/sudoku$ ./sudoku.sh normal
reading from standard input:
asdfghjkl
234567890ß
asdfghjkl
xcvbnm,.
line ended before reading all characters in row
```



X-Sudoku oder Squiggly

Da XSudoku auf die bestehende Lösung aufbaut werden hier keine Eingabefehler angeführt.

```
./sudoku.sh x-sudoku xsudo1.csv xsudokuOut.csv  
got initial puzzle:
```

```
004790060  
068005000  
000080134  
000076000  
080029506  
056010079  
093000052  
600000980  
002900000
```

```
found solution 1 in 499 moves:
```

```
124793865  
368145297  
579682134  
931576428  
487329516  
256814379  
893461752  
645237981  
712958643
```

```
total 5013 moves, 1 solutions  
schueler@debianlamp:~/Desktop/test$ █
```



Quellenangaben

- [1] Matt Zucker, "Sudoku.c", zuletzt aktualisierte: 2010, online verfügbar:
<http://www.swarthmore.edu/NatSci/mzucker1/e15/sudoku.html>, zuletzt besucht am:
30.10.2013

