

Aufgabe 1: Kassiopeias Weg

Tobias Nöthlich - Teilnahme-ID: 6396

29.11.2015

Inhaltsverzeichnis

I	Lösungsidee	1
II	Umsetzung	2
1	<code>int main()</code>	2
2	<code>void finde_weg(feld_typ& feld, vek pos, unsigned knoten = 0)</code>	2
3	<code>vek operator+(vek a, vek b) & char& at(feld_typ& feld, vek pos)</code>	2
4	<code>void zeige(int höhe, int breite, const feld_typ& feld)</code>	3
III	Beispiele	3
IV	Quelltext	4

Teil I

Lösungsidee

Da die Aufgabe explizit erforderte das jedes Feld nur einmal besucht werden kann, ist es nur bedingt möglich das Programm aus der Junioraufgabe 2 zu verwenden. Als Algorithmus zum Finden der Lösung habe ich mich für den Tiefensuchalgorithmus entschieden, da er relativ einfach zu implementieren ist und auch von Seiten der Größe bzw. Komplexität der vorgegebenen Beispiele nichts den Einsatz von Heuristik zwingend notwendig macht. Dabei wird zwar jeder mögliche Weg bis zum Finden einer Lösung getestet, die Laufzeit ist aber trotzdem gering da die Beispiele

nicht sonderlich komplex sind. Ein weiterer Vorteil der Tiefensuche ist die 100% Trefferquote, sollte es einen Weg geben wird er gefunden.

Teil II

Umsetzung

1 `int main()`

Vor der Ausführung der Funktion `int main()` definiere ich global ein struct namens `vek` welches die beiden Integer `x` und `y` beinhaltet. Desweiteren definiere ich für `std::vector<std::string>` den Alias `feld_typ`. Ich beginne mit der Funktion `int main()`, welche einen Vektor vom Typ `std::string` namens `Quadratien` initialisiert und daraufhin die Funktion `void finde_weg(feld_typ& feld, vek pos, unsigned knoten = 0)` mit den Parametern `Quadratien` und `start_pos` aufruft. Die letzte Aufgabe der Funktion `int main()` ist schließlich nur noch der Test ob ein Weg gefunden wurde und eine entsprechende Ausgabe.

2 `void finde_weg(feld_typ& feld, vek pos, unsigned knoten = 0)`

Die Funktion `void finde_weg(feld_typ& feld, vek pos, unsigned knoten = 0)` leistet nun den Hauptteil der Arbeit indem sie die Wege welche Kassiopaea benutzen kann durchtestet. Dazu wird in einer for-Schleife überprüft ob alle Felder `Quadratien` gefüllt sind, was einem gefundenen Weg gleichkommt. Sollte das zutreffen, wird der gefundene Weg ausgegeben und das Programm terminiert. Ist dies allerdings nicht der Fall, so wird in einer bereichsbasierten for-Schleife die Variable `richt_vek` vom Typ `vek` einem Element aus einem weiteren vek-Array namens `Richtungen`, welcher die Tupel `{0, -1}`, `{0, 1}`, `{-1, 0}`, `{1, 0}` enthält zugeordnet. In der for-Schleifen wird dann überprüft welcher der 4 Tupel momentan gewählt ist und dem String `Weg` wird die entsprechende Richtung (N, O, S oder W) hinzugefügt. Danach wird die Funktion `finde_weg` mit der neuen Position im Vektor erneut aufgerufen. Verläuft der Weg in eine Sackgasse, gibt die Funktion "`Backtracking...`" aus und setzt die Zellen, welche besucht wurden, bis zu dem Punkt wo es eine andere Möglichkeit zum "abbiegen" gab, zurück. Das wiederholt sich nun solange bis entweder ein Weg gefunden wurde oder das Programm keinen Weg finden konnte.

3 `vek operator+(vek a, vek b) & char& at(feld_typ& feld, vek pos)`

Die beiden oben beschriebenen Funktionen sind das Grundgerüst des Programmes, jedoch benötigt das Programm noch zwei kleine "Hilfsfunktionen" um richtig zu funktionieren. zum einen wäre das die Funktion `vek operator+(vek a, vek b)` welche das addieren von zwei Variablen des Typs `vek` ermöglicht. Desweiteren fehlt noch die

Funktion `char& at(feld_typ& feld, vek pos)`, welche zurückgibt was an der Stelle `pos` im Vektor `feld` liegt.

4 void zeige(int höhe, int breite, const feld_typ& feld)

Um *Quadratien* auf dem Bildschirm ausgeben zu können wird ausserdem noch die Funktion `void zeige(int höhe, int breite, const feld_typ& feld)` verwendet, welche in einer geschachtelten for-Schleife den Vektor ausgibt.

Teil III

Beispiele

1.

```
#####
#   #   #
#   #   #
#   K   #
#   #   #
#   #   #
#####
```

 Ausgabe: "WNNWSSSOOONNNNOOSSSONNN",

2.

```
#####
#       #
#   ### #
#   #   #
#   K   #
#   #   #
#####
```

 Ausgabe: "Kein Weg gefunden!"

3.

```
#####
#   K   #
#   ### #
#       #
#####
```

 Ausgabe: "Kein Weg gefunden!"

4.

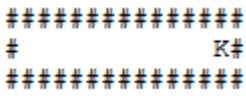
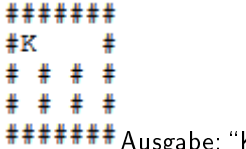
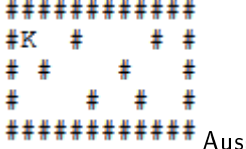
```
#####
#   K   #
#   ### #
#       #
#####
```

 Ausgabe: "WWSSOOOONOSONNWWW"

5.

```
#####
#       #
#       K
#####
```

 Ausgabe: "Kein Weg gefunden!"

6.  Ausgabe: "WWWWWWWWWWWWWW"
7.  Ausgabe: "Kein Weg gefunden!"
8.  Ausgabe: "Kein Weg gefunden!"

Teil IV

Quelltext

Kassiopeia.cpp

```

1  #include <iostream> //Ein-/Ausgabe
2  #include <cstdlib>   //system("...")
3  #include <vector>
4  #include <string>
5
6  std::string Weg;
7
8  struct vek //struct für alle wichtigen Operationen im Vektor
9  {
10     int x, y;
11 };
12
13 using feld_typ = std::vector<std::string>; //Alias für std::vector<std::string>
14
15 /*ADDITION VON VEK'S*/
16 vek operator+(vek a, vek b)
17 {
18     return{ a.x + b.x, a.y + b.y };
19 }
20
21 vek Richtung[] = { { 0, -1 }, { 0, 1 }, { -1, 0 }, { 1, 0 } }; //Die Richtungen
22
23 /*ZEICHEN AN BESTIMMTER STELLE*/
24 char& at(feld_typ& feld, vek pos)
25 {
26     return feld[pos.y][pos.x];
27 }
28
29 /*AUSGABE DES VECTORS*/
30 void zeige(int höhe, int breite, const feld_typ& feld)
31 {
32     for (int i = 0; i < höhe; i++)
33     {
34         for (int j = 0; j < breite; j++)
35         {
36             std::cout << feld[i][j];

```

```

37         }
38         std::cout << std::endl;
39     }
40     std::cout << std::endl;
41 }
42
43 /*ERMITTELN DES WEGES*/
44 void finde_weg(feld_typ& feld, vek pos, unsigned knoten = 0)
45 {
46     int leeresFeld = 0;
47     int Breite = feld[0].length();
48     int Höhe = feld.size();
49     if (at(feld, pos) != ' ')
50     {
51         return;
52     }
53
54     std::cout << "Reihe: " << pos.y << ", Spalte: " << pos.x << '\n';
55     at(feld, pos) = 'O' + knoten;
56
57     zeige(Höhe, Breite, feld);
58
59     for (int i = 0; i < Höhe; i++)
60     {
61         for (int j = 0; j < Breite; j++)
62         {
63             if (feld[i][j] == ' ')
64             {
65                 leeresFeld++;
66             }
67         }
68     }
69
70     if (leeresFeld == 0)
71     {
72
73         std::cout << Weg << std::endl << Weg.length() << " Felder besucht." << std::endl;
74         system("Pause");
75         exit(0);
76     }
77
78     for (auto richt_vek : Richtung){
79
80         if (richt_vek.y == -1)
81         {
82             Weg.append("N");
83         }
84         if (richt_vek.y == 1)
85         {
86             Weg.append("S");
87         }
88         if (richt_vek.x == 1)
89         {
90             Weg.append("O");
91         }
92         if (richt_vek.x == -1)
93         {
94             Weg.append("W");
95         }
96         finde_weg(feld, pos + richt_vek, knoten + 1);
97         Weg.pop_back();
98     }
99
100     std::cout << "Backtracking..\n";

```

```

105         at(feld, pos) = ' ';
106     }
107
108
109
110     int main()
111     {
112         /*VARIABLENDEKLARATION*/
113         std::vector<std::string> feld =
114         {
115             "#####",
116             "#   #",
117             "#####",
118             "#   #",
119             "#####",
120
121         };
122
123         int Breite = feld[0].length();
124         int Höhe = feld.size();
125         int leeresFeld = 0;
126         vek start_pos = { 3, 1 }; //Startpunkt, erst x dann y
127
128         /*BEGINN DER ANWEISUNGEN*/
129
130         finde_weg(feld, start_pos); //Aufru
131
132         for (int i = 0; i < Höhe; i++)
133         {
134             for (int j = 0; j < Breite; j++)
135             {
136                 if (feld[i][j] == ' ')
137                 {
138                     leeresFeld++;
139                 }
140             }
141         }
142
143         if (leeresFeld != 0)
144         {
145             std::cout << std::endl << "Kein Weg gefunden!" << std::endl;
146         }
147
148         system("Pause");
149         return 0;
150     }

```