Aufgabe 2: Baulwürfe

Valentin Stephan Zwerschke

22. November 2020

Inhaltsverzeichnis

1 Lösungsidee

In einem ersten Schritt suche ich prinzipiell in allen Zeilen des Feldes nach den Dreierkombinationen, aus denen die Rechtecke aufgebaut sind. Ich unterscheide dabei volle (XXX) und leere ($X\ X$) Dreierkombinationen. Für beide Typen lege ich separate Listen an, deren Einträge jeweils Zeilen- und Spalteninformation der Postion auf dem Feld sind.
In einem zweiten Schritt durchlaufe ich dann die Liste der gefundenen XXX-Kombinationen und prüfe, ob es entsprechend darunter liegende $X\ X$- und XXX-Kobinationen in den beiden Listen gibt. Falls dem so ist erhöhe ich einen Zähler, der bei Null startet. Diese Methode nutzt die Bedingung, dass sich die Rechtecke nicht überlappen.

2 Umsetzung

Mein Programm habe ich in der Programmiersprache Python umgesetzt.
Zuerst werden mittels einer kleinen Routine die Baulwurffelder aus der gewünschten Textdatei, die ich der Routine als String übergebe, in ein zweidimensionales Array eingelesen. Dabei speichere ich es durch Boolsche Variablen False bzw. True, um Speicherplatz zu sparen.
Für den ersten Schritt, dem Auffinden von Dreierkombinationen, lege ich zwei Listen an

- *combs_middle:* enthält alle gefundenen Dreierkombinationen vom Typ XXX (also die oberen und unteren Deckel der Rechtecke). Die Liste besteht aus Listen mit zwei Elementen, die jeweils die Zeile und Spalte des ersten Zeichens der Kombination beinhalten
- *combs_leer:* enthält alle gefundenen Dreierkombinationen vom Typ $X\ X$ (also die mittleren Reihen der Rechtecke). Die Liste enthält jeweils die Zeile und Spalte des ersten Zeichens der Kombination

Ich arbeite mit zwei verschachtelten for-Schleifen. Die erste, mit der Variablen x, durchläuft die Zeilen des Arrays. Für jede Zeile durchläuft die zweite Variable, i, die Spalten. Wird eine Baulwurfhügel an der Position x,i gefunden, also ist an der Stelle *True* im Array *file_list[x][i]*, dann wird mit mehreren if-Abfragen geprüft, ob es eine der gesuchten Dreierkombos ist. Im Grunde gibt es ein definiertes Vorgehen für XXX und eines für $X\ X$, die ich allerdings in der Schleife zusammengefasst habe, was es tatsächlich etwas komplizierter aussehen lässt, als es tatsächlich ist. In der Schleife nutze ich mehrere Flags bzw. Variablen.

- *last_i:* Variable, der eine Zahl zugewiesen wird (Spaltenindex eines möglichen ersten Hügels in einer Reihe), wenn ein Hügel vorhanden ist; die Variable wird bei jeder neuen Zeile mit *None* initialisiert und beim Durchlaufen der Spalten entsprechend aktualisiert

- *last_ i_ 2:* Variable, der eine Zahl zugewiesen wird (Spaltenindex eines zweiten Hügels in einer Reihe), wenn auf der vorigen Position (*last_ i*) ein Hügel ist; die Variable wird bei jeder neuen Zeile mit *None* initialisiert und beim Durchlaufen der Spalten entsprechend aktualisiert
- *comb:* Variable, der eine Zahl zugeswiesen wird (Spaltenindex eines dritten bzw. letzten Hügels in einer Reihe), wenn auf der vorigen Position (*last_ i_ 2*) ein Hügel ist; die Variable wird bei jeder neuen Zeile mit *None* initialisiert und beim Durchlaufen der Spalten entsprechend aktualisiert
- *leer_ combo:* Flag (Boolsche Variable), das mir eine mögliche *X X* Kombination anzeigt
- *end:* Flag, dass verwendet wird, um den Spezialfall von vier oder mehr Hügeln hintereinander in einer Reihe zu behandeln. Die Variable wird *True* gesetzt, sobald vier Hügel hintereinander folgen.

Im Detail: Fangen wir mit dem einfachen Fall an:

- An der Stelle *file_ list[x][i]* wird kein Baulwurfhügel gefunden. In diesem Fall kann es sein, dass wir in der Mitte einer *X X* sind. Dies prüfen wir mit der Variablen *last_ i*. Ist diese mit einer Zahl definiert, so setze ich das Flag *leer_ combo* auf *True*, um anzuzeigen, dass wir genau diesen Fall haben. Ansonsten setzen ich *leer_ combo* auf *False*, um zu wissen, dass wir nicht in diesem Fall sind. Wir wissen aber in beiden Fällen, dass wir keinesfalls in einer Folge von Baulwurfhügeln sind und setzen daher die Variablen *last_ i*, *last_ i_ 2* und *comb* auf *None*. Wir können nun in der verschachtelten Schleife ein Feld weiter gehen.
- An der Stelle *file_ list[x][i]* wird ein Baulwurfhügel gefunden. Wir setzen zunächst das Flag *end* auf *False*. Dann prüfen wir, ob es an der vorherigen Stelle einen Baulwurfhügel gab. Fangen wir wieder mit dem einfacheren Fall an: es gab keinen. Dann mache ich in die Liste der *X X*-Kombinationen (also *combs_ leer*) einen Eintrag, sofern das Flag *leer_ combo* gesetzt ist. In jedem Fall setze ich für den nächsten Schritt die Variable *last_ i* auf die Spaltenposition *i* und setze *leer_ combo* wieder auf *False*. Im Fall, dass das letzte Feld auch einen Baulwurfhügel hatte gibt es wieder zwei Fälle zu unterscheiden. Es gab entweder nur einen oder bereits zwei oder mehr Hügel.
  - Es gab nur einen (meine Variable *last_ i_ 2* ist nicht gesetzt): wir setzen *last_ i_ 2* auf den aktuellen Wert *i*
  - Es gab bereits zwei (meine Variable *last_ i_ 2* ist gesetzt): Wir sind also fündig geworden mit einer *XXX* Kombination. Wieder gibt es zwei Möglichkeiten:
    - Wir sind genau bei einem dritten Hügel in der Reihe (Die Variable *comb* ist nicht gesetzt). Wir setzen *comb* auf die akutellen Wert *i*.
    - Wir sind bei vier oder mehr Hügeln in einer Reihe (die Variable *comb* ist gesetzt): Ich speichere die gefundenen Kombination *XXX* in die Liste *combs_ middle* ab und bereite den check der nächsten Position vor, indem ich nacheinander *last_ i* auf *last_ i_ 2* und *last_ i_ 2* auf *comb* und *comb* auf *i* setze. Zudem setze ich das Flag *end*, um festzuhalten, dass die Kombination bereits in die Liste *combs_ middle* aufgenommen wurde.

  Bevor ich zum nächsten Feld gehe, prüfe ich, ob wir Dreierkombination gefunden haben (die Variable *comb* ist gesetzt) und nicht der Sonderfall von vier oder mehr Hügeln in einer Reihe vorhanden ist (das Flag *end* ist *False*). Wenn beides zutrifft, nehme ich die Kombination in die Liste der *XXX*-Kombinationen (Liste *combs_ middle*) auf. Dann geht es in der Schleife weiter, bis alle Felder ausgewertet sind.

Im zweiten Schritt werte ich die Listen *combs_ middle* und *combs_ leer* aus und suche die Rechtecke. Dazu verwende ich noch eine dritte Liste, *full_ leer_ comb*. Diese enthält alle 9-er Kominationen vom Typ *XXX*, *X X*, *X X* in jeweils aufeinander folgenden Zeilen, die in ein und derselben Spalte sind. Abgespeichert werden jeweils zwei Werte, die Zeile und die Spalte, in der eine weitere *XXX* Kobination gefunden werden müsste, um das Rechteck zu schließen, bildlich gesprochen also der untere Deckel des Baulwurf-Rechtecks. Mit einer verschachtelten for-Schleife, die in der ersten Ebenen alle Elemente der *combs_ middle* Liste (also *XXX*-Kombinationen) und in der zweiten Ebene die *combs_ leer* Liste (also *X X*-Kombinationen) durchgeht, wird mit if-Abfragen geprüft, ob diese Dreierkombinationen tatsächlich passend untereinander liegen (also in der gleiche Spalte und übereinander bezüglich Zeile). Mit einem Hilfsflag *half_ com* (Typ Boolean) markiere ich zwischendurch, ob bei einer zwei zeilen unter einer *XXX* Kombination bereits eine *X X* Kombination existiert. Diese einfache Methode funktioniert, da die Einträge der Liste in erster Ordnung nach aufsteigender Zeilenfolge (in zweiter Ordnung nach Spalte) sortiert sind. In diesem und

nur diesem Fall (also dem der oben beschriebenen 9er-Kombination), schreibe ich einen Eintrag in die Liste *full_ leer_ comb* und prüfe mit einer weiteren if-Abfrage, ob diese Stelle in der Liste *combs_ middle*, also den *XXX*-Deckeln enthalten ist. Wenn das der Fall ist, habe ich ein Rechteck identifiziert und erhöhe den Zähler *count*. Die Routinen lasse ich für allen 6 Karten hintereinander ablaufen und gebe jeweils die gefundene Zahl der Rechtecke aus.

3 Beispiele

Hier die Ergebnisse, die mir das Programm für alle 6 Felder liefert:

- Karte 0: 7 Rechtecke
- Karte 1: 37 Rechtecke
- Karte 2: 32 Rechtecke
- Karte 3: 318 Rechtecke
- Karte 4: 3193 Rechtecke
- Karte 5: 20 Rechtecke

4 Quellcode

Ich füge an dieser Stelle nur die beiden Teile der Hauptroutine ein. Den Aufruf und die File-Einleseroutine brauche ich sicherlich nicht abzudrucken.
Erster Teil der Hauptroutine, in der ich die Dreierkombinationen im Array auffinde und die Positionen in die Liste speichere:

```
combs_middle = []
combs_leer = []
for x in range(len(file_list)):              # x durchlaeuft die Zeilen im Feld
    last_i = None                            # Erster Huegel einer moeglichen Dreierkombination XXX
    last_i_2 = None                          # Zweiter Huegel einer moeglichen Dreierkombination XXX
    comb = None                              # Dritter Huegel einer Dreierkombination XXX
    leer_combo = False                       # Flag, dass mir sagt, ob ich an einem leeren Feld
    end = False                              # Diese Flag wird True in dem Fall, dass vier oder mehr
    for i in range(len(file_list[x])):       # i durchlaeuft die Spalten in der x'ten Zeile
        if file_list[x][i]:                  # True bedeutet Baulwurfhuegel (also X in der Textdatei)
            end = False                      # Flag zum Start Initialisieren
            if last_i is not None:           # Wir sind an der 2. Stelle einer moegl. Kombination XXX
                if last_i_2 is not None:     # Wir sind an der 3. Stelle einer XXX Kombination
                    if comb is not None:     # Wir haben 4 oder mehr Huegel hintereinander - Sonderfall
                        end = True           # Flag des Sonderfalls wird gehisst
                        combs_middle.append([x, comb]) # Sonderfall in der Liste eintragen
                        last_i = last_i_2    # Vorbereitung naechstes Feld
                        last_i_2 = comb      # Vorbereitung naechstes Feld
                    comb = i                 # Vorbereitung naechstes Feld, unabh. vom letzten if
                else:                        # Das ist der Fall, dass wir auf einem Huegel sind und
                    last_i_2 = i             # Vorbereitung naechstes Feld
            else:                            # Das ist der Fall, dass wir auf einem Huegel stehen,
                                             # aber das vorige Feld leer ist
                if leer_combo:               # Wir schauen in den Zwischenspeicher leer_combo,
                                             # ob der aktuelle Huegel eine Komb. X X vervollstaendigt
                    combs_leer.append([x, i - 1])
                last_i = i                   # Wir vemerken den letzten Huegel und weiter geht es
                leer_combo = False           # In der naechsten Position sicher kein X X mehr
            if comb is not None and not end:# hier tragen wir alle XXX Kombinationen ein,
                                             # die nicht Sonderfall mehr als vier sind
                combs_middle.append([x, last_i_2])
        else:                                # Wenn kein Baulwurfhuegel auf der Postion ist,
                                             # gibt es zwei Moeglichkeiten
            if last_i is not None:
                leer_combo = True            # Es koennte noch eine X X Kombination sein,
                                             # das merken wir uns fuer spaeter
            else:
                leer_combo = False           # Es ist keine X X Kombination, da zwei Positionen
            last_i = None                    # In jedem Fall resetten wir jetzt wieder alle
```

```
            last_i_2 = None
41          comb = None
```

Zweiter Teil der Hauptroutine, in der ich aus den Listen die Rechtecke identifiziere und zähle:

```
count = 0
half_comb = False
full_leer_comb = []
for comb in combs_middle:
    for new_comb in combs_leer:
        if new_comb[0] == comb[0] + 1 and new_comb[1] == comb[1]:
            if not half_comb:
                half_comb = True
                continue

            elif new_comb[0] == comb[0] + 2 and new_comb[1] == comb[1] and half_comb:
                full_leer_comb.append([comb[0] + 3, comb[1]])
                half_comb = False

        if comb in full_leer_comb:
        count += 1
```