Aufgabe 1: Blumenbeet

Team-ID: 00485

Team-Name: FXC3

Bearbeiter/-innen dieser Aufgabe:
Leonardo Benini

3. November 2019

Inhaltsverzeichnis

Lösungsidee

Die Anzahl an verschiedenen Kombinationen des Beetes sind überschaubar und aufgrund dessen bietet sich ein Brute Force Attacke sehr gut zur Lösung dieser Aufgabe an. Das heißt das Programm probiert rekursiv alle möglichen Farbkombinationen aus, evaluiert die Punktezahl und merkt sich die beste Kombination. Um die rekursive Methode aber zu beschleunigen und nicht unnötige Kombinationen auszuprobieren, habe ich eine Einschränkung eingebaut, die die Laufzeit deutlich verbessert: Es werden nur Kombinationen betrachtet, die dir richtige Anzahl an verschiedenen Farben haben. Wenn sich zum Beispiel der Kunde 7 verschiedene Farben wünscht, macht es überhaupt keinen Sinn die Kombinationen für 6 verschiedene Farben auszuprobieren, usw. Zum einen ist die Auswahl der Farben auf die schon benutzten Farben beschränkt, wenn die gewünschte Anzahl an Farben erreicht ist. Umgekehrt erzwingt der Algorithmus, dass immer die gewünschte Anzahl an verschiedenen Farben erreicht werden.

Jede betrachtete Kombination wird gewertet und mit der bis zu diesem Zeitpunkt besten Kombination verglichen. Falls die neue Kombination eine höhere Punktzahl aufweist, wird diese als „Beste“ abgespeichert.

Umsetzung

Die Lösungsidee wird in Python implementiert. Zuerst wird das Beispiel eingelesen und es werden die Verbindungen, die die einzelnen Töpfe im Beet haben, definiert. Das Beet wird als Liste dargestellt, wobei Index 0 für die oberste Pflanze steht und der Rest Reihe für Reihe durchgegangen wird (siehe nebenstehende Abbildung).

Anfangs wird die opt_beet Methode aufgerufen, die wiederum eigentlich nur die rekursive Methode loop in Gang setzt. Diese besteht aus einer for Schleife, die durch alle möglichen Farben („colours_ava“) durchgeht und dann jeweils wieder die loop Methode aufruft, dann aber für den nächsten Topf. Sobald der Vorgang am letzten Topf angekommen ist, werden die

Punkte für das Beet berechnet und falls dieses besser ist als das zuvor Beste, wird dies gespeichert. Daraufhin returned die Methode und so werden rekursiv alle Möglichkeiten ausprobiert.

Damit wie in der Lösungsidee angesprochen keine unnötigen Berechnungen durchgeführt werden, kann sich die Methode nicht immer irgendeine von den sieben Farben aussuchen. Dies wird garantiert durch eine Liste an Farben, die noch „erhältlich“ sind. Wenn zum Beispiel die ersten drei Töpfe eine blaue Blume beinhalten und der Kunde 7 verschiedene Farben will, dann müssen die restlichen sechs Töpfe jeweils mit einer anders farbigen Blume gefüllt werden. Außerdem werden die erhältlichen Farben den benutzten Farben gleichgesetzt, wenn die Anzahl an verschiedenen Farben schon erreicht ist.

```
def loop(col_count, colours_ava, colours_used, i): # recursive method
    d = colours_ava.copy()
    z = colours_used.copy()
    f = col_count
    global points, beet, preferred, best_beet, max_colours
    for e in range(len(colours_ava)):
        beet[i] = colours_ava[e]
        colours_used.add(colours_ava[e])
        dif = True
        for plant in beet[:i]:
            if colours_ava[e] == plant:
                dif = False
        if dif != False: col_count -= 1
        if max_colours == max_colours - col_count:
            colours_ava = list(colours_used).copy()
        if col_count == 8 - i:
            for plant in beet[:i + 1]:
                try:
                    del colours_ava[colours_ava.index(plant)]
                except: pass
        if i < 8:
            loop(col_count, colours_ava, colours_used, i+1)
        if i == 8:
            p = calc_points(beet, preferred)
            if p > points:
                points = p
                best_beet = beet.copy()
        colours_ava = d.copy()
        col_count = f
        colours_used = z.copy()
    return
```

Beispiele

Beispiel 1(blumen.txt):

```
[1, 6, 6, 2, 1, 7, 3, 4, 5] oder ['blau', 'rot', 'rot', 'gelb', 'blau', 'tuerkis',
'gruen', 'orange', 'rosa'] = 14 Punkte
```

Beispiel 2(blumen1.txt):

```
[6, 7, 7, 6, 6, 6, 7, 7, 6] oder ['rot', 'tuerkis', 'tuerkis', 'rot', 'rot', 'rot',
'tuerkis', 'tuerkis', 'rot'] = 36 Punkte
```

Beispiel 3(blumen2.txt):

```
[3, 6, 6, 7, 7, 7, 6, 6, 7] oder ['gruen', 'rot', 'rot', 'tuerkis', 'tuerkis',
'tuerkis', 'rot', 'rot', 'tuerkis'] = 32 Punkte
```

Beispiel 4(blumen3.txt):

```
[1, 2, 3, 4, 6, 5, 7, 7, 6] oder ['blau', 'gelb', 'gruen', 'orange', 'rot', 'rosa',
'tuerkis', 'tuerkis', 'rot'] = 13 Punkte
```

Beispiel 5(blumen4.txt):

```
[1, 5, 5, 3, 6, 6, 4, 7, 2] oder ['blau', 'rosa', 'rosa', 'gruen', 'rot', 'rot',
'orange', 'tuerkis', 'gelb'] = 22 Punkte
```

Beispiel 6(blumen5.txt):

```
[1, 5, 5, 3, 6, 6, 4, 7, 2] oder ['blau', 'rosa', 'rosa', 'gruen', 'rot', 'rot',
'orange', 'tuerkis', 'gelb'] = 22 Punkte
```

Quellcode

Das Projekt ist auf Github einsehbar.

Link: https://github.com/FloxiGH/BWINF2019