28. Bundeswettbewerb Informatik

Sebastian Morr

11. April 2010

Inhaltsverzeichnis

Allgemeines

In dieser Arbeit werden die Aufgaben 1 und 2 gelöst.

Zur Umsetzung der Programmieraufgaben benutze ich auch in der zweiten Runde die von mir heißgeliebte Skriptsprache Ruby. Sie ermöglicht es, Programme sehr elegant zu formulieren, und (wichtig!) unterliegt einer freien Lizenz.

Entwickelt und auf Funktionsfähigkeit überprüft wurde mit dem Standardinterpreter[1] in Version 1.9.0, auf einer x86-64-Architektur unter GNU/Linux, Kernel 2.6.31.

Die Programmdokumentation wird teilweise im Quelltext selbst vorgenommen, ein allgemeiner Überblick über die Bestandteile und deren Zusammenspiel ist jeweils vorangestellt.

[1] http://www.ruby-lang.org/de/

Aufgabe 1

Lösungsidee

Definitionen

Ich verwende in der Beschreibung der Lösungsidee folgende Begriffe:

Der **Zustand** (`state`) beschreibt die Schaltposition eines Schalters. Die Schalter, die in der Aufgabe beschrieben sind, haben entweder den Zustand 0 oder 1.

Eine **Stellung** (`combination`/`comb`) beschreibt die Zustände aller n Schalter eines Schlosses. Sie symbolisiert also das Schloss zu einem bestimmten Zeitpunkt und drückt aus, welche der Schalter auf 1 und welche auf 0 stehen. Beispiel: `101001010111110101001011`1.

Von diesen n Schaltern sind nur m aktiv. Die (richtigen) Positionen und Zustände dieser Schalter nenne ich **Schlüssel** (`key`). Kennt man den Schlüssel, kann man das Schloss öffnen, für die restlichen Schalter spielt der Zustand dann keine Rolle mehr. Die inaktiven Schalterpositionen, deren Zustand „egal“ ist, markiere ich im Programm und in der Dokumentation durch Punkte (angeleht an die Syntax regulärer Ausdrücke).

Beispiel: `...0....01.1.....0` (in diesem Schlüssel sind die Schalter 4, 9, 10, 12 und 25 aktiv).

Der Schlüsselbaum

In der Aufgabe wird gesagt, es sei nicht notwendig, alle möglichen Stellungen durchzugehen. Stimmt, es reicht aus, wenn die Stellungen des Öffnungscodes alle möglichen *Schlüssel* enthalten. Mein Lösungsansatz basiert darauf, dem Code wiederholt diejenige Stellung hinzuzufügen, die die meisten neuen Schlüssel abdeckt.

Mein Programm generiert also zunächst alle möglichen Schlüssel. Diese einfach in einer Liste abzulegen würde zu fürchterlichen Laufzeiten führen - da könnte man genausogut doch alle möglichen Stellungen ausprobieren. Stattdessen legt das Programm die Schlüssel in einer Baumstruktur ab. Ein Knoten kann den Wert 0, 1 oder „.“ besitzen, die Ebene steht für die Position im Schlüssel. Durchläuft man den Baum von der Wurzel bis zu einem Blatt, hat man genau einen Schlüssel ausgelesen.

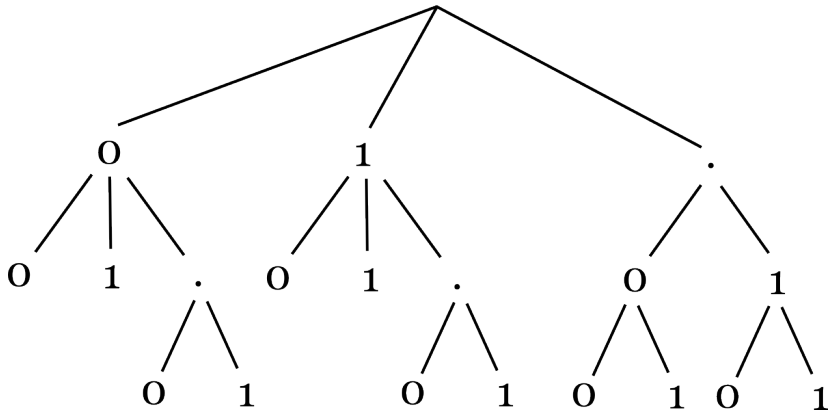

Abbildung 1: Schlüsselbaum für $n = 3$ und $m = 2$

Wenn die Schlüssel auf diese Weise angeordnet sind, kann man mithilfe von rekursiven Algorithmen sehr elegant und zeiteffizient Such- sowie Löschvorgänge durchführen. Ich implementiere insbesondere eine Methode, die zählt, wie viele Schlüssel von einer bestimmten Stellung abgedeckt werden, d.h., wie viele Schlüssel in dieser Stellung enthalten sind.

Erzeugung des Codes

Anfangs deckt jede Stellung gleich viele Schlüssel ab, nämlich $\binom{n}{m}$. Denn der Schlüsselbaum ist absolut symmetrisch aufgebaut, es ist an jeder Stelle egal, ob man mit einer 0 oder einer 1 weitermacht. Man kann also mit einer beliebigen, zufälligen Stellung beginnen. Der Schlüsselbaum überprüft, welche seiner Schlüssel durch diese Stellung abgedeckt wurden und löscht diese.

Es gilt nun, als zweite Stellung diejenige zu finden, die am meisten Schlüssel im Baum abdeckt. Eine Optimierung an dieser Stelle basiert auf der Beobachtung, dass eine Stellung genau so viele Schlüssel abdeckt wie ihre Inversion: Aufgrund der Symmetrie des Baumes erreicht eine Stellung, bei der Nullen und Einsen vertauscht wurden, genau so viele wie das Original. Eine solche wird also als zweites ausgewählt und aus dem Baum gelöscht.

Erst ab der dritten Stellung kommt das Programm ins Grübeln. Für alle möglichen Stellungen zu prüfen, wie viele Schlüssel sie abdecken, kommt nicht in Frage. Stattdessen geht es schrittweise vor: Es überprüft, ob der Code besser mit Null oder Eins beginnen sollte - es wird der Zustand ausgewählt, über den man im Baum mehr Schlüssel erreichen kann.[2] Nun wird für die zweite Position überprüft, welcher Zustand günstiger ist. So wird für alle n Positionen der Stellung verfahren, am Schluss hat man diejenige Stellung, die von allen möglichen am meisten Schlüssel abdeckt. Sie wird genauso wie oben beschrieben dem Code hinzugefügt. Der Vorgang wird wiederholt, bis alle Schlüssel im Code enthalten sind, d.h., der Schlüsselbaum leer ist.

Anmerkung: Es kann mehrere Stellungen geben, die gleich viele Schlüssel abdecken. Welche davon ausgewählt wird, kann sich jedoch im weiteren Verlauf als mehr oder weniger günstig herausstellen. Mir ist keine zeiteffiziente Möglichkeit eingefallen, Vorhersagen darüber zu treffen, daher wähle ich stets die erste der besten Stellungen. Abhängig von der angangs zufällig generierten Stellung *kann* das Programm den optimalen, kürzestmöglichen Code ausgeben, *muss* aber nicht. Es findet aber immerhin *ziemlich* kurze Codes, tut dies in vertretbarer Zeit und ist elegant aufgebaut, und mit dieser Kombination bin ich zufrieden.

Implementation

Erweiterungen

Die Aufgabe fordert einen Öffnungscode für ein Schloss mit 25 Schaltern, von denen 5 aktiv sind. Diese Einschränkung hebe ich auf, n und m sind frei wählbar (natürlich muss n >= m gelten). Das ist sinnvoll, da so beliebig große und komplexe Schlösser dieser Art geknackt werden können, und schwierig, weil die verwendeten Algorithmen so allgemein gehalten werden müssen, dass sie mit jeder n/m-Kombination umgehen können.

Auch die Vorgabe, dass ein Schalter nur den Zustand 0 oder 1 haben kann, erweitere ich: Ein Schloss kann beliebig viele Zustände einnehmen, zum Beispiel auch den Zustand 2. Ein Zustandsbereich von 0 bis 9 würde ein Zahlenschloss nachbilden, ein Bereich von a bis z ein Kryptex, wie es in Dan Browns *The Da Vinci Code* vorkommt. Offensichtlich ist diese Erweiterung deswegen sinnvoll, weil sie den Umgang mit verschiedensten Schloss-Typen ermöglicht. Auch an dieser Stelle habe ich die Algorithmen und die Baum-Datenstruktur nachträglich verallgemeinert, um beliebige Zustände speichern und verarbeiten zu können.

[2] Wie oben schon gesagt: Tatsächlich ist die Null an der ersten Position genauso gut wie die Eins, wegen der Erweiterungen macht es aber Sinn, das hier so allgemein zu halten.

KeyTree

Der oben beschriebene Schlüsselbaum wird in der Klasse `KeyTree` implementiert. Schon das Erzeugen des Baums gestaltete ich rekursiv: Jeder Knoten ist ein Hash (in anderen Programmiersprachen: Dictionary), der unter seinen Schlüsseln für die verschiedenen Zustände (etwa 0 und 1), sowie „.“ weitere KeyTree-Instanzen speichert. Jede Instanz speichert die Anzahl seiner Unterschlüssel zwischen, um spätere Abfragen zu beschleunigen.

Listing 1: keytree.rb

```
# Berechne "n über r"
def Math.nCr(n,m)
    a, b = m, n-m
    a, b = b, a if a < b # a ist der größere Wert
    numer = (a+1..n).inject(1) {|t,v| t*v } # n!/m!
    denom = (2..b).inject(1) {|t,v| t*v } # (n-m)!
    numer/denom
end

# Eine Baumstruktur, die Informationen über verbleibende Schlüssel
# enthält.
class KeyTree < Hash

    # 'count' gibt darüber Auskunft, wieviele Schlüssel der Baum
    # noch enthält.
    attr_reader :count

    # Erzeuge einen Baum, der die Schlüssel für n Schalter enthält,
    # von denen m aktiv sind. Der Baum wird (n über m)*(2^m)
    # Schlüssel enthalten. Wenn m = 0, folgen keine aktiven
    # Schalter mehr, dieser Zweig ist fertig. Ansonsten erzeuge für
    # jeden möglichen Zustand einen untergeordneten Baum, der um
    # einen Schalter kürzer ist und einen aktiven Schalter weniger
    # hat. Wenn der Schlüssel außerdem noch beliebige Schalter
    # enthalten kann, füge einen Baum dafür hinzu
    def initialize n, m, states
        @count = Math::nCr(n,m)*(states.size**m)

        return if m == 0

        states.each do |state|
            self[state] = KeyTree.new(n-1, m-1, states)
        end

        if n > m
            self["."] = KeyTree.new(n-1, m, states)
        end
    end

    # Berechne, wie viele Schlüssel von der Stellung 'comb' abgedeckt
    # würden.
    # Ist ein Blatt des Baumes erreicht, zähle einen neuen Schlüssel.
    # Wenn 'comb' nur noch beliebige Schalter enthält, passt sie auf
    # alle Unterschlüssel, gib den zuvor gespeicherten Wert 'count'
    # zurück. Ansonsten summiere die covered_by-Werte der Unterbäume.
    def covered_by comb
        return 1 if empty?
```

```
        return @count if comb =~ /^\.+$/

        sum = 0
        each_reached_by(comb) do |state, tree|
            sum += tree.covered_by(comb[1..-1])
        end
        sum
    end

    # Führe einen Codeblock für jeden Unterbaum aus, der von der
    # Stellung 'comb' erreicht wird. Das ist dann der Fall, wenn der
    # Baum über jeden Zustand erreicht wird (".") oder wenn
    # der Zustand zum ersten Wert aus 'comb' passt.
    def each_reached_by comb
        each do |state, tree|
            if state == comb[0] or state == "."
                yield state, tree
            end
        end
    end

    # Lösche alle Schlüssel, die von 'comb' abgedeckt werden.
    # Durchlaufe hierzu alle passenden Unterbäume und rufe diese
    # Methode für die vorne um ein Element verkürzte Stellung auf.
    # Wenn ein Unterbaum dadurch geleert wird, lösche ihn ganz.
    # Verringere 'count' entsprechend.
    def done comb
        each_reached_by(comb) do |state, tree|
            @count -= tree.count
            tree.done comb[1..-1]
            if tree.empty?
                delete(state)
            else
                @count += tree.count
            end
        end
    end
end
```

CodeGenerator

Die eigentliche Berechnung des Öffnungscodes findet in der Klasse `CodeGenerator` statt. Sie erzeugt eine passende `KeyTree`-Instanz und führt auf dieser den oben beschriebenen Algorithmus aus. Die Methode `generate` gibt ein Array zurück, das die Stellungen des Öffnungscodes als Strings enthält.

Listing 2: codegen.rb

```
require "keytree"

# Diese Klasse implementiert die eigentliche Lösungsidee.
class CodeGenerator

    # Speichere n, m und die verfügbaren Schalterstellungen und
    # erzeuge den KeyTree.
    def initialize n, m, states
```

```
        @n = n
        @m = m
        @states = states
        @tree = KeyTree.new(n, m, states)
        @code = []
    end

    # Nimm die beste nächste Kombination und seine Inversion in den
    # Öffnungscode auf. Wiederhole das so lange, bis alle Schlüssel
    # abgedeckt sind.
    def generate
        loop do
            break if @tree.empty?

            comb = next_comb
            take_comb comb

            if @states.size == 2
                take_comb comb.tr(@states.join, @states.reverse.join)
            end
        end
        return @code
    end

    # Lösche alle Schlüssel aus dem Baum, die die Stellung 'comb'
    # abdeckt, und füge diese dem Öffnungscode hinzu.
    def take_comb comb
        @tree.done(comb)
        @code << comb
    end

    # Berechne, welche Stellung am meisten neue Schlüssel abdeckt.
    # Beginne mit einer zufälligen Stellung (jede deckt gleich viele
    # Schlüssel ab).
    # Danach gehe schrittweise vor: Prüfe, wie viele Schlüssel
    # abgedeckt würden, wenn die Stellung mit "0" bzw. "1" beginnen
    # würde und füge die bessere Variante der Stellung hinzu.
    # Wiederhole das so lange, bis die Stellung die Länge n hat.
    def next_comb
        if @code.empty?
            return Array.new(@n){@states[rand(@states.size)]}.join(""
               )
        end

        comb = ""
        while comb.size < @n
            best_state = @states[0]
            best_count = 0
            fill = "."*(@n-comb.size-1)
            @states.each do |state|
                count = @tree.covered_by(comb+state+fill)
                if count > best_count
                    best_state = state
                    best_count = count
                end
            end
```

```
            comb << best_state
        end
        return comb
    end
end
```

Beispielläufe

Die Datei *crack.rb* ermöglicht es, den CodeGenerator komfortabel von der Kommandozeile aus zu steuern. Sie nimmt drei Parameter entgegen, die Größe und Eigenschaften des zu knackenden Schlosses festlegen:

Listing 3: crack.rb

```
require "codegen"

n = ARGV[0].to_i
m = ARGV[1].to_i
states = (ARGV[2] || "01").split("")

puts code = CodeGenerator.new(n, m, states).generate
puts "Length: #{code.size}."
```

Standardmäßig werden die Zustände 0 und 1 verwendet. Ich setze in den Beispielen außerdem einen `time`-Aufruf vor, um die Laufzeit zu dokumentieren. Rechenknecht war ein Core 2 Duo P9600 mit 2.53GHz. Der 25/5-er Code liegt nach etwa 5 Minuten vor, dieser Wert ließe sich durch den Einsatz einer statisch typisierten, kompilierbaren Programmiersprache weiter optimieren. Dadurch, dass der gesamte Schlüsselbaum im Speicher behalten werden muss und die Algorithmen große Stacks aufbauen müssen, frisst das Programm viel Arbeitsspeicher, der Computer, auf dem es ausgeführt wird, muss deshalb über ausreichende Auslagerungspartitionen oder -dateien verfügen.

Alle Beispiele befinden sich auch auf der CD im Ordner *code/examples/*.

Beispiellauf für 9/3:

```
$ time ruby crack.rb 9 3
110011100
001100011
010101000
101010111
000000100
111111011
011001110
100110001
010010010
101101101
000111110
111000001
010001111
101110000
001011001
110100110
Length: 16.

real    0m0.053s
user    0m0.040s
sys     0m0.000s
```

Ein Code für 19/4. Deutlich unter 100 Stellungen, jupp:

```
$ time ruby crack.rb 19 4
1001001101101101000
0110110010010010111
0000010100000100010
1111101011111011101
0011000000110001001
1100111111001110110
0101011001010111011
1010100110101000100
0100000011101001010
1011111100010110101
0001100111011100001
1110011000100011110
0010001111010011100
1101110000101100011
0111010111100110000
1000101000011001111
0001111010100011000
1110000101011100111
0100101100110101100
1011010011001010011
0010111001101100101
1101000110010011010
0111101100001010010
1000010011110101101
0001001010001110101
1111011010111001010
0100011110111000001
1011100001000111110
0010101011110110010
1101010100001001101
0001110101111010110
1110001010000101001
0111011010011101110
1000100101100010001
0100100001000000100
1011011110111111011
0010010100001111000
```

```
110110101111000011 1
001100111110000011 1
110011000001111100 0
011010011010111111 1
100101100101000000 0
000000000011111011 0
```

```
111111111100000100 1
000011110100101101 1
111100001011010010 0
000110011010000101 1
111001100101111010 0
001000001000100000 0
110111101110111111 1
```

```
000000010000001010 1
111111101111110101 0
Length: 52.

real    0m4.354s
user    0m4.320s
sys     0m0.030s
```

Der in der Aufgabe geforderte Code für 25/5:

```
$ time ruby crack.rb 25 5
1111011110011110101000111
0000100001100001010111000
0101000100000100000001010
1010111011111011111110101
0011001000110010001100000
1100110111001101110011111
0110010001011000011001100
1001101110100111100110011
0001010011000010101011001
1110101100111101010100110
0100001010010101111111100
1011110101101010000000011
0111100011110111000010101
1000011100001000111101010
0010000110101110110011100
1101111001010001001100011
0000111110010010010000101
1111000001101101101111010
0101111101101110111101101
1010000001001000100001001 0
0011111010001101001001000
1100000101110010110110111
0110110110100000101100110
1001001001011111010011001
0011011101010101110110000
1100100010101010001001111
0110011000100111010000011
1001100111011000101111100
0000010100111101001110101
1111101011000010110001010
0101010010111001110000000
1010101101000110001111111
0001100000011110111010010
1110011111100001000101101
0111101100001001100010101
1000010011110110011101010
0010100111011111100101011
1101011000100000011010100
0100101011111100000011001 0
1011010100000111111001101
0001001111101001011000111
1110110000010110100111000
0111000111001011011110000
1000111000110100100001111
0100001101010011101001000
1011110010101100010110111
0010001001001100100100101
1101110110110011011011010
0111111101110100011011011
1000000010001011100100100
0010110001000011111010110
1101001110111100000101001
0100111011001110000110110
1011000100110001111001001
0001110100011011000101110
1110001011100100111010001
0100100100000101011100001
```

```
1011011011111010100011110
0010011110111011001011011
1101100001000100110100100
0111010010010000110111111
1000101101101111001000000
0001011001100111100101110
1110100110011000011010001
0100010101111110100010000
1011101010000001011101111
0010101000111010110101001
1101010111000101001010110
0001101111110100111000110
1110010000001011000111001
0101111010101011111111011
1010000101010100000000100
0011110111100100000101000
1100001000011011111010111
0110101110000110010111000
1001010001110011010000111
0111000000101110001111111
1000111111010001110000000
0000001010000100001011111
1111110101110111101000000
0100110011110101101001001
1011001100001010010110110
0001000111010010010100011
1110111000101101101011100
0010111101011000001110011
1101000010100111110001100
0111001111111111011101110
1000110000000000100010001
0010000011101001110001011
1101111100010110001110100
0100011110001100110000010
1011100001110011001111101
0001100000111101010001110
1110011111000010101110001
0101001001001000000011101
1010110110110111111100010
0000010100100011010111110
1111101011011100101000001
0111100100100000101011010
1000011011011111010100101
0011011010010101100101111
1100100101101010011001000
0110101001010011100001110
1001010110101100011110001
0101110010011110011100101
1010001101100001100011010
0000101110110001101111101
1111010001001110010000010
0011111101101000110011100
1100000001001011100110011
0101011100110011100100011
1010100011001100011011100
0010010101111100111100111
1101101010000011000011000
0001000100000110101000101
1110111011111001010111010
```

```
0110000111110010000000100
1001111000001101111111011
0111100000100001010100101
1000011111011110101011010
0010101001100110111100011
1101010110011001000011100
0100010001101101001110010
1011101110010010110001101
0000101011100010010010100
1111010100011101101101011
0111011010101010101100000
1000100101010101010011111
0001110011001111110110001
1110001100110000001001110
0101100111000001101101011
1010011000111110010010100
0000111001011000011001001
1111000110100111100110110
0100110101101011000000101
1011001010010100111111010
0010000110001111001100100
1101111001110000110011011
0111011001010111011001101
1000100110101000100110010
0100101100011001110111111
1011010011100110001000000
0101100101110111101010000
1010011010001000010101111
0011111111001101000010011
1100000000110010111101100
0111101000011001101100010
1000010111100110010011101
0100001011010000000101100
1011110100101111111010011
0111001101001011110011011
1000110010110100001100100
0010010100110000000011001
1101101011001111111100110
0011100110110010011010001
1100011001001101100101110
0000000101011000010111000
1111110101001111010000111
0100010010110010000110110
1011101101001011111001001
0111000000100000100100011
1000111111011111011011100
0110001000100101100110000
1001110111010100110011111
0100000101001000000000011
1011111010110111111111100
0010101101000101101001111
1101010010111010010110000
0000101000100011000000010
1111010111011100111111101
0000001000100001001111000
1111110111011110110000111
0100101010010111110000010
1011010101101000001111101
0101000001100010000110100
```

```
101011111001110111100101101000000001000000000000100010111111111011111111110111
```

```
Length: 178.
```

```
real    5m12.837s
user    5m12.000s
sys     0m1.080s
```

Ein Zahlenschloss mit 3 Zylindern, von denen 2 relevant sind:

```
$ time ruby crack.rb 3 2
    0123456789
902
010
120
230
340
450
560
670
780
890
001
111
221
331
441
551
661
771
881
991
022
103
212
352
432
542
682
762
872
913
093
134
204
323
463
```

```
573
643
753
805
984
035
192
245
314
424
585
654
794
864
975
044
155
266
306
415
526
636
707
816
927
076
186
297
365
477
537
695
746
857
938
087
147
258
```

```
378
408
518
628
719
829
969
059
168
283
399
489
509
600
733
848
940
617
725
067
179
274
387
496
594
833
956
038
239
649
788
098
Length: 105.

real    0m0.113s
user    0m0.100s
sys     0m0.010s
```

Ein „Vokalschloss“ der Länge 5 mit 2 aktiven Rädern:

```
$ time ruby crack.rb 5 2
    aeiou
uaiea
aeaaa
eieia
ioooa
ouuua
aaeoe
eeiue
iiaee
oooae
uuuie
aoiii
```

```
euaoi
iauai
oeeei
uioui
auoeo
eoueu
ieeio
oaaiu
uiiao
ueeou
aiuoo
eaouo
iuiau
ooauo
```

```
oeoiu
oieuu
aeeau
oeioa
uoaaa
aeuua
eoeaa
iueua
Length: 33.

real    0m0.052s
user    0m0.040s
sys     0m0.000s
```

Aufgabe 2

Lösungsidee

Wie lange sich eine Gruppe im Restaurant aufhält, kann nicht vorhergesagt werden: Manche kommen nur kurz für einen Drink, andere feiern den ganzen Tag über ihren Geburtstag. Die Verweildauer ist genauso unbestimmt wie unwesentlich: Für die Algorithmen sind die Momente interessant, in denen etwas passiert, sprich, in denen eine neue Gruppe das Restaurant betritt oder sich eine solche auflöst.

Teilaufgabe 1

Es ist klar, dass der Ober eine neu eintreffende Gruppe direkt neben eine schon vorhandene setzen sollte - anderenfalls entstehen Lücken, die er hätte vermeiden können.

Mein Algorithmus stellt also eine Liste der momentan vorhandenen zusammenhängenden freien Sitzplätze zusammen - eine Liste von „Lücken“. Alle Lücken, die zu klein sind, um die neue Gruppe aufzunehmen, scheiden aus.

Es gibt nun mehrere Möglichkeiten, wie mit eintreffenden Gruppen verfahren werden soll.

- Man könnte sie in die erste Lücke setzen, in die sie hineinpasst (*FirstFit*).
- Man könnte sie in die kleinste Lücke setzen, in die sie noch passt - im Optimalfall also in eine Lücke, die genauso groß ist, wie die Gruppe selbst. Bei diesem Vorgehen bleibt immer eine möglichst große Lücke frei, denn die neue Gruppe wird niemals „zufälligerweise“ in die größte verfügbare Lücke gesetzt werden (*BestFit*).
- Man könnte sie auch in die größte Vorhandene Lücke setzen (*WorstFit*).

Die verschiedenen Methoden haben Vor- und Nachteile. Welche Strategie sollte der Ober wählen? Das können wir durch empirische Untersuchung herausfinden. In der folgenden Tabelle vergleiche ich die verschiedenen Strategien. Eintreffen und Verlassen der Gruppen wurde komplett zufällig gesteuert (`RandomEnvironment`), was einen durchschnittlichen Abend simulieren soll. Maßeinheit ist „Wutanfälle pro Ereignis“, kurz „WpE“, wobei ein Ereignis das Erscheinen oder Verlassen einer Gruppe ist. Es wurden jeweils 1.000.000 Ereignisse simuliert.

Strategie	WpE
BestFit	0.0547
FirstFit	0.0552
WorstFit	0.0570

Die Strategie, Gruppen in möglichst kleine Lücken zu setzen, stellt sich also am effektivsten heraus - das ist praktischerweise auch das, was man intuitiv erwarten würde.

Übrigens weichen die Strategien überraschend wenig voneinander ab. Der Ober ärgert sich bei etwa jeder 20. eintreffenden Gruppe. Hoffentlich hat er ein starkes Herz.

Ich stelle es im Programm dem Ober frei, welche Strategie er auswählt. Wie meine Untersuchungen gezeigt haben, wird er mit der „BestFit“-Strategie am glücklichsten.

Teilaufgabe 2

Mir ist wichtig, dass die Strategie der Schüler unabhängig von der Strategie des Obers und vor allem auch ohne deren Kenntnis funktioniert. Natürlich können die Töchter ihren Vater beobachten, bis sie seine Strategie herausgefunden haben - aber was hindert den Ober daran,

während des Streichs seine Strategie zu ändern? Also noch einmal: Die Strategie der Töchter muss gegen jede Oberstrategie erfolgreich sein.

Ich gehe davon aus, dass das Restaurant während des Streichs nicht von weiteren Gästen besucht wird. Wenn die Schüler auf sich allein gestellt sind, haben sie es am schwersten. Jede zusätzliche Gästegruppe würde ihre Reihen nur verstärken, sprich, die Anzahl der verfügbaren Gäste erhöhen. Ich beginne mit einem leeren Restaurant und lasse bis auf die Schüler keine äußeren Einflüsse zu.

Eine Strategie, die immer funktioniert, benötigt $\lceil \frac{n}{2} \rceil + 2$ Schüler.

1. $\lceil \frac{n}{2} \rceil + 1$ Schüler betreten nacheinander und einzeln das Restaurant.

2. Egal, wie gut die Strategie des Obers sein mag - mit Sicherheit werden sich nun zwei Schüler genau gegenübersitzen. Das liegt daran, dass es nur $\lceil \frac{n}{2} \rceil$ Paare von gegenüberliegenden Stühlen gibt; wenn es mehr Gäste gibt als diese Zahl, *muss* mindestens ein Paar komplett besetzt sein.[3] Wenn alle Schüler bis auf diese beiden das Restaurant nun wieder verlassen, erzeugen die beiden zwei Lücken von höchstens $\lceil \frac{n}{2} \rceil - 1$ Plätzen.

3. Es sind aber noch genau $\lceil \frac{n}{2} \rceil$ Schüler verfügbar. Diese bilden nun eine große Gruppe und betreten das Restaurant - der Ober wird sich fürchterlich ärgern! Die Auslastung beträgt $\frac{2}{n}$-tel, was der Anforderung der Aufgabe, möglichst geringe Auslastungen zu erzeugen, wohl Genüge tut.

In Simulationen, bei denen die Gruppen komplett zufällig auftauchen und wieder verschwinden, kann sich der Ober übrigens auch schon bei $\lceil \frac{n}{2} \rceil + 1$ teilnehmenden Schülern ärgern - ich kann jedoch nicht ausschließen, dass der Ober hierzu eine Gegenstrategie finden könnte. Die oben vorgestellte Methode hingegen funktioniert bewiesenermaßen immer.

Teilaufgabe 3

Wie erwähnt ärgert sich der Ober auf jeden Fall, wenn $\lceil \frac{n}{2} \rceil + 2$ Schüler mitmachen. Nehmen weniger teil, werden die Lücken nicht groß genug; Nehmen mehr teil, stehen sich einige Löcher in den Bauch - der Streich funktioniert aber trotzdem.

Beispiellauf meiner Programme für ein Restaurant mit 28 Plätzen:

```
Unser Turmrestaurant mit 28 Sitzplätzen öffnet.
----------------------------
Eine neue Gruppe von 1 Person betritt das Restaurant!
Sie erhält die ID 'A' und wird auf den Platz 0 gesetzt.
A---------------------------
Eine neue Gruppe von 1 Person betritt das Restaurant!
Sie erhält die ID 'B' und wird auf den Platz 1 gesetzt.
AB--------------------------
Eine neue Gruppe von 1 Person betritt das Restaurant!
Sie erhält die ID 'C' und wird auf den Platz 2 gesetzt.
ABC-------------------------
Eine neue Gruppe von 1 Person betritt das Restaurant!
Sie erhält die ID 'D' und wird auf den Platz 3 gesetzt.
ABCD------------------------
Eine neue Gruppe von 1 Person betritt das Restaurant!
Sie erhält die ID 'E' und wird auf den Platz 4 gesetzt.
ABCDE-----------------------
Eine neue Gruppe von 1 Person betritt das Restaurant!
Sie erhält die ID 'F' und wird auf den Platz 5 gesetzt.
ABCDEF----------------------
Eine neue Gruppe von 1 Person betritt das Restaurant!
Sie erhält die ID 'G' und wird auf den Platz 6 gesetzt.
ABCDEFG---------------------
```

[3] Bei ungradem n denkt man sich irgendwo noch einen leeren Stuhl dazu.

```
Eine neue Gruppe von 1 Person betritt das Restaurant!
Sie erhält die ID 'H' und wird auf den Platz 7 gesetzt.
ABCDEFGH____________________
Eine neue Gruppe von 1 Person betritt das Restaurant!
Sie erhält die ID 'I' und wird auf den Platz 8 gesetzt.
ABCDEFGHI___________________
Eine neue Gruppe von 1 Person betritt das Restaurant!
Sie erhält die ID 'J' und wird auf den Platz 9 gesetzt.
ABCDEFGHIJ__________________
Eine neue Gruppe von 1 Person betritt das Restaurant!
Sie erhält die ID 'K' und wird auf den Platz 10 gesetzt.
ABCDEFGHIJK_________________
Eine neue Gruppe von 1 Person betritt das Restaurant!
Sie erhält die ID 'L' und wird auf den Platz 11 gesetzt.
ABCDEFGHIJKL________________
Eine neue Gruppe von 1 Person betritt das Restaurant!
Sie erhält die ID 'M' und wird auf den Platz 12 gesetzt.
ABCDEFGHIJKLM_______________
Eine neue Gruppe von 1 Person betritt das Restaurant!
Sie erhält die ID 'N' und wird auf den Platz 13 gesetzt.
ABCDEFGHIJKLMN______________
Eine neue Gruppe von 1 Person betritt das Restaurant!
Sie erhält die ID 'O' und wird auf den Platz 14 gesetzt.
ABCDEFGHIJKLMNO_____________
Die Gruppe 'B' verlässt das Restaurant.
A_CDEFGHIJKLMNO_____________
Die Gruppe 'C' verlässt das Restaurant.
A__DEFGHIJKLMNO_____________
Die Gruppe 'D' verlässt das Restaurant.
A___EFGHIJKLMNO_____________
Die Gruppe 'E' verlässt das Restaurant.
A____FGHIJKLMNO_____________
Die Gruppe 'F' verlässt das Restaurant.
A_____GHIJKLMNO_____________
Die Gruppe 'G' verlässt das Restaurant.
A______HIJKLMNO_____________
Die Gruppe 'H' verlässt das Restaurant.
A_______IJKLMNO_____________
Die Gruppe 'I' verlässt das Restaurant.
A________JKLMNO_____________
Die Gruppe 'J' verlässt das Restaurant.
A_________KLMNO_____________
Die Gruppe 'K' verlässt das Restaurant.
A__________LMNO_____________
Die Gruppe 'L' verlässt das Restaurant.
A___________MNO_____________
Die Gruppe 'M' verlässt das Restaurant.
A____________NO_____________
Die Gruppe 'N' verlässt das Restaurant.
A_____________O_____________
Eine neue Gruppe von 14 Personen betritt das Restaurant!
Sie findet leider keinen Platz mehr. Der Ober ÄRGERT sich!
```

Implementation

Objekte der Klasse `Area` werden im gesamten Programm verwendet, um die Position und Größe von Gruppen, aber auch von Lücken zu speichern.

Listing 4: area.rb

```
class Area

    # Der Bereich fängt bei Platz 'first' an und erstreckt sich über
        'size' Plätze.
    attr_reader :first, :size

    def initialize first, size
```

```
        raise "An empty group is not possible." if size <= 0

        @first = first
        @size = size
    end

    # Der letzte Platz des Bereichs.
    def last
        @first+@size-1
    end

    # Überscheidet sich der Bereich mit einem anderen?
    def overlap? other
        (other.first >= first and other.first <= last) or (other.last
            >= first and other.last <= last)
    end

    # Verschiebe den Bereich um 'amount' nach hinten.
    def shift amount
        Area.new(@first+amount, @size)
    end
end
```

Das Programm kann im Wesentlichen durch das Zusammenspiel dreier weiterer Klassen beschrieben werden:

Restaurant

Diese Klasse speichert die im Restaurant anwesenden Gruppen, nimmt neue Gruppen entgegen und entlässt andere wieder. Das Restaurant stellt nach außen ein Interface zur Verfügung, das einerseits Informationen über die momentane Besetzung liefert und andererseits das Eintreffen und Verlassen von Gruppen verwaltet. Zur Idenfikation besitzt jede Gruppe eine eindeutige ID - im echten Betrieb könnte das etwa „Familie Meier“ oder „Otto Hempel“ sein, zur Bearbeitung dieser Aufgabe beschränke ich mich auf einzelne alphanumerische Zeichen, denn diese lassen später eine übersichtliche grafische Darstellung zu.

Die Darstellung des Restaurants erfolgt auf die denkbar einfachste Weise: Durch einen String, in dem leere Stühle durch Unterstriche, Gruppen durch Buchstaben dargestellt werden. Beispiel: Das Restaurant aus der Aufgabe würde so dargestellt:

```
AAAA_BB_CCCC___DD____EE___AA
```

Man sieht, wie die Gruppe A über den rechten Rand hinausgeht und vorne fortgesetzt wird. Diese Form der Darstellung wird keinen Preis gewinnen[4], ist aber wegen ihrer Übersichtlichkeit und Kompaktheit zur Dokumentation von Abläufen umso zweckmäßiger.

Listing 5: restaurant.rb

```
require "area"

class Restaurant
```

[4]Ich bewerbe mich hiermit ausdrücklich *nicht* um den MCI-Sonderpreis.

```
    # Gib Zugriff auf Anzahl der Sitzplätze und anwesende Gruppen.
    attr_reader :seats, :groups

    # Initialisiere das Restaurant, seine Gruppen und die verfügbaren
        Gruppen-IDs.
    def initialize seats
        @seats = seats
        @groups = {}
        @group_ids = ("A".."Z").to_a.concat(("a".."z").to_a)
    end

    # Berechne die Anzahl der belegten Plätze aus den Gruppengrößen.
    def taken_seats
        @groups.inject(0) {|sum, (id, area)| sum + area.size}
    end

    # Gib Anzahl der freien Plätze zurück.
    def free_seats
        @seats-taken_seats
    end

    # Gib eine noch freie ID zurück.
    def free_id
        raise "No more IDs available." if @group_ids.empty?

        @group_ids.shift
    end

    # Weise einer Gruppe eine bestimmte Stelle zu. Verschiedene
       Fehlerabfragen.
    def assign area
        raise "The restaurant can't take this group anymore." if area
            .size > free_seats
        raise "First seat is bigger than actual capacity." if area.
            first > @seats
        raise "Areas overlap!" unless gap? area

        id = free_id
        @groups[id] = area
        return id
    end

    # Überprüfe, ob ein bestimmter Bereich frei ist. Verschiebe die
       Gruppen auch um eine Kreisdrehung, um das zu überprüfen.
    def gap? other_area
        @groups.each do |id, area|
            if area.overlap?(other_area) or area.shift(-@seats).
                overlap?(other_area)
                 return false
            end
        end
        true
    end

    # Entferne eine Gruppe aus dem Restaurant und gib ihre ID frei.
    def leave id
```

```
        raise "This group does not exist." unless @groups.has_key? id
        @groups.delete(id)
        @group_ids << id
    end

    # Gib einen String der momentanen Besetzung zurück.
    def seat_list
        seats = Array.new(@seats){"_"}
        @groups.each do |id, area|
            area.first.upto(area.last) do |seat|
                seat = seat % @seats
                seats[seat] = id
            end
        end
        seats.join("")
    end

    # Gib ein Array der Lücken zurück. Füge dabei vordere und hintere
        Lücken zusammen.
    def gaps
        return [Area.new(0, @seats)] if @groups.empty?
        return [] if free_seats == 0

        seats = seat_list
        max = seats.size
        gaps = []
        in_gap = false
        size = 0
        first = nil
        seats.split("").each_with_index do |char, i|
            if char == "_"
                size += 1
                if in_gap
                else
                    first = i
                    in_gap = true
                end
            end
            if char != "_" or i == max-1
                if in_gap
                    area = Area.new(first, size)
                    gaps << area
                    size = 0
                    in_gap = false
                end
            end
            seats = seats[1..-1]
        end

        if gaps.first.first == 0 and gaps.last.last == max-1
            first = gaps.delete_at(0)
            last = gaps.delete_at(-1)
            area = Area.new(last.first, first.size+last.size)
            gaps = gaps + [area]
        end
```

```
        gaps
    end
end
```

Waiter

Eine Klasse, die von `Waiter` erbt, kann gefragt werden, wo eine neu eintreffende Gruppe platziert werden soll. Sie antwortet mit der Nummer des ersten zu besetzenden Platzes. Die Simulation nimmt diese Entscheidung entgegen und belegt die Plätze. Geht eine Gruppe, wird dies ohne weitere Fragen durchgeführt.

Listing 6: waiter.rb

```
require "restaurant"
require "area"

# Abstrakte Waiter-Klasse, die für verschiedene Strategien abgeleitet
    wird.
class Waiter
    def initialize restaurant
        @r = restaurant
    end

    # Rein: Größe der Gruppe, die platziert werden soll.
    # Raus: Erster Sitzplatz, ab dem die Gruppe sitzen soll.
    # Die select_gap-Funktion kann in den Unterklassen implementiert
       werden.
    def place size
        return false if size > @r.free_seats

        possible_gaps = @r.gaps.select{|gap| gap.size >= size}
        return false if possible_gaps.empty?

        select_gap(possible_gaps).first
    end
end

class FirstFitWaiter < Waiter
    def select_gap gaps
        gaps.first
    end
end

class WorstFitWaiter < Waiter
    def select_gap gaps
        gaps.sort_by{|gap| gap.size}.last
    end
end

class BestFitWaiter < Waiter
    def select_gap gaps
        gaps.sort_by{|gap| gap.size}.first
    end
end
```

Environment

Das `Environment` ist die „Gastquelle“, die darüber bestimmt, welches Ereignis als nächstes passiert. Auch hier gibt es verschiedene Typen. Ich habe ein zufälliges Environment implementiert, das zum Testen der Oberstrategie verwendet wurde, eine interaktive Simulation, sowie die Tochter-Strategie.

Listing 7: environment.rb

```
# coding: utf-8

class Hash
    # Wähle einen zufälligen Wert aus dem Hash.
    def choose_one_key
        my_keys = keys
        my_keys[rand(my_keys.size)]
    end
end

class Environment

    # Gib Zugriff auf die noch verfügbare Menge an Gästen.
    attr_accessor :available

    def initialize restaurant, simulation, available
        @restaurant = restaurant
        @simulation = simulation
        @available = available
        setup if respond_to? :setup
    end

    def enter size
        @simulation.enter size
    end

    def leave id
        @simulation.leave id
    end
end

# Mit diesem Environment kann der Benutzer beliebige Gruppen
# ins Restaurant schicken und diese wieder auflösen. Sehr
# praktisch zum Rumspielen.
class InteractiveEnvironment < Environment
    def next_action
        print "? "
        cmd, arg = gets.chomp.split("", 2)

        case cmd
        when "+"
            enter(arg.to_i)
        when "-"
            leave(arg)
        else
            puts "Unbekannter Befehl."
        end
    end
```

```
end

# Ein zufälliges Environment, das entweder eine neue Gruppe
# bildet oder eine auflöst. Es respektiert die verfügbare
# Gäste-Anzahl.
class RandomEnvironment < Environment
    def setup
        @detail = false
    end
    def next_action
        if (rand(2) < 1 or @restaurant.groups.empty?) and @available
           > 0
            size = [rand(5)+1, @available].min
            success = enter(size)
        else
            key = @restaurant.groups.choose_one_key
            leave(key)
        end
    end
end

# Dieses Environment implementiert die Strategie der Töchter.
class EvilEnvironment < Environment

    # Prüfe, ob der Streich durchgeführt werden kann und
       initialisiere Werte.
    def setup
        min = (@restaurant.seats/2.0).ceil+2
        @diff = @available - min
        raise "Diese Anzahl Schüler reicht nicht aus, um den Streich
           durchzuführen!" if @diff < 0
        @ids = []
        @step = 0
    end

    # Führe die drei Schritte des Streichs aus, gesteuert von
    # '@step'.
    def next_action
        case @step
        when 0
            if @available > @diff + 1
                # Fülle das Restaurant zur Hälfte mit
                # Einzelgruppen. Speichere deren IDs.
                @ids << enter(1)
            else
                # Vergiss erste und letzte Gruppe, leite
                # nächsten Schritt ein.
                @ids.delete_at(0)
                @ids.delete_at(-1)
                @step = 1
                next_action
            end
        when 1
            if @ids.size > 0
                # Entferne alle bis auf zwei Schüler, die
                # sich gegenübersitzen.
```

```
                leave(@ids.shift)
            else
                @step = 2
                next_action
            end
        when 2
            # Schick nun alle verfügbaren Schüler als eine
            # Gruppe rein. ÄRGER!
            enter(@available)
        end
    end
end
```

Simulation

Die Klasse `Simulation` schließlich fasst die drei Klassen zusammen und schreibt auf den Bildschirm, was die einzelnen Komponenten so machen. Am Ende der Datei wird eine beispielhafe Simulation erzeugt und gestartet.

Listing 8: simulation.rb

```
# coding: utf-8
require "restaurant"
require "waiter"
require "environment"

class Simulation

    # Erzeuge Restaurant, Ober und Gastquelle.
    def initialize seats, waiter, environment, available
        @restaurant = Restaurant.new(seats)
        @waiter = waiter.new(@restaurant)
        @environment = environment.new(@restaurant, self, available)
    end

    # Starte die Simulation. Zeige die aktuelle Besetzung
    # vor jedem Schritt an.
    def run
        puts "Unser Turmrestaurant mit #{@restaurant.free_seats}
           Sitzplätzen öffnet."
        loop do
            puts @restaurant.seat_list
            @environment.next_action
        end
    end

    # Eine Gruppe möchte im Restaurant essen! Frag den Ober
    # nach einem freien Platz!
    def enter size
        puts "Eine neue Gruppe der Größe #{size} betritt das
           Restaurant!"

        unless size.to_i > 0
            puts "Es gibt keine leeren Gruppen."
            return false
        end
```

```
        if size > @environment.available
            puts "So viele Gäste stehen nicht mehr zur Verfügung."
            return false
        end

        seat = @waiter.place(size)

        if seat
            id = @restaurant.assign Area.new(seat, size)
            @environment.available -= size
            puts "Sie erhält die ID '#{id}' und wird den Plätzen #{
                seat} bis #{seat+size-1} zugewiesen."
            return id
        else
            puts "Sie findet leider keinen Platz mehr. Der Ober Ä
                RGERT sich!"
            return false
        end
    end

    # Eine Gruppe möchte das Restaurant verlassen! Erhöhe
    # die verfügbare Gästeanzahl!
    def leave id
        unless @restaurant.groups.has_key? id
            puts "Es gibt keine Gruppe mit der ID '#{id}'."
            return
        end
        puts "Die Gruppe '#{id}' verlässt das Restaurant."
        @environment.available += @restaurant.groups[id].size
        @restaurant.leave(id)
    end
end

# Setze Anzahl der Stühle, verfügbare Gäste-Anzahl,
# Ober-Strategie und Environment-Typ. Starte dann die
# Simulation.

seats = 32
available_guests = 99

waiter = BestFitWaiter
#waiter = WorstFitWaiter
#waiter = FirstFitWaiter

environment = RandomEnvironment
#environment = InteractiveEnvironment
#environment = EvilEnvironment

Simulation.new(seats, waiter, environment, available_guests).run
```

Der folgende Testlauf zeigt eine interaktive Sitzung mit einem BestFit-Ober, in der einige Funktionen vorgeführt werden (Zuweisung einer Gruppe „um die Kante herum“, Füllen der kleinsten Lücken und zum Schluss ein schöner Wutanfall).

```
$ ruby simulation.rb
Unser Turmrestaurant mit 32 Sitzplätzen öffnet.
--------------------------------
```

```
? +2
Eine neue Gruppe der Größe 2 betritt das Restaurant!
Sie erhält die ID 'A' und wird den Plätzen 0 bis 1 zugewiesen.
AA______________________________
? +10
Eine neue Gruppe der Größe 10 betritt das Restaurant!
Sie erhält die ID 'B' und wird den Plätzen 2 bis 11 zugewiesen.
AABBBBBBBBBB____________________
? +3
Eine neue Gruppe der Größe 3 betritt das Restaurant!
Sie erhält die ID 'C' und wird den Plätzen 12 bis 14 zugewiesen.
AABBBBBBBBBBCCC_________________
? +5
Eine neue Gruppe der Größe 5 betritt das Restaurant!
Sie erhält die ID 'D' und wird den Plätzen 15 bis 19 zugewiesen.
AABBBBBBBBBBCCCDDDDD____________
? +2
Eine neue Gruppe der Größe 2 betritt das Restaurant!
Sie erhält die ID 'E' und wird den Plätzen 20 bis 21 zugewiesen.
AABBBBBBBBBBCCCDDDDDEE__________
? -A
Die Gruppe 'A' verlässt das Restaurant.
__BBBBBBBBBBCCCDDDDDEE__________
? -C
Die Gruppe 'C' verlässt das Restaurant.
__BBBBBBBBBB___DDDDDEE__________
? +8
Eine neue Gruppe der Größe 8 betritt das Restaurant!
Sie erhält die ID 'F' und wird den Plätzen 22 bis 29 zugewiesen.
__BBBBBBBBBB___DDDDDEEFFFFFFFF__
? -E
Die Gruppe 'E' verlässt das Restaurant.
__BBBBBBBBBB___DDDDD__FFFFFFFF__
? +2
Eine neue Gruppe der Größe 2 betritt das Restaurant!
Sie erhält die ID 'G' und wird den Plätzen 20 bis 21 zugewiesen.
__BBBBBBBBBB___DDDDDGGFFFFFFFF__
? -D
Die Gruppe 'D' verlässt das Restaurant.
__BBBBBBBBBB________GGFFFFFFFF__
? +4
Eine neue Gruppe der Größe 4 betritt das Restaurant!
Sie erhält die ID 'H' und wird den Plätzen 30 bis 33 zugewiesen.
HHBBBBBBBBBB________GGFFFFFFFFHH
? -F
Die Gruppe 'F' verlässt das Restaurant.
HHBBBBBBBBBB________GG________HH
? +5
Eine neue Gruppe der Größe 5 betritt das Restaurant!
Sie erhält die ID 'I' und wird den Plätzen 12 bis 16 zugewiesen.
HHBBBBBBBBBBIIIII___GG________HH
? +4
Eine neue Gruppe der Größe 4 betritt das Restaurant!
Sie erhält die ID 'J' und wird den Plätzen 22 bis 25 zugewiesen.
HHBBBBBBBBBBIIIII___GGJJJJ____HH
? +2
```

```
Eine neue Gruppe der Größe 2 betritt das Restaurant!
Sie erhält die ID 'K' und wird den Plätzen 17 bis 18 zugewiesen.
HHBBBBBBBBBBIIIIIKK_GGJJJJ____HH
? +2
Eine neue Gruppe der Größe 2 betritt das Restaurant!
Sie erhält die ID 'L' und wird den Plätzen 26 bis 27 zugewiesen.
HHBBBBBBBBBBIIIIIKK_GGJJJJLL__HH
? +3
Eine neue Gruppe der Größe 3 betritt das Restaurant!
Sie findet leider keinen Platz mehr. Der Ober ÄRGERT sich!
HHBBBBBBBBBBIIIIIKK_GGJJJJLL__HH
```

Ein Ausschnitt eines Zufalls-Laufes mit der BestFit-Strategie, in dem sich der Ober lange gegen die Gästemassen behaupten kann:

```
________________________________
Eine neue Gruppe der Größe 5 betritt das Restaurant!
Sie erhält die ID 'l' und wird den Plätzen 0 bis 4 zugewiesen.
lllll____________________________
Eine neue Gruppe der Größe 4 betritt das Restaurant!
Sie erhält die ID 'i' und wird den Plätzen 5 bis 8 zugewiesen.
lllllllll_________________________
Eine neue Gruppe der Größe 3 betritt das Restaurant!
Sie erhält die ID 'k' und wird den Plätzen 9 bis 11 zugewiesen.
lllllllllkkk______________________
Die Gruppe 'l' verlässt das Restaurant.
_____iiiikkk______________________
Eine neue Gruppe der Größe 2 betritt das Restaurant!
Sie erhält die ID 'o' und wird den Plätzen 12 bis 13 zugewiesen.
_____iiiikkkoo____________________
Eine neue Gruppe der Größe 4 betritt das Restaurant!
Sie erhält die ID 'q' und wird den Plätzen 14 bis 17 zugewiesen.
_____iiiikkkooqqqq________________
Eine neue Gruppe der Größe 3 betritt das Restaurant!
Sie erhält die ID 'r' und wird den Plätzen 18 bis 20 zugewiesen.
_____iiiikkkooqqqqrrr_____________
Die Gruppe 'i' verlässt das Restaurant.
_________kkkooqqqqrrr_____________
Die Gruppe 'k' verlässt das Restaurant.
____________ooqqqqrrr_____________
Eine neue Gruppe der Größe 2 betritt das Restaurant!
Sie erhält die ID 'n' und wird den Plätzen 21 bis 22 zugewiesen.
____________ooqqqqrrrnn__________
Eine neue Gruppe der Größe 4 betritt das Restaurant!
Sie erhält die ID 'p' und wird den Plätzen 23 bis 26 zugewiesen.
____________ooqqqqrrrnnpppp_____
Eine neue Gruppe der Größe 3 betritt das Restaurant!
Sie erhält die ID 'v' und wird den Plätzen 27 bis 29 zugewiesen.
____________ooqqqqrrrnnppppvvv__
Eine neue Gruppe der Größe 1 betritt das Restaurant!
Sie erhält die ID 'u' und wird den Plätzen 30 bis 30 zugewiesen.
____________ooqqqqrrrnnppppvvvu_
Die Gruppe 'q' verlässt das Restaurant.
____________oo____rrrnnppppvvvu_
Eine neue Gruppe der Größe 2 betritt das Restaurant!
Sie erhält die ID 't' und wird den Plätzen 14 bis 15 zugewiesen.
____________oott__rrrnnppppvvvu_
```

```
Die Gruppe 'o' verlässt das Restaurant.
______________tt__rrrnnppppvvvu_
Die Gruppe 'r' verlässt das Restaurant.
______________tt_____nnppppvvvu_
Die Gruppe 'v' verlässt das Restaurant.
______________tt_____nnpppp___u_
Die Gruppe 'n' verlässt das Restaurant.
______________tt_______pppp___u_
Die Gruppe 't' verlässt das Restaurant.
________________________pppp___u_
Eine neue Gruppe der Größe 5 betritt das Restaurant!
Sie erhält die ID 'w' und wird den Plätzen 31 bis 35 zugewiesen.
wwww____________________pppp___uw
Eine neue Gruppe der Größe 3 betritt das Restaurant!
Sie erhält die ID 's' und wird den Plätzen 27 bis 29 zugewiesen.
wwww____________________ppppsssuw
Eine neue Gruppe der Größe 2 betritt das Restaurant!
Sie erhält die ID 'x' und wird den Plätzen 4 bis 5 zugewiesen.
wwwwxx__________________ppppsssuw
Die Gruppe 'u' verlässt das Restaurant.
wwwwxx__________________ppppsss_w
Die Gruppe 'x' verlässt das Restaurant.
wwww____________________ppppsss_w
Eine neue Gruppe der Größe 5 betritt das Restaurant!
Sie erhält die ID 'z' und wird den Plätzen 4 bis 8 zugewiesen.
wwwwzzzzz_______________ppppsss_w
Die Gruppe 'w' verlässt das Restaurant.
____zzzzz_______________ppppsss__
Die Gruppe 'z' verlässt das Restaurant.
________________________ppppsss__
Eine neue Gruppe der Größe 5 betritt das Restaurant!
Sie erhält die ID 'y' und wird den Plätzen 30 bis 34 zugewiesen.
yyy_____________________ppppsssyy
Eine neue Gruppe der Größe 1 betritt das Restaurant!
Sie erhält die ID 'A' und wird den Plätzen 3 bis 3 zugewiesen.
yyyA____________________ppppsssyy
Eine neue Gruppe der Größe 1 betritt das Restaurant!
Sie erhält die ID 'C' und wird den Plätzen 4 bis 4 zugewiesen.
yyyAC___________________ppppsssyy
Eine neue Gruppe der Größe 1 betritt das Restaurant!
Sie erhält die ID 'B' und wird den Plätzen 5 bis 5 zugewiesen.
yyyACB__________________ppppsssyy
Eine neue Gruppe der Größe 5 betritt das Restaurant!
Sie erhält die ID 'J' und wird den Plätzen 6 bis 10 zugewiesen.
yyyACBJJJJJ_____________ppppsssyy
Eine neue Gruppe der Größe 1 betritt das Restaurant!
Sie erhält die ID 'L' und wird den Plätzen 11 bis 11 zugewiesen.
yyyACBJJJJJL____________ppppsssyy
Eine neue Gruppe der Größe 1 betritt das Restaurant!
Sie erhält die ID 'E' und wird den Plätzen 12 bis 12 zugewiesen.
yyyACBJJJJJLE___________ppppsssyy
Die Gruppe 'A' verlässt das Restaurant.
yyy_CBJJJJJLE___________ppppsssyy
Eine neue Gruppe der Größe 2 betritt das Restaurant!
Sie erhält die ID 'D' und wird den Plätzen 13 bis 14 zugewiesen.
yyy_CBJJJJJLEDD_________ppppsssyy
```

```
Die Gruppe 'E' verlässt das Restaurant.
yyy_CBJJJJJL_DD________ppppsssyy
Eine neue Gruppe der Größe 2 betritt das Restaurant!
Sie erhält die ID 'F' und wird den Plätzen 15 bis 16 zugewiesen.
yyy_CBJJJJJL_DDFF______ppppsssyy
Eine neue Gruppe der Größe 4 betritt das Restaurant!
Sie erhält die ID 'H' und wird den Plätzen 17 bis 20 zugewiesen.
yyy_CBJJJJJL_DDFFHHHH__ppppsssyy
Die Gruppe 's' verlässt das Restaurant.
yyy_CBJJJJJL_DDFFHHHH__pppp___yy
Eine neue Gruppe der Größe 5 betritt das Restaurant!
Sie findet leider keinen Platz mehr. Der Ober ÄRGERT sich!
yyy_CBJJJJJL_DDFFHHHH__pppp___yy
Eine neue Gruppe der Größe 2 betritt das Restaurant!
Sie erhält die ID 'I' und wird den Plätzen 21 bis 22 zugewiesen.
yyy_CBJJJJJL_DDFFHHHHIIpppp___yy
Eine neue Gruppe der Größe 3 betritt das Restaurant!
Sie erhält die ID 'T' und wird den Plätzen 27 bis 29 zugewiesen.
yyy_CBJJJJJL_DDFFHHHHIIppppTTTyy
Die Gruppe 'J' verlässt das Restaurant.
yyy_CB_____L_DDFFHHHHIIppppTTTyy
Die Gruppe 'D' verlässt das Restaurant.
yyy_CB_____L___FFHHHHIIppppTTTyy
Die Gruppe 'T' verlässt das Restaurant.
yyy_CB_____L___FFHHHHIIpppp___yy
Die Gruppe 'B' verlässt das Restaurant.
yyy_C______L___FFHHHHIIpppp___yy
Die Gruppe 'H' verlässt das Restaurant.
yyy_C______L___FF____IIpppp___yy
Eine neue Gruppe der Größe 1 betritt das Restaurant!
Sie erhält die ID 'U' und wird den Plätzen 3 bis 3 zugewiesen.
yyyUC______L___FF____IIpppp___yy
Die Gruppe 'I' verlässt das Restaurant.
yyyUC______L___FF______pppp___yy
Eine neue Gruppe der Größe 3 betritt das Restaurant!
Sie erhält die ID 'M' und wird den Plätzen 27 bis 29 zugewiesen.
yyyUC______L___FF______ppppMMMyy
Die Gruppe 'p' verlässt das Restaurant.
yyyUC______L___FF__________MMMyy
Eine neue Gruppe der Größe 5 betritt das Restaurant!
Sie erhält die ID 'P' und wird den Plätzen 5 bis 9 zugewiesen.
yyyUCPPPPP_L___FF__________MMMyy
Die Gruppe 'U' verlässt das Restaurant.
yyy_CPPPPP_L___FF__________MMMyy
Eine neue Gruppe der Größe 1 betritt das Restaurant!
Sie erhält die ID 'G' und wird den Plätzen 3 bis 3 zugewiesen.
yyyGCPPPPP_L___FF__________MMMyy
Die Gruppe 'C' verlässt das Restaurant.
yyyG_PPPPP_L___FF__________MMMyy
Eine neue Gruppe der Größe 2 betritt das Restaurant!
Sie erhält die ID 'Z' und wird den Plätzen 12 bis 13 zugewiesen.
yyyG_PPPPP_LZZ_FF__________MMMyy
Eine neue Gruppe der Größe 3 betritt das Restaurant!
Sie erhält die ID 'N' und wird den Plätzen 17 bis 19 zugewiesen.
yyyG_PPPPP_LZZ_FFNNN_______MMMyy
Die Gruppe 'y' verlässt das Restaurant.
```

```
___G_PPPPP_LZZ_FFNNN_______MMM__
Die Gruppe 'N' verlässt das Restaurant.
___G_PPPPP_LZZ_FF__________MMM__
Eine neue Gruppe der Größe 1 betritt das Restaurant!
Sie erhält die ID 'V' und wird den Plätzen 4 bis 4 zugewiesen.
___GVPPPPP_LZZ_FF__________MMM__
Eine neue Gruppe der Größe 2 betritt das Restaurant!
Sie erhält die ID 'd' und wird den Plätzen 30 bis 31 zugewiesen.
___GVPPPPP_LZZ_FF__________MMMdd
Eine neue Gruppe der Größe 2 betritt das Restaurant!
Sie erhält die ID 'Y' und wird den Plätzen 0 bis 1 zugewiesen.
YY_GVPPPPP_LZZ_FF__________MMMdd
Die Gruppe 'L' verlässt das Restaurant.
YY_GVPPPPP__ZZ_FF__________MMMdd
Eine neue Gruppe der Größe 5 betritt das Restaurant!
Sie erhält die ID 'R' und wird den Plätzen 17 bis 21 zugewiesen.
YY_GVPPPPP__ZZ_FFRRRRR_____MMMdd
Eine neue Gruppe der Größe 1 betritt das Restaurant!
Sie erhält die ID 'O' und wird den Plätzen 2 bis 2 zugewiesen.
YYOGVPPPPP__ZZ_FFRRRRR_____MMMdd
Die Gruppe 'P' verlässt das Restaurant.
YYOGV_______ZZ_FFRRRRR_____MMMdd
Die Gruppe 'V' verlässt das Restaurant.
YYOG________ZZ_FFRRRRR_____MMMdd
Eine neue Gruppe der Größe 2 betritt das Restaurant!
Sie erhält die ID 'K' und wird den Plätzen 22 bis 23 zugewiesen.
YYOG________ZZ_FFRRRRRKK___MMMdd
Eine neue Gruppe der Größe 2 betritt das Restaurant!
Sie erhält die ID 'W' und wird den Plätzen 24 bis 25 zugewiesen.
YYOG________ZZ_FFRRRRRKKWW_MMMdd
Die Gruppe 'O' verlässt das Restaurant.
YY_G________ZZ_FFRRRRRKKWW_MMMdd
Die Gruppe 'R' verlässt das Restaurant.
YY_G________ZZ_FF_____KKWW_MMMdd
Eine neue Gruppe der Größe 5 betritt das Restaurant!
Sie erhält die ID 'j' und wird den Plätzen 17 bis 21 zugewiesen.
YY_G________ZZ_FFjjjjjKKWW_MMMdd
Die Gruppe 'M' verlässt das Restaurant.
YY_G________ZZ_FFjjjjjKKWW____dd
Eine neue Gruppe der Größe 2 betritt das Restaurant!
Sie erhält die ID 'e' und wird den Plätzen 26 bis 27 zugewiesen.
YY_G________ZZ_FFjjjjjKKWWee__dd
Eine neue Gruppe der Größe 1 betritt das Restaurant!
Sie erhält die ID 'X' und wird den Plätzen 2 bis 2 zugewiesen.
YYXG________ZZ_FFjjjjjKKWWee__dd
Eine neue Gruppe der Größe 4 betritt das Restaurant!
Sie erhält die ID 'S' und wird den Plätzen 4 bis 7 zugewiesen.
YYXGSSSS____ZZ_FFjjjjjKKWWee__dd
Die Gruppe 'K' verlässt das Restaurant.
YYXGSSSS____ZZ_FFjjjjj__WWee__dd
Eine neue Gruppe der Größe 1 betritt das Restaurant!
Sie erhält die ID 'a' und wird den Plätzen 14 bis 14 zugewiesen.
YYXGSSSS____ZZaFFjjjjj__WWee__dd
Die Gruppe 'G' verlässt das Restaurant.
YYX_SSSS____ZZaFFjjjjj__WWee__dd
Die Gruppe 'd' verlässt das Restaurant.
```

```
YYX_SSSS____ZZaFFjjjjj__WWee____
Eine neue Gruppe der Größe 5 betritt das Restaurant!
Sie findet leider keinen Platz mehr. Der Ober ÄRGERT sich!
YYX_SSSS____ZZaFFjjjjj__WWee____
Die Gruppe 'Y' verlässt das Restaurant.
__X_SSSS____ZZaFFjjjjj__WWee____
Eine neue Gruppe der Größe 5 betritt das Restaurant!
Sie erhält die ID 'b' und wird den Plätzen 28 bis 32 zugewiesen.
b_X_SSSS____ZZaFFjjjjj__WWeebbbb
Eine neue Gruppe der Größe 5 betritt das Restaurant!
Sie findet leider keinen Platz mehr. Der Ober ÄRGERT sich!
```

Erweiterungen

Die Ober-Strategie könnte so erweitert werden, dass sie wenn eine eintreffende Gruppe keinen Platz mehr findet, prüft, ob das durch das Weiterrücken einer anderen Gruppe zu lösen wäre. Sie hätte berechnet, was die geringste Belästigung für die anwesenden Gäste wäre. Ich habe diese Erweiterung nicht implementiert, und mir scheint auch zweifelhaft, ob sie in der Realität überhaupt zum Einsatz kommen würde. In meiner Vorstellung ist das Turmrestaurant ein sehr feines Restaurant, da wäre ein „Könn' se bitte ma aufrücken?“ natürlich undenkbar.