Übung 7 - Musterlösung Abgabe 14.6.2012

Aufgabe 1 – Denial-of-service-Angriff 8 Punkte

a) Eine Möglichkeit die Aufgabe zu lösen ist folgende: Die Hash-Funktion lautet

```
def hhash(s):
    h = 0
    for k in s:
        h = 23*h + ord(k)
    return h
```

Durch sukzessives Einsetzen erhalt man für $string = s_1 s_2 s_3 s_4$ und $ord(s_k) = o_k$:

$$h(s_1, s_2, s_3, s_4) = 23^3 \cdot o_1 + 23^2 \cdot o_2 + 23^1 \cdot o_3 + 23^0 \cdot o_4$$

Ziel ist es, die Variablen o_1, o_2, o_3, o_4 so zu variieren, dass h konstant bleibt. Man nehme z.B $o_1 = o_2 = o_3 = o_4 = 50$, also den String "2222". Dann ist

$$23^3 \cdot 50 + 23^2 \cdot 50 + 23^1 \cdot 50 + 23^0 \cdot 50 = 636000$$

Verringert man die erste Variable um 1, kann man die zweite zum Ausgleich um 23 erhöhen und hat dasselbe Ergebnis:

$$23^3 \cdot (50 - 1) + 23^2 \cdot (50 + 23) + 23^1 \cdot 50 + 23^0 \cdot 50 = 636000$$

$$23^3 \cdot 50 - 23^3 + 23^2 \cdot 50 + 23^2 \cdot 23 + 23^1 \cdot 50 + 23^0 \cdot 50 = 636000$$

$$23^3 \cdot 50 - 23^3 + 23^2 \cdot 50 + 23^3 + 23^1 \cdot 50 + 23^0 \cdot 50 = 636000$$

$$23^3 \cdot 50 + 23^2 \cdot 50 + 23^1 \cdot 50 + 23^0 \cdot 50 = 636000$$

Eine Lösung, die denselben Hashwert hat, ist demnach $49{,}73{,}50{,}50$, also der String "1I22". Auf diesem Prinzip bauen beide implementierte Lösungen auf. Die erste Variante variiert Strings der Länge 2 und kombiniert diese dann zu Strings der Länge 4. Die zweite Variante verallgemeinert das Variationsprinzip: Die zwei Funktionen `vary()` und `compensate()` funktionieren folgendermaßen: `compensate()` bekommt einen Rest und verteilt ihn entsprechend der Koeffizienten auf die Werte. `vary()` variiert ab einre gegebenen Position eine Stelle in einem gewissen Rahmen und ruft dann `compensate()` auf, um die Variation auszugleichen. Das Ergebnis wird in ein Dictionary geschrieben, die Position wird nach rechts verschoben und die Variation wird für alle Ergebnisse erneut durchgeführt. Der angebene Rahmen umfasst die alphanumerischen Zeichen. Zum Schluss werden die gewonnen Werte in Strings übersetzt, nach Sonderzeichen gefiltert, in eine Datei geschrieben und mit ihren Hashwerten ausgegeben.

Beide Implementationen befinden sich im Anhang unter `collisions.py`. **Implementation 1** liefert die Strings: '5z5z', '5z6c', '5z7L', '5z85', '6c5z', '6c6c', '6c7L', '6c85', '7L5z', '7L6c', '7L7L', '7L85', '855z', '856c', '857L', '8585' mit dem Hash 710730. **Implementation 2** ergibt 'Akzx', 'Alcx', 'Alda', 'AleJ', 'Alf3', 'AmLx', 'AmMa', 'AmNJ', 'AmO3', 'An5x', 'An6a', 'An7J', 'An83', 'BTzx', 'BUda', 'BVLx', 'BVMa', 'BVNJ', 'BVO3', 'BW5x', 'BW6a', 'BW7J', 'BW83' mit Hash 850384.

Aufgabe 2 – Cocktail-Datenbank 26 Punkte

a) Eine mögliche Implementation befindet sich im Anhang unter `cocktails.py`. Zunächst wird das dictionary auf eine angenehme Form gebracht, die Strings werden normalisiert und einige Manual Mappings werden durchgeführt. Dann werden die Zutaten entfernt, die ohnehin auf der Ignore-List[1] stehen, so spart man sich später Arbeit. Damit kommt man auf 462 Zutaten.

b) Das dictionary wird invertiert und eine List mit den 15 häufigsten Zutaten ausgegeben (abzüglich derer auf der Ignore-List). Die Ausgabe:

```
__________________Die 15 gebraeuchlichsten Zutaten__________________________


orangensaft    214
ananassaft    155
limettensaft    127
grenadine    118
rum    116
gin    113
weißerrum    113
bluecuracao    98
orangen    82
erdbeeren    80
wermut    75
limette    74
maracujasaft    66
minze    55
amaretto    52
```

c) Diese Funktion wird mit sets realisiert. Das Beispiel gibt den Cocktail „Metthattan“ aus.

d) Da der Raum, über den hier iteriert wird, wirklich ausufernde Dimensionen hat, lohnt es sich, sich zunächst enige Gedanken über dessen Struktur zu machen. Zunächst werden die Zutaten in einzelne Buckets eingefügt: Jeder Bucket hat die Eigenschaft, dass jeweils zwei paarweise kombinierte Zutaten aus demselben in keinem möglichen Cocktail zusammen zu finden sind. Man spart sich also unnötige Kombinationen. Zudem sind die buckets nach relativer Wichtigkeit der Zutaten geordnet und im bucket selbst sind die Zutaten nach absoluter Wichtigkeit geordnet.

Relativ trivial ist, dass alle Cocktails, die mehr als 5 Zutaten (+ Ignore-List) benötigen, vorher aus den dictionaries genommen werden, weil sie ohnehin nicht in Frage kommen.

Es ist nun relativ wahrscheinlich, dass man eine gute Lösung direkt am Anfang findet. Nach wenigen Sekunden findet man eine Kombination mit 22 Cocktails. Dies kann man folgendermaßen ausnutzen: Hat man 5 Zutaten, die sich zu 22 Cocktails kombinieren lassen, so impliziert dies, dass jede Zutat in mindestens 11 verschiedenen Cocktails enthalten sein muss. Denn: Aus 4 Zutaten kann man maximal

$$\binom{4}{2} + \binom{4}{3} + \binom{4}{4} = 11$$

[1] Anmerkung: Die Zutaten auf dieser Ignore-List sind für den Haushalt eines angehenden Akademikers absolut essentiell. Der Autor empfiehlt die sofortige Anschaffung, falls nicht vorhanden.

mixen. Wäre die nächste Zutat, welche man hinzufügt, nur in einem Cocktail, käme man maximal auf 12 verschiedene Cocktails. Mit einer Zutat, die in zwei Cocktails vorkommt, käme man auf 13 usf.

So kann man alle Zutaten streichen, die in weniger als 11 Cocktails vorkommen. Das schränkt den Raum der Möglichkeiten beträchtlich ein! Im Anhang befindet sich unter `cocktails.py` eine Möglichkeit für eine Kombination aus 5 Zutaten und 18 buckets. Nach 10 Sekunden hat man folgende Ausgabe:

```
Zutaten:  [u'bluecuracao', u'orangensaft', u'gin', u'erdbeeren', u'wermut']

benutzt in folgenden  23  cocktails:

--------------------------------------------------

["Pimm's Reef", 'Walzer - Cocktail', 'Zitronen-Sektcocktail mit Erdbeeren', 'Sekt
mit Blue Curacao', 'James Bond', 'Zitronen - Sekt - Cocktail mit Erdbeeren',
'Extra Dry Martini', 'Erdbeer - Sekt', 'Grüne Wiese', 'Screwdriver', 'Dry
Martini', 'Erdbeeren Aperitif', 'Martini Cocktail', 'Wodka Martini', 'Peekaboo
Cocktail', 'Perfect Cocktail', 'Blue Lagoon', 'Aqua de Valencia', 'Martini
Aperitif', 'Blue Lady', 'Green Widow Cocktail', 'Martini', 'Martini Cocktail
(plain vanilla)']
```

Anhang – collisions.py

```
from copy import copy
from numpy import unique
from itertools import product

#the hash function
def hhash(s):
      h = 0
      for k in s:
            h = 23 * h + ord(k)
      return h

print '#--------------------first approach----------------------#'
def construct2():
      k = 122 # choose starting character codes
      j = k - 3*23

      res = []
      for i in xrange(4):
            res.append(chr(j)+chr(k))
            j += 1   # vary the first character ...
            k -= 23  # and compensate in the second
      return res

def construct4():
      a = construct2() # create four 2-strings
      # out of every pair of 2-strings build a 4-string
      return [''.join(k) for k in product(a, a)]

def testCollisions(a):#test it!
      h = hhash(a[0])
      for k in a:
            if hhash(k) != h:
                  print "Error: Hash values do not collide!"
                  return
      print "All keys have hash value:", h

def exportCollisions(a):#write it to file
      file('collisions1.txt', 'w').write('\n'.join(a))

solutions = construct4()
print solutions
testCollisions(solutions)
exportCollisions(solutions)
```

```
print '#--------------------second approach----------------------# '
#compensate a given residue by changing the values!
def compensate(rest,init,coeffs):
      if len(init) > 0:
            ret = []
            initcopy = copy(init)    #copy the lists so pop does not destroy the originals
            coeffcopy = copy(coeffs)
            while rest != 0 and len(initcopy) > 0: # while there is still a rest and initcopy
isnt empty
                  c = coeffcopy.pop(0) # get coefficient
                  s = rest/c              # how much of the rest fits at this position
                  r = initcopy.pop(0)   # how much is already there
                  if r+s > 123:           # if its too much (out of range)
                        ret.append(123)   # append maximum
                        rest = rest-(123-r)*c #and recalculate the rest
                  elif r+s < 48:          # likewise for too little
                        ret.append(48)
                        rest = rest-(48+r)*c
                  else:
                        ret.append(r+s)   # likewise for fitting amounts
                        rest = rest-(s)*c
      if rest ==0:                        # if rest is 0
            return ret+init[len(ret):] # return the corrected positions + what is missing if so
      else:
            return None                  # return None when it is too much

def vary(init,coeffs,deg): # vary a solution from a certain point on (deg)
      solutions = {deg:[init]} # save solutions in a dic, key is degree
      while deg > 1:   # while the degree is reasonable
            deg -= 1      # iter down
            solutions[deg] = [] #initialize new solution list
            for sol in solutions[deg+1]: #for every solution!
                  t = 1 #iterparams
                  ind = len(sol)-deg-1
                  while True:
                        plus = compensate(t*coeffs[ind],sol[ind+1:],coeffs[ind+1:]) #vary
positive, get changed values
                        if plus is not None:
                              solutions[deg].append(sol[:ind]+[sol[ind]-t]+plus)   #build   solution
from not-changed and changed values for
                        minus   =   compensate(-t*coeffs[ind],sol[ind+1:],coeffs[ind+1:])   #likewise
negative
                        if minus is not None:
                              solutions[deg].append(sol[:ind]+[sol[ind]+t]+minus)
                        if plus is None and minus is None: #if nothing is possible, break
                              break
                        t += 1 #iter vary amount
      ret = []
      for value in solutions.values(): #unpack dic
            ret += value
      return ret

def stringFilter(strings): #just strings using numbers, Big and small letters
      allowed = range(97,123)+range(65,91)+range(48,58)
      ret = []
      for string in strings:
```

```
            flag = 0
            for char in string:
                  if ord(char) in allowed:
                        continue
                  else:
                        flag = 1
                        break
            if flag == 0:
                  ret.append(string)
      return ret


def to_String(solutions): #turn numerical solution inter a string
      ret = []
      for s in solutions:
            string = ''
            for char in s:
                  string += chr(char)
            ret.append(string)
      return list(unique(ret))


init = [ord(char) for char in list('Alda')] #init string
coeffs = [23**3,23**2,23**1,23**0] #coeffs of hash function
solutions = [] #solutions
for i in range(1,5): #vary from all positions
      solutions += vary(init,coeffs,i)
strings = stringFilter(to_String(solutions)) #convert to string, filter
print strings
f = open('collisions2.txt','w') #write to file
for string in strings:
      f.write(string+'\n')
f.close()
testCollisions(strings) #show to the tutor it works!
```

Anhang – cocktails.py

```
# -*- coding: latin-1 -*-
import json, codecs, numpy, re, sys, itertools
from collections import OrderedDict
from functools import partial

def get_CocktailDic(path):
    f = codecs.open(path, 'r')
    res = json.load(f)
    #restructure dictionary
    for key in res.keys():#use sets to make sure every ingredient appears once
        res[key] = set([normalize(string) for string in res[key]['ingredients'].keys()])
    return res

def normalize(string):
    string = string.lower() #only lower case
    string = string.encode('latin-1')
    string = str(re.sub('\(.*\)', '', string)) #remove brackets and their contents
    string = str(re.sub(',.*', '', string)) #remove everything after a comma
    for k in list(', :('):#remove special chars
        string = string.replace(k, '')
    string = string.decode('latin-1')
    return string

def sortByLength(dic):
    ordered_d = sorted(dic.viewitems(), key=lambda x: len(x[1]))
    ordered_d.reverse()
    ordered_d = OrderedDict(ordered_d)
    return ordered_d

def sortByKeys(dic):
    ordered_d = sorted(dic.viewitems(), key=lambda x: x[0])
    ordered_d.reverse()
    ordered_d = OrderedDict(ordered_d)
    return ordered_d

def inverse(d):
    inv = {}
    for k, v in d.iteritems():
        for j in v:
            inv[j] = inv.get(j, [])
            inv[j].append(k)
    return inv
#replace ing1 with ing2
def replace_Ingredient(cocktailDic, ing1, ing2):
    for cocktail in cocktailDic.keys():
        if ing1 in cocktailDic[cocktail]:
            cocktailDic[cocktail].remove(ing1)
            cocktailDic[cocktail].add(ing2)
#some manual replacements which cannot be done by string operations in a simple
#way
def manual_Mappings(cocktailDic):
    replace_Ingredient(cocktailDic,u'crushedice',u'eiswrfel')
```

```
	replace_Ingredient(cocktailDic,u'eigelb',u'ei')
	replace_Ingredient(cocktailDic,u'eiwei',u'ei')
	replace_Ingredient(cocktailDic,u'erdbeere',u'erdbeeren')
	replace_Ingredient(cocktailDic,u'erdbeer',u'erdbeeren')
	replace_Ingredient(cocktailDic,u'redbull',u'energydrink')
	replace_Ingredient(cocktailDic,u'zuckerrohrsirup',u'zuckersirup')
	replace_Ingredient(cocktailDic,u'zitronensaft',u'zitrone')
	replace_Ingredient(cocktailDic,u'zitronenschale',u'zitrone')
	replace_Ingredient(cocktailDic,u'schokoladenstreusel',u'schokoladenraspel')
	replace_Ingredient(cocktailDic,u'vanillemark',u'vanilleschote')
	replace_Ingredient(cocktailDic,u'schokoladenraspel',u'schokolade')
	replace_Ingredient(cocktailDic,u'orangenscheibe',u'orangen')
	replace_Ingredient(cocktailDic,u'orange',u'orangen')
	replace_Ingredient(cocktailDic,u'orangenschale',u'orangen')
	replace_Ingredient(cocktailDic,u'orangenlimonade',u'fanta')
	replace_Ingredient(cocktailDic,u's sahne',u'sahne')
	replace_Ingredient(cocktailDic,u'natur-joghurt',u'joghurt')
	replace_Ingredient(cocktailDic,u'ananas-bl er',u'ananas')
	replace_Ingredient(cocktailDic,u'ananas-schnitz',u'ananas')
	replace_Ingredient(cocktailDic,u'sahneersatz',u'sahne')
	replace_Ingredient(cocktailDic,u'milch-schaum',u'milch')
	replace_Ingredient(cocktailDic,u'rohrzucker',u'zucker')
	replace_Ingredient(cocktailDic,u'meersalz',u'salz')
	replace_Ingredient(cocktailDic,u'saftundzuckergemischt*',u'saft')
	replace_Ingredient(cocktailDic,u'saftvontropischenfrchten',u'saft')
	return cocktailDic
#delete every ingredient from ignoreList in this dictionary
def ignoreDic(cocktailDic, ignoreList):
	for ignore in ignoreList:
		for cocktail in cocktailDic.keys():
			if ignore in cocktailDic[cocktail]:
				cocktailDic[cocktail].remove(ignore)
	return cocktailDic
#delete every ingredient from ignoreList in this list
def ignoreL(ingredientList, ignoreList):
	for ig in ignoreList:
		if ig in ingredientList:
			ingredientList.remove(ig)
#return a list of possible cocktails for this ingredientlist
def possible_cocktails(inverseD, Dic, available):
	cocktails = set([])
	possible = []
	availableSet = set(available)
	for ingredient in available:
		cocktails = cocktails.union(set(inverseD[ingredient]))
	for cocktail in cocktails:
		if set(Dic[cocktail]).issubset(availableSet):
			possible.append(cocktail)
	return possible
#check if two ingredients appear in a common cocktail
def commonCocktail(ing0, ing1, revDic):
	if ing0 is None or ing1 is None:
		return False
	elif len(set(revDic[ing1]).intersection(set(revDic[ing0]))) != 0:
		return True
	return False
#bucket all ingredients so that in every bucket every possible combination of two
#ingredients does not appear in a common cocktail. this saves unnecessary iterations
```

```
def bucket_ingredients(cocktailDic, ingredients):
      revDic = inverse(cocktailDic)
      buckets = [[]]
      while len(ingredients) > 0:
            ing0 = ingredients.pop()#get ingredient from list
            for i in range(len(buckets)):#check every bucket
                  flag = 0
                  for ing1 in buckets[i]:#check with every ingredient in that bucket
                        if commonCocktail(ing0, ing1, revDic):#if it has a common cocktail...
                              flag = 1 #...set the flag
                              break #...and break
                  if flag == 0:#if no common cocktail exists, ingredient goes into that bucket
                        buckets[i].append(ing0)
                        break#since it has its bucket, others dont need to be checked
                  elif flag == 1 and i == len(buckets) - 1:#if it didnt fit in a bucket and all
buckets have been tried
                        buckets.append([ing0])#append a new bucket
                        break
      return buckets
#return a list from ingredients from a cocktailDic
def get_ingredients(dic):
      ingredients = []
      for cocktail in dic.keys():
            ingredients += dic[cocktail]
      return list(numpy.unique(ingredients))
#get priorities for each ingredient, priority = # of cocktails for ingredient which ingredient is
used
def get_priorities(dic):
      sortedInverse = sortByLength(inverse(dic))
      for key in sortedInverse.keys():
            sortedInverse[key] = len(sortedInverse[key])
      return sortedInverse
#cancel all ingredients which arent used in a least "limit" cocktails
def limit_To(dic, limit):
      for key in dic.keys():
            if len(dic[key]) > limit:
                  del dic[key]
      return dic


ignoreList = [u'salz', u'pfeffer',u'wasser',u'cocktailkirsche',u'eiswrfel',u'zucker', \
                  u'sahne',u'zitrone',u'wodka',u'ei',u'schnaps',u'milch',u'bier',u'zimt',u'tee',
\
                  u'sekt',u'/span>,u'saft',u'olive',u'mett']



#a)
#get dic and eliminate superfluous ingredients
res = get_CocktailDic('cocktails.json')
res = manual_Mappings(res)
res = ignoreDic(res, ignoreList)
ingredients = get_ingredients(res)

print '\n'
print '_____Anzahl der Zutaten_____'
print '\n'
```

```
print 'Anzahl: ', len(ingredients)

#b)
#get inverse dic and sort it by length of the cocktail-List
resInv = inverse(res)
resInv = sortByLength(resInv)

print '\n'
print '_________________Die 15 gebraeuchlichsten Zutaten_____________________'
print '\n'
c = 0
for key, value in resInv.items():
     print key, '  ', len(value)
     c += 1
     if c == 15:
          break
print '\n'

#c)
#set ingredient List before looking for possible cocktails. more effective later
#on
ingList = [u'genever', u'pfeffer', u'mett', u'zwiebel']
#filter ingredient list
ignoreL(ingList, ignoreList)
print '\n'
print '_________________moegliche cocktails fuer_____________________________'
print ingList
print '\n'
for cocktail in possible_cocktails(resInv, res, ingList):
     print cocktail
print '\n'

#the Metthattan!

#d)
#limit to cocktails with 5 ingredients, because there are only 5 available
res = limit_To(res, 5)
ingredients = get_ingredients(res)
resInv = inverse(res)
#limit to ingredients which can provide 12 or more cocktails
for ing in ingredients:
     if len(resInv[ing]) < 12:
          ingredients.remove(ing)
#get priorities for each ingredient
ingredNum = len(ingredients)
priorities = get_priorities(res)
bucketDic = {}
#bucket ingredients. each bucket has the property that for every ingredient
#contained in it, it is impossible to be combined with any other ingredient in
#the same bucket
buckets = bucket_ingredients(res, ingredients)
#make dict with key -> bucket: averageUsefullness -> bucket
for b in range(len(buckets)):
     priority = 0
     for ingredient in buckets[b]:
          priority += priorities[ingredient]
```

```
      key = float(priority) / len(buckets[b])
      bucketDic[key] = buckets[b]
#sort from high to low
bucketDic = sortByKeys(bucketDic)
#sort buckets by usefulness from high to low
for key in bucketDic.keys():
      bucketDic[key] = sorted(bucketDic[key], key=lambda x: priorities[x])
      bucketDic[key].reverse()
#initiate counting parameters and a partial possible cocktail function, so
#it does not have to be initiated every time
bestCocktailNum = 0
bestCombination = None
bucketList = [value for value in bucketDic.values()]
partialPossible_Cocktails = partial(possible_cocktails, resInv, res)
#make a list with ranges for a multi-index
bucketIndex = [range(len(bucket)) for bucket in bucketList]
print '_____best Combination_____'
print 'Zutaten: ', ingredNum, 'Buckets: ', len(bucketList)
for indeces in itertools.product(*bucketIndex):#iter through all possible ingredient
combinations...
      for bucketCombo in itertools.permutations(bucketList, 5):#...for 5 ingredients
            ings = []
            for bucket in bucketCombo:
                  ings.append(bucket[indeces[bucketList.index(bucket)]])#look up the current
ingrdients index
            possible = partialPossible_Cocktails(ings)
            if len(possible) > bestCocktailNum:
                  bestCocktailNum = len(possible)
                  bestCombination = ings
                  print '_____'
                  print ''
                  print 'Zutaten: ', bestCombination
                  print 'benutzt in folgenden ',bestCocktailNum,' cocktails:'
                  print '-----'
                  print [cocktail.encode('latin-1') for cocktail in possible_cocktails(resInv,
res, bestCombination)]
```