Junioraufgabe 2: **Baulwürfe**

Team-ID: **00255**

Team: **Solo**

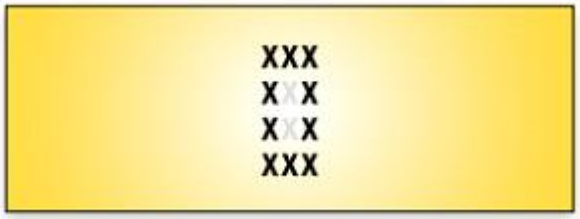
Bearbeiter dieser Aufgabe: **Walther Trgovac**

11. Oktober 2020

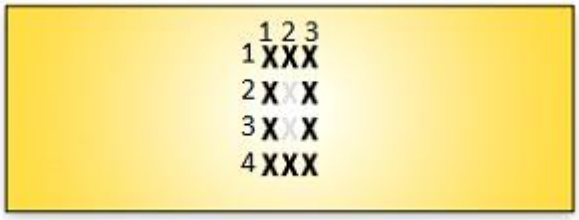
Inhaltsverzeichnis

Lösungsidee

Das Ziel der Aufgabe ist es, alle Muster, die so aussehen, zu zählen:



Dieser Baulwurf besteht aus Hügeln, die in diesem Fall, mit „X“ markiert sind. Da die Baulwürfe sich nicht überlappen, alles was man machen muss ist, einen X aussuchen und dann überprüfen, ob die anderen Buchstaben X in seiner Nähe dem Muster entsprechen. Mit Hilfe des Koordinaten Systems kann man diesen Muster besser darstellen:



Jetzt kann man zum Beispiel den X an der Position (1,1) aussuchen und überprüfen, ob sich an den Positionen (x,y): (2,1), (3,1), (1,2), (3,2), (1,3), (3,3), (1,4), (2,4), (3,4) auch ein X befindet. Falls ja, dann haben wir einen Baulwurf. Man muss nur aufpassen, wenn die Baulwürfe sich überlappen, aber das wird deutlicher in Beispielen.

Umsetzung

Die Lösungsidee habe ich in C++ implementiert.

Zuerst liest das Programm die Textdatei ein und speichert die in die 2D Array „matrix“.

Dann mit Hilfe von double For Loops wird jede Stelle in der Array besucht und zuerst überprüfen, ob sich da ein X befindet und falls ja, dann wird überprüft, ob sich an den Positionen (wie oben geschrieben) auch ein X befindet. Falls ja, wird die Antwort um 1 erhöht.

Ich habe auch noch eine 2D Array „used“, in der alle schon benutzte Positionen mit „1“ markiert sind. Das hilft bei den Baulwürfen, die sich überlappen.

Man muss nicht jede Stelle in der Array überprüfen, da der Baulwurf 4 Stellen „hoch“ und 3 Stellen „breit“ ist. Deswegen wenn man im Koordinaten System X-Achse betrachtet, kann man die letzten 2 Stellen ignorireren, genauso wie bei Y-Achse, kann man die letzten 3 Stellen ignorieren.

Beispiele

Bei dem ersten Beispiel karte0.txt:

```
24
6
XXX              X
X X   X    XXX       X
X X        X XXXX
XXX        X XX X      X
     X     XXXX X
           XXX
```

Man sieht deutlich, dass sich da 3 Baulwürfe befinden und das ist auch die Ausgabe dieses Programms.

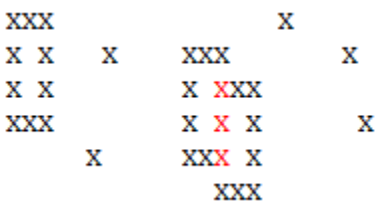
Bei den folgenden Beispielen habe ich folgenden Ergebnisse:
Karte1.txt: 37
Karte2.txt: 32
...

Karte6.txt zeigt die grösste möglichste Anzahl der Baulwürfe in diesem „Koordinaten System“. Der Baulwurf ist 3 Stellen breit und in diesem Fall haben wir als Breite die Zahl 129. 129/3 = 43. Es gibt

maximal 43 Baulwürfe an der X-Achse und maximal 52/4 = 13 Baulwürfe an der Y-Achse. 43*13=559. Das ist die grösste möglichste Anzahl der Baulwürfe und das ist auch die Ausgabe des Programms.

Es gibt einen besonderen Fall, wenn die Baulwürfe sich überlappen. Ich habe karte0 modifiziert:

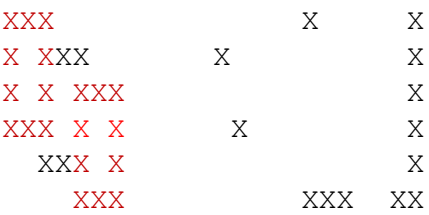
24
6



Die rot markierten Stellen sind die Positionen, an den sich 2 Baulwürfe überlappen. In diesem Fall ist die Antwort 2 und nicht 3, da die Baulwürfe sich nicht überlappen dürfen. Von den 2 Baulwürfen rechts muss man nur einen als “richtigen” aussuchen und zählen.

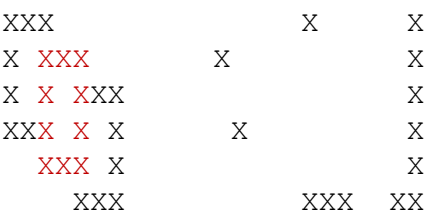
Noch ein Beispiel, bei dem man aufpassen muss:

24
6



Hier sieht man, dass die richtige Antwort 2 ist, aber es kann auch sein, dass die Antwort 1 lautet, wenn man so zählt:

24
6



Jetzt sieht man, dass die Anwort 1 ist. Da in der Aufgabenstellung nicht steht, dass man die Anzahl der Baulwürfe maximiseren soll, sind die beiden Antworten richtig. Die Sache ist nur, wie man zählt. Die Ausgabe meines Programms ist in diesem Fall 2, aber 1 soll auch richtig sein.

Quellcode

```
//Input
char matrix[h][b]; // Array für den Plan
int used[h][b]; // Array für die Positionen, an den sich Baulwürfe schon
befinden
// Textdatei wird eingelesen und gespeichert
for (int i = 0; i < h; i++){
      string s; getline(cin, s);
      for (int j = 0; j < b; j++){
            matrix[i][j] = s[j];
            used[i][j] = 0;
      }
}



// Man überprüft alle Positionen in der Array, an den sich ein Baulwurf
befinden kann
for (int i = 0; i < h-3; i++){
      for (int j = 0; j < b-2; j++){
      // Falls sich an der Position ein X befindet und ist kein Teil eines
      Baulwurfs
      if (matrix[i][j] == 'X' && used[i][j] == 0){
      // Falls diese Positonen dem "X" entsprechen, Antwort um 1 erhöhen und
      alle diesen Positionen als benutzt markieren
            if (matrix[i+1][j] == 'X' && matrix[i+2][j] == 'X' &&
            matrix[i+3][j] == 'X' && matrix[i][j+1] == 'X' &&
            matrix[i+1][j+1] == ' ' && matrix[i+2][j+1] == ' ' &&
            matrix[i+3][j+1] == 'X' && matrix[i][j+2] == 'X' &&
            matrix[i+1][j+2] == 'X' && matrix[i+2][j+2] == 'X' &&
            matrix[i+3][j+2] == 'X'){
                  ans++;
                  used[i][j] = 1;
                  used[i+1][j] = 1;
                  used[i+2][j] = 1;
                  used[i+3][j] = 1;
                  used[i][j+1] = 1;
                  used[i+3][j+1] = 1;
                  used[i][j+2] = 1;
                  used[i+1][j+2] = 1;
```

```
                    used[i+2][j+2] = 1;
                    used[i+3][j+2] = 1;
                }
            }
            }
}
```