# 32. Bundeswettbewerb Informatik, Runde 2 Ausarbeitungen zu den Aufgaben "Buschfeuer" und "Lebenslinien"

Philip Wellnitz

# Vorwort

# Inhaltsverzeichnis

1	Auf	gabe 1	- Buschfeuer	3
	1.1	Lösung	gsidee	 3
		1.1.1	Korrektheit	 6
		1.1.2	Laufzeitanalyse	 7
	1.2	Umsetz	zung	 8
	1.3	Beispie	ele	 9
		1.3.1	Beispiel 0	 9
		1.3.2	Beispiel 1	 11
		1.3.3	Beispiel 2	 15
		1.3.4	Beispiel 3	 19
	1 4	Quellte	ext.	24

# 1 Aufgabe 1 - Buschfeuer

# 1.1 Lösungsidee

Ein Feld ist ein quadratisches Stück Land, welches genau einen folgender Zustände inne Feld haben kann:

BRENNBAR Das Stück Land ist ind der Lage, zu brennen.

BRENNEND Ein brennendes Stück Land.

GELÖSCHT Ein Stück Land, welches nie wieder brennen wird.

LEER Ein leeres Stück Land.

Alle Felder haben die selbe Fläche.

Ein Wald ist nun die rechteckig-gitterförmige Anordnung von  $n \times m$  Feldern. Die  $Umgebung\ U(f)$  eines Feldes f in einem Wald W ist dabei die Menge an Feldern, welche in W eine gemeinsame Kante mit f haben.

Wald Umgebung

Der Wald wird nun diskret beobachtet. Es ist dabei sichergestellt, dass nur sofern ein Feld bei einer Beobachtung brennend ist, dieses und jedes brennbare Feld seiner Umgebung bei der nächsten Beobachtung brennen werden, sofern diese nicht schon brennen. Diese Eigenschaft des Waldes sei mit Feuerausbreitung bezeichnet.

Ab der 2. Beobachtung kann pro Beobachtung genau 1 (brennendes) Feld gelöscht werden. Wird ein brennendes Feld gelöscht, so fängt seine Umgebung bis zur nächsten Beobachtung nicht an zu brennen.

Die erste Beobachtung, ab der ein Feld f brennt, heiße Entflammung von f.

Ziel ist es nun, eine Folge von zu löschenden Feldern anzugeben, sodass bei deren Einhaltung die Anzahl der brennenden Felder minimiert wird.

Im Folgenden seinen diejenigen Felder, welche bei mindestens 2 Beobachtungen brennend waren, als *verkohlt* bezeichnet.

Nach der Feuerausbreiteung muss jedes Feld der Umgebung eines verkohlten Feldes c brennend sein oder gewesen sein oder seit der Entflammung von c nicht brennbar gewesen sein.

Sei nun zunächst der Fall betrachtet, dass nur brennende Feler gelöscht werden können.

Es ist leicht zu erkennen, dass es die Lösung nicht verschlechtert, wenn ab der 2. Beobachtung bei jeder Beobachtung 1 brennendes Feld gelöscht wird. Daher wird im Folgenden davon ausgegangen, dass bei jeder Beobachtung (ab der 2.) 1 brennendes Feld gelöscht wird. Es gilt nun also für jede dieser Beobachtungen dasjenige brennende Feld zu finden, durch dessen Löschung die Anzahl der im Folgenden (nicht unbedingt umittelbar folgend) zu brennen anfangenden Felder minimiert.

Sei nun eine Beobachtung fixiert.

Nun soll für ein brennendes Feld F ein Maß  $\mu(F)$  dafür gefunden werden, mit dem bestimmt werden kann, welches Feld zum Löschen in obigem Sinne am Besten ist. Sei  $\mu(F)$  daher die Anzahl der brennbaren Felder, zu denen F das brennende Feld mit dem kleinster Abstand ist. Dieser kürzeste Abstand ist dabei die minimale Anzahl an Beobachtungen, bis das Feld anfängt zu brennen. (Unter der Annahme, dass keine weiteren Felder gelöscht

werden.)

Löscht man nun F, so wird der kleinste Abstand aller Felder höchstens größer; bei allen Feldern, bei deren kürzestem Abstand F jedoch keine Rolle spielte (bei denen der Abstand zu einem anderen brennenden Feld also kleiner oder gleich dem Abstand zu F ist), tritt keine Veränderung auf.

Für 2 Werte  $\mu(F_1)$  und  $\mu(F_2)$  gilt nun: ist  $\mu(F_1) < \mu(F_2)$ , so erzeugte  $F_2$  bei mehr Feldern eine Vergrößerung des kleinster Abstands als  $F_1$ .

Die *minimale Lebenszeit* eines Feldes sei nun eben der kleinste Abstand zu einem brennenden Feld. Es ist leicht zu erkennen, dass nach mindesten so vielen Beobachtungen, wie die minimale Lebenszeit eines Feldes ist, das Feld zu brennen beginnt.

 $\mu(F)$  gibt also auch die Anzahl der Felder an, deren minimale Lebenszeit allein durch F besitmmt ist. Löscht man F, so wird, wie schon gesehen, die minimale Lebenszeit all dieser Felder höchstens größer, es ist also am Besten, dasjenige Feld  $F^*$  zum Löschen auszuwählen, welches  $\mu(\cdot)$  für alle aktuell brennenden Felder maximiert.

Es gilt nun noch  $\mu$  effizient zu bestimmen. Da ein Wald eine rechteckige Gitterform besitzt, ist der kürzeste Abstand zwischen 2 Feldern 1, ganau dann, wenn diese Felder eine gemeinsame Kante haben.

Fasse man das Gitter nun als Graphen auf, wobei die Felder die Knoten sind und zwischen 2 Knoten eine Kante ist, genau dann, wenn zwischen diesen Feldern eine Kante ist. Es nun offensichtlich, dass dieser Graph ungewichtet und ungerichtet ist. Somit ist das Finden von kleinsten Abständen mittels einer *Breitensuche* möglich.

Dabei sind die Startfelder der Breitensuche die brennenden Felder. Dabei muss für jedes dieser brennenden Felder eine eigene Breitensuche gestartet werden; wobei für alle Breitensuchen gemeinsam die ermittelten kleinsten Abstände gespeichert werden müssen. Zusätzlich zu den kleinsten Abständen müssen auch die dazugehörigen brennenden Felder gespeichert werden, von denen pro Feld eventuell mehr als 1 existiert. Weiterhin muss die Breitensuche nur brennbare Felder besuchen.

Sind die kleinsten Abstände gefunden, so kann  $\mu$  ermittelt werden, mithilfe simplem durchiterieren über alle Felder und gleichzeitigem Zählen der Felder, für die nur 1 brennendes Feld gespeichert wurde.

In Pseudocode:

```
1
   Wald
           ; //Der Wald; ein 2D-Container
2
3
   AnfangsBrennendeFelder()
                                    { //Ermittelt die von Anfang
      brennenden Felder
     brennendeFelder := null; //1D-Container für Positionen
4
        brennender Felder
     for (i = 0..Wald.Höhe())
5
       for (j = 0..Wald.Breite())
6
7
          if (Wald[i,j] == BRENNEND)
            brennendeFelder.Add((i;j)); //Gefundene Position
8
               hinzufügen
9
10
     return brennendeFelder; //Alle gefundenen Positionen
        zurückgeben
  }
11
12
  NächsteBeobachtung(aktBrennendeFelder) { //Ermittelt die bei der
13
      nächsten Beonachtung brennenden Felder, aus den Feldern, die
      aktuell brennen
```

```
14
     neuBrennendeFelder := null;
15
     for all((x;y) from aktBrennendeFelder)
       if (Wald[x,y] == GELÖSCHT)
16
17
         continue; //Feld kann kein Feuer verteilen
18
       Wald[x,y] := VERKOHLT; //2 mal brennende Felder sind verkohlt
19
20
       for all((x';y') from Umgebung((x;y)))
         if(Wald[x',y'] == BRENNBAR)
21
22
           neuBrennendeFelder.Add((x';y')); //Gefundene Position
              hinzufügen
23
           Wald[x',y'] := BRENNEND; //Wald beginnt zu brennen
24
25
     return neuBrennendeFelder;
26 }
27
  GetOptBewässerungspunkt(aktBrennendeFelder) { //Ermittelt den
      besten Bewässerungspunkt
29
     kleinsterAbstand := null; //Speichert für alle Felder des
        Waldes den kleinsten Abstand zu jedem Feld aus
        aktBrennendeFelder
30
31
     for(i = 0..kleinsterAbstand.Size())
           Fülle kleinsterAbstand[i] mithilfe einer Breitensuche
32
33
34
     anzEindeutigKleinstAbstände := null;
35
36
     for (i = 0..Wald.Höhe())
       for (j = 0..Wald.Breite())
37
         if(Es ex. k mit kleinsterAbstand[k][i,j] eindeutiges
38
            Minimum für alle mögliche k)
39
           anzEindeutigKleinstAbstände[k]++;
40
41
     return aktBrennendeFelder[k, sodass
        anzEindeutigKleinstAbstände[k] maximal];
42 }
43
  SimuliereFeuer() { //Die eigentliche Berechnung
44
45
     aktBrennendeFelder := AnfangsBrennendeFelder(); //Anfangs
        interessante Felder; Kann brennende, von Feuer umschlossene
        Felder beinhalten
46
     while (!aktBrennendeFelder.Empty()) //Solange es brennende
        Felder gibt
47
       aktBrennendeFelder := NächsteBeobachtung(
          aktBrennendeFelder) //Ermittle die bei nächster
          Beobachtung brennenden Felder
48
           if (aktBrennendeFelder.Empty())
                             //Keine Felder brennen mehr
49
             break;
50
     Wald[GetOptBewässerungspunkt(aktBrennendeFelder)] := GELÖSCHT;
51
        //Lösche das aktuell beste Feld
52 }
```

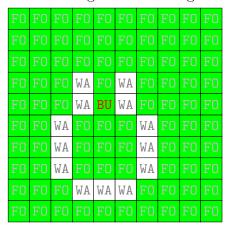
#### 1.1.1 Korrektheit

Wie schon beschrieben, wird bei jeder Beobachtung das für diese Beobachtung nach  $\mu$  beste Feld zum Löschen ausgewählt.

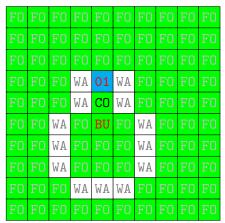
Es gilt also zu zeigen, dass insgesamt nicht weniger Felder abbrennen, sollte bei einer Beobachtung nicht das für diese Beobachtung nach  $\mu$  optimalste Feld gelöscht werden.

Es lässt sich jedoch ein einfaches Beispiel konstruieren, indem eben dies der Fall ist; eine bessere Lösung also gefunden werden kann, wird nicht das nach  $\mu$  optimalste Feld gelöscht:

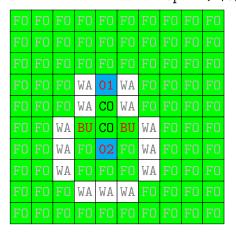
Die Löschung nach dem Algorithmus:



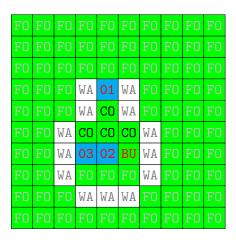
--- At time 1: Water spot (4|3)



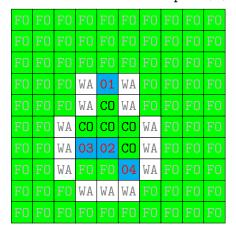
--- At time 2: Water spot (4|6)



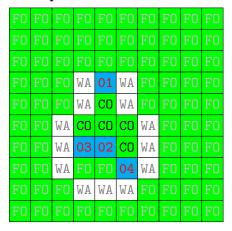
--- At time 3: Water spot (3|6)



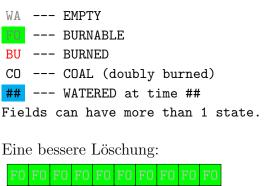
--- At time 4: Water spot (5|7)

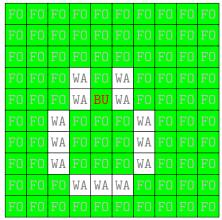


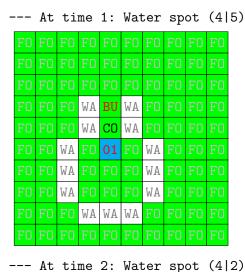
--- And you'll find 5 pieces of coal and 4 pieces of watered coal

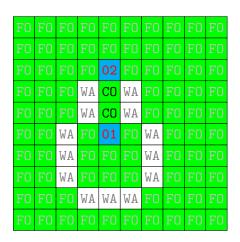


Explanation:

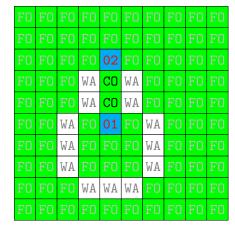








--- And you'll find 2 pieces of coal and 2 pieces of watered coal



#### Explanation:

WA --- EMPTY

BU --- BURNABLE

CO --- COAL (doubly burned)

## --- WATERED at time ##

Fields can have more than 1 state.

Der Algorithmus ist also nicht optimal, es handelt sich um eine Heuristik. Dabei liefert sie bei vielen Eingaben *ziemlich* gute Ergebnisse<sup>1</sup>. Zum Vergleich habe ich den Brute-Force-Ansatz implementiert, der garantiert optimale Löschungen liefert.

# 1.1.2 Laufzeitanalyse

Der Brute-Force-Ansatz probiert alle Möglichkeiten an verschiedenen Löschungen und wählt die optimalste. Grob überschlagen gibt es für jede Löschung 4 Möglichkeiten, somit ergibt sich eine grobe obere Schranke für den Worst-Case von  $\mathcal{O}(4^b)$ , mit b der Anzahl der Löschungen der Lösung<sup>2</sup>. Dieser Ansatz hat also eine exponenzielle Laufzeit, im Gegensatz zu der Heuristik wie im Folgenden gezeigt wird.

<sup>&</sup>lt;sup>1</sup>Siehe dazu Sektion Beispiele

 $<sup>^2\</sup>mathrm{Es}$  gibt wohl Pfade im Suchbaum, die länger als b sind; durch geschicktes Pruning ist diese Schranke jedoch einhaltbar

Eine Breitensuche hat eine Laufzeit von  $\mathcal{O}(V+E)$  in einem Graphen mit E Kanten und V Knoten. Speziell hat der Graph bei dieser Aufgabe  $n \cdot m$  Knoten und  $(n-1) \cdot (m-1)$  Kanten.

Eine Breitensuche wird nach obigem Algorithmus bei jeder der insgesamt b Beobachtungen  $f(b_i)$ -mal benötigt, wobei  $f(b_i)$  die Anzahl der zu betrachtenden brennenden Felder bei Beobachtung  $b_i$  sei.

Eine Breitensuche besucht nach obigem Algorithmus höchstens  $n \cdot m - f(b_i)$  Felder; die Breitensuchen haben also eine Laufzeit von  $\mathcal{O}(f(b_i) \cdot (2 \cdot n \cdot m - f(b_i)))$ . Es ist leicht zu erkennen, dass die Funktion F(x) = x(a-x) das Maximum an der Stelle  $x_{max} = \frac{a}{2}$  hat. Somit gilt  $\mathcal{O}(f(b_i) \cdot (2 \cdot n \cdot m - f(b_i))) = \mathcal{O}(\frac{nm}{2}(2nm - \frac{nm}{2})) = \mathcal{O}(\frac{3n^2m^2}{4}) = \mathcal{O}(n^2m^2)$  Es ergibt sich eine Gesamtlaufzeit von  $\mathcal{O}(n^2 \cdot m^2 \cdot b)$ . Mit  $b = \mathcal{O}(n \cdot m)$  ergibt sich eine (wohl sehr grobe) obere Schranke der Laufzeit von  $\mathcal{O}(n^3 \cdot m^3)$ .

Mit diesem Algorithmus lassen sich also Lösungen für Wälder gut berechnen, deren Dimensionen 200 nicht überschreiten, bei denen also  $\max n, m \le 200$ .

# 1.2 Umsetzung

Für die Umsetzung habe ich die Sprache C++ verwendet. Dabei habe ich sowohl den Brute-Force-Ansatz als auch die Heuristik implementiert.

Zunächst habe ich mir für Wälder eine Klasse Woods geschrieben. Deren Deklaration findet sich in der Datei Woods.h, die Implementierung in der Datei Woods.cpp. Jeder Wald hat dabei eine Breite (Width) und eine Höhe (Height).

Dabei benutzen Wälder für die Representierung eines Feldes einen FIELDSTATE, welcher als char definiert ist. <sup>3</sup> Dabei kann ein FIELDSTATE einen oder mehrere, ebenfalls definierter, Zustände annehmen. Dabei handelt es sich um die in der Lösungsidee beschriebenen Zustände eines Feldes, EMPTY, BURNABLE, BURNED, WATERED und COAL.

Ein Wald hält sich nun ein 2-dimensional, variabel großes Feld von FIELDSTATEs, der eigentliche Wald.

Durch geschickte Operatorenüberladung und geeigntete Akzessormethoden können diese Attribute vollständig gekapselt werden.

Der eigentliche Algorithmus findet sich in der Datei Buschfeuer.cpp; die Ein- und Ausgabe steht in der Datei Buschfeuer.h

Das Lesen der Eingabe übernimmt die Prozedur parseInput, welche die Daten in eine globale Instanz der Klasse Woods Forest einliest.

Ist die Eingabe gelesen, werden aus dieser die zu Beginn brennenden Felder mithilfe der Funktion getInitialBurningFields ermittelt und dann gleich an die Prozedur simulateFire weitergereicht. Diese Prozedur simulateFire simuliert nun das Feuer und ermittelt die zu löschenden Felder unter Zuhilfenahme der Funktion getOptimalWaterSpot. Dabei wird nach jedem Löschvorgang eine Ausgabe getätigt, welche die zu löschende Position (oben links mit (0—0) beginnend) ausgibt. Auch wird unter Verwendung von ASCII-Escape-Sequenzen ein Bild in der Konsole angezeigt, welches den Wald darstellt.

Ist das Feuer gelöscht (kann es sich also nicht weiter ausbreiten), wird dem NUtzer eine Meldung ausgegeben, wie viele Felder verbrannten und wie viele Felder verbrannt und gelöscht wurden. (Diese beiden Zahlen beschreiben disjunkte Mengen.) Auch hier wird wieder ein Bild erzeugt und ausgegeben.

<sup>&</sup>lt;sup>3</sup>Das Wort "definiert" ist durchaus ernst zu nehmen, da es hier beschreiben soll, dass etwas mittels #define "gelöst" wurde.

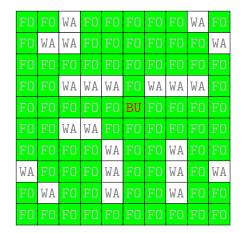
# 1.3 Beispiele

# 1.3.1 Beispiel 0

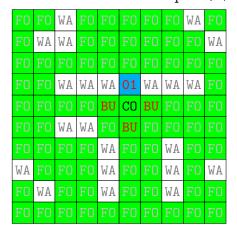
Die ist das Beispiel aus der Aufgabenstellung. Umgewandelt für mein Programm sieht diese Eingabe folgendermaßen aus<sup>4</sup>:

- 1 10 10
- 2 1101111101
- 3 1001111110
- 4 1111111111
- 5 1100010001
- 6 1111131111
- 7 1100111111
- 8 1111011011
- 9 0111011010
- 10 1011011011
- 11 111111111

Die Heuristik produziert folgende Ausgabe<sup>56</sup>:



--- At time 1: Water spot (5|3)



--- At time 2: Water spot (3|4)

<sup>&</sup>lt;sup>4</sup>Diese Eingabe finden Sie auch in der Datei O.in

<sup>&</sup>lt;sup>5</sup>Diese Ausgabe finden Sie auch in der Datei 0.out.tex; Eine Datei 0.out mit den ASCII-Escape-Sequenzen exisitert ebenfalls.

 $<sup>^6</sup>$ Um die ASCII-Escape-Sequenzen in TEX korrekt darzustellen, habe ich spezielle Ausgabemethoden geschrieben. Diese produzieren anstatt der ASCII-Sequenzen TEX-Befehle, welche optisch zu ähnlichen Ergebnissen führen.

FO	FO	WA	FO	FO	FO	FO	FO	WA	FO
FO	WA	WA	FO	FO	FO	FO	FO	FO	WA
FO									
FO	FO	WA	WA	WA	01	WA	WA	WA	FO
FO	FO	FO	02	CO	CO	CO	BU	FO	FO
FO	FO	WA	WA	BU	CO	BU	FO	FO	FO
FO	FO	FO	FO	WA	BU	FO	WA	FO	FO
WA	FO	FO	FO	WA	FO	FO	WA	FO	WA
FO	WA	FO	FO	WA	FO	FO	WA	FO	FO
FO									

--- At time 3: Water spot (8|4)

FO	FO	WA	FO	FO	FO	FO	FO	WA	FO
FO	WA	WA	FO	FO	FO	FO	FO	FO	WA
FO									
FO	FO	WA	WA	WA	01	WA	WA	WA	FO
FO	FO	FO	02	CO	CO	CO	CO	03	FO
FO	FO	WA	WA	CO	CO	CO	BU	FO	FO
FO	FO	FO	FO	WA	CO	BU	WA	FO	FO
WA	FO	FO	FO	WA	BU	FO	WA	FO	WA
FO	WA	FO	FO	WA	FO	FO	WA	FO	FO
FO									

--- At time 4: Water spot (8|5)

FO	FO	WA	FO	FO	FO	FO	FO	WA	FO
FO	WA	WA	FO	FO	FO	FO	FO	FO	WA
FO									
FO	FO	WA	WA	WA	01	WA	WA	WA	FO
FO	FO	FO	02	CO	CO	CO	CO	03	FO
FO	FO	WA	WA	CO	CO	CO	CO	04	FO
FO		FO	FO	WA	CO	CO	WA	FO	FO
WA	FO	FO	FO	WA	CO	BU	WA	FO	WA
FO	WA	FO	FO	WA	BU	FO	WA	FO	FO
FO									

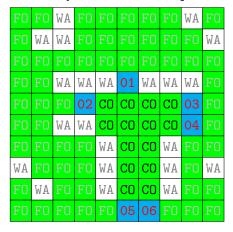
--- At time 5: Water spot (5|9)

FO	FO	WA	FO	FO	FO	FO	FO	WA	FO
FO	WA	WA	FO	FO	FO	FO	FO	FO	WA
FO									
FO	FO	WA	WA	WA	01	WA	WA	WA	FO
FO	FO	FO	02	CO	CO	CO	CO	03	FO
FO	FO	WA	WA	CO	CO	CO	CO	04	FO
FO	FO	FO	FO	WA	CO	CO	WA	FO	FO
WA	FO	FO	FO	WA	CO	CO	WA	FO	WA
FO	WA	FO	FO	WA	CO	BU	WA	FO	FO
FO	FO	FO	FO	FO	05	FO	FO	FO	FO

--- At time 6: Water spot (6|9)

FO	FO	WA	FO	FO	FO	FO	FO	WA	FO
FO	WA	WA	FO	FO	FO	FO	FO	FO	WA
FO									
FO	FO	WA	WA	WA	01	WA	WA	WA	FO
FO	FO	FO	02	CO	CO	CO	CO	03	FO
FO	FO	WA	WA	CO	CO	CO	CO	04	FO
FO	FO	FO	FO	WA	CO	CO	WA	FO	FO
WA	FO	FO	FO	WA	CO	CO	WA	FO	WA
FO	WA	FO	FO	WA	CO	CO	WA	FO	FO
FO	FO	FO	FO	FO	05	06	FO	FO	FO

--- And you'll find 14 pieces of coal and 6 pieces of watered coal



# Explanation:

WA --- EMPTY

FO --- BURNABLE

BU --- BURNED

CO --- COAL (doubly burned)

## --- WATERED at time ##

Fields can have more than 1 state.

Diese Ausgabe deckt sich auch mit der des Brute-Force-Ansatzes, weshalb ich dessen Ausgabe hier weglasse.

# 1.3.2 Beispiel 1

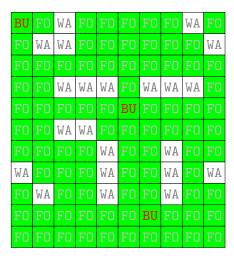
Eine Situation mit mehr als einem Feuer bei der ersten Beobachtung<sup>7</sup>:

- 1 10 11
- 2 3101111101
- 3 1001111110
- 4 1111111111
- 5 1100010001
- 6 1111131111
- 7 1100111111
- 8 1111011011

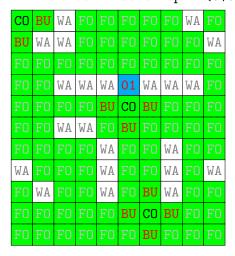
<sup>&</sup>lt;sup>7</sup>Diese Eingabe finden Sie auch in der Datei 1.in

- 9 0111011010
- 10 1011011011
- 11 1111113111
- 12 1111111111

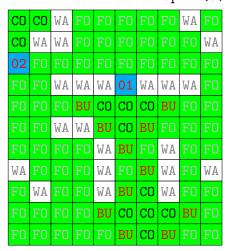
Mein Programm produziert folgende Ausgabe<sup>8</sup>:



--- At time 1: Water spot (5|3)

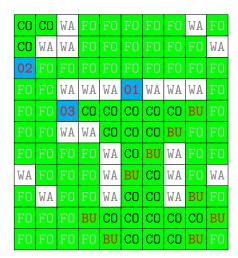


--- At time 2: Water spot (0|2)

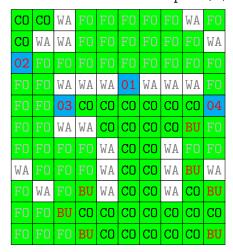


--- At time 3: Water spot (2|4)

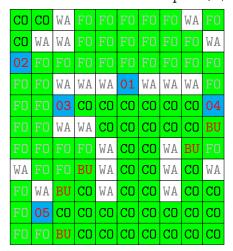
<sup>&</sup>lt;sup>8</sup>Diese Ausgabe finden Sie auch in der Datei 1.out.tex; Eine Datei 1.out mit den ASCII-Escape-Sequenzen exisitert ebenfalls.



--- At time 4: Water spot (9|4)

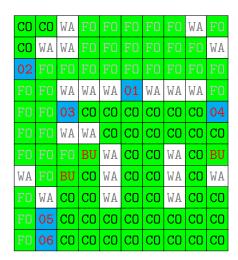


--- At time 5: Water spot (1|9)

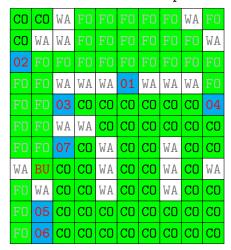


--- At time 6: Water spot (1|10)

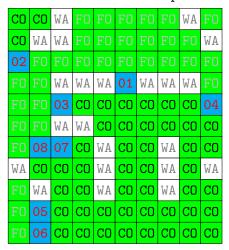
1



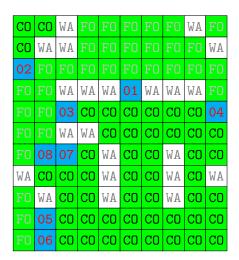
--- At time 7: Water spot (2|6)



--- At time 8: Water spot (1|6)



--- And you'll find 48 pieces of coal and 8 pieces of watered coal



#### Explanation:

WA --- EMPTY

FO --- BURNABLE

BU --- BURNED

CO --- COAL (doubly burned)

## --- WATERED at time ##

Fields can have more than 1 state.

#### 1.3.3Beispiel 2

9.

- 13 13
- 11111111111111
- 1000001000001
- 1011111111101
- 1011111111101
- 6 1011111111101
- 1011111111101
- 1111113111111 9

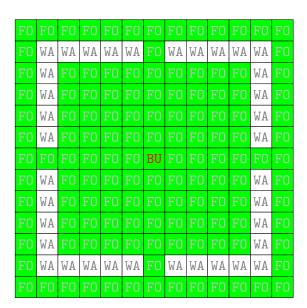
1011111111101

- 10 1011111111101
- 11 1011111111101
- 1011111111101 12
- 13 1000001000001
- 14 11111111111111

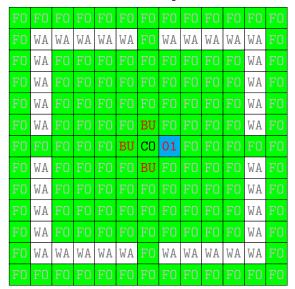
Mein Programm produziert folgende Ausgabe<sup>10</sup>:

<sup>&</sup>lt;sup>9</sup>Diese Eingabe finden Sie auch in der Datei 2.in

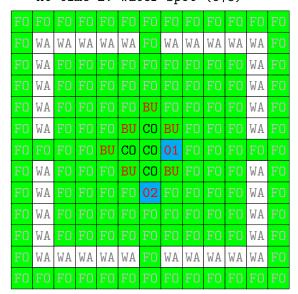
<sup>&</sup>lt;sup>10</sup>Diese Ausgabe finden Sie auch in der Datei 2.out.tex; Eine Datei 2.out mit den ASCII-Escape-Sequenzen exisitert ebenfalls.



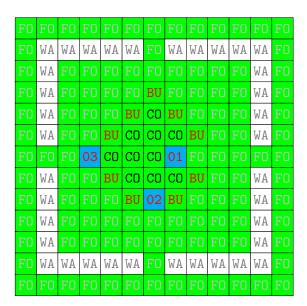
--- At time 1: Water spot (7|6)



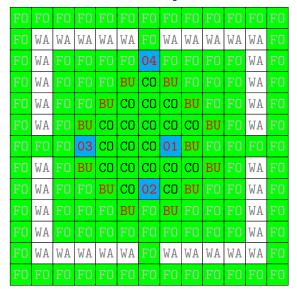
--- At time 2: Water spot (6|8)



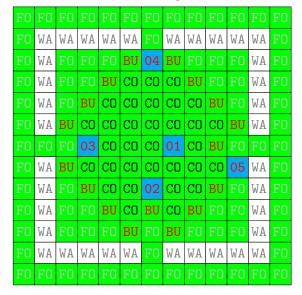
--- At time 3: Water spot (3|6)



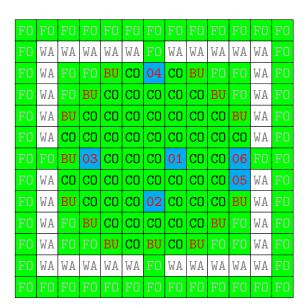
--- At time 4: Water spot (6|2)



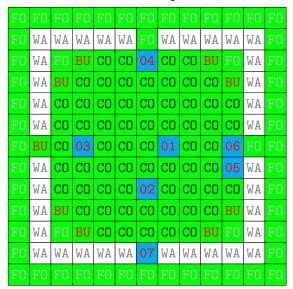
--- At time 5: Water spot (10|7)



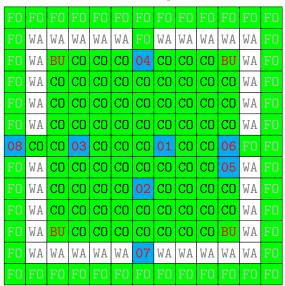
--- At time 6: Water spot (10|6)



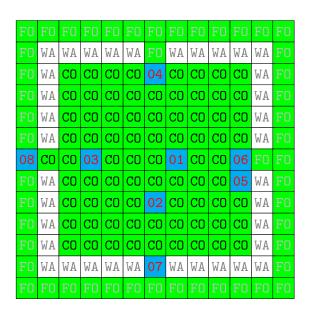
--- At time 7: Water spot (6|11)



--- At time 8: Water spot (0|6)



--- And you'll find 76 pieces of coal and 8 pieces of watered coal



#### Explanation:

WA --- EMPTY

--- BURNABLE

BU --- BURNED

CO --- COAL (doubly burned)

## --- WATERED at time ##

Fields can have more than 1 state.

# 1.3.4 Beispiel 3

1 100 100

16

Ein (etwas) größeres Beispiel.<sup>11</sup>:

```
2
7
9
10
13
```

<sup>&</sup>lt;sup>11</sup>Diese Eingabe finden Sie auch in der Datei 3.in

```
23
26
27
28
29
30
31
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
```

```
77
80
81
82
83
84
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
```

Mein Programm produziert folgende Ausgabe<sup>12</sup> Dabei hat die Berechnung wenige Sekunden in Anspruch genommen, sofern nicht die Ausgabe der ASCII-Escape-Sequenzen gefordert wird. Dies erhöhte die Laufzeit auf ca. 30s.:

```
At time 1: Water spot (45|51)

At time 2: Water spot (45|52)

At time 3: Water spot (46|50)

At time 4: Water spot (46|53)

At time 5: Water spot (47|49)

At time 6: Water spot (47|54)

At time 7: Water spot (48|48)

At time 8: Water spot (48|55)

At time 9: Water spot (49|47)

At time 10: Water spot (49|56)

At time 11: Water spot (50|46)

At time 12: Water spot (50|57)

At time 13: Water spot (51|45)

At time 14: Water spot (52|44)
```

<sup>&</sup>lt;sup>12</sup>Diese Ausgabe finden Sie auch in der Datei 3.out.tex2;

```
At time 16: Water spot (52|59)
At time 17: Water spot (53|43)
At time 18: Water spot (53|60)
At time 19: Water spot (54|42)
At time 20: Water spot (54|61)
At time 21: Water spot (55|41)
At time 22: Water spot (55|62)
At time 23: Water spot (56|40)
At time 24: Water spot (56|63)
At time 25: Water spot (57|39)
At time 26: Water spot (57|64)
At time 27: Water spot (58|38)
At time 28: Water spot (58|65)
At time 29: Water spot (59|37)
At time 30: Water spot (59|66)
At time 31: Water spot (60|36)
At time 32: Water spot (60|67)
At time 33: Water spot (61|35)
At time 34: Water spot (61|68)
At time 35: Water spot (62|34)
At time 36: Water spot (62|69)
At time 37: Water spot (63|33)
At time 38: Water spot (63|70)
At time 39: Water spot (64|32)
At time 40: Water spot (64|71)
At time 41: Water spot (65|31)
At time 42: Water spot (65|72)
At time 43: Water spot (66|30)
At time 44: Water spot (66|73)
At time 45: Water spot (67|29)
At time 46: Water spot (67|74)
At time 47: Water spot (68|28)
At time 48: Water spot (68|75)
At time 49: Water spot (69|27)
At time 50: Water spot (69|76)
At time 51: Water spot (70|26)
At time 52: Water spot (70|77)
At time 53: Water spot (71|25)
```

```
At time 54: Water spot (71|78)
At time 55: Water spot (72|24)
At time 56: Water spot (72|79)
At time 57: Water spot (73|23)
At time 58: Water spot (73|80)
At time 59: Water spot (74|22)
At time 60: Water spot (74|81)
At time 61: Water spot (75|21)
At time 62: Water spot (75|82)
At time 63: Water spot (76|20)
At time 64: Water spot (76|83)
At time 65: Water spot (77|19)
At time 66: Water spot (77|84)
At time 67: Water spot (78|18)
At time 68: Water spot (78|85)
At time 69: Water spot (79|17)
At time 70: Water spot (79|86)
At time 71: Water spot (80|16)
At time 72: Water spot (80|87)
At time 73: Water spot (81|15)
At time 74: Water spot (81|88)
At time 75: Water spot (82|14)
At time 76: Water spot (82|89)
At time 77: Water spot (83|13)
At time 78: Water spot (83|90)
At time 79: Water spot (84|12)
At time 80: Water spot (84|91)
At time 81: Water spot (85|11)
At time 82: Water spot (85|92)
At time 83: Water spot (86|10)
At time 84: Water spot (86|93)
At time 85: Water spot (87|9)
At time 86: Water spot (87|94)
At time 87: Water spot (88|8)
At time 88: Water spot (88|95)
At time 89: Water spot (89|7)
At time 90: Water spot (89|96)
At time 91: Water spot (90|6)
```

```
At time 92: Water spot (90|97)
At time 93: Water spot (91|5)
At time 94: Water spot (91|98)
At time 95: Water spot (92|4)
At time 96: Water spot (92|99)
At time 97: Water spot (93|3)
At time 98: Water spot (93|2)
At time 99: Water spot (93|1)
At time 100: Water spot (93|0)
And you'll find 6948 pieces of coal and 100 pieces of watered coal
```

# 1.4 Quelltext

#### Woods.h

```
#include <vector>
^2
3 #define FIELDSTATE
                            char
4 #define EMPTY
5 #define BURNABLE
                            1
6 #define BURNED
7 #define WATERED
  #define COAL
9
  class Woods{
10
11
    private:
12
     int Width, Height;
13
     std::vector<std::vector<FIELDSTATE> > Fields;
14
15
  public:
16
     Woods(int width, int height);
17
     int width() const;
18
19
     int height() const;
20
21
     FIELDSTATE& operator() (int x, int y);
     FIELDSTATE operator() (int x, int y) const;
22
23 };
```

### Woods.cpp

```
1 #include <vector>
2 #include <cstdio>
3
4 #include "Woods.h"
5
6 FIELDSTATE ERROR = -1;
7
```

Woods::Woods(int width, int height) : Width(width),

```
Height(height) {
9
     Fields.assign(height, std::vector<FIELDSTATE>(width, 0));
10 }
11
12 int Woods::width() const { return this->Width; }
  int Woods::height() const { return this->Height; }
13
14
15 FIELDSTATE& Woods::operator() (int x, int y) {
16
     if (x < 0 | | y < 0 | | x >= width() | | y >= height()){
       printf("OUT OF BOUNDS 1");
17
18
       return ERROR;
19
     }
20
     return this->Fields[y][x];
21 }
22 FIELDSTATE Woods::operator() (int x, int y) const {
23
     if (x < 0 | | y < 0 | | x >= width() | | y >= height()){
       printf("OUT OF BOUNDS 2");
24
25
       return ERROR;
26
     }
27
     return this->Fields[y][x];
28 }
   Buschfeuer.h Dies ist die Ein- und Ausgabe; sowie einige Definitionen.
1 #include <cstdio>
2 #include <vector>
3
4 #include "Woods.h"
5
6 typedef std::pair<int,int> PII;
7
  const int oo = (1 << 29);</pre>
                                                                  11
      The infinity
9
   Woods Forest(0, 0);
10
11 struct Point {
12 public:
    int x, y;
13
14
     Point(int _x,int _y) : x(_x), y(_y) { }
15
  int dir [4] [2] = \{\{1,0\},\{0,1\},\{-1,0\},\{0,-1\}\};
16
17
18
  std::vector<Point> Solution;
  std::FILE* OUT;
      // The file to mirror the output to
20
  void (*printSolution)(std::FILE*, bool);
21
22 //BEGIN OF INPUT
23 void parseInput(std::FILE* f) {
24
     int acFieldWidth, acFieldHeight;
25
     std::fscanf(f, "%i %i\n",&acFieldWidth, &acFieldHeight);
26
```

```
27
     Forest = Woods(acFieldWidth, acFieldHeight);
28
29
     for(int i = 0; i < acFieldHeight; ++i){</pre>
30
       for(int j= 0; j < acFieldWidth; ++j){</pre>
31
          char c;
32
         std::fscanf(f, "%c",&c);
         c -= '0';
33
34
         Forest(j, i) = (FIELDSTATE) c;
35
36
       if(i < acFieldHeight-1)</pre>
          std::fscanf(f, "\n");
37
38
     }
39 }
40
  //END OF INPUT
  //BEGIN OF OUTPUT
41
   void printSolution_TEX(std::FILE* f, bool finalOut) {
42
43
     std::fprintf(f, "\\\\n");
44
45
     std::fprintf(f, "\\begin{tikzpicture}\n");
     std::fprintf(f, "\\tikzset{square matrix/.style={\n");
46
     std::fprintf(f, "matrix of nodes,\n");
47
48
     std::fprintf(f, "column sep=-\\pgflinewidth, row
        sep=-\\pgflinewidth,\n");
     std::fprintf(f, "nodes={draw,\n");
49
     std::fprintf(f, "minimum height=#1,\n");
50
     std::fprintf(f, "anchor=center,\n");
51
52
     std::fprintf(f, "text width=#1,\n");
     std::fprintf(f, "align=center,\n");
53
     std::fprintf(f, "inner sep=0pt\n");
54
     std::fprintf(f, "},\n");
55
     std::fprintf(f, "},\n");
56
     std::fprintf(f, "square matrix/.default=1.2cm\n");
57
     std::fprintf(f, "}\n");
58
59
60
     std::fprintf(f, "\\matrix[square matrix=1.4em] {\n");
     for(int j= 0; j < Forest.height(); ++j) {</pre>
61
62
       for(int i= 0; i < Forest.width(); ++i) {</pre>
63
          if(i)
64
            std::fprintf(f," &");
65
66
            FIELDSTATE acField = Forest(i, j);
67
            if(acField == EMPTY)
68
              std::fprintf(f, "|[fill=white]|");
            else if(acField & WATERED)
69
              std::fprintf(f, "|[fill=cyan]|");
70
71
            else if(acField & BURNABLE)
72
              std::fprintf(f, "|[fill=green]|");
73
74
           if(acField & COAL)
              std::fprintf(f, "\\color[rgb]{0,0,0}");
75
76
            else if(acField & BURNED)
77
              std::fprintf(f, "\\color[rgb]{1,0,0}");
78
            else if(acField == EMPTY)
```

```
79
              std::fprintf(f, "\\color[gray]{0.5}");
80
            else if(acField & BURNABLE)
              std::fprintf(f, "\\color[gray]{0.75}");
81
82
            if(acField & WATERED){
83
              for (int t = 0; t < Solution.size(); ++t)</pre>
84
85
                 if (Solution[t].x == i && Solution[t].y == j) {
                   std::fprintf(f, "\\textbf{\\texttt{%02d}}}",t+1);
86
87
                   break;
88
                }
            }
89
90
            else if(acField & COAL)
91
              std::fprintf(f, "\\textbf{\\texttt{CO}}}");
92
            else if(acField & BURNED)
93
              std::fprintf(f, "\\textbf{\\texttt{BU}}}");
94
            else if(acField == EMPTY)
95
              std::fprintf(f, "\\texttt{WA}\");
            else if(acField & BURNABLE)
96
97
              std::fprintf(f, "\\texttt{FO}\");
98
               std::fprintf(f, "\\phantom{AA}");
99
100
          std::fprintf(f, "%%\n");
        }
101
102
103
        std::fprintf(f, "\\\\n");
104
      }
105
106
      std::fprintf(f, "};\n\\end{tikzpicture}\\\\n");
107
108
      if(finalOut){
109
        std::fprintf(f, "\\\\nExplanation:");
110
        std::fprintf(f,
           "\\\\n\\colorbox{white}{\\color[gray]{0.5}WA}
           EMPTY");
111
        std::fprintf(f,
           "\\\\n\\colorbox{green}{\\color[gray]{0.5}F0}
           BURNABLE");
112
        std::fprintf(f,
           "\\\n\\colorbox{white}{\\color[rgb]{1,0,0}\\textbf{BU}}
           --- BURNED");
        std::fprintf(f,
113
           "\\\\n\\colorbox{white}{\color[rgb]{0,0,0}\\textbf{CO}}
           --- COAL (doubly burned)");
        std::fprintf(f, "\\\\n\\colorbox{cyan}{\\#\\#}
114
           WATERED at time \\#\\#");
115
        std::fprintf(f, "\\\\nFields can have more than 1 state.");
      }
116
117 }
118
119 void printSolution_TERMINAL(std::FILE* f, bool finalOut) {
120
      fprintf(f, "\n");
121
      //The ASCII-magic starts here:
122
      for(int j= 0; j < Forest.height(); ++j) {</pre>
```

```
123
        for(int i= 0; i < Forest.width(); ++i) {</pre>
124
          FIELDSTATE acField = Forest(i, j);
125
          int waterval = 0;
126
          fprintf(f, "\x1b[s ");
127
          if (acField == EMPTY)
128
            std::fprintf(f, "\x1b[u\x1b[37;47mWA");
129
130
          if (acField & BURNABLE)
131
            std::fprintf(f, "\x1b[u\x1b[32;42mF0");
132
          if (acField & BURNED)
            std::fprintf(f, "\x1b[u\x1b[1;5;31m/\\");
133
134
          if (acField & COAL)
135
            std::fprintf(f, "\x1b[u\x1b[1;4;5;30m/\\");
136
          if (acField & WATERED)
            for (int t = 0; t < Solution.size(); ++t)</pre>
137
               if (Solution[t].x == i && Solution[t].y == j) {
138
139
                 std::fprintf(f, "\x1b[u\x1b[46m%02d", t+1);
140
                break:
              }
141
142
          std::fprintf(f, "\x1b[0;39;49m");
143
144
145
        std::fprintf(f, "\n");
      }
146
147
      if (finalOut) {// An Explanation shall be printed
148
149
        std::fprintf(f, "\nExplanation:");
        std::fprintf(f, "\n\x1b[37;47mWA\x1b[39;49m --- EMPTY");
150
        std::fprintf(f, "\n\x1b[32;42mF0\x1b[39;49m ---
151
        std::fprintf(f, "\n\x1b[1;5;31m/\\x1b[0;39m --- BURNED");
152
        std::fprintf(f, "\n\x1b[1;4;5;30m/\\x1b[0;39m ---
153
           (doubly burned)");
154
        std::fprintf(f, \n\x1b[46m##\x1b[0;39m --- WATERED at
           time ##");
155
        std::fprintf(f, "\nFields can have more than 1 state.");
156
      std::fprintf(f, "\n");
157
158
   }
159
160 void dontPrintSolution(std::FILE* f,bool finalOut) { return; }
161 //END OF OUTPUT
    Buschfeuer.cpp Dies ist die Implementierung der Heuristik.
 1 #include <cstdio>
 2 #include <vector>
 3 #include <queue>
 4 #include <set>
 5 #include <string>
 6 #include <cstring>
 7
   #include <algorithm>
 8
```

9 #include "Buschfeuer.h"

```
10
  Point getOptimalWaterSpot(std::vector < Point > & candidates) {
11
     std::queue<std::pair<PII,Point> > q;
12
                                        // ((distance | color) |
        Location)
     for(int i= 0; i < candidates.size(); ++i)</pre>
13
14
       q.push(std::pair <PII, Point > (PII(0,i), candidates[i]));
          // insert all the candidates as start points for the BFS
15
16
     std::vector<std::vector<std::set<int> > >
        visited(Forest.width(),
                                       // remember all nearest points
        first
17
       std::vector<std::set<int> >(Forest.height()));
18
     std::vector<std::vector<int> > shortDis(Forest.width(),
                    // shortest distant to any burning field
19
       std::vector<int>(Forest.height(),oo));
20
21
     //BFS to calculate shortest paths
22
     while(!q.empty()){
23
       std::pair<PII,Point> ac = q.front();
24
       Point acPoint = ac.second;
25
       int acDistance = ac.first.first;
26
       int acColor = ac.first.second;
27
28
       q.pop();
       if (visited[acPoint.x][acPoint.y].count(acColor))
29
30
       visited[acPoint.x][acPoint.y].insert(acColor);
31
       shortDis[acPoint.x][acPoint.y] =
32
          std::min(acDistance, shortDis[acPoint.x][acPoint.y]);
33
       for(int i= 0; i < 4; ++i){</pre>
34
35
         int newx = acPoint.x + dir[i][0];
         int newy = acPoint.y + dir[i][1];
                                                                    //
36
             calculate new field's indexes
37
         if(newx < 0 || newy < 0 || newy >= Forest.height() || newx
38
             >= Forest.width())
39
            continue:
                                                                    11
               new field is outside the woods
         if (Forest(newx, newy) != BURNABLE)
40
41
           continue;
                                                                    //
               Field is not of interest
42
         if(visited[newx][newy].count(acColor) == 0)
                                                                    //
43
            Don't compute things twice
           if(acDistance + 1 <= shortDis[newx][newy]){</pre>
44
45
              shortDis[newx][newy] = acDistance + 1;
              q.push(std::pair<PII,Point>(PII(acDistance +
46
                 1,acColor),Point(newx,newy)));
47
           }
48
       }
49
     }
```

```
50
51
     //determine the field to be watered
52
     std::vector<PII> waterval(candidates.size(),PII(0,0));
53
     std::vector<std::vector<int> > farthDist(candidates.size(),
        std::vector<int>(2*(Forest.width()+Forest.height()),0));
54
55
     //Count the number of fields that have an unique fire spot
        a.k.a. waterval
56
     //Reckon the farthest field
57
     for(int i= 0; i < Forest.width(); ++i)</pre>
       for(int j= 0; j < Forest.height(); ++j)</pre>
58
59
          if(visited[i][j].size() == 1){
60
            waterval[*visited[i][j].begin()].first++;
61
            farthDist[*visited[i][j].begin()][shortDis[i][j]]++;
62
63
     for(int i = 0; i < waterval.size(); ++i)</pre>
64
       waterval[i].second = i;
65
66
     std::sort(waterval.begin(), waterval.end(), std::greater <PII > ());
67
     //BEGIN HOTFIX
     for(int i = 0; i < waterval.size(); ++i)</pre>
68
69
       for(int j = 1; j < candidates.size(); ++j)</pre>
70
          if(farthDist[waterval[i].second][j] > 1)
            return candidates[waterval[i].second];
71
     //END HOTFIX
72
73
74
     return candidates[waterval[0].second];
75 }
76
77
   std::vector<Point>& getInitialBurningFields() {
78
     static std::vector<Point> burnedFields;
79
80
     for(int i = 0; i < Forest.height(); ++i)</pre>
81
       for(int j= 0; j < Forest.width(); ++j)</pre>
82
          if(Forest(j, i) & BURNED){
            burnedFields.push_back(Point(j, i));
83
            printf("Initially burning: (%i|%i)\n",j, i);
84
85
86
     return burnedFields;
  }
87
88
89
  void simulateFire(const std::vector<Point>&
      initiallyBurningFields) {
     std::vector<Point> burnedFields = initiallyBurningFields;
90
91
     if(printSolution != dontPrintSolution)
92
       printSolution_TERMINAL(stdout, false);
     if (OUT != 0)
93
94
       printSolution(OUT, false);
95
96
     int time = 0;
97
     while(!burnedFields.empty()) {
                                                                   //
        Simulate as long as there's still fire in the world
```

```
98
        std::vector<Point> newBurnedFields;
           // The burning fields at the next point of time
99
100
        //Calculate the new burning fields
        for(size_t i = 0; i < burnedFields.size(); ++i){</pre>
101
          int acx = burnedFields[i].x;
102
103
          int acy = burnedFields[i].y;
104
105
          if(Forest(acx, acy) & WATERED)
            continue:
                                                                    11
106
                The field got watered and does not spread fire
          Forest(acx, acy) |= COAL;
                                                                    //
107
             Field burned down to coal...
108
109
          for(int j = 0; j < 4; ++j) {
110
            int newx = acx + dir[j][0];
111
            int newy = acy + dir[j][1];
112
            if(newx < 0 || newy < 0 || newy >= Forest.height() ||
113
                newx >= Forest.width())
114
               continue:
                                                                    //
                  new field is outside the woods
            if (Forest(newx, newy) == BURNABLE){
115
               Forest(newx, newy) |= BURNED;
                                                                    //
116
                  Field starts burning
117
               newBurnedFields.push_back(Point(newx,newy));
118
               printf(" From now on burning: (%i|%i)\n",newx,newy);
119
   //
           // log the happenings
120
          }
121
122
        }
123
        if (newBurnedFields.empty())
                                                                   //
           Nothing to water, all plants happy...
124
            break:
125
126
        burnedFields = newBurnedFields;
127
128
        Point toWater = getOptimalWaterSpot(newBurnedFields); //
           Determine the field to water
                                                                   // ...
129
        Forest(toWater.x, toWater.y) |= WATERED;
           and water it
130
        Solution.push_back(toWater);
131
132
133
        //Output / mirror the partial solution
134
135
        std::printf("---\nAt time %i: Water spot
           (\%i \mid \%i) \setminus n'', ++time, toWater.x, toWater.y);
        if(printSolution != dontPrintSolution)
136
137
          printSolution_TERMINAL(stdout, false);
138
        if (OUT) {
139
```

```
140
          std::fprintf(OUT, "---\nAt time %i: Water spot
              (%i|%i)\n",time,toWater.x,toWater.y);
          printSolution(OUT, false);
141
142
143
144
         printf("Fire died.\n");
145
   //
146
147
      //Count the total number of burned or coaled
148
      int wcnt = 0. ccnt = 0:
      for(int i= 0; i < Forest.width(); ++i)</pre>
149
150
        for(int j= 0; j < Forest.height(); ++j)</pre>
151
          if(Forest(i, j) & WATERED)
152
            wcnt++;
153
          else if(Forest(i, j) & COAL)
154
            ccnt++;
155
      //Output / Mirror the solution
156
      std::printf("---\nAnd you'll find %i pieces of coal and %i
157
         pieces of watered coal\n",ccnt,wcnt);
      if(printSolution != dontPrintSolution)
158
159
        printSolution_TERMINAL(stdout, true);
      if (OUT) {
160
        std::fprintf(OUT, "---\nAnd you'll find %i pieces of coal
161
           and %i pieces of watered coal\n",ccnt,wcnt);
162
        printSolution(OUT, true);
163
164 }
165
166
    int main(int argc, char** argv){
      if (argc > 1) {
167
        std::freopen(argv[1], "r", stdin);
168
169
        std::printf("Using %s as input.\n", argv[1]);
170
      }
171
      if (argc > 2){
        printf("Mirroring output to %s.\n", argv[2]);
172
173
        if (strstr(argv[2], ".tex2")) {
174
          std::printf("I reckon you want me to produce some
             graphicless TeX stuff...\n");
175
          printSolution = dontPrintSolution;
176
        else if (strstr(argv[2], ".tex")) {
177
178
          std::printf("I reckon you want me to produce some TeX
             stuff...\n");
179
          printSolution = printSolution_TEX;
180
        else if (strstr(argv[2], ".raw")) {
181
          std::printf("I reckon you want me to surpress
182
             graphics...\n");
          printSolution = dontPrintSolution;
183
184
        }
185
        else
186
          printSolution = printSolution_TERMINAL;
```

9

10

#include "Buschfeuer.h"

11 int solve(vector < Point > )

```
OUT = std::fopen(argv[2], "w");
187
      }
188
189
      else{
190
        OUT = 0;
        printSolution = printSolution_TERMINAL;
191
192
193
      parseInput(stdin);
194
195
      simulateFire(getInitialBurningFields());
196 }
    Buschfeuer2.cpp Dies ist die Implementierung des Brute-Force-Ansatzes.
 1 #include <cstdio>
 2 #include <vector>
 3 #include <queue>
 4 #include <set>
 5 #include <string>
 6 #include <cstring>
   #include <algorithm>
```