Junioraufgabe 2: Baulwürfe

J2.1 Lösungsidee

Darstellung der Maulwurfshügel

Unser Programm bekommt als Eingabe eine Karte der Höhe h und Breite b. Die Karte der Forscher kann als zweidimensionales boolesches Array A der Größe $h \times b$ dargestellt werden. Wir lesen unsere Eingabe so ein, dass in $A[i][j]$ der boolesche Wert `true` steht, wenn in der Karte der Forscher in der i-ten Zeile und der j-ten Spalte ein Hügel verzeichnet ist.

Eine Erste Lösungsidee

Eine der ersten Lösungsideen, die sich beim Durchlesen der Aufgabenstellung ergibt, ist die folgende: Überprüfe für jedes Planquadrat, ob es die linke obere Ecke eines Baulwurfshügels darstellt. Das Programm in Pseudocode sieht also folgendermaßen aus:

```
 1: procedure BAULWURFSBAUE(A, h, b)
 2:     baulwurfsbaue ← 0
 3:     for z ← 0,..,h−4 do                                  ▷ Zeilenindex
 4:         for s ← 0,..,b−3 do                              ▷ Spaltenindex
 5:             if A[z][s] and A[z][s+1] and A[z][s+2] and
 6:               A[z+1][s] and not A[z+1][s+1] and A[z+1][s+2] and
 7:               A[z+2][s] and not A[z+2][s+1] and A[z+2][s+2] and
 8:               A[z+3][s] and A[z+3][s+1] and A[z+3][s+2] then
 9:                 baulwurfsbaue ← baulwurfsbaue+1
10:             end if
11:         end for
12:     end for
13:     return baulwurfsbaue
14: end procedure
```

Aber Achtung! Diese Methode hat noch ein Problem. Schauen wir uns einen Ausschnitt aus der Beispielkarte 5 an:

Wie die Forscher schnell herausfinden können, gibt es auf dieser Karte nur 4 Baulwurfsbaue. Trotzdem findet der oben beschriebene Algorithmus 5 Baue. Warum ist das der Fall? Neben

den 4 „echten“ Bauen (Abbildung J2.1) gibt es auf der Karte ein fünftes Mal die gesuchte Struktur (Abbildung J2.2). Weil sich aber, wie die Aufgabenstellung sagt, Baulwurfsbaue nicht überlappen, kann es sich hier nicht um einen Bau handeln.

Abbildung J2.1: Die 4 Baulwurfsbaue

Abbildung J2.2: Ein Extrabau[5]

Unser Programm findet hier also alle wirklichen Baulwurfsbaue, erkennt allerdings auch noch eine weitere Kombination an Hügeln als Baulwurfsbau. Die Lösung muss also noch ein bisschen angepasst werden, um dieses Problem zu verhindern. Eine Möglichkeit zur Anpassung ist beispielsweise die folgende: Sobald ein Baulwurfsbau gefunden wird, markiere jeden Hügel, der bereits in diesem Bau vorkommt. Wird nun ein neuer Baulwurfsbau gefunden, überprüfe zuerst, ob in diesem Bau bereits verwendete Hügel vorkommen. Nur wenn dies nicht der Fall ist, wird der neu gefundene Bau gezählt.

Mit dieser Lösung kann es natürlich passieren, dass in unserem Beispiel nur ein Baulwurfsbau gefunden wird, wenn der Bau in der Mitte zuerst gefunden wird. Dies kann allerdings nicht passieren, wenn beispielsweise alle Felder von links nach rechts und oben nach unten durchsucht werden. Startet man allerdings in der Mitte mit der Suche, kann dieser Ansatz falsche Ergebnisse liefern.

Einfachere Lösung

Die Aufgabenstellung macht noch eine weitere Vorgabe: Ein Planquadrat mit einem Maulwurfshügel ist immer rundum von Quadraten ohne Hügel umgeben. Damit kann die folgende Beobachtung angestellt werden: Sobald ein Quadrat mit Hügel an ein anderes Quadrat mit Hügel angrenzt, muss es automatisch ein Planquadrat mit einem Baulwurf sein! Eine weitere Beobachtung hilft unserem Lösungsansatz noch weiter: Jeder Hügel, welcher von Baulwürfen gebaut wird, grenzt in mindestens einer Richtung (oben, unten, links, rechts) an ein weiteres Planquadrat mit Hügel an!

Daher kann für jeden Hügel einfach überprüft werden, ob von diesem aus nach oben, unten, links oder rechts mindestens ein weiterer Hügel verzeichnet ist. Wenn ja, wird der Hügel als *Baulwurfshügel* markiert. Dieses System findet alle Hügel, die von Baulwürfen (als Teil ihrer Baue) gebaut wurden, und keinen weiteren Hügel.

Nun wissen wir, dass ein Baulwurfsbau immer die folgende Struktur besitzt:

[5]Es ist nicht möglich, dass auf der ganzen Karte nur der rot markierte Baulwurfsbau zu finden ist. Wäre dies

```
XXX
X X
X X
XXX
```

Damit besteht ein Baulwurfsbau immer aus genau zehn Baulwurfshügeln. Die Anzahl an Baulwurfsbauen ist also einfach die Anzahl der Baulwurfshügel geteilt durch zehn.

In Pseudocode sieht unser Algorithmus also nun wie folgt aus:

```
1:  procedure BAULWURFSBAUE(A, h, b)
2:      baulwurfshuegel ← 0
3:      for z ← 0,..,h−1 do                                   ▷ Zeilenindex
4:          for s ← 0,..,b−1 do                               ▷ Spaltenindex
5:              if A[z][s] then
6:                  if A[z−1][s] or A[z][s+1] or A[z+1][s] or A[z][s−1] then
7:                      baulwurfshuegel ← baulwurfshuegel + 1
8:                  end if
9:              end if
10:         end for
11:     end for
12:     return baulwurfshuegel ÷ 10
13: end procedure
```

In diesem Code wird an einigen Stellen auf das Array an Stellen zugegriffen, die außerhalb des Kartenbereichs liegen. Dies kann zu Fehlern im Programm führen. In unserem Pseudocode wird immer erwartet, dass an Stellen außerhalb des Kartenbereichs der Wert `false` in A steht.

Variante

Genauso gut kann natürlich die Anzahl an Maulwurfshügeln gezählt werden, indem überprüft wird, dass kein angrenzendes Quadrat einen Hügel aufweist. Die Anzahl der Baulwurfsbaue ergibt sich dann als

$$\frac{\text{Gesamtanzahl Hügel - Maulwurfshügel}}{10}$$

J2.2 Beispiele

Angewandt auf die von BWINF bereitgestellten Eingaben liefert unser Programm die folgenden Ergebnisse:

der Fall, sind alle anderen schwarzen `X` normale Maulwürfe. Allerdings wissen wir, dass Maulwürfe ihre Baue nicht aneinander angrenzend bauen.

Eingabe	**Ergebnis**	Ergebnis einfache Lösung
`karte0.txt`	**3**	3
`karte1.txt`	**37**	37
`karte2.txt`	**32**	32
`karte3.txt`	**318**	318
`karte4.txt`	**3189**	3193
`karte5.txt`	**16**	20
`karte6.txt`	**559**	559

J2.3 Bewertungskriterien

Die Bewertungskriterien (Fettdruck) vom Bewertungsbogen werden hier näher erläutert (Punktabzug in []).

- $[-1]$ **Modellierung ungeeignet**
  Die offensichtliche Möglichkeit zur Repräsentation der Hügelkarte ist ein zweidimensionales Array. Ob boolesche Werte, die Zeichen der Eingabe (Leerzeichen und X) oder ein anderes Wertepaar verwendet werden: das ist alles gut brauchbar. Darstellungen, die das Lösungsverfahren unnötig verkomplizieren, führen zum Punktabzug.
- $[-1]$ **Lösungsverfahren fehlerhaft**
  Die Aussage der Aufgabenstellung, dass sich Baulwurfsbaue nicht überlappen, konnte auch als Versprechen verstanden werden, dass sich in der Eingabe auch keine Baulfwurfsbaue so berühren, dass es zu scheinbaren Überlappungen der Baulfwurfsbau-Strukturen kommt. In diesem Fall funktioniert auch das zuerst beschriebene, einfache Verfahren. Es wird deshalb nicht verlangt, dass das Verfahren scheinbar überlappende Strukturen erkennt und ausschließt. Damit sind auch die Werte in der rechten Spalte der Ergebnistabelle akzeptabel. Ansonsten sollten Baulwurfsbaue aber zuverlässig erkannt werden, auch wenn sie aneinander angrenzen oder am Rand der Karte liegen. Abgesehen von den scheinbar überlappenden Strukturen dürfen aber natürlich auch nicht mehr Baue erkannt werden als vorhanden sind.
- $[-1]$ **Verfahren bzw. Implementierung unnötig aufwendig / ineffizient**
  Nur besonders umständliche Verfahren bzw. Implementierungen führen zu Punktabzug. Das Programm sollte in der Lage sein, alle BWINF-Beispiele in sehr kurzer Zeit zu lösen.
- $[-1]$ **Beispiele fehlerhaft bzw. zu wenige oder ungeeignete Beispiele**
  Es sollten die Ausgaben zu mindestens drei der sechs zusätzlichen Beispiele zu der Aufgabe (`karte1.txt` bis `karte6.txt`) in der Dokumentation abgedruckt sein.