

32. Bundeswettbewerb Informatik, Runde 2

Ausarbeitungen zu den Aufgaben „Buschfeuer“ und „Lebenslinien“

Philip Wellnitz

Vorwort

Armer Korrektor!¹²

Auf den folgenden Seiten sind die vor meiner unendlichen Genialität strotzenden Lösungen der diesjährigen Zweitrundenaufgaben niedergeschrieben und zu bewundern. Die glorreichen Programme zu den einzelnen Aufgaben sollten sich auf einem normalen Rechner³ aus einem Terminal problemlos starten lassen; speziell für Aufgabe 1 eignet sich besonders eines, welches die göttlichen, also mir gleichen, ASCII-Escape-Sequenzen unterstützt. Weiterhin habe ich (noch) darauf verzichtet, meine überragenden Programmierfähigkeiten in wunderschönen Programmen im gut lesbaren ASM auszudrücken; auch habe ich kein C++ mit inline-ASM verwendet⁴.

Nach diesem mit übermäßiger Bescheidenheit glänzenden Vorwort möchte ich Ihnen nun viel Freude bei der Korrektur meiner Lösungen wünschen...
und eine erholsame Zeit danach.

Inhaltsverzeichnis

1 Aufgabe 1 - Buschfeuer	3
1.1 Lösungsidee	3
1.1.1 Korrektheit	6
1.1.2 Laufzeitanalyse	7
1.2 Umsetzung	7
1.3 Beispiele	8
1.3.1 Beispiel 0	8
1.3.2 Beispiel 1	10
1.3.3 Beispiel 2	14
1.4 Quelltext	18

¹Dieses Vorwort existiert nur, weil es scheinbar zur Struktur gehören muss. Außerdem wollte ich sowas auch mal schreiben...

²Und es ist essenziell, um meine ständige Präsenz in den *Perlen der Informatik* zu wahren...

³Das sei im Folgenden ein Rechner mit einem Linux-artigen OS

⁴Schade eigentlich.

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. Verallgemeinernd muss gezeigt werden, dass kein μ' existiert, welches bei mindestens 1 Beobachtung 1 anderes Feld als μ vorschlägt und bei der insgesamt weniger Felder abbrennen als bei μ ; dass μ also *optimal* ist.

Außerdem muss gezeigt werden, dass der Algorithmus terminiert. Da der Algorithmus jedoch nur brennende und nicht verkohlte Felder betrachtet und jedes brennendes Feld nach endlicher Zeit in den Zustand verkohlt übergeht, gibt es einen Zeitpunkt, ab dem alle einst brennenden Felder verkohlt sind. Dann gibt es jedoch keine Felder, auf denen der Algorithmus operieren kann, der Algorithmus terminiert dann, und somit immer.

Nach der Definition von μ wird dasjenige, beliebige Feld F_i aus allen möglichen Feldern $F_1..F_n$ zum Löschen ausgewählt, welches die minimale Lebenszeit von den meisten Feldern erhöht. Wählte man ein beliebiges Feld $F_i^<$ aus $\{F_1, \dots, F_n\}$, mit $\mu(F_i^<) < \mu(F_i)$ so erhöht sich nach Definition der minimalen Lebenszeit diese bei $\mu(F_i) - \mu(F_i^<) > 0$ Feldern weniger, als wenn man F_i wählte. Erhalten diese Felder in den nächsten z_o Beobachtungen keine Lebenszeitverlängerung, so brennen sie ab, wobei z_o der kleinste Abstand des Feldes o zu F_i ist.

Es verbleibt also zu zeigen, dass es keine Situation geben kann, bei der die Wahl von $F_i^<$ zu einer insgesamt geringeren Anzahl an verbrannten Feldern führt.

Angenommen es gäbe solch eine Situation.

Dies heißt jedoch, dass es eine oder mehrere Löschungen von Feldern gibt, welche insgesamt dazu führen, dass die Lebenszeit von $\mu(F_i) - \mu(F_i^<) + 1$ Feldern verlängert wird. Außerdem dürfen diese Löschungen nicht möglich sein, wenn F_i anstatt $F_i^<$ gelöscht wird. Dies im Speziellen heißt jedoch, dass Felder gelöscht werden, welche sonst durch die Löschung von F_i eine Lebenszeitverlängerung erhielten. Somit wäre es aber besser gewesen, F_i zu löschen, da bei den Beobachtungen danach auch andere Felder gelöscht werden könnten und die insgesamt Anzahl an verbrannten Feldern so insgesamt gesunken wäre.⁵ Es ist also optimal, ein Feld mit maximalem $\mu(F_i)$ auszuwählen. Bleibt zu zeigen, dass die Wahl eines speziellen F_i mit maximalem $\mu(F_i)$ an der gesamten Anzahl an verbrannten Feldern nichts ändert.

Sei $F_i^=$ ein beliebiges Feld aus $\{F_1, \dots, F_n\}$, mit $\mu(F_i^=) = \mu(F_i)$ und $F_i^= \neq F_i$.⁶

Es genügt zu zeigen, dass das Wählen von $\mu(F_i^=)$ keine Verringerung der am Ende insgesamt brennenden Felder gegenüber $\mu(F_i)$ darstellt, da $\mu(F_i)$ und $\mu(F_i^=)$ beliebig gewählt sind.

Angenommen dies sei der Fall.

Der Algorithmus ist also korrekt und optimal.

⁵Es ist theoretisch möglich, dass die Wahl zwischen $F_i^<$ und F_i keinen Unterschied macht, beispielsweise, wenn alle Felder innerhalb der nächsten o Beobachtungen verkohlen oder gelöscht werden. Dabei sei o der maximale Abstand, der in $\mu(F_i^<)$ Berücksichtigung fand. Dies stellt jedoch keinen Widerspruch zur Behauptung dar.

⁶Sei vorausgesetzt, dass ein solches $F_i^=$ existiert. Andernfalls existiert dieser Fall nicht, der Beweis ist dann hier beendet.

1.1.2 Laufzeitanalyse

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

1.3 Beispiele

1.3.1 Beispiel 0

Die ist das Beispiel aus der Aufgabenstellung. Umgewandelt für mein Programm sieht diese Eingabe folgendermaßen aus⁷:

```

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

```

Mein Programm produziert folgende Ausgabe⁸⁹:

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):

⁷Diese Eingabe finden Sie auch in der Datei 0.in

⁸Diese Ausgabe finden Sie auch in der Datei 0.out.tex; Eine Datei 0.out mit den ASCII-Escape-Sequenzen existiert ebenfalls.

⁹Um 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 --- WALL

FO --- FOREST

BU --- BURNED

CO --- COAL (doubly burned)

--- WATERED at time

Fields can have more than 1 state.

1.3.2 Beispiel 1

Eine Situation mit mehr als einem Feuer bei der ersten Beobachtung¹⁰:

```

1  10 11
2  3101111101
3  1001111110
4  1111111111
5  1100010001
6  1111131111
7  1100111111
8  1111011011
9  0111011010
10 1011011011
11 1111113111

```

¹⁰Diese Eingabe finden Sie auch in der Datei 1.in

12 1111111111

Mein Programm produziert folgende Ausgabe¹¹:

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):

¹¹Diese 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 (1|7):

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	BU	CO	WA	CO	CO	WA	CO	CO
WA	07	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	CO	CO	WA	CO	CO	WA	CO	CO
WA	07	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	CO	CO	WA	CO	CO	WA	CO	CO
WA	07	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 --- WALL

FO --- FOREST

BU --- BURNED

CO --- COAL (doubly burned)

--- WATERED at time

Fields can have more than 1 state.

1.3.3 Beispiel 2

¹²:

```

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 Ausgabe¹³:

¹²Diese Eingabe finden Sie auch in der Datei 2.in

¹³Diese 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	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
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
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	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
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	CO	05	WA	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:

FO	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	WA	FO
FO	WA	CO	CO	CO	CO	04	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	
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	CO	CO	CO	CO	CO	CO	CO	CO	CO	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	FO

Explanation:

WA --- WALL

FO --- FOREST

BU --- BURNED

CO --- COAL (doubly burned)

--- WATERED at time

Fields can have more than 1 state.

1.4 Quelltext

```

1  #include <cstdio>
2  #include <vector>
3  #include <queue>
4  #include <set>
5  #include <string>
6  #include <cstring>
7  using namespace std;
8
9  typedef pair<int,int> PII;
10
11 #define FIELDSTATE      char
12 #define WALL            0
13 #define WOODS           1
14 #define BURNED          2
15 #define WATERED        4
16 #define COAL            8
17
18 const int oo = (1 << 29);           //
    The infinity
19
20 class Woods{
21 private:
22     int Width, Height;
23     vector<vector<FIELDSTATE> > Fields;
24

```

```

25 public:
26     Woods(int width, int height) : Width(width), Height(height) {
27         Fields.assign(height, vector<FIELDSTATE>(width, 0));
28     }
29
30     int width() const { return Width; }
31     int height() const { return Height; }
32
33     FIELDSTATE& operator() (int x, int y) {
34         if (x < 0 || y < 0 || x >= width() || y >= height())
35             printf("OUT OF BOUNDS 1");
36         return Fields[y][x];
37     }
38     FIELDSTATE operator() (int x, int y) const {
39         if (x < 0 || y < 0 || x >= width() || y >= height())
40             printf("OUT OF BOUNDS 2");
41         return Fields[y][x];
42     }
43 } Forest(0, 0);
44
45 struct Point {
46 public:
47     int x, y;
48     Point(int _x, int _y) : x(_x), y(_y) { }
49 };
50 int dir[4][2] = {{1,0},{0,1},{-1,0},{0,-1}};
51
52 vector<Point> Solution;
53 FILE* OUT; //
54     The file to mirror the output to
55 void (*printSolution)(FILE*, bool);
56
57 Point getOptimalWaterSpot(vector<Point>& candidates){
58     queue<pair<PII,Point> > q; //
59     ((distance | color) | Location)
60     for(int i= 0; i < candidates.size(); ++i)
61         q.push(pair<PII,Point>(PII(0,i),candidates[i])); //
62         insert all the candidates as start points for the BFS
63
64     vector<vector<set<int> > > visited(Forest.width(), //
65         remember all nearest points first
66         vector<set<int> >(Forest.height()));
67     vector<vector<int> > shortDis(Forest.width(), //
68         shortest distant to any burning field
69         vector<int>(Forest.height(),oo));
70
71     //BFS to calculate shortest paths
72     while(!q.empty()){
73         pair<PII,Point> ac = q.front();
74         Point acPoint = ac.second;
75         int acDistance = ac.first.first;
76         int acColor = ac.first.second;

```

```
73     q.pop();
74     if(visited[acPoint.x][acPoint.y].count(acColor))
75         continue;
76     visited[acPoint.x][acPoint.y].insert(acColor);
77
78     for(int i= 0; i < 4; ++i){
79         int newx = acPoint.x + dir[i][0];
80         int newy = acPoint.y + dir[i][1];                //
81         calculate new field's indexes
82
83         if(newx < 0 || newy < 0 || newy >= Forest.height() || newx
84             >= Forest.width())
85             continue;                                    //
86         new field is outside the woods
87         if (Forest(newx, newy) != WOODS)
88             continue;                                    //
89         Field is not of interest
90
91         if(visited[newx][newy].count(acColor) == 0)      //
92             Don't compute things twice
93         if(acDistance + 1 <= shortDis[newx][newy]){
94             shortDis[newx][newy] = acDistance + 1;
95             q.push(pair<PII,Point>(PII(acDistance +
96                 1,acColor),Point(newx,newy)));
97         }
98     }
99 }
100
101 //determine the field to be watered
102 vector<int> waterval(candidates.size(),0);
103
104 //Count the number of fields that have an unique fire spot
105     a.k.a. waterval
106 for(int i= 0; i< Forest.width(); ++i)
107     for(int j= 0; j < Forest.height(); ++j)
108         if(visited[i][j].size() == 1)
109             waterval[*visited[i][j].begin()]++;
110
111 //determine the field of the candidates which has the highest
112     waterval
113 int maxv = waterval[0];                                //
114     maximal value
115 int maxi = 0;                                           //
116     index of maximal value
117
118 for(int i= 1; i < candidates.size(); ++i)
119     if(waterval[i] > maxv){
120         maxv = waterval[i];
121         maxi = i;
122     }
123
124 return candidates[maxi];
125 }
```

```
116
117 //BEGIN OF INPUT
118 void parseInput(FILE* f) {
119     int acFieldWidth, acFieldHeight;
120     fscanf(f, "%i %i\n",&acFieldWidth, &acFieldHeight);
121
122     Forest = Woods(acFieldWidth, acFieldHeight);
123
124     for(int i = 0; i < acFieldHeight; ++i){
125         for(int j = 0; j < acFieldWidth; ++j){
126             char c;
127             fscanf(f, "%c",&c);
128             c -= '0';
129             Forest(j, i) = (FIELDSTATE) c;
130         }
131         if(i < acFieldHeight-1)
132             fscanf(f, "\n");
133     }
134 }
135 //END OF INPUT
136 //BEGIN OF OUTPUT
137 void printSolution_TEX(FILE* f, bool finalOut) {
138     fprintf(f, "\\n");
139
140     fprintf(f, "\\begin{tikzpicture}\n");
141     fprintf(f, "\\tikzset{square matrix/.style={\n");
142     fprintf(f, "matrix of nodes,\n");
143     fprintf(f, "column sep=-\\pgflinewidth, row\n");
144     fprintf(f, "sep=-\\pgflinewidth,\n");
145     fprintf(f, "nodes={draw,\n");
146     fprintf(f, "minimum height=#1,\n");
147     fprintf(f, "anchor=center,\n");
148     fprintf(f, "text width=#1,\n");
149     fprintf(f, "align=center,\n");
150     fprintf(f, "inner sep=0pt\n");
151     fprintf(f, "},\n");
152     fprintf(f, "square matrix/.default=1.2cm\n");
153     fprintf(f, "}\n");
154
155     fprintf(f, "\\matrix[square matrix=1.4em] {\n");
156     for(int j = 0; j < Forest.height(); ++j) {
157         for(int i = 0; i < Forest.width(); ++i) {
158             if(i)
159                 fprintf(f, " &");
160
161             FIELDSTATE acField = Forest(i, j);
162             if(acField == WALL)
163                 fprintf(f, "|[fill=white]|");
164             else if(acField & WATERED)
165                 fprintf(f, "|[fill=cyan]|");
166             else if(acField & WOODS)
167                 fprintf(f, "|[fill=green]|");
```

```

168
169     if(acField & COAL)
170         fprintf(f, "\\color{rgb}{0,0,0}");
171     else if(acField & BURNED)
172         fprintf(f, "\\color{rgb}{1,0,0}");
173     else if(acField == WALL)
174         fprintf(f, "\\color{gray}{0.5}");
175     else if(acField & WOODS)
176         fprintf(f, "\\color{gray}{0.75}");
177
178     if(acField & WATERED){
179         for (int t = 0; t < Solution.size(); ++t)
180             if (Solution[t].x == i && Solution[t].y == j) {
181                 fprintf(f, "\\textbf{%02d}",t+1);
182                 break;
183             }
184     }
185     else if(acField & COAL)
186         fprintf(f, "\\textbf{CO}");
187     else if(acField & BURNED)
188         fprintf(f, "\\textbf{BU}");
189     else if(acField == WALL)
190         fprintf(f, "WA");
191     else if(acField & WOODS)
192         fprintf(f, " FO");
193     else
194         fprintf(f, "\\phantom{AA}");
195     fprintf(f, "%%\n");
196 }
197
198     fprintf(f, "\\n");
199 }
200
201 fprintf(f, "};\n\\end{tikzpicture}\\n");
202
203 if(finalOut){
204     fprintf(f, "\\n\\nExplanation:");
205     fprintf(f, "\\n\\n\\colorbox{white}{\\color{gray}{0.5}WA}
206         --- WALL");
207     fprintf(f, "\\n\\n\\colorbox{green}{\\color{gray}{0.5}FO}
208         --- FOREST");
209     fprintf(f, "\\n\\n\\color{rgb}{1,0,0}\\textbf{BU} ---
210         BURNED");
211     fprintf(f, "\\n\\n\\color{rgb}{0,0,0}\\textbf{CO} ---
212         COAL (doubly burned)");
213     fprintf(f, "\\n\\n\\colorbox{cyan}{\\#\\#} --- WATERED at
214         time \\#\\#");
215     fprintf(f, "\\n\\nFields can have more than 1 state.");
216 }
217 }
218
219 void printSolution_TERMINAL(FILE* f, bool finalOut) {
220     fprintf(f, "\n");

```

```

216 //The ASCII-magic starts here:
217 for(int j= 0; j < Forest.height(); ++j) {
218     for(int i= 0; i < Forest.width(); ++i) {
219         FIELDSTATE acField = Forest(i, j);
220         int waterval = 0;
221
222         fprintf(f, "\x1b[s ");
223         if (acField == WALL)
224             fprintf(f, "\x1b[u\x1b[37;47mWA");
225         if (acField & WOODS)
226             fprintf(f, "\x1b[u\x1b[32;42mFO");
227         if (acField & BURNED)
228             fprintf(f, "\x1b[u\x1b[1;5;31m/\");
229         if (acField & COAL)
230             fprintf(f, "\x1b[u\x1b[1;4;5;30m/\");
231         if (acField & WATERED)
232             for (int t = 0; t < Solution.size(); ++t)
233                 if (Solution[t].x == i && Solution[t].y == j) {
234                     fprintf(f, "\x1b[u\x1b[46m%02d", t+1);
235                     break;
236                 }
237         fprintf(f, "\x1b[0;39;49m");
238     }
239
240     fprintf(f, "\n");
241 }
242
243 if (finalOut) { // An Explanation shall be printed
244     fprintf(f, "\nExplanation:");
245     fprintf(f, "\n\x1b[37;47mWA\x1b[39;49m --- WALL");
246     fprintf(f, "\n\x1b[32;42mFO\x1b[39;49m --- FOREST");
247     fprintf(f, "\n\x1b[1;5;31m/\x1b[0;39m --- BURNED");
248     fprintf(f, "\n\x1b[1;4;5;30m/\x1b[0;39m --- COAL (doubly
        burned)");
249     fprintf(f, "\n\x1b[46m#\x1b[0;39m --- WATERED at time
        ##");
250     fprintf(f, "\nFields can have more than 1 state.");
251 }
252 fprintf(f, "\n");
253 }
254
255 //END OF OUTPUT
256
257 vector<Point>& getInitialBurningFields() {
258     static vector<Point> burnedFields;
259
260     for(int i = 0; i < Forest.height(); ++i)
261         for(int j= 0; j < Forest.width(); ++j)
262             if(Forest(j, i) & BURNED){
263                 burnedFields.push_back(Point(j, i));
264             }
265     printf("Initially burning: (%i|%i)\n",j, i);
266     return burnedFields;

```

```
267 }
268
269 void simulateFire(const vector<Point>& initiallyBurningFields) {
270     vector<Point> burnedFields = initiallyBurningFields;
271     printSolution_TERMINAL(stdout, false);
272     if (OUT != 0)
273         printSolution(OUT, false);
274
275     int time = 0;
276     while(!burnedFields.empty()) {
277         // Simulate as long as there's still fire in the world
278         vector<Point> newBurnedFields;
279         // The burning fields at the next point of time
280
281         //Calculate the new burning fields
282         for(size_t i = 0; i < burnedFields.size(); ++i){
283             int acx = burnedFields[i].x;
284             int acy = burnedFields[i].y;
285
286             if(Forest(acx, acy) & WATERED)
287                 continue;
288             // The field got watered and does not spread fire
289             Forest(acx, acy) |= COAL;
290             // Field burned down to coal...
291
292             for(int j = 0; j < 4; ++j) {
293                 int newx = acx + dir[j][0];
294                 int newy = acy + dir[j][1];
295
296                 if(newx < 0 || newy < 0 || newy >= Forest.height() ||
297                    newx >= Forest.width())
298                     continue;
299                 // new field is outside the woods
300                 if(Forest(newx, newy) == WOODS){
301                     Forest(newx, newy) |= BURNED;
302                     // Field starts burning
303                     newBurnedFields.push_back(Point(newx, newy));
304
305                     // printf(" From now on burning: (%i|%i)\n", newx, newy);
306                     // log the happenings
307                 }
308             }
309         }
310         if(newBurnedFields.empty())
311             // Nothing to water, all plants happy...
312             break;
313
314         burnedFields = newBurnedFields;
315
316         Point toWater = getOptimalWaterSpot(newBurnedFields);
317         // Determine the field to water
318         Forest(toWater.x, toWater.y) |= WATERED;
319         // ... and water it
320     }
```



```
309     Solution.push_back(toWater);
310
311
312     //Output / mirror the partial solution
313     printf("At time %i: Water spot
314           (%i|%i):\n", ++time, toWater.x, toWater.y);
315     printSolution_TERMINAL(stdout, false);
316
317     if (OUT) {
318         fprintf(OUT, "At time %i: Water spot
319               (%i|%i):\n", time, toWater.x, toWater.y);
320         printSolution(OUT, false);
321     }
322 //     printf("Fire died.\n");
323
324     //Count the total number of burned or coaled
325     int wcnt = 0, ccnt = 0;
326     for(int i = 0; i < Forest.width(); ++i)
327         for(int j = 0; j < Forest.height(); ++j)
328             if(Forest(i, j) & WATERED)
329                 wcnt++;
330             else if(Forest(i, j) & COAL)
331                 ccnt++;
332
333     //Output / Mirror the solution
334     printf("And you'll find %i pieces of coal and %i pieces of
335           watered coal:\n", ccnt, wcnt);
336     printSolution_TERMINAL(stdout, true);
337     if (OUT) {
338         fprintf(OUT, "And you'll find %i pieces of coal and %i
339               pieces of watered coal:\n", ccnt, wcnt);
340         printSolution(OUT, true);
341     }
342
343     int main(int argc, char** argv){
344         if (argc > 1) {
345             freopen(argv[1], "r", stdin);
346             printf("Using %s as input.\n", argv[1]);
347         }
348         if (argc > 2){
349             printf("Mirroring output to %s.\n", argv[2]);
350             if (strstr(argv[2], ".tex")) {
351                 printf("I reckon you want me to produce some TeX
352                       stuff...\n");
353                 printSolution = printSolution_TEX;
354             }
355             else
356                 printSolution = printSolution_TERMINAL;
357             OUT = fopen(argv[2], "w");
358         }
359     }
```

```
357     else{
358         OUT = 0;
359         printSolution = printSolution_TERMINAL;
360     }
361
362     parseInput(stdin);
363     simulateFire(getInitialBurningFields());
364 }
```