Junioraufgabe 1

Lösungsidee

Als erstes werden erst einmal alle Fächer befüllt. Wenn die Summe aller Fächer unter 20 liegt soll das erste volle Fach solange entleert werden bis entweder keine Ballons mehr vorhanden sind oder ein funktionierend Subset gefunden wurde. Ansonsten soll das Programm alle Subsets der Fächer generieren und überprüfen ob (mindestens) eines davon eine bestimmte (i-BallonsInsAusgabe, wobei i bei 20 anfängt) an Ballons besitzt. Wenn dies der Fall ist sollen alle Fächer aus dem gefunden Subset entleert und neu befüllt werden. Danach wird das Ausgabefach entleert. Sonst wird i solange erhöht bis ein passendes Subset gefunden wurde, welches dann entleert wird.

Umsetzung

Das Problem habe ich in Ruby gelöst. Die Fächer werden durch eine eigene Klasse repräsentiert, welche in *src/lib/storage.rb* zu finden ist. Der eigentliche Code ist in *src/app/main.rb*. (*src/bin/app.rb* ist nur ein „bootstrap“ für das Script/Programm)

Die Methode *make_round* wird so lange aufgerufen bis keine möglichen Schritte mehr vorhanden sind. Diese Methode wiederum wartet, falls die Summe der Fächer plus das Ausgabefach unter 20 liegen, auf „bessere Daten“ (*hope_for_better_data*), ansonsten entleert sie eines der „optimalen“ Subsets die vorhanden sind (*empty_optimal_data*).

Wichtige Teile des Codes

```
def hope_for_better_data
  sto = first_non_empty_storage
  return false if sto == false
  self.used += sto.amount
  sto.to output_storage
  print "FACH(#{sto.id + 1}), "
  sto.amount = data.pop unless data.empty?
  true
end

def empty_optimal_data
  subs = subsets(storages)
  i = 20 - output_storage.amount
  i += 1 while find_fitting(subs, i).empty?
  fitting = find_fitting(subs, i)
  fitting.each do |x|
    print "FACH(#{x.id + 1}), "
    self.used += x.amount
    x.to output_storage
    x.amount = data.pop unless data.empty?
  end
  $stdout.flush
  puts "VERPACKEN() = #{output_storage.amount}"
  self.packages += 1
  output_storage.amount = 0
end

def make_round
  if (sum_storage storages) + output_storage.amount < 20
    hope_for_better_data
  else
    empty_optimal_data
  end
end
```

Beispiele

Beispiel 1

Eingabe: **1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1** (40 Einsen)

Ausgabe:

```
Fach(1), Fach(2), Fach(3), Fach(4), Fach(5), Fach(6), Fach(7), Fach(8), Fach(9), Fach(10),  (Erstes Befüllen fertig)
FACH(1), FACH(1), FACH(1), FACH(1), FACH(1), FACH(1), FACH(1), FACH(1), FACH(1), FACH(2), FACH(3),
FACH(4), FACH(5), FACH(6), FACH(7), FACH(8), FACH(9), FACH(10), VERPACKEN() = 20
FACH(1), FACH(1), FACH(1), FACH(1), FACH(1), FACH(1), FACH(1), FACH(1), FACH(1), FACH(1), FACH(2), FACH(3),
FACH(4), FACH(5), FACH(6), FACH(7), FACH(8), FACH(9), FACH(10), VERPACKEN() = 20

Hergestellte Packungen: 2
Verbrauchte Ballons: 40
Übrig in Fächern: 0
Übrig in Ausgabe: 0

paul@arch ~/doc/dev/BWInf/Juniorаufgabe 1/src                                  ✓ 5530 23:01:39
```

Diesem Screenshot kann man entnehmen, dass 2 Packungen mit jeweils 20 Ballons (und einem Gesamtverbrauch von 40) ohne übriggebliebene Ballons verpackt wurden. Dazu gibt es dann auch noch die Reihenfolge in der die Fächer entleert wurden, wobei die erste Zeile nur für das erste Auffüllen steht.

Beispiel 2

Eingabe: luftballons3.txt (von der BWInf Seite)

Ausgabe:

```
Fach(1), Fach(2), Fach(3), Fach(4), Fach(5), Fach(6), Fach(7), Fach(8), Fach(9), Fach(10),  (Erstes Befüllen fertig)
FACH(1), FACH(3), FACH(7), VERPACKEN() = 20
FACH(3), FACH(4), FACH(6), FACH(9), VERPACKEN() = 20
FACH(1), FACH(3), FACH(4), FACH(6), FACH(9), VERPACKEN() = 20
FACH(1), FACH(3), FACH(6), FACH(9), VERPACKEN() = 20
FACH(1), FACH(3), FACH(4), VERPACKEN() = 20
FACH(3), FACH(7), FACH(8), FACH(9), VERPACKEN() = 20
FACH(1), FACH(3), FACH(5), FACH(7), VERPACKEN() = 20
FACH(1), FACH(3), FACH(4), VERPACKEN() = 20
FACH(1), FACH(4), FACH(5), VERPACKEN() = 20
FACH(3), FACH(5), FACH(7), VERPACKEN() = 20
FACH(3), FACH(4), FACH(7), VERPACKEN() = 20
FACH(1), FACH(3), FACH(5), FACH(6), FACH(7), VERPACKEN() = 20
FACH(1), FACH(3), FACH(5), FACH(6), VERPACKEN() = 20
FACH(3), FACH(6), FACH(7), VERPACKEN() = 20
FACH(3), FACH(6), FACH(7), FACH(8), VERPACKEN() = 20
FACH(1), FACH(1), FACH(1), FACH(3), FACH(7), FACH(10), VERPACKEN() = 20
FACH(1), FACH(10), VERPACKEN() = 20
FACH(1), FACH(3), FACH(5), FACH(7), FACH(10), VERPACKEN() = 21
FACH(1), FACH(1), FACH(1), FACH(10), VERPACKEN() = 21
FACH(1), FACH(1), FACH(5), FACH(8), VERPACKEN() = 21
FACH(1), FACH(1), FACH(5), FACH(8), VERPACKEN() = 20
FACH(1), FACH(5), FACH(8), FACH(10), VERPACKEN() = 20
FACH(1), FACH(1), FACH(5), FACH(8), VERPACKEN() = 21
FACH(1), FACH(1), FACH(5), FACH(8), FACH(10), VERPACKEN() = 20
FACH(1),
Hergestellte Packungen: 24
Verbrauchte Ballons: 491
Übrig in Fächern: 0
Übrig in Ausgabe: 7

paul@arch ~/doc/dev/BWInf/Juniorаufgabe 1/src                                  ✓ 5530 23:04:38
```

Diesem Screenshot ist zu entnehmen, dass 24 Packungen verpackt wurden. Alle abgesehen von 4 enthalten exakt 20, die anderen 21. Es wurden 491 Ballons verarbeitet und 7 Ballons blieben übrig.

Verwendung

Da dieses Programm in Ruby geschrieben ist muss Ruby auch auf dem Host System vorhanden sein, da es eine Skriptsprache ist und somit nicht kompiliert werden kann. Bundler wird nicht benötigt, da alles in der Gemfile nur für Entwicklungszwecke genutzt wurde (z.B. Guard+RuboCop um mich dazu zu zwingen mich an die „best practices“ in Ruby zu halten).

Angenommen man befindet sich im src Ordner, dann würde man es so ausführen:

ruby bin/app.rb <pfad_zur_datei>

Wenn überhaupt kein Argument vorhanden ist, oder die Datei nicht existiert gibt es einen Error.