39. Bundeswettbewerb Informatik

Runde 1

Jugendwettbewerb Informatik 2020

Runde 3

Junioraufgabe 2: Baulwürfe

07. November 2020

Inhaltsverzeichnis

Lösungsidee

Die Hügelkarte wird zuerst eingelesen. Das Programm geht jedes Planquadrat durch. Sofern sich ein Hügel (X) auf dem jetzigen Planquadrat befindet, wird angenommen, dass dieser Hügel auch ein Teil eines Baulwurfsbaues ist, und zwar genau der Teil, der die linke obere Ecke des Baues ist (vgl. Abb. 1, das rot markierte X). Dann testet man die anderen Stellen des möglichen Baulwurfsbaues (vgl. Abb. 1, die Koordinaten aller anderen X) und die beiden Planquadrate ohne Hügel im Zentrum. Falls auf allen diesen Planquadraten, die nicht Teil anderer Baulwurfsbaue sein dürfen, jeweils auch einen Hügel (X) steht und das Zentrum leer (also ohne Hügel) ist, wird festgestellt, dass es sich hier um einen Baulwurfsbau handelt.

Abb. 1, ein Baulwurfsbau

Da jedes Planquadrat einmal mit dem obigen Verfahren geprüft wird, können alle Baulwurfsbaue der Hügelkarte herausgefunden werden. Da die Baue sich nicht überklappen, wird kein Baulwurfsbau mehrmals oder niemals gezählt wird. Dabei handelt es von einer Zeitkomplexität von *O(n)*.

Jedoch lässt sich das Programm noch schneller funktionieren, indem man speichert, ob das jetzige Planquadrat schonmal getestet wurde. Es sei der Fall, wenn dieses Planquadrat schon als Teil eines anderen Baulwurfsbaues (aber nicht die linke obere Ecke) identifiziert wurde. In diesem Fall wird dieser Hügel nicht mehr als den linken oberen Teil des Baues angenommen und alle anderen Planquadrate werden auch nicht getestet. Somit wird eine Vielzahl von Vergleichen durch diesen Einzelnen ersetzt. Das funktioniert besonders gut, wenn auf der Karte viele Baulwurfsbaue stehen.

Darüber hinaus wird vorher geprüft, ob dieses Quadrat schon am Rand steht und damit nicht mehr möglich als den linken oberen Teil des Baues sein wird. In diesem Fall werden die anderen Teile auch nicht getestet.

Umsetzung

Das Programm wird in C# 8.0 implementiert und mit .Net Core 3.1 CLI kompiliert. Um .Net Core 3.1 SDK herunterzuladen, klicken Sie hier.

Am Anfang wird eine Integer-Variable namens *spezialBaulwuerfenCount* deklariert, die die Anzahl von diesen speziellen Baulwurfbaue speichert.

```
    int spezialBaulwuerfenCount = 0;
```

Die Test-Datei wird durch *Streamreader* eingelesen. Hier wird erwartet, dass der/die TesterIn den Namen der Test-Datei nach dem Öffnen des Programms eingibt, sodass das Programm weiß, welche Datei es einlesen sollte. Die Test-Datei soll in demselben Ordner stehen, wo auch das Programm steht.

```
        string testFilePath = Console.ReadLine();
    ……
        using (StreamReader sr = File.OpenText(testFilePath))
```

Zuerst wird die beiden Integer b (Breite) und h (Höhe) von den ersten beiden Zeilen gelesen und in zwei Variablen namens *b* und *h* gespeichert:

```
          int b = Convert.ToInt32(sr.ReadLine());
          int h = Convert.ToInt32(sr.ReadLine());
```

Dann werden zwei zwei-dimensionalen Arrays (vgl. Abb. 2), *map* und *besucht*, initialisiert. Das Array *map* (map: char[h, b]) speichert die Hügelkarte in Form von *char*. *map[x, y]* repräsentiert das Planquadrat auf der Koordinate (x, y), wobei x = Zeile - 1 und y = Spalte - 1. Das Array *besucht* (besucht: bool[h, b]) speichert, ob dieses Planquadrat schonmal geprüft wurde und damit nicht mehr getestet werden muss, ob dieses auch die linke obere Ecke eines Baulwurfsbaues ist. Analog heißt *besucht[x, y]* ein Boolean Wert für das Planquadrat, ob dieses bereits geprüft wurde, wobei x = Zeile - 1 und y = Spalte - 1.

```
          char[,] map = new char[h, b];
          bool[,] besucht = new bool[h, b];
```

	0	1	2	3	4	...	...	b-4	b-3	b-2	b-1	*j=*
0	''	''	'X'	'X'	'X'	''	''	''	''	''	''	
1	''	''	'X'	''	'X'	''	''	''	'X'	''	''	
2	''	''	'X'	''	'X'	''	''	''	''	''	''	
3	''	''	'X'	'X'	'X'	'X'	'X'	''	'X'	'X'	'X'	
...	''	'X'	''	''	''	''	''	''	'X'	''	'X'	
...	''	''	''	''	''	''	'X'	''	'X'	''	'X'	
h-2	'X'	''	''	''	''	''	''	''	'X'	'X'	'X'	
h-1	''	''	''	''	''	''	''	''	'X'	''	''	
i=												

Abb. 2, das zwei-dimensionale Array map[h,b], nachdem die Hügelkarte eingelesen und gespeichert wird

Die ganze Hügelkarte wird dann eingelesen und in das Array *map* gespeichert (vgl. Abb. 2).

Danach wird jedes Planquadrat durch zwei verschachtelte Schleifen einmal geprüft, indem man die Erweiterungsmethode *IstHuegel* für jedes Planquadrat aufruft. Jedoch wird ein Planquadrat nicht geprüft, wenn das schon am Rand steht (also wenn $i + 3 >= h$ oder $j + 2 >= b$) und damit kein gültigen Baulwurfsbau bilden kann, oder, wenn das Planquadrat bereits Teil eines anderen Baulwurfsbaues ist (bzw. dass dieses Planquadrat schonmal vorher „besucht“ wurde).

```
        for (int i = 0; i < h; i++) {
            for (int j = 0; j < b; j++) {
              // Falls es schon ganz am Rand steht
              // und ist unmoeglich, eine Baulwuerfenbau zu bilden
              if (i + 3 >= h || j + 2 >= b) {
                // Dann ueberspringt man das Planquadrat direkt, ohne zu pruefen
                continue;
              }

              // Falls diese Quadrat ist noch nicht besucht
        // und auf diesem Quadrat
              // gibt es einen X bzw. Huegel
              if (!besucht[i, j] && map[i, j].IstHuegel()) {
        ..........
              }
            }
          }
        }
```

Die Erweiterungsmethode *IstHuegel* gibt einen Boolean-Wert zurück, der darauf hinweist, ob auf diesem Planquadrat ein Hügel, nämlich char "X", steht.

```
private static bool IstHuegel(this char mapKoordinate)
{
  return mapKoordinate.CompareTo(HUEGEL_ZEICHEN) == 0;
}
```

Nachdem es festgestellt wird, dass auf dem jetzigen Planquadrat ein unbesuchter Hügel existiert, werden dieser Hügel als die linke obere Ecke eines gültigen Baulwurfsbaues (vgl. Abb. 1) angenommen und dementsprechend alle andere Stellen, an denen für diesen Baulwurfsbau ein Hügel nötig wird, getestet, ob auf allen diesen Stellen auch genau jeweils ein Hügel steht.

```
// Testen, ob es sich von solchen speziellen Baulwuerfen handelt
bool istSpezial = map[i, j + 1].IstHuegel() &&
map[i, j + 2].IstHuegel() &&
map[i + 1, j].IstHuegel() &&
map[i + 1, j + 2].IstHuegel() &&
map[i + 2, j].IstHuegel() &&
map[i + 2, j + 2].IstHuegel() &&
map[i + 3, j].IstHuegel() && map[i + 3, j + 1].IstHuegel() &&
map[i + 3, j + 2].IstHuegel();
// Testet, ob die beiden Planquadraten im Zentrum leer sind
// sodass es genau den Muster eines Baulwurfsbaues bildet
bool centerIstLeer = !(map[i+1, j+1].IstHuegel() ||
map[i+2, j+1].IstHuegel());
// Testet, ob die Huegel des Baues schon Teil eines anderen Baues ist
bool sindBesucht = besucht[i, j + 1] || besucht[i, j + 2] ||
besucht[i + 1, j] || besucht[i + 1, j + 2] || besucht[i + 2, j] ||
besucht[i + 2, j + 2] || besucht[i + 3, j] || besucht[i + 3, j + 1] ||
besucht[i + 3, j + 2];
```

Ist es der Fall, erhöht man die Variable *spezialBaulwuerfenCount* um eins und markiert **alle** diesen Planquadraten als „besucht“. Ist es aber hingegen nicht der Fall, bleibt die Variable *spezialBaulwuerfenCount* unberührt und **nur** das jetzige Quadrat wird als „besucht“ markiert.

```
if (istSpezial && !sindBesucht && centerIstLeer)
{
  // Erhoeht man den Counter und markiert
  // alle Huegel des Baues als besucht
  spezialBaulwuerfenCount++;
  besucht[i, j] = true;
  besucht[i, j + 1] = true;
  besucht[i, j + 2] = true;
  besucht[i + 1, j] = true;
  besucht[i + 1, j + 2] = true;
  besucht[i + 2, j] = true;
  besucht[i + 2, j + 2] = true;
  besucht[i + 3, j] = true;
  besucht[i + 3, j + 1] = true;
  besucht[i + 3, j + 2] = true;
}
// Falls es aber nicht der Fall ist
else
{
  // markiert man nur das jetzigen Quadrat als besucht
  besucht[i, j] = true;
}
```

Am Ende wird die Anzahl der Baulwurfsbaue in der Console gezeigt.

```
Console.WriteLine($"Es gibt insgesamt {spezialBaulwuerfenCount} Baul-
wurfsbaue.");
```

Beispiele

Erwartete Formen von Daten

Angabe für das Programm:

Name der Test-Datei sollte nach Anforderung des Programms eingegeben werden.

Nach der Ausgabe des Ergebnisses wird das Programm den Nutzer auffordern, eine beliebige Taste zu drücken, um das Programm zu schließen.

Beispiel 1, vorgegeben, karte0.txt

Ausgabe:

```
Es gibt insgesamt 3 Baulwurfsbaue.
```

Erklärung:

Das Programm fängt mit dem linken oberen Quadrat an (rot markiert) und prüft, ob sich auf dem Planquadrat ein Hügel befindet. Jetzt ist der Zeiger der Schleife an der Position *[0,0]*.

```
XXX              X
X X   X    XXX       X
X X        X XXXX
XXX        X XX X      X
     X     XXXX X
              XXX
```

Es ist hier der Fall. Dann nehmen wir an, dass dieser Hügel der linken oberen Ecke eines Baulwurfsbaues ist (vgl. Abb. 1) und prüft, ob auf den anderen Quadraten des angenommenen Baues auch tatsächlich ein Hügel stehen und ob in der Mitte des Baues kein Hügel steht. Die zu prüfende Quadrate markieren wir hier mit **dunkelblau**. Achte darauf, dass die von den blauen Quadraten umgebende beide Quadrate auch geprüft wird, um zu sehen, ob die auch leer sind und damit dem Muster eines Baulwurfbaues entsprechen.

```
XXX              X
X X   X    XXX       X
X X        X XXXX
XXX        X XX X      X
     X     XXXX X
              XXX
```

Es ist auch der Fall, so erhöht man die Variable *spezialBaulwuerfenCount* um eins und markieren alle dieser Quadrate als „besucht“. „besuchte“ Quadrate markieren wir mit gelb. Die werden nicht noch mal geprüft.

Das obige rote Quadrat ist nun geprüft. Der Zeiger der Schleife ändert sich zu *[0,1]*. Da dieses Quadrat bereit als „besucht“ (gelb) markiert wurde, wird es nicht nochmal geprüft und der Zeiger ändert sich zu *[0,2]*. Bei diesem Quadrat ist eben dem Fall, deswegen wird der Zeiger der Schleife zu *[0,3]*. Dieses Quadrat ist nicht „besucht“, und wird deshalb auch nach dem obigen Verfahren geprüft. Da sich auf diesem Quadrat kein Hügel befindet, setzt man den „besucht“-Wert des Quadrats zu *true* und ändert den Zeiger zu *[0,4]*.

```
XXX             X
X X   X    XXX       X
X X        X XXXX
XXX        X XX X      X
     X     XXXX X
              XXX
```

Wenn der Zeiger zu *[0,17]* wird, prüft man wie gewohnt, ob auf diesem Quadrat ein Hügel steht. Hier ist der Fall, und nach der Behauptung, dass dieser Hügel auch der linke obere Teil eines Baulwurfbaues ist, wird die andere Quadrate (markiert mit **dunkelblau**) geprüft.

```
XXX             X□□
X X   X    XXX   □  □X
X X        X XXXX□  □
XXX        X XX X□□□  X
     X     XXXX X
              XXX
```

Da auf den anderen zu prüfenden Quadraten (**dunkelblau** Kasten markierte) kein Hügel stehen, ist diese Behauptung also falsch. Nun setzt man der „besucht“-Wert dieses Quadrats zu *true* und verschiebt den Zeiger auf *[0, 18]*.

Analog macht man weiter, bis der Zeiger das Ende, in diesem Fall *[5,23]*, erreicht hat. Der Wert der Variable *spezialBaulwuerfenCount* ist genau die Anzahl der gesuchten Baulwurfsbaue.

Beispiel 2, vorgegeben, karte1.txt

Ausgabe:

```
Es gibt insgesamt 37 Baulwurfsbaue.
```

Beispiel 3, vorgegeben, karte2.txt

Ausgabe:

```
Es gibt insgesamt 32 Baulwurfsbaue.
```

Beispiel 4, vorgegeben, karte3.txt

Ausgabe:

```
Es gibt insgesamt 318 Baulwurfsbaue.
```

Beispiel 5, vorgegeben, karte4.txt

Ausgabe:

```
Es gibt insgesamt 3189 Baulwurfsbaue.
```

Beispiel 6, vorgegeben, karte5.txt

Ausgabe:

```
Es gibt insgesamt 16 Baulwurfsbaue.
```

Beispiel 7, vorgegeben, karte6.txt

Ausgabe:

```
Es gibt insgesamt 559 Baulwurfsbaue.
```

Beispiel 8, weiteres Beispiel

Testdatei:

```
10
10
XXXXXXXXXX
XXXXXXXXXX
XXXXXXXXXX
XXXXXXXXXX
XXXXXXXXXX
XXXXXXXXXX
XXXXXXXXXX
XXXXXXXXXX
XXXXXXXXXX
XXXXXXXXXX
```

Ausgabe:

```
Es gibt insgesamt 0 Baulwurfsbaue.
```

Quellcode

```
1.  using System;
2.  using System.IO;
3.
4.  namespace JuniorAufgabe2
5.  {
6.    public static class BaulWuerfeCounter
7.    {
8.      private const char HUEGEL_ZEICHEN = 'X';
9.
10.     public static void Main(string[] args)
11.     {
12.       int spezialBaulwuerfenCount = 0;
13.
14.       Console.WriteLine("Bitte Name der Test-Datei eingeben...");
15.       string testFilePath = Console.ReadLine();
16.
17.       if (File.Exists(testFilePath))
18.       {
19.         using (StreamReader sr = File.OpenText(testFilePath))
20.         {
21.           int b = Convert.ToInt32(sr.ReadLine());
22.           int h = Convert.ToInt32(sr.ReadLine());
23.
24.           char[,] map = new char[h, b];
25.           bool[,] besucht = new bool[h, b];
26.
27.           // Einlesen der Daten
28.           for (int i = 0; i < h; i++)
29.           {
30.             char[] lineChars = sr.ReadLine().ToCharArray();
31.             for (int j = 0; j < b; j++)
32.             {
33.               map[i, j] = lineChars[j];
34.               besucht[i, j] = false;
35.             }
36.           }
37.
38.           for (int i = 0; i < h; i++)
39.           {
40.             for (int j = 0; j < b; j++)
41.             {
42.               // Falls es schon ganz am Rand steht und ist unmoeglich, eine
    Baulwuerfenbau zu bilden
43.               if (i + 3 >= h || j + 2 >= b)
44.               {
45.                 // Dann ueberspringt man das Planquadrat direkt, ohne zu pru
    efen
46.                 continue;
47.               }
48.
49.               // Falls diese Quadrat ist noch nicht besucht und auf diesem Q
    uadrat
50.               // gibt es ein X bzw. Huegel
51.               if (!besucht[i, j] && map[i, j].IstHuegel())
52.               {
53.                 // Testen, ob es sich von solchen speziellen Baulwuerfen han
    delt
54.                 bool istSpezial = map[i, j + 1].IstHuegel() && map[i, j + 2]
    .IstHuegel() && map[i + 1, j].IstHuegel() && map[i + 1, j + 2].IstHuegel() &
    & map[i + 2, j].IstHuegel() && map[i + 2, j + 2].IstHuegel() && map[i + 3, j
    ].IstHuegel() && map[i + 3, j + 1].IstHuegel() && map[i + 3, j + 2].IstHuege
    l();
```

```
55.                 // Testet, ob die beiden Planquadraten im Zentrum leer sind
56.                 // sodass es genau den Muster eines Baulwurfsbaues bildet
57.                 bool centerIstLeer = !(map[i+1, j+1].IstHuegel() || map[i+2,
    j+1].IstHuegel());
58.                 // Testet, ob die Huegel des Baues schon Teil eines anderen
   Baues ist
59.                 bool sindBesucht = besucht[i, j + 1] || besucht[i, j + 2] ||
    besucht[i + 1, j] || besucht[i + 1, j + 2] || besucht[i + 2, j] || besucht[
   i + 2, j + 2] || besucht[i + 3, j] || besucht[i + 3, j + 1] || besucht[i + 3
   , j + 2];
60.                 // Falls es der Fall ist
61.                 if (istSpezial && !sindBesucht && centerIstLeer)
62.                 {
63.                   // Erhoeht man den Counter und markiert
64.                   // alle Huegel des Baues als besucht
65.                   spezialBaulwuerfenCount++;
66.                   besucht[i, j] = true;
67.                   besucht[i, j + 1] = true;
68.                   besucht[i, j + 2] = true;
69.                   besucht[i + 1, j] = true;
70.                   besucht[i + 1, j + 2] = true;
71.                   besucht[i + 2, j] = true;
72.                   besucht[i + 2, j + 2] = true;
73.                   besucht[i + 3, j] = true;
74.                   besucht[i + 3, j + 1] = true;
75.                   besucht[i + 3, j + 2] = true;
76.                 }
77.                 // Falls es aber nicht der Fall ist
78.                 else
79.                 {
80.                   // markiert man nur das jetzigen Quadrat als besucht
81.                   besucht[i, j] = true;
82.                 }
83.               }
84.             }
85.           }
86.         }
87.         Console.WriteLine($"Es gibt insgesamt {spezialBaulwuerfenCount} Baul
   wurfsbaue.");
88.       }
89.       else
90.       {
91.         Console.WriteLine("Die gegebene Datei-
   Namen ist nicht gueltig. Das Programm und die Datei muessen in demselben Ord
   ner stehen. ");
92.       }
93.       Console.WriteLine("Drueck eine beliebige Taste zu schliessen...");
94.       Console.ReadKey();
95.     }
96.
97.     private static bool IstHuegel(this char mapKoordinate)
98.     {
99.       return mapKoordinate.CompareTo(HUEGEL_ZEICHEN) == 0;
100.          }
101.
102.        }
103.      }
```