34. Bundeswettbewerb Informatik
1. Aufgabe

Dokumentation

TEAM „Der LAMABÄR“

30. November 2015

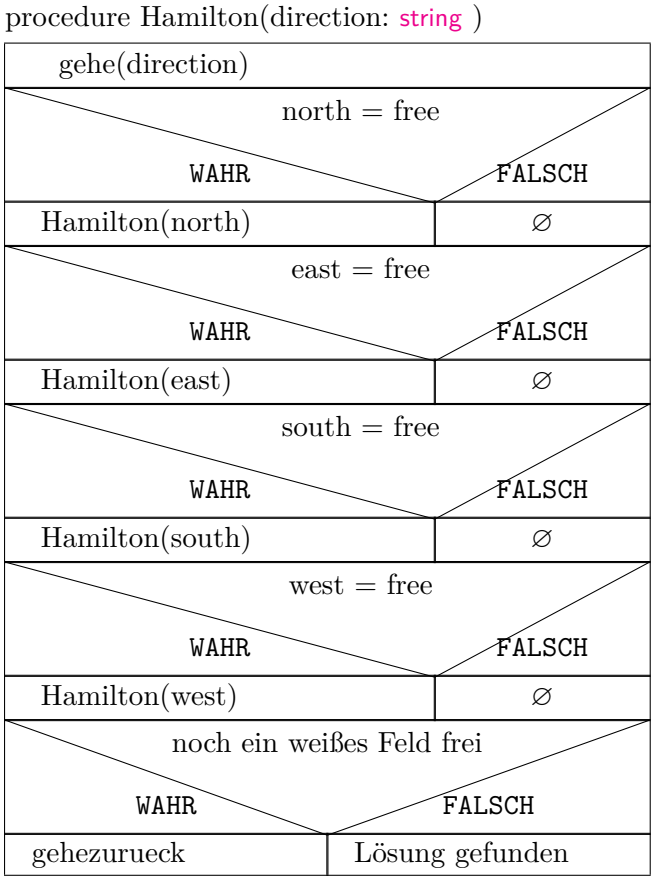
Inhaltsverzeichnis

1 Lösungsidee

Die Aufgabe beinhaltet zwei verschiedene Aufgaben, die Junioraufgabe 2 und die Hauptaufgabe 1. Entgegen der vorgeschlagenen Richtung habe ich zuerst die Hauptaufgabe bearbeitet und danach die Junioraufgabe 2.
Formal wird das Problem, wie es in der 1. Aufgabe beschrieben ist, als „Hamiltonpfadproblem“ bezeichnet. Es wird also ein Pfad gesucht, der jeden Knoten eines Graphens genau einmal enthält. Ein Knoten entspricht hier ein weißes Feld, auf den die Schildkröte gehen kann.
Die Idee zur Lösung des Problems besteht im Wesentlichen in einer einfachen Rekursion, dargestellt in Abbildung 1. Die Idee dahinter ist, dass die Schildkröte rekursiv in alle Richtungen läuft und dabei eine Art „Duftspur“ hinterlässt, sodass sie den Weg nicht doppelt gehen kann. Wenn die Schildkröte dann nicht mehr in eine Richtung laufen kann, dann wird geprüft, ob es noch ein weißes Feld gibt, auf welchem noch keine Duftspur hinterlassen wurde. Wenn dem so ist, dann geht die Schildkröte so lange zurück, bis sie ganz getreu der Rekursion in eine andere Richtung weitergehen kann.

Die Junioraufgabe 2 habe ich dann damit gelöst, in dem das Hamiltonproblem ausgeführt wird und wenn die Schildkröte auf einem weißen Feld landet, dann wird eine Variable gesetzt, dass dieses Feld betretbar ist. Da durch die Rekursion alle weißen Felder begangen werden, die die Schildkröte begehen kann, kann man nach der Rekursion herausfinden, welche weißen Felder nicht begangen werden können, die die Schildkröte also nie erreichen kann. Es gibt sicherlich einfachere Wege dies zu tun, aber da ich die Lösung für das Hamiltonproblem bereits geschrieben habe, war dieser Weg einfacher und schneller zu programmieren.

Abbildung 1: Nassi-Shneiderman-Diagramm zum Hamiltonsproblem



1.1 Laufzeitkomplexität

Bei dem Hamiltonpfadproblem besteht ein exponentieller Zusammenhang (s. Beispiele), was mit der Komplexitätsklasse des Hamiltonpfadproblems (NP-vollständig) übereinstimmt, was impliziert, dass es keinen effizienteren Algorithmus mit z.B. $\mathcal{O}(n)$ oder gar $\mathcal{O}(\log n)$ gibt. Jegliche Verbesserung des Codes würde die Steigung der exponentielle Funktion nur um ein Faktor verringern, sie kann jedoch nicht das exponentielle Wachstum verhindern. Daher habe ich mir gar nicht die Mühe gegeben, einen effizienteren Algorithmus zu schreiben, zumal dieser auch so schon die Beispiele in ein paar Millisekunden lösen kann. Die einzige Optimierung, die es gibt, ist die, dass der Algorithmus abbricht, wenn eine Lösung gefunden wurde.
Die genaue Beschreibung der Laufzeitkomplexität in Form einer Gleichung war mir leider nicht möglich, da es von zu vielen Faktoren abhängt. So ist zum Beispiel der genaue Aufbau des Feldes, die Länge und Breite des Feldes und auch die Startposition der Schildkröte relevant. Selbst in dem WorstCase, in dem das Feld ein Quadrat ist und es keine hereinragenden schwarze „Hindernisse“ gibt, kann man nichts genaueres ermitteln, da die schlimmste Startposition der Schildkröte von der Länge/Breite des Quadrats abhängt.

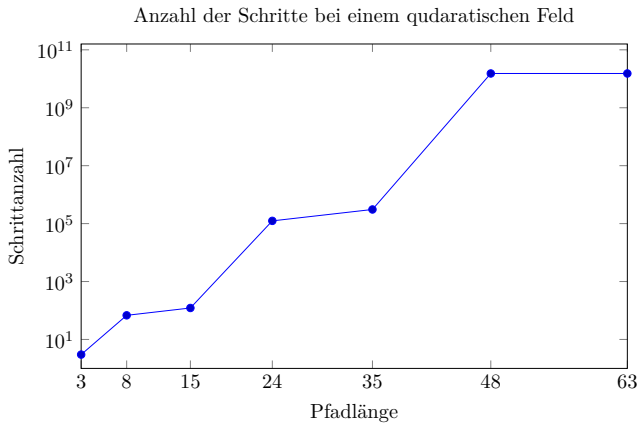


Abbildung 2: Pfadlänge vs. Schrittanzahl

Um aber trotzdem einmal beispielhaft zu verdeutlichen, dass hier ein exponentieller Zusammenhang besteht, habe ich einmal bei 2*2,3*3,4*4,5*5,6*6,7*7 und 8*8 Quadraten die Schildkröte an der Startposition starten lassen, an der diese die meisten „Schritte“ gehen muss. Dann habe ich die Schritte von meinem Programm zählen lassen und diese in Relation zu der Pfadlänge gesetzt: Abbildung 2. Eine Tendenz ist hier sicherlich erkennbar, auch wenn diese nicht exakt linear ist. Man beachte, dass die Ordinate in dem Fall logarithmisch eingeteilt ist, sodass ein linearer Graph in dem Fall einem exponentiellen Graph entspricht, was meine obige These unterstützt.
Die Berechnung der Anzahl der Schritte dauerte bei einem 7*7 und 8*8-Feld zwar ca. 4 Stunden, im Regelbetrieb, bei mehreren Hindernissen und keinen Quadraten, so wie es bei den Beispielen der Fall ist, dauert die Berechnung allerdings nie länger als eine Sekunde auf einem mittelmäßigen Rechner.

2 Umsetzung

Für die Umsetzung habe ich das Modell-View-Controller(MVC)-Prinzip verwendet, wobei ich View und Controller der Einfachheit halber zu einer Instanz vereint habe. Hinzu kommt eine objektorientierte Programmierung. Der View-Controller erfasst die Daten zu dem Feld und gibt eine Erstellung von diesem bei dem Modell in Auftrag; ein Objekt der Klasse TSpielfeld wird erstellt. Diese erstellt für jedes Feld in einem dynamischen Array ein Objekt der Klasse TFeld, welche Daten, wie die Eigenschaft, ob das Feld begehbar ist (s. Junioraufgabe 2) und Daten, die für das Hamiltonproblem benötigt werden, wie schwarzes, freies oder bereits begangenes Feld, enthält. Die Klasse TSpielfeld erzeugt auch die grafische Darstellung des aktuell gewählten Feld. Danach führt der Benutzer die Procedure „Hamilton“ entweder direkt oder indirekt durch die Junioraufgabe 2 aus, welche dann wiederum die Rekursion ausführt. Als grafische Eingabemöglichkeit stehen einmal ein Editor und einmal das Öffnen mit einer Textdatei zur Verfügung.
Ein UML-Klassendiagramm erachte ich aufgrund des recht einfachen Aufbaus und Lösungsweg für nicht notwendig, da dies keine neuen Erkenntnisse einbringt, die man nicht schon in dem obigen Text, in den Beispielen oder im Quelltext erlangen würde. Außerdem ist das, was ich noch nicht genannt habe, nämlich die grafische Ausgabe, die einen nicht unwesentlichen Anteil des Codes übernimmt, nicht so relevant.

3 Beispiele

Ich verwende die acht Beispiele (s. Tabelle 1), die vorgegeben wurden, wobei ich auf eine grafische Darstellung der Beispiele verzichte, da man dies entweder in meinem Programm nachsehen kann

oder sich anderweitig ansehen kann.
Da es für viele Beispiele mehrere Lösungen gibt, wurde hier nur eine beispielhafte Lösung zum Hamiltonpfad aufgeschrieben.

3.1 Beispieltabelle

Tabelle 1: Beispiele

	Höhe * Breite	Alle Felder erreichbar?	Hamiltonpfad (Beispiel)	Schritt-anzahl
Beispiel 0	6*9	WAHR	WNNWSSSOOONNNOOOSSSWNN	198
Beispiel 1	7*11	FALSCH	FALSCH	1108
Beispiel 2	5*8	WAHR	FALSCH	76
Beispiel 3	5*9	WAHR	OOOOSSWNWSWWWWNNO	17
Beispiel 4	3*15	WAHR	FALSCH	12
Beispiel 5	3*15	WAHR	WWWWWWWWWWWW	12
Beispiel 6	5*7	WAHR	FALSCH	10
Beispiel 7	5*12	WAHR	FALSCH	169

4 Quelltext

Im Folgenden wird neben dem zentralen Quelltext, die eigentliche Rekursion, wie sie auch schon in dem Nassi-Sneiderman-Diagramm dargestellt wurde (s. Abbildung 1), dargelegt. Außerdem werden die Funktionen „gehe“, „gehezurueck“ und „frei“, die für die Rekursion benötigt werden, im Folgendem dargelegt. Da der restliche Code entweder für die Kommunikation zwischen den Klassen (Getter, Setter und Constructor) oder für die grafische Aufbereitung inklusive das vorherige Erkennen von falschen Eingaben von dem Benutzer zuständig ist, wäre es nicht sinnvoll, diesen hier zu integrieren, da diese Komponente die eigentliche Lösungsfindung nicht beeinflusst. Da bei der Erzeugung jedes Objekts der Klasse TFeld „nur“ zwei einfache in sich geschachtelte for-Schleifen ausgeführt werden, füge ich den Quelltext hier nicht hinzu.
Aber zum besseren Verständnis habe ich einmal alle wichtigen Variablen der Klasse TSpielfeld samt Erklärungen hier noch hingeschrieben:

4.1 Variablen

```
1   breite : integer; //Die Breite des Spielfeld
2   hoehe : integer; //Die Hoehe des Spielfeld
3   moeglich : boolean; //Gibt an, ob das
        Beteten aller weissen Felder moeglich ist
4   loesung : boolean; //True=Loesung gefunden;
        False=noch keine gefunden/keine verfuegbar
5   pfad : string;  //Gibt den Pfad im Format
        'NWOONSO' o.ae. an
6   Felder : array of array of TFelder; //
        dynamisches Array: x=breite;
        y=hoehe
7   aktx : integer; //aktuelle x-Position der
        turtle
8   akty : integer; //aktuelle y-Position der
        turtle
9   schritte : extended; //gibt die Anzahl der
        Geh-Vorgaenge an s. Laufzeitoptimierung
10  mehrmalsausgefuehrt : boolean; //false:
        einmal den Hamiltonalgorithmus ausgefuehrt;
        true: zweimal oder mehrmals ausgefuehrt
```

Listing 1: Wichtige Variablen der Klasse TSpielfeld in Turbo Pascal

Abbildungsverzeichnis

4.2 Rekursion

```
1  procedure TSpielfeld.hamilton(direction: Integer);
2  var i,j:integer;weissda:boolean;
3    procedure gehe(direction:integer);
4    begin
5      schritte:=schritte+1;
6      if Felder[aktx,akty].getbetretbar=false then Felder[aktx,akty].setbetretbar(true);
7      //Gehen
8      if direction=0 then begin    //Gehe nach Norden
9        akty:=akty-1;
10       pfad:=pfad+'N';
11     end;
12     if direction=1 then begin    //Gehe nach Osten
13       aktx:=aktx+1;
14       pfad:=pfad+'O';
15     end;
16     if direction=2 then begin    //Gehe nach Sueden
17       akty:=akty+1;
18       pfad:=pfad+'S';
19     end;
20     if direction=3 then begin    //Gehe nach Westen
21       aktx:=aktx-1;
22       pfad:=pfad+'W';
23     end;
24     Felder[aktx,akty].setstatus(1);
25     if Felder[aktx,akty].getbetretbar=false then Felder[aktx,akty].setbetretbar(true);
26   end;
27   function frei(direction:integer):boolean;
28   begin
29     result:=false;
30     if direction=0 then begin
31       if Felder[aktx,akty-1].getStatus=0 then result:=true; //true=frei; false=belegt
32     end;
33     if direction=1 then begin
34       if Felder[aktx+1,akty].getStatus=0 then result:=true; //true=frei; false=belegt
35     end;
36     if direction=2 then begin
37       if Felder[aktx,akty+1].getStatus=0 then result:=true; //true=frei; false=belegt
38     end;
39     if direction=3 then begin
40       if Felder[aktx-1,akty].getStatus=0 then result:=true; //true=frei; false=belegt
41     end;
42   end;
43   procedure gehezurueck(direction:string);
44   begin
45       Felder[aktx,akty].setstatus(0);
46       if direction='N' then begin
47         akty:=akty+1;
48         Delete(pfad, Length(pfad), 1);
49       end;
50       if direction='O' then begin
51         aktx:=aktx-1;
52         Delete(pfad, Length(pfad), 1);
53       end;
54       if direction='S' then begin
55         akty:=akty-1;
56         Delete(pfad, Length(pfad), 1);
57       end;
58       if direction='W' then begin
59         aktx:=aktx+1;
60         Delete(pfad, Length(pfad), 1);
61       end;
62   end;
```

Listing 2: Die Rekursion mit sämtlichen verwendeten Prozeduren der Klasse TSpielfeld in Turbo Pascal (Teil 1)

```
1  begin
2    weissda:=false;
3    if loesung=true then exit;  //Abbruch: Um Rechenleistung zu sparen, da das Checken nach dem frei sein,
         nichts mehr bringt s. Kommentar unten
4    if mehrmalsausgefuehrt=true then exit; //Abbruch: Es aendert nichts den Algorithmus mehrmals
         auszufuehren
5    if direction>=0 then gehe(direction);
6    if frei(0)=true then hamilton(0); //Wenn im Norden frei ist, dann hamilton(north)
7    if frei(1)=true then hamilton(1); //Wenn im Osten frei ist, dann hamilton(east)
8    if frei(2)=true then hamilton(2); //Wenn im Sueden frei ist, dann hamilton(south)
9    if frei(3)=true then hamilton(3); //Wenn im Westen frei ist, dann hamilton(west)
10     for I := 1 to hoehe do begin
11       for J := 1 to breite do begin
12         if Felder[j,i].getStatus=0 then begin
13           if weissda=false then begin
14             if pfad<>'' then begin
15               gehezurueck(pfad[length(pfad)]); //Wenn noch irgendwo ein weisses Feld ist, dann gehe
                     zurueck und fahre dort fort
16             end;
17             weissda:=true;
18          end;
19         end;
20       end;
21     end;
22   if weissda=false then loesung:=true else weissda:=false;    //sage mir den Pfad, wenn der
       richtige Weg gefunden wurde
23 end; //Wenn die Loesung gefunden wurde, dann wird nicht mehr zurueckgegangen, sodass dann keine unoetige
     Strecke gelaufen wird, obwohl schon eine Loesung existiert.
```

Listing 3: Die Rekursion mit sämtlichen verwendeten Prozeduren der Klasse TSpielfeld in Turbo Pascal (Teil 2)

4.3 Junioraufgabe 2

```
1  function TSpielfeld.sommer():boolean;
2  var i,j:integer;
3  begin
4    hamilton(-1);
5    for I := 1 to hoehe do begin
6      for J := 1 to breite do begin
7        if (Felder[j,i].getbetretbar=false) AND (Felder[j,i].getStatus<>2) then begin
8          result:=false;
9          exit;
10       end else result:=true;
11     end;
12   end;
13 end;
```

Listing 4: Der Quelltext zur Lösung der Junioraufgabe 2 in: Klasse TSpielfeld in Turbo Pascal