Aufgabe 1: Blumenbeet

Team-ID: 00198

Team-Name: Die Dullis

Bearbeiter/-innen dieser Aufgabe:
Lasse Friedrich

24. November 2019

Aufgabenstellung 1
Lösungsidee 1
Umsetzung 2
Beispiele 3
Quellcode 5
Literaturverzeichnis 8

Aufgabenstellung

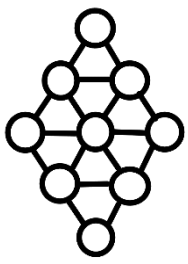
Es soll aufgrund angegebener Präferenzen und der Anzahl unterschiedlicher Blumenfarben, das beste Blumenbeet, mit der höchsten Bewertung, herausgefunden werden.

Lösungsidee

Hierzu wird als Kerngedanke das Prinzip der Permutation[1] verwendet. Permutation bezeichnet das Durchprobieren aller möglichen Kombinationen. Ähnlich zu einem Fahrradschloss, welches man durch Ausprobieren aller Kombination öffnen kann, ist es auch möglich, das beste Blumenbeet zu ermitteln, in dem man alle Blumenfarben für jeden Topf systematisch durchprobiert. Dabei müssen dann 9! (362.880) Kombinationen durchprobiert werden.

Um die Anzahl der Möglichkeiten zu reduzieren, werden gewisse Blumentöpfe vor der Permutation, entsprechen einer einfachen Logik vorab „bepflanzt“. Somit ist es möglich, die vorherige Anzahl von 9! (362.880) auf 5! (120), außer in gewissen Grenzfällen, zu reduzieren.

Die Bewertung des Blumenbeets basiert auf einem einfachen Knoten-Kanten-Modell. Hierzu wird für jeden Blumentopf (Knoten) eine Verbindung (Kante) zu jedem seiner Nachbarn erstellt. Diese Kanten bekommen ihre Bewertungen basierend auf den Blumenfarben, zwischen denen sie liegen. Die Summe aller dieser einzelnen Kantenbewertungen ergibt die Gesamtbewertung für das Blumenbeet.



Umsetzung

Die oben beschriebene Lösung wurde in der Programmiersprache C# und den dazugehörigen Standard Bibliotheken umgesetzt.

Als erster Schritt wird für jede einzelne, angegebene Farbe eine individuelle **Farbenbewertung** kurz ***FB*** erstellt. Dies geschieht durch einfache Addition der hinter den einzelnen Kombinationen genannten Bonuspunkte. Wenn zum Beispiel:

- Rot Blau **3**
- Blau Tuerkis **2**

als Kombinationen genannt werden, ist die Farbenbewertung *FB* von Rot = **3** (3 + 0), die *FB* von Blau = **5** (3 + 2) und zuletzt die *FB* von Tuerkis = **2** (0+2).

Auf der Basis dieser Bewertungen kann im Anschluss eine **Kombinationsbewertung** kurz ***KB*** erstellt werden. Diese wird dafür benutzt, die noch verbleibenden Farben des Blumenbeets mit den Farben aufzufüllen, die die höchste Gesamtbewertung für das Blumenbeet bringen. Die *KB* setzt dich dabei aus der Summe der einzelnen *FB* zusammen. In Bezug auf das genannte Beispiel ist die *KB* von **(Rot Blau)** = ((***FB* Rot**) + (***FB* Blau**) = (5 + 3) = 8 und die *KB von* (Blau Tuerkis) = (5 + 2) = 7.

Nach diesem Schritt wird eine ***Farbenliste*** erstellt. Diese enthält maximal neun Farben, welche ausreichen, um jeden **Knoten** (einfache Klasse mit dem Attribut Farbe) eine Farbe zu zuweisen. Die ***Farbenliste*** wird zuerst mit den Farben aus den Kombinationen *(hier: Blau, Rot, Tuerkis)* gefüllt. Danach werden weitere, zufällig ausgewählte Farben hinzugefügt, bis die geforderte Farbanzahl erreicht ist. Zum Schluss wird die Liste mit den Farben aus der Kombination mit der höchsten *KB* aufgefüllt. Falls sich das nicht ausgeht, weil noch eine Farbe fehlt, wird die Farbe mit der besten *FB* hinzugefügt.

Da es **maximal sieben** verschiedene **Farben** gibt, es jedoch **neun** verschiedene **Knoten** zu besetzen gilt, kann man den zwei **freien Knoten** immer die Farben der **besten Farbenkombination** (höchste *KB(hier: Rot Blau)*) zuweisen.

Dadurch erhält man die beste Farbenkombination immer **zweimal** auf einem Blumenbeet. Damit kann man diese beiden Farbkombinationen immer gebündelt an **dieselbe Position** setzen. Dabei muss darauf geachtet werden, dass **Knoten** mit möglichst **wenig Nachbarn** gewählt werden, um keine anderen günstigen Kombinationen zu verbauen. Darüber hinaus sind die Farben so anzuordnen, dass sie einerseits die **maximale Bewertung** bekommen und andererseits möglichst viele Kombinationsmöglichkeiten offenlassen.

Das wird erreicht, wenn eine Farbe aus der Kombination mit der höchsten ***KB*** *(hier: Blau/Rot)* auf den **Knoten** mit den **meisten Nachbarn** gesetzt wird und dieser als **Startpunkt** verwendet wird. Veranschaulicht wird dieses Vorgehen mit folgendem Beispiel:

Im nächsten Schritt werden diese vier Farben aus der ***Farbenliste*** entfernt und den oben gezeigten Knoten zugewiesen. Die daraus resultierende neu konvertierte ***Farbenliste*** wird mit der **Farbe** von **jedem Knoten,** der noch **nicht besetzt** ist gleichgesetzt. Im Anschluss wird diese ***Farbenliste*** permutiert.

Dabei wird nach jeder Permutation[1] die Bewertung für die aktuellen Farben der Knoten neu berechnet. Dies geschieht durch die **Klasse Kante**, welche Referenzen zu den Knoten hat, zwischen denen sie liegt. Durch die Methode ***GewichtBestimmen()*** vergleicht die Kante die Farben der Knoten in **beiden Varianten.**

Im gezeigten Beispiel könnte bei der ersten Kombination entweder der erste Knoten Blau und der zweite Rot sein, oder es könnte der erste Knoten Rot und der zweite Knoten Blau sein. In beiden Fällen würde das als richtige Kombination gelten.

Falls es zu einer **Übereinstimmung** kommt, werden die **Bonuspunkte** der Kombination als eine ***Kantenbewertung*** verwendet. Die **Summe** aller ***Kantenbewertungen*** ergibt die ***Gesamtbewertung*** für dieses Blumenbeet.

Zuletzt wird noch die ***maximale Bewertung*** definiert. Diese wird immer dann ermittelt und aktualisiert, wenn eine Blumenbeet Variante eine höhere Bewertung erzielen kann als alle vorangegangenen. Diese neue Bewertung wird nun zur ***maximalen Bewertung.*** Das zu dieser Zeit beste Blumenbeet wird dann zusammen mit der ***maximalen Bewertung*** und einem Permutationszähler ausgegeben.

Beispiele

Hier werden nun die vorgegebenen Beispiele und im Anschluss noch Grenzfälle betrachtet.

```
7 (Farben Anzahl)
2 (Kombinationen Anzahl)
rot blau 3
rot tuerkis 2
```

rot

blau blau

rosa rot tuerkis

gruen gelb

orange

Bewertung: 14 Counter: 1

>>> Sehr ähnlich zum oben genannten Beispiel. (Beispielaufgabe blumen.txt)

```
2
2
rot tuerkis 3
gruen rot 1
```

rot

tuerkis tuerkis

rot rot rot

tuerkis tuerkis

rot

Bewertung: 36 Counter: 13

>>> Weil mehr Farben in den Kombinationen angegeben wurden, als eigentlich erlaubt waren, wurde die Farbe mit dem schlechtesten ***FB*** aus der ***Farbenliste*** zuvor entfernt. (Beispielaufgabe blumen1.txt)

```
3
2
tuerkis rot 3
rot gruen 1
```

tuerkis

rot rot

tuerkis tuerkis gruen

rot rot

tuerkis

Bewertung: 32 Counter: 169

>>> Hier kam das erste Mal das Prinzip zum Tragen, dass man beide Farben der besten Kombination in der Mitte probieren muss, wenn man nur die Farbe mit dem höchsten ***FB*** *(hier: rot)* ausprobieren würde wäre die beste Bewertung 31. (Beispielaufgabe blumen2.txt)

```
7
2
rot tuerkis 3
gruen rot 1
```

rot

tuerkis tuerkis

rosa rot gruen

gelb blau

orange

Bewertung: 13 Counter: 1

>>> Auch wieder sehr ähnlich zum ersten Beispiel. (Beispielaufgabe blumen3.txt)

```
7
6
rot tuerkis 3
gruen rot 1
rot rosa 3
blau rosa 2
gelb orange 1
gruen orange 1
```

rot

rosa rosa

blau rot tuerkis

orange gruen

gelb

Bewertung: 20 Counter: 61

>>>Das erste Mal mit etwas mehr Kombinationen, auch hier funktioniert die vierer Kombination oben im Blumenbeet. (Beispielaufgabe blumen4.txt)

```
7
7
rot tuerkis 3
```

```
rot gruen 1
rot rosa 3
blau rosa 2
gelb orange 1
gruen orange 1
blau tuerkis 2
```

rot

tuerkis tuerkis

gruen rot blau

orange rosa

gelb

Bewertung: 22 Counter: 64

>>>Der erste Grenzfall. Es werden alle Farben und alle Kombinationen in Anspruch genommen. Auch hier gab es keine Probleme. (Beispielaufgabe blumen5.txt)

```
1
1
rot rot 3
```

rot

rot rot

rot rot rot

rot rot

rot

Bewertung: 48 Counter: 1

>>>Der zweite Grenzfall, bei welchem nur eine Farbe und nur eine Kombination gegeben ist. Hier sieht man das die allerhöchste Bewertung 48 beträgt.

Quellcode

```
//Modulimport
using System;
using System.Linq;
using System.Text.RegularExpressions;
   static class Program
   {
      //Die später gestellte Farbenliste
      static List<string> farbenliste = new List<string>();
      //Liste aller Knoten (Blumentöpfe)
      public static List<KNOTEN> knotenpunkte = new List<KNOTEN>();
       //Liste aller Kanten (Verbindung zweier Blumentöpfe)
      static List<KANTE> kanten = new List<KANTE>();
      //Aufzählung der Farben aus dem Input mit ihrem Farbenwert
      static Dictionary<string, int> farbenwerte = new Dictionary<string, int>();
      }
      private static void FarbenListeStellen()
      {
         //Entfernt eine Farbe falls es weniger Farben geben soll als in den kombinationen angegeben
         for (int e = farben_anzahl; e < farbenwerte.Keys.Count; e++)
```

```
{
   int min_farbe_wert = 9999;
   string min_farbe = null;
   foreach (string f in farbenwerte.Keys)
   {
      if (farbenwerte[f] < min_farbe_wert)
      {
         min_farbe_wert = farbenwerte[f];
         min_farbe = f;
      }
   }
   farbenwerte.Remove(min_farbe);
}
//fügt die farben der Kombinationen hinzu
foreach (string k in farbenwerte.Keys)
{
   for (int i = 0; i < alle_farben.Count; i++)
   {
      if (alle_farben[i] == k)
      {
         alle_farben.Remove(k);
      }
   }
   farbenliste.Add(k);
}
//fügt andere Farben hinzu, wenn die farben anzahl noch nicht erreicht wurde
for (int j = 0; j < farben_anzahl - farbenliste.Count; j++)
{
   int r = j;
   farbenliste.Add(alle_farben[r]);
   j--;
   alle_farben.Remove(alle_farben[r]);
}

//speichert die beste Farbe aus farbenwerte
max_farbe = farbenwerte.Aggregate((x, y) => x.Value > y.Value ? x : y).Key;

//fügt die besten Farben so lange hinzu bis alle Blumentöpfe eine Farbe hätten
while (farbenliste.Count < knotenpunkte.Count)
{
   //Wenn mehr als 2 Farben zu füllen sind, immer die beste Kombination nochmal hinzufügen
   if (knotenpunkte.Count - farbenliste.Count >= 2)
   {
      for (int b = 0; b < kombinationen.GetLength(0); b++)
      {
         if (int.Parse(kombinationen[b, 3]) == max_kombinations_bewertung)
         {
            if (farbenliste.Count < knotenpunkte.Count)
            {
               farbenliste.Add(kombinationen[b, 0]);
               farbenliste.Add(kombinationen[b, 1]);
               max_kombination[0] = kombinationen[b, 0];
               max_kombination[1] = kombinationen[b, 1];
            }
         }
      }
   }
   //sonst nur die beste Farbe hinzufügen
   else
   {
      int max_farbe_wert = 0;
      foreach (string f in farbenwerte.Keys)
```

```
                {
                    if (farbenwerte[f] > max_farbe_wert)
                    {
                        max_farbe_wert = farbenwerte[f];
                        max_farbe = f;
                    }
                }
                farbenliste.Add(max_farbe);
            }
        }
}
        //ändert die Position der einzelnen Elemente innerhalb einer Liste
        private static IList<IList<T>> Permutationen<T>(IList<T> list)
        {
            List<IList<T>> perms = new List<IList<T>>();
            if (list.Count == 0)
                return perms;
            int factorial = 1;
            for (int i = 2; i <= list.Count; i++)
                factorial *= i;
            for (int v = 0; v < factorial; v++)
            {
                List<T> s = new List<T>(list);
                int k = v;
                for (int j = 2; j <= list.Count; j++)
                {
                    int other = (k % j);
                    T temp = s[j - 1];
                    s[j - 1] = s[other];
                    s[other] = temp;
                    k = k / j;
                }
                perms.Add(s);
            }
            return perms;
        }
        //addiert das Gewicht aller Kanten und gibt diese zurück
        private static int BewertungBestimmen()
        {
            int score = 0;
            foreach (KANTE k in kanten)
            {
                k.gewicht = 0;
                k.GewichtBestimmen();
                score += k.gewicht;
            }
            return score;
        }
    }
}
//Knoten Klasse repräsentiert die einzelnen Blumentöpfe
    class KNOTEN
    {
        //Parameter zum überprüfen ob dieser Knoten noch besetzt werden muss
        public bool isLocked = false;
        public string farbe { get; set; }

    }
 //befindet sich immer zwischen zwei Knoten um mögliche kombinationen zu finden und eine Gewichtung zu
bestimmen
    class KANTE
    {
```

```
public int gewicht { get; set; }

private KNOTEN[] knoten = new KNOTEN[2];

public void GewichtBestimmen()
{
   if (knoten[0] != null && knoten[1] != null && kombinationen != null)
   {
      for (int i = 0; i < kombinationen.GetLength(0); i++)
      {
         if (kombinationen[i, 0] == knoten[0].farbe && kombinationen[i, 1] == knoten[1].farbe)
         {
            gewicht += int.Parse(kombinationen[i, 2]);
         }
         else if (kombinationen[i, 0] == knoten[1].farbe && kombinationen[i, 1] == knoten[0].farbe)
         {
            gewicht += int.Parse(kombinationen[i, 2]);

         }
      }
```

Literaturverzeichnis

[1]Stack Overflow
https://stackoverflow.com/questions/756055/listing-all-permutations-of-a-string-integer
(abgerufen am 10.11.2019)