

32. Bundeswettbewerb Informatik, Runde 2

Ausarbeitungen zu den Aufgaben „Buschfeuer“ und „Lebenslinien“

Philip Wellnitz

Vorwort

Inhaltsverzeichnis

1	Aufgabe 1 - Buschfeuer	3
1.1	Lösungsidee	3
1.1.1	Korrektheit	6
1.1.2	Laufzeitanalyse	7
1.2	Umsetzung	8
1.3	Beispiele	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	Quelltext	24

1 Aufgabe 1 - Buschfeuer

1.1 Lösungsidee

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

BRENNBAR Das Stück Land ist in 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 seien diejenigen Felder, welche bei mindestens 2 Beobachtungen brennend waren, als *verkohlt* bezeichnet.

Nach der Feuerausbreitung 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 Felder 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 unmittelbar 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 *kleinsten 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 kleinsten 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 bestimmt ist. Löscht man F , so wird, wie schon gesehen, die minimale Lebenszeit aller 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 μ effizient zu bestimmen. Da ein Wald eine rechteckige Gitterform besitzt, ist der kürzeste Abstand zwischen 2 Feldern 1, genau 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 μ 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
4    brennendeFelder := null; //1D-Container für Positionen
    brennender Felder
5    for (i = 0..Wald.Höhe())
6      for (j = 0..Wald.Breite())
7        if (Wald[i,j] == BRENNEND)
8          brennendeFelder.Add((i,j)); //Gefundene Position
          hinzufügen
9
10   return brennendeFelder; //Alle gefundenen Positionen
      zurückgeben
11 }
12
13 NächsteBeobachtung(aktBrennendeFelder) { //Ermittelt die bei der
    nächsten Beobachtung brennenden Felder, aus den Feldern, die
    aktuell brennen

```

```
14 neuBrennendeFelder := null;
15 for all((x;y) from aktBrennendeFelder)
16     if(Wald[x,y] == GELÖSCHT)
17         continue; //Feld kann kein Feuer verteilen
18
19     Wald[x,y] := VERKOHLT; //2 mal brennende Felder sind verkohlt
20     for all((x';y') from Umgebung((x;y)))
21         if(Wald[x',y'] == BRENNBAR)
22             neuBrennendeFelder.Add((x',y')); //Gefundene Position
                hinzufügen
23             Wald[x',y'] := BRENNEND; //Wald beginnt zu brennen
24
25     return neuBrennendeFelder;
26 }
27
28 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())
32         Fülle kleinsterAbstand[i] mithilfe einer Breitensuche
33
34     anzEindeutigKleinstAbstände := null;
35
36     for (i = 0..Wald.Höhe())
37         for (j = 0..Wald.Breite())
38             if(Es ex. k mit kleinsterAbstand[k][i,j] eindeutiges
                Minimum für alle mögliche k)
39                 anzEindeutigKleinstAbstände[k]++;
40
41     return aktBrennendeFelder[k, sodass
        anzEindeutigKleinstAbstände[k] maximal];
42 }
43
44 SimuliereFeuer() { //Die eigentliche Berechnung
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())
49             break; //Keine Felder brennen mehr
50
51     Wald[GetOptBewässerungspunkt(aktBrennendeFelder)] := GELÖSCHT;
        //Lösche das aktuell beste Feld
52 }
```

1.1.1 Korrektheit

Wie schon beschrieben, wird bei jeder Beobachtung das für diese Beobachtung nach μ 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 μ 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 μ optimalste Feld gelöscht:

Die Löschung nach dem Algorithmus:

FO	FO	FO	FO	FO	FO	FO	FO	FO	FO
FO	FO	FO	FO	FO	FO	FO	FO	FO	FO
FO	FO	FO	FO	FO	FO	FO	FO	FO	FO
FO	FO	FO	WA	FO	WA	FO	FO	FO	FO
FO	FO	FO	WA	BU	WA	FO	FO	FO	FO
FO	FO	WA	FO	FO	FO	WA	FO	FO	FO
FO	FO	WA	FO	FO	FO	WA	FO	FO	FO
FO	FO	WA	FO	FO	FO	WA	FO	FO	FO
FO	FO	WA	FO	FO	FO	WA	FO	FO	FO
FO	FO	FO	WA	WA	WA	FO	FO	FO	FO
FO	FO	FO	FO	FO	FO	FO	FO	FO	FO

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

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

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

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

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

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

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

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

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

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

Explanation:

WA --- EMPTY
 FO --- BURNABLE
 BU --- BURNED
 CO --- COAL (doubly burned)
 ## --- WATERED at time ##
 Fields can have more than 1 state.

Eine bessere Löschung:

FO	FO	FO	FO	FO	FO	FO	FO	FO	FO
FO	FO	FO	FO	FO	FO	FO	FO	FO	FO
FO	FO	FO	FO	FO	FO	FO	FO	FO	FO
FO	FO	FO	WA	FO	WA	FO	FO	FO	FO
FO	FO	FO	WA	BU	WA	FO	FO	FO	FO
FO	FO	WA	FO	FO	FO	WA	FO	FO	FO
FO	FO	WA	FO	FO	FO	WA	FO	FO	FO
FO	FO	WA	FO	FO	FO	WA	FO	FO	FO
FO	FO	FO	WA	WA	WA	FO	FO	FO	FO
FO	FO	FO	FO	FO	FO	FO	FO	FO	FO

--- At time 1: Water spot (4|5)

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

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

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

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

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

Explanation:

WA --- EMPTY
 FO --- BURNABLE
 BU --- BURNED
 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 Ergebnisse¹. Zum Vergleich habe ich den Brute-Force-Ansatz implementiert, der garantiert optimale Lösungen liefert.

1.1.2 Laufzeitanalyse

Der Brute-Force-Ansatz probiert alle Möglichkeiten an verschiedenen Lösungen 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 Lösungen der Lösung². Dieser Ansatz hat also eine exponentielle Laufzeit, im Gegensatz zu der Heuristik wie im Folgenden gezeigt wird.

¹Siehe dazu Sektion Beispiele

²Es 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 $\max n, m \leq 200$.

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.³ 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 geeignete 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.

³Das 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 aus⁴:

```

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

```

Die Heuristik produziert folgende Ausgabe⁵⁶:

FO	FO	WA	FO	FO	FO	FO	FO	WA	FO
FO	WA	WA	FO	FO	FO	FO	FO	FO	WA
FO	FO	FO	FO	FO	FO	FO	FO	FO	FO
FO	FO	WA	WA	WA	FO	WA	WA	WA	FO
FO	FO	FO	FO	FO	BU	FO	FO	FO	FO
FO	FO	WA	WA	FO	FO	FO	FO	FO	FO
FO	FO	FO	FO	WA	FO	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	FO	FO	FO	FO	FO	FO	FO	FO	FO

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

FO	FO	WA	FO	FO	FO	FO	FO	WA	FO
FO	WA	WA	FO	FO	FO	FO	FO	FO	WA
FO	FO	FO	FO	FO	FO	FO	FO	FO	FO
FO	FO	WA	WA	WA	O1	WA	WA	WA	FO
FO	FO	FO	FO	BU	CO	BU	FO	FO	FO
FO	FO	WA	WA	FO	BU	FO	FO	FO	FO
FO	FO	FO	FO	WA	FO	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	FO	FO	FO	FO	FO	FO	FO	FO	FO

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

⁴Diese Eingabe finden Sie auch in der Datei 0.in

⁵Diese Ausgabe finden Sie auch in der Datei 0.out.tex; Eine Datei 0.out mit den ASCII-Escape-Sequenzen existiert ebenfalls.

⁶Um die ASCII-Escape-Sequenzen in T_EX korrekt darzustellen, habe ich spezielle Ausgabemethoden geschrieben. Diese produzieren anstatt der ASCII-Sequenzen T_EX-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	FO	FO	FO	FO	FO	FO	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	FO	FO	FO	FO	FO	FO	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	FO	FO	FO	FO	FO	FO	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	FO	FO	FO	FO	FO	FO	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	FO	FO	FO	FO	FO	FO	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	BU	WA	FO	WA
FO	WA	FO	FO	WA	BU	FO	WA	FO	FO
FO	FO	FO	FO	FO	FO	FO	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	FO	FO	FO	FO	FO	FO	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	FO	FO	FO	FO	FO	FO	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

FO	FO	WA	FO	FO	FO	FO	FO	WA	FO
FO	WA	WA	FO	FO	FO	FO	FO	FO	WA
FO	FO	FO	FO	FO	FO	FO	FO	FO	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

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 weglassen.

1.3.2 Beispiel 1

Eine Situation mit mehr als einem Feuer bei der ersten Beobachtung⁷:

```

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

```

⁷Diese Eingabe finden Sie auch in der Datei 1.in

```

9  0111011010
10 1011011011
11 1111113111
12 1111111111

```

Mein Programm produziert folgende Ausgabe⁸:

BU	FO	WA	FO	FO	FO	FO	FO	WA	FO
FO	WA	WA	FO	FO	FO	FO	FO	FO	WA
FO	FO	FO	FO	FO	FO	FO	FO	FO	FO
FO	FO	WA	WA	WA	FO	WA	WA	WA	FO
FO	FO	FO	FO	FO	BU	FO	FO	FO	FO
FO	FO	WA	WA	FO	FO	FO	FO	FO	FO
FO	FO	FO	FO	WA	FO	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	FO	FO	FO	FO	FO	BU	FO	FO	FO
FO	FO	FO	FO	FO	FO	FO	FO	FO	FO

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

CO	BU	WA	FO	FO	FO	FO	FO	WA	FO
BU	WA	WA	FO	FO	FO	FO	FO	FO	WA
FO	FO	FO	FO	FO	FO	FO	FO	FO	FO
FO	FO	WA	WA	WA	O1	WA	WA	WA	FO
FO	FO	FO	FO	BU	CO	BU	FO	FO	FO
FO	FO	WA	WA	FO	BU	FO	FO	FO	FO
FO	FO	FO	FO	WA	FO	FO	WA	FO	FO
WA	FO	FO	FO	WA	FO	FO	WA	FO	WA
FO	WA	FO	FO	WA	FO	BU	WA	FO	FO
FO	FO	FO	FO	FO	BU	CO	BU	FO	FO
FO	FO	FO	FO	FO	FO	BU	FO	FO	FO

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

CO	CO	WA	FO	FO	FO	FO	FO	WA	FO
CO	WA	WA	FO	FO	FO	FO	FO	FO	WA
O2	FO	FO	FO	FO	FO	FO	FO	FO	FO
FO	FO	WA	WA	WA	O1	WA	WA	WA	FO
FO	FO	FO	BU	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	BU	WA	FO	WA
FO	WA	FO	FO	WA	BU	CO	WA	FO	FO
FO	FO	FO	FO	BU	CO	CO	CO	BU	FO
FO	FO	FO	FO	FO	BU	CO	BU	FO	FO

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

⁸Diese Ausgabe finden Sie auch in der Datei 1.out.tex; Eine Datei 1.out mit den ASCII-Escape-Sequenzen existiert ebenfalls.

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

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

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

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

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

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

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

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

CO	CO	WA	FO	FO	FO	FO	FO	WA	FO
CO	WA	WA	FO	FO	FO	FO	FO	FO	WA
02	FO	FO	FO	FO	FO	FO	FO	FO	FO
FO	FO	WA	WA	WA	01	WA	WA	WA	FO
FO	FO	03	CO	CO	CO	CO	CO	CO	04
FO	FO	WA	WA	CO	CO	CO	CO	CO	CO
FO	FO	07	CO	WA	CO	CO	WA	CO	CO
WA	BU	CO	CO	WA	CO	CO	WA	CO	WA
FO	WA	CO	CO	WA	CO	CO	WA	CO	CO
FO	05	CO	CO	CO	CO	CO	CO	CO	CO
FO	06	CO	CO	CO	CO	CO	CO	CO	CO

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

CO	CO	WA	FO	FO	FO	FO	FO	WA	FO
CO	WA	WA	FO	FO	FO	FO	FO	FO	WA
02	FO	FO	FO	FO	FO	FO	FO	FO	FO
FO	FO	WA	WA	WA	01	WA	WA	WA	FO
FO	FO	03	CO	CO	CO	CO	CO	CO	04
FO	FO	WA	WA	CO	CO	CO	CO	CO	CO
FO	08	07	CO	WA	CO	CO	WA	CO	CO
WA	CO	CO	CO	WA	CO	CO	WA	CO	WA
FO	WA	CO	CO	WA	CO	CO	WA	CO	CO
FO	05	CO	CO	CO	CO	CO	CO	CO	CO
FO	06	CO	CO	CO	CO	CO	CO	CO	CO

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

CO	CO	WA	FO	FO	FO	FO	FO	WA	FO
CO	WA	WA	FO	FO	FO	FO	FO	FO	WA
02	FO	FO	FO	FO	FO	FO	FO	FO	FO
FO	FO	WA	WA	WA	01	WA	WA	WA	FO
FO	FO	03	CO	CO	CO	CO	CO	CO	04
FO	FO	WA	WA	CO	CO	CO	CO	CO	CO
FO	08	07	CO	WA	CO	CO	WA	CO	CO
WA	CO	CO	CO	WA	CO	CO	WA	CO	WA
FO	WA	CO	CO	WA	CO	CO	WA	CO	CO
FO	05	CO	CO	CO	CO	CO	CO	CO	CO
FO	06	CO	CO	CO	CO	CO	CO	CO	CO

Explanation:

WA --- EMPTY

FO --- BURNABLE

BU --- BURNED

CO --- COAL (doubly burned)

--- WATERED at time

Fields can have more than 1 state.

1.3.3 Beispiel 2

9:

```

1  13 13
2  11111111111111
3  1000001000001
4  10111111111101
5  10111111111101
6  10111111111101
7  10111111111101
8  11111131111111
9  10111111111101
10 10111111111101
11 10111111111101
12 10111111111101
13 1000001000001
14 11111111111111

```

Mein Programm produziert folgende Ausgabe¹⁰:

⁹Diese Eingabe finden Sie auch in der Datei 2.in

¹⁰Diese Ausgabe finden Sie auch in der Datei 2.out.tex; Eine Datei 2.out mit den ASCII-Escape-Sequenzen existiert ebenfalls.

FO	FO	FO	FO	FO	FO	FO	FO	FO	FO	FO	FO	FO
FO	WA	WA	WA	WA	WA	FO	WA	WA	WA	WA	WA	FO
FO	WA	FO	FO	FO	FO	FO	FO	FO	FO	FO	WA	FO
FO	WA	FO	FO	FO	FO	FO	FO	FO	FO	FO	WA	FO
FO	WA	FO	FO	FO	FO	FO	FO	FO	FO	FO	WA	FO
FO	WA	FO	FO	FO	FO	FO	FO	FO	FO	FO	WA	FO
FO	FO	FO	FO	FO	FO	BU	FO	FO	FO	FO	FO	FO
FO	WA	FO	FO	FO	FO	FO	FO	FO	FO	FO	WA	FO
FO	WA	FO	FO	FO	FO	FO	FO	FO	FO	FO	WA	FO
FO	WA	FO	FO	FO	FO	FO	FO	FO	FO	FO	WA	FO
FO	WA	FO	FO	FO	FO	FO	FO	FO	FO	FO	WA	FO
FO	WA	FO	FO	FO	FO	FO	FO	FO	FO	FO	WA	FO
FO	WA	FO	FO	FO	FO	FO	FO	FO	FO	FO	WA	FO
FO	WA	WA	WA	WA	WA	FO	WA	WA	WA	WA	WA	FO
FO	FO	FO	FO	FO	FO	FO	FO	FO	FO	FO	FO	FO

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

FO	FO	FO	FO	FO	FO	FO	FO	FO	FO	FO	FO	FO
FO	WA	WA	WA	WA	WA	FO	WA	WA	WA	WA	WA	FO
FO	WA	FO	FO	FO	FO	FO	FO	FO	FO	FO	WA	FO
FO	WA	FO	FO	FO	FO	FO	FO	FO	FO	FO	WA	FO
FO	WA	FO	FO	FO	FO	FO	FO	FO	FO	FO	WA	FO
FO	WA	FO	FO	FO	FO	BU	FO	FO	FO	FO	WA	FO
FO	FO	FO	FO	FO	BU	CO	O1	FO	FO	FO	FO	FO
FO	WA	FO	FO	FO	FO	BU	FO	FO	FO	FO	WA	FO
FO	WA	FO	FO	FO	FO	FO	FO	FO	FO	FO	WA	FO
FO	WA	FO	FO	FO	FO	FO	FO	FO	FO	FO	WA	FO
FO	WA	FO	FO	FO	FO	FO	FO	FO	FO	FO	WA	FO
FO	WA	FO	FO	FO	FO	FO	FO	FO	FO	FO	WA	FO
FO	WA	WA	WA	WA	WA	FO	WA	WA	WA	WA	WA	FO
FO	FO	FO	FO	FO	FO	FO	FO	FO	FO	FO	FO	FO

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

FO	FO	FO	FO	FO	FO	FO	FO	FO	FO	FO	FO	FO
FO	WA	WA	WA	WA	WA	FO	WA	WA	WA	WA	WA	FO
FO	WA	FO	FO	FO	FO	FO	FO	FO	FO	FO	WA	FO
FO	WA	FO	FO	FO	FO	FO	FO	FO	FO	FO	WA	FO
FO	WA	FO	FO	FO	FO	BU	FO	FO	FO	FO	WA	FO
FO	WA	FO	FO	FO	BU	CO	BU	FO	FO	FO	WA	FO
FO	FO	FO	FO	BU	CO	CO	O1	FO	FO	FO	FO	FO
FO	WA	FO	FO	FO	BU	CO	BU	FO	FO	FO	WA	FO
FO	WA	FO	FO	FO	FO	O2	FO	FO	FO	FO	WA	FO
FO	WA	FO	FO	FO	FO	FO	FO	FO	FO	FO	WA	FO
FO	WA	FO	FO	FO	FO	FO	FO	FO	FO	FO	WA	FO
FO	WA	WA	WA	WA	WA	FO	WA	WA	WA	WA	WA	FO
FO	FO	FO	FO	FO	FO	FO	FO	FO	FO	FO	FO	FO

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

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

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

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

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

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

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

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

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

FO	FO	FO	FO	FO	FO	FO	FO	FO	FO	FO	FO	FO
FO	WA	WA	WA	WA	WA	FO	WA	WA	WA	WA	WA	FO
FO	WA	FO	BU	CO	CO	04	CO	CO	BU	FO	WA	FO
FO	WA	BU	CO	CO	CO	CO	CO	CO	CO	BU	WA	FO
FO	WA	CO	CO	CO	CO	CO	CO	CO	CO	CO	WA	FO
FO	WA	CO	CO	CO	CO	CO	CO	CO	CO	CO	WA	FO
FO	BU	CO	03	CO	CO	CO	01	CO	CO	06	FO	FO
FO	WA	CO	CO	CO	CO	CO	CO	CO	CO	05	WA	FO
FO	WA	CO	CO	CO	CO	02	CO	CO	CO	CO	WA	FO
FO	WA	BU	CO	CO	CO	CO	CO	CO	CO	BU	WA	FO
FO	WA	FO	BU	CO	CO	CO	CO	CO	BU	FO	WA	FO
FO	WA	WA	WA	WA	WA	07	WA	WA	WA	WA	WA	FO
FO	FO	FO	FO	FO	FO	FO	FO	FO	FO	FO	FO	FO

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

FO	FO	FO	FO	FO	FO	FO	FO	FO	FO	FO	FO	FO
FO	WA	WA	WA	WA	WA	FO	WA	WA	WA	WA	WA	FO
FO	WA	BU	CO	CO	CO	04	CO	CO	CO	BU	WA	FO
FO	WA	CO	CO	CO	CO	CO	CO	CO	CO	CO	WA	FO
FO	WA	CO	CO	CO	CO	CO	CO	CO	CO	CO	WA	FO
FO	WA	CO	CO	CO	CO	CO	CO	CO	CO	CO	WA	FO
08	CO	CO	03	CO	CO	CO	01	CO	CO	06	FO	FO
FO	WA	CO	CO	CO	CO	CO	CO	CO	CO	05	WA	FO
FO	WA	CO	CO	CO	CO	02	CO	CO	CO	CO	WA	FO
FO	WA	CO	CO	CO	CO	CO	CO	CO	CO	CO	WA	FO
FO	WA	BU	CO	CO	CO	CO	CO	CO	CO	BU	WA	FO
FO	WA	WA	WA	WA	WA	07	WA	WA	WA	WA	WA	FO
FO	FO	FO	FO	FO	FO	FO	FO	FO	FO	FO	FO	FO

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

[illegible]

[illegible]

Mein Programm produziert folgende Ausgabe¹² 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 (51|58)
At time 15: Water spot (52|44)

¹²Diese 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

```
1 #include <vector>
2
3 #define FIELDSTATE      char
4 #define EMPTY           0
5 #define BURNABLE        1
6 #define BURNED          2
7 #define WATERED         4
8 #define COAL            8
9
10 class Woods{
11     private:
12         int Width, Height;
13         std::vector<std::vector<FIELDSTATE> > Fields;
14
15     public:
16         Woods(int width, int height);
17
18         int width() const;
19         int height() const;
20
21         FIELDSTATE& operator() (int x, int y);
22         FIELDSTATE operator() (int x, int y) const;
23 };
```

Woods.cpp

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



```

8 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; }
13 int Woods::height() const { return this->Height; }
14
15 FIELDSTATE& Woods::operator() (int x, int y) {
16     if (x < 0 || y < 0 || x >= width() || y >= height()){
17         printf("OUT OF BOUNDS 1");
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()){
24         printf("OUT OF BOUNDS 2");
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
8 const int oo = (1 << 29); //
    The infinity
9 Woods Forest(0, 0);
10
11 struct Point {
12 public:
13     int x, y;
14     Point(int _x,int _y) : x(_x), y(_y) { }
15 };
16 int dir[4][2] = {{1,0},{0,1},{-1,0},{0,-1}};
17
18 std::vector<Point> Solution;
19 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){
30     for(int j= 0; j < acFieldWidth; ++j){
31         char c;
32         std::fscanf(f, "%c",&c);
33         c -= '0';
34         Forest(j, i) = (FIELDSTATE) c;
35     }
36     if(i < acFieldHeight-1)
37         std::fscanf(f, "\n");
38 }
39 }
40 //END OF INPUT
41 //BEGIN OF OUTPUT
42 void printSolution_TEX(std::FILE* f, bool finalOut) {
43     std::fprintf(f, "\\\\n");
44
45     std::fprintf(f, "\\begin{tikzpicture}\n");
46     std::fprintf(f, "\\tikzset{square matrix/.style={\n");
47     std::fprintf(f, "matrix of nodes,\n");
48     std::fprintf(f, "column sep=-\\pgflinewidth, row
        sep=-\\pgflinewidth,\n");
49     std::fprintf(f, "nodes={draw,\n");
50     std::fprintf(f, "minimum height=#1,\n");
51     std::fprintf(f, "anchor=center,\n");
52     std::fprintf(f, "text width=#1,\n");
53     std::fprintf(f, "align=center,\n");
54     std::fprintf(f, "inner sep=0pt\n");
55     std::fprintf(f, "},\n");
56     std::fprintf(f, "},\n");
57     std::fprintf(f, "square matrix/.default=1.2cm\n");
58     std::fprintf(f, "}\n");
59
60     std::fprintf(f, "\\matrix[square matrix=1.4em] {\n");
61     for(int j= 0; j < Forest.height(); ++j) {
62         for(int i= 0; i < Forest.width(); ++i) {
63             if(i)
64                 std::fprintf(f, " &");
65
66             FIELDSTATE acField = Forest(i, j);
67             if(acField == EMPTY)
68                 std::fprintf(f, "|[fill=white]|");
69             else if(acField & WATERED)
70                 std::fprintf(f, "|[fill=cyan]|");
71             else if(acField & BURNABLE)
72                 std::fprintf(f, "|[fill=green]|");
73
74             if(acField & COAL)
75                 std::fprintf(f, "\\color[rgb]{0,0,0}");
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)
81         std::fprintf(f, "\\color{gray}{0.75}");
82
83     if(acField & WATERED){
84         for (int t = 0; t < Solution.size(); ++t)
85             if (Solution[t].x == i && Solution[t].y == j) {
86                 std::fprintf(f, "\\textbf{\\texttt{%02d}}",t+1);
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}");
96     else if(acField & BURNABLE)
97         std::fprintf(f, "\\texttt{FO}");
98     else
99         std::fprintf(f, "\\phantom{AA}");
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, "\\n\\Explanation:");
110     std::fprintf(f,
111         "\\n\\colorbox{white}{\\color{gray}{0.5}WA} ---
112         EMPTY");
111     std::fprintf(f,
112         "\\n\\colorbox{green}{\\color{gray}{0.5}FO} ---
113         BURNABLE");
112     std::fprintf(f,
113         "\\n\\colorbox{white}{\\color{rgb}{1,0,0}\\textbf{BU}}
114         --- BURNED");
113     std::fprintf(f,
114         "\\n\\colorbox{white}{\\color{rgb}{0,0,0}\\textbf{CO}}
115         --- COAL (doubly burned)");
114     std::fprintf(f, "\\n\\colorbox{cyan}{\\#\\#} ---
115         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) {

```

```

123     for(int i= 0; i < Forest.width(); ++i) {
124         FIELDSTATE acField = Forest(i, j);
125         int waterval = 0;
126
127         fprintf(f, "\x1b[s ");
128         if (acField == EMPTY)
129             std::fprintf(f, "\x1b[u\x1b[37;47mWA");
130         if (acField & BURNABLE)
131             std::fprintf(f, "\x1b[u\x1b[32;42mFO");
132         if (acField & BURNED)
133             std::fprintf(f, "\x1b[u\x1b[1;5;31m/\");
134         if (acField & COAL)
135             std::fprintf(f, "\x1b[u\x1b[1;4;5;30m/\");
136         if (acField & WATERED)
137             for (int t = 0; t < Solution.size(); ++t)
138                 if (Solution[t].x == i && Solution[t].y == j) {
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
148 if (finalOut) { // An Explanation shall be printed
149     std::fprintf(f, "\nExplanation:");
150     std::fprintf(f, "\n\x1b[37;47mWA\x1b[39;49m --- EMPTY");
151     std::fprintf(f, "\n\x1b[32;42mFO\x1b[39;49m --- BURNABLE");
152     std::fprintf(f, "\n\x1b[1;5;31m/\x1b[0;39m --- BURNED");
153     std::fprintf(f, "\n\x1b[1;4;5;30m/\x1b[0;39m --- COAL
154                   (doubly burned)");
155     std::fprintf(f, "\n\x1b[46m#\x1b[0;39m --- WATERED at
156                   time ##");
157     std::fprintf(f, "\nFields can have more than 1 state.");
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
11 Point getOptimalWaterSpot(std::vector<Point>& candidates){
12     std::queue<std::pair<PII,Point> > q;
13                                     // ((distance | color) |
14                                     Location)
15     for(int i= 0; i < candidates.size(); ++i)
16         q.push(std::pair<PII,Point>(PII(0,i),candidates[i]));
17         // insert all the candidates as start points for the BFS
18
19     std::vector<std::vector<std::set<int> > >
20         visited(Forest.width(), // remember all nearest points
21         first
22     std::vector<std::set<int> >(Forest.height()));
23     std::vector<std::vector<int> > shortDis(Forest.width(),
24         // shortest distant to any burning field
25     std::vector<int>(Forest.height(),oo));
26
27     //BFS to calculate shortest paths
28     while(!q.empty()){
29         std::pair<PII,Point> ac = q.front();
30         Point acPoint = ac.second;
31         int acDistance = ac.first.first;
32         int acColor = ac.first.second;
33
34         q.pop();
35         if(visited[acPoint.x][acPoint.y].count(acColor))
36             continue;
37         visited[acPoint.x][acPoint.y].insert(acColor);
38         shortDis[acPoint.x][acPoint.y] =
39             std::min(acDistance,shortDis[acPoint.x][acPoint.y]);
40
41         for(int i= 0; i < 4; ++i){
42             int newx = acPoint.x + dir[i][0];
43             int newy = acPoint.y + dir[i][1]; //
44             calculate new field's indexes
45
46             if(newx < 0 || newy < 0 || newy >= Forest.height() || newx
47                 >= Forest.width())
48                 continue; //
49             new field is outside the woods
50             if (Forest(newx, newy) != BURNABLE)
51                 continue; //
52             Field is not of interest
53
54             if(visited[newx][newy].count(acColor) == 0) //
55                 Don't compute things twice
56                 if(acDistance + 1 <= shortDis[newx][newy]){
57                     shortDis[newx][newy] = acDistance + 1;
58                     q.push(std::pair<PII,Point>(PII(acDistance +
59                         1,acColor),Point(newx,newy)));
60                 }
61         }
62     }
63 }

```

```
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(),
54     std::vector<int>(2*(Forest.width()+Forest.height()),0));
55 //Count the number of fields that have an unique fire spot
56     a.k.a. waterval
57 //Reckon the farthest field
58 for(int i = 0; i < Forest.width(); ++i)
59     for(int j = 0; j < Forest.height(); ++j)
60         if(visited[i][j].size() == 1){
61             waterval[*visited[i][j].begin()].first++;
62             farthDist[*visited[i][j].begin()][shortDis[i][j]]++;
63         }
64 for(int i = 0; i < waterval.size(); ++i)
65     waterval[i].second = i;
66 std::sort(waterval.begin(),waterval.end(),std::greater<PII>());
67 //BEGIN HOTFIX
68 for(int i = 0; i < waterval.size(); ++i)
69     for(int j = 1; j < candidates.size(); ++j)
70         if(farthDist[waterval[i].second][j] > 1)
71             return candidates[waterval[i].second];
72 //END HOTFIX
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)
81         for(int j = 0; j < Forest.width(); ++j)
82             if(Forest(j, i) & BURNED){
83                 burnedFields.push_back(Point(j, i));
84 //                 printf("Initially burning: (%i|%i)\n",j, i);
85             }
86     return burnedFields;
87 }
88
89 void simulateFire(const std::vector<Point>&
90     initiallyBurningFields) {
91     std::vector<Point> burnedFields = initiallyBurningFields;
92     if(printSolution != dontPrintSolution)
93         printSolution_TERMINAL(stdout, false);
94     if (OUT != 0)
95         printSolution(OUT, false);
96
97     int time = 0;
98     while(!burnedFields.empty()) {
99         //
100         Simulate as long as there's still fire in the world
```

```

98     std::vector<Point> newBurnedFields;
99         // The burning fields at the next point of time
100
101     //Calculate the new burning fields
102     for(size_t i = 0; i < burnedFields.size(); ++i){
103         int acx = burnedFields[i].x;
104         int acy = burnedFields[i].y;
105
106         if(Forest(acx, acy) & WATERED)
107             continue; //
108             // The field got watered and does not spread fire
109         Forest(acx, acy) |= COAL; //
110         // Field burned down to coal...
111
112         for(int j = 0; j < 4; ++j) {
113             int newx = acx + dir[j][0];
114             int newy = acy + dir[j][1];
115
116             if(newx < 0 || newy < 0 || newy >= Forest.height() ||
117                newx >= Forest.width())
118                 continue; //
119                 // new field is outside the woods
120             if(Forest(newx, newy) == BURNABLE){
121                 Forest(newx, newy) |= BURNED; //
122                 // Field starts burning
123                 newBurnedFields.push_back(Point(newx,newy));
124
125                 // printf(" From now on burning: (%i|%i)\n",newx,newy);
126                 // log the happenings
127             }
128         }
129     }
130     if(newBurnedFields.empty()) //
131         // Nothing to water, all plants happy...
132         break;
133
134     burnedFields = newBurnedFields;
135
136     Point toWater = getOptimalWaterSpot(newBurnedFields); //
137     // Determine the field to water
138     Forest(toWater.x, toWater.y) |= WATERED; // ...
139     // and water it
140     Solution.push_back(toWater);
141
142     //Output / mirror the partial solution
143
144     std::printf("---\nAt time %i: Water spot
145                (%i|%i)\n",++time,toWater.x,toWater.y);
146     if(printSolution != dontPrintSolution)
147         printSolution_TERMINAL(stdout, false);
148
149     if (OUT) {

```

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



```
187     OUT = std::fopen(argv[2], "w");
188 }
189 else{
190     OUT = 0;
191     printSolution = printSolution_TERMINAL;
192 }
193
194 parseInput(stdin);
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>
7 #include <algorithm>
8
9 #include "Buschfeuer.h"
10
11 int solve(vector<Point> )
```