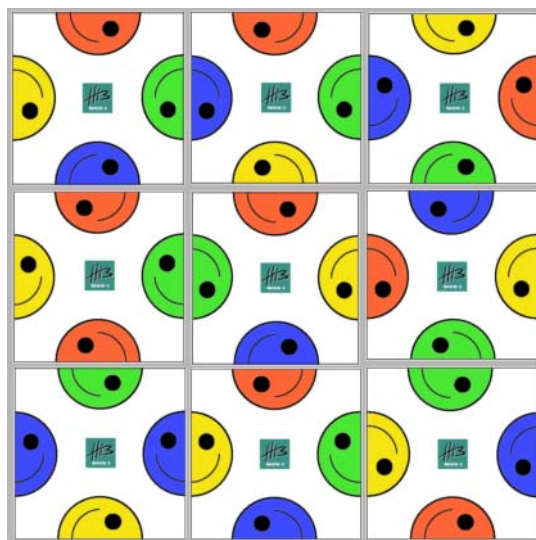


Das HIB Smiley-Puzzle

Die Aufgabe: Du erhältst 9 quadratische Kärtchen, auf denen bunte halbe Smileys zu sehen sind.



Lege diese Kärtchen in Form eines 3 mal 3 Quadrates auf, sodass die Smileys aneinander grenzender Kärtchen gleiche Farbe haben und ein vollständiges Gesicht erhalten.



Ein Ausschnitt aus einer Lösung:

Wenn Du willst, drucke die Seite farbig, schneide die Kärtchen aus und versuch Dein Glück.

Mathematiker wenden sich mit Grausen, da man 9 Kärtchen in 4 Verdrehungen auf 9 Plätze (bei je 4 Symmetrielösungen) auf $36 \times 32 \times 28 \times \dots \times 8 \times 4 / 4$, das sind über 23 Milliarden Arten auflegen kann. Und all das durchzuprobieren kann auch einen fleißigen Mathematiker ins Schwitzen bringen.

Wir Informatiker werden diese Aufgabe selbstverständlich mit dem Computer lösen. Schließlich geben wir uns auch nicht mit der Suche nach 'einer Lösung' zufrieden, sondern wollen gleich wissen, *wie viele* unterschiedliche Lösungen es denn gibt (wenn es überhaupt welche gibt)!

Algorithmische Lösung

Wie so oft – hat man erst einmal eine gute Möglichkeit gefunden, die Daten (Kärtchen) in eine dem Computer zugängliche Struktur umzuwandeln, ist der Rest ein Kinderspiel.

In bewährter 'Bottom-Up' Programmiermethode klären wir einfach der Reihe nach die Begriffe, mit denen wir den Computer konfrontieren wollen. Wenn uns dabei alles sonnenklar ist, ist es dem Computer später auch klar.

Die Daten

1.) Jedes Kärtchen hat vier Bilder.

Beginnen wir traditionell oben (Norden) und gehen im Gegenuhrzeigersinn, so erhalten wir eine Liste von Bildeigenschaften. Diese bestehen aus der **Farbe** des Smileys und der Information, ob es sich um eine **linke oder rechte** Gesichtshälfte handelt. (Legen wir später Kärtchen aneinander, so ist nur wichtig, dass die Farbe gleich ist, und die Hälfte unterschiedlich.) Nehmen wir für 'rechts' die Zahl 0 und für 'links' die Zahl 1, so können wir das linke obere Kärtchen bereits beschreiben. Wir wählen für die Bildeigenschaften ein Tupel (*farbe*, *haelfte*) und verpacken das ganze in eine Liste:

```
[ ("rot",0), ("gelb",1), ("blau",1), ("gruen",0) ]
```

Und wie beschreiben wir alle Kärtchen? Einfach als Liste von 9 solchen Einzelkartenlisten.

Ich gebe Dir hier die Liste zum Herauskopieren:

```
karten = [ ("rot",0), ("gelb",1), ("blau",1), ("gruen",0) ],
           [ ("rot",1), ("blau",1), ("gelb",0), ("gruen",0) ],
           [ ("gelb",0), ("blau",0), ("gruen",1), ("rot",1) ],
           [ ("rot",1), ("gelb",0), ("rot",0), ("gruen",1) ],
           [ ("rot",0), ("gruen",1), ("blau",1), ("gelb",0) ],
           [ ("blau",1), ("rot",1), ("gruen",0), ("gelb",0) ],
           [ ("gruen",0), ("blau",0), ("gelb",1), ("blau",1) ],
           [ ("rot",1), ("gelb",0), ("blau",0), ("gruen",1) ],
           [ ("gruen",0), ("gelb",1), ("rot",1), ("blau",0) ] ]
```

2.) Jedes Kärtchen kann gedreht werden.

Wir müssen also zu jeder Karte ihre **Orientierung** berücksichtigen. Wählen wir einfach 0 für die Ausgangsstellung (Norden), dann erscheint bei einer 90°-Drehung gegen den Uhrzeiger das Bild Nummer 3 oben, nach 2 Drehungen die Nummer 2, nach drei die Nummer 1, nach 4 die Nummer 0. Rechnerisch: $\text{neuesoben} = 4 - \text{altesoben}$

3.) Jedes Kärtchen kann auf jedem von 9 Plätzen liegen.

Numerieren wir das quadratische Feld in üblicher Weise von Null beginnend:

0	1	2
3	4	5
6	7	8

4.) Somit können wir alle Details präzise beschreiben:

Ein Kärtchen: das Tupel `karte=(bild,haelfte)`

Der Kärtchenvorrat: die Liste `karten = [karte0, karte1,, karte8]`

Die Platzierung eines Kärtchens als Tupel (`platznummer,orientierung`) in einer Liste, die diese Informationen für alle aufgelegten Kärtchen mitführt.

Funktionen

Und welche Funktionen werden wir benötigen? Wir bilden einfach nach, was wir beim Probieren von Hand tun!

– Funktion `oben(karte,orientierung)`

(zur Übung:) sie soll uns sagen, welches Bild wir in diesem Falle oben sehen können

- Funktion `bild(karte,orientierung,richtung)`
sie soll sagen, welches Bild die orientierte Karte in einer gewissen Richtung zeigt. Denk an die Modulo-Funktion!
- Funktion `passt(karte1,orient1,richtung,karte2,orient2)`
sie sagt uns, ob diese beiden orientierten Karten in einer Richtung (bezogen auf die erste Karte) passt. Klar: gleiche Farbe, andere Hälfte

```
def bild(karte,orient,richtung):
    return karten[karte][(-orient+richtung)%4]

entgegen = [2,3,0,1] # die Gegenrichtungen zu N W S O

def passt(k1,o1,r,k2,o2):
    bild1 = bild(k1,o1,r)
    bild2 = bild(k2,o2,entgegen[r])
    return (bild1[0]==bild2[0]) and (bild1[1]!=bild2[1])
```

Nun müssen wir aber doch über einen geeigneten Algorithmus nachdenken. Die Antwort ist aber leicht, wenn wir uns etwa an das Damenproblem oder die Rössel-Sprünge erinnern: **Backtracking**. Wir simulieren das schrittweise Anlegen: "**ein Kärtchen nach dem anderen hinlegen, und wenns nicht passt, drehen oder wieder wegnehmen und ein anderes versuchen**".

Wir betrachten ein Quadrat-Feld nach dem anderen und legen probeweise eine Karte hin. Wenn sie passt, machen wir mit dieser bisherigen Stellung weiter, sonst probieren wir eine andere Orientierung oder eine andere Karte. (Ich hoffe Du erkennst: Beim Damenproblem war es ganz genauso)

Welche Parameter brauchen wir?

An welcher Platznummer wir momentan arbeiten müssen wir wissen, und welche Kartennummern noch zur Verfügung stehen. Eine Liste mit den Einträgen der bisher angelegten Karten brauchen wir auch, sonst können wir ja gar keine Entscheidung treffen, ob die neue Karte passt oder nicht.

<pre>def anlegen(platz,vorrat,bisher): if vorrat == []: anzeigen(bisher) return for karte in vorrat: uebrig = vorrat[:] uebrig.remove(karte) for orient in range(4): if erlaubt(platz,karte,orient,bisher): anlegen(platz+1,uebrig, bisher+[(karte,orient)])</pre>	<p>kein vorrat mehr? Fertig und Lösung ausgeben!</p> <p>jede vorrätige Karte</p> <p>aus Vorrat entfernen in jeder Orientierung versuchen wenn Anlegen erlaubt mit dieser Stellung weitermachen</p> <p>sonst nichts tun,</p>
---	---

Ein Aufruf wie `anlegen(0,[0,1,2,3,4,5,6,7,8],[])` ergibt dann alle Lösungen!

Hier kamen allerdings noch zwei neue Funktionen vor:

- `anzeigen`: gibt eine Lösung (möglichst in hübscher Form) aus. `print` bisher klappt aber natürlich auch...
- `erlaubt`: testet, ob die Karte zu den bisherigen passt. Beachte: Du musst nur den linken und/oder oberen Nachbarn (falls vorhanden) berücksichtigen, da wir schrittweise anlegen. Jede neue Karte hat also ein recht kleines 'Umfeld'.

Jetzt kann die Fertigstellung des Programms nicht mehr schwierig sein Du musst nur `erlaubt` beibringen, für die jetzt gültige Umgebung in geeigneter Weise `passt` zu befragen...

Wie viele Lösungen findest Du?

Erweiterungen:

- Filtere die Lösungen aus, die sich nur durch Drehung der gesamten Anordnung von einer schon angezeigten Lösung unterscheiden. (Das reduziert die Gesamtanzahl auf ein Viertel)
- Ein Trick, der hier anwendbar ist: Beginne die Platzierungen in der Mitte. Dieses Kärtchen wird nicht gedreht und Du erhältst nur eindeutige Lösungen. Dafür wird allerdings die Bestimmung der zu prüfenden Umgebung eines neuen Kärtchens etwas aufwendiger.

– **ERFINDE NEUE PUZZLES**

Ändere die Farben und kontrolliere, ob das Puzzle noch lösbar ist. Und ob sich auch nicht zu viele Lösungen ergeben...

Nimm mehr Farben

Nimm mehr Figuren

Ändere die Figuren: Wie wärs mit bunten Kätzchen, Kühen,...

Du kannst vielleicht auch am Computer arbeiten und Photos von Verwandten einsetzen!

Was an unserem Programm zu ändern ist, ist nur die Liste der `karten`. Alles übrige bleibt wie es war!

Versuche ein **4x4 Puzzle!** (Vorsicht – längere Rechenzeit! Optimiere die Funktionen: möglichst wenige Aufrufe, statt die Farben als Strings zu behandeln, nimm besser Zahlen.)

- Beachte aber: der Mittelweg ist richtig. Zu wenige Farben: dann gibt's gleiche Kärtchen. Zu viele Farben: da gibt es weniger Alternativen zum Anlegen und das Puzzle wird einfacher zu lösen. Du wirst aber sicher rasch ein Gefühl für die günstigsten Verhältnisse entwickeln!