Übung 7 Abgabe 14.6.2012

Aufgabe 1 – Denial-of-Service-Angriff 10 Punkte

Hashtabellen werden ineffizient, wenn die verwendete Hashfunktion viele Kollisionen verursacht. Ein Hacker, der die Hashfunktion kennt, kann diese Tatsache ausnutzen, um absichtlich ineffiziente Anfragen zu konstruieren: Er wählt die in der Anfrage verwendeten Schlüssel gezielt so, dass alle auf den gleichen Hashwert abgebildet werden. Werden viele ineffiziente Anfragen dieser Art gleichzeitig gestartet, kann die angegriffene Webseite (die die gegebene Hashfunktion intern benutzt) zusammenbrechen (denial-of-service). In der Praxis verhindert man dies durch die Verwendung von universellem Hashing, indem man die Hashfunktion per Zufall aus einem großen Pool erlaubter Hashfunktionen auswählt. Dann ist es nicht mehr möglich, im Vorhinein ein ungünstiges Set von Schlüsseln zu konstruieren. Für diese Übungsaufgabe wollen wir aber annehmen, dass stets die folgende einfache Hashfunktion verwendet wird:

```
def hhash(s): # s ist ein Schlüssel vom Typ string
    h = 0     # der Hashwert wird mit 0 initialisiert
    for k in s:
        h = 23*h + ord(k)  # Aktualisieren des Hashs mit dem Zeichencode
    return h
```

Dies ist eine Variante der Bernsteinfunktion aus der Vorlesung, wo der Multiplikator „33“ durch „23“ ersetzt wurde. Finden Sie mindestens 16 Schlüssel (Strings) der Länge 4, die alle den gleichen Hashwert haben. Geben Sie diese Schlüssel im File "`collisions.txt`" ab (ein String pro Zeile) und beschreiben Sie, wie Sie vorgegangen sind, um diese Schlüssel zu finden. Hinweis: Beginnen Sie damit, Kollisionen mit Schlüsseln der Länge 2 zu konstruieren und verwenden Sie diese Ergebnisse zur Konstruktion von Schlüsseln der Länge 4.

Aufgabe 2 – Cocktail-Datenbank 26 Punkte

In dieser Aufgabe wollen wir üben, wie man mit dem Python-Datentyp `dict` und dem JSON-Format umgeht. Das File http://hci.iwr.uni-heidelberg.de/Staff/ukoethe/download/cocktails.json enthält eine Liste von Cocktail-Rezepten im JSON-Format (die Zeichencodierung ist, wie bei JSON gefordert, UTF-8). Das Einlesen des Files in Python erfolgt am einfachsten mit dem Standardmodul `json` (siehe Dokumentation unter http://docs.python.org/library/json.html). Das File hat folgende Struktur:

```
{
   "Name des Cocktails": {
       "ingredients": {
           "Zutat": "Menge",
           "Whisky": "2 cl",
           "Cola": "10 cl"
        },
        "instructions": "Anleitung zum Mixen",
        "summary": "Kurzbeschreibung (optional)"
   }
   ...
}
```

Die Datei enthält insgesamt 1158 Rezepte, die beim Einlesen mittels `json`-Modul in einem Python-`dict` abgelegt werden. Ihre Lösungen sollen im File "`cocktails.py`" abgegeben werden. Verwenden Sie Kommentare im Code, um Ihr Vorgehen kurz und prägnant zu begründen.

a) Lesen Sie die Datei "`cocktails.json`" in ein Dictionary `recipes` ein und schreiben Sie eine Funktion `all_ingredients(recipes)`, die eine Gesamtliste aller Zutaten zurückgibt. Versuchen Sie, die Namen der Zutaten soweit wie möglich zu normalisieren, indem Sie Unwichtiges *6 Punkte*

weglassen (z.B. Sonderzeichen, Großschreibung, zusätzliche Informationen). Beispielsweise sollen "Sahne", "Sahne (flüssig)" und "Sahne (30%)" alle gleichermaßen als "sahne" gezählt werden. Implementieren Sie dazu eine Hilfsfunktion `normalizeString(s)`. Wie viele verschiedene Zutaten werden (nach der Normalisierung) insgesamt gefunden?

b) Schreiben Sie eine Funktion `inverse_recipes = cocktails_inverse(recipes)`, die das Dictionary invertiert: Die in a) gefundenen Zutaten sollen jetzt die Schlüssel sein, und die zugehörigen Werte sind Listen der Cocktailnamen, in denen die jeweilige Zutat benötigt wird. Exportieren Sie das Ergebnis in ein File "`cocktails_inverse.json`", das etwa so aussehen soll: *4 Punkte*

```
{
   "sahne": [ "Pimm's Reef", "Alexandra Cocktail", ...],
   ...
}
```

Benutzen Sie das inverse Dictionary außerdem, um die Zutaten nach Häufigkeit zu sortieren. Welche 15 Zutaten werden am häufigsten benötigt?

c) Schreiben Sie eine Anfragefunktion `possible_cocktails(inverse_recipes, available_ingredients)` mit folgendem Verhalten: Gegeben sei eine Liste von Zutaten `available_ingredients`, die Sie zufällig im Hause haben. Die Funktion gibt eine Liste von Cocktails zurück, die man aus den gegebenen Zutaten (und nur diesen) herstellen kann. Die Ergebnisliste ist leer, wenn sich kein möglicher Cocktail findet. Um Ihre Erfolgschancen zu vergrößern, sollten Sie unwichtige Zutaten ignorieren. Das gilt insbesondere für Zutaten, die nur zur Dekoration des Glases dienen (wie z.B. Cocktailkirschen) – die kann man zur Not durch etwas anderes ersetzen oder ganz weglassen. Erstellen Sie dazu eine `ignore_list`. *6 Punkte*

d) Benutzen Sie die Anfragefunktion aus c), um herauszufinden, welche 5 (oder 7) Zutaten man im Hause haben sollte, damit man möglichst viele verschiedene Cocktails herstellen kann. Erweitern Sie dazu die `ignore_list` um triviale Zutaten, die ohnehin überall vorrätig sind (wie z.B. Wasser, Salz und Pfeffer) und deshalb nicht mitgezählt werden sollen. Schreiben Sie eine Funktion `optimal_ingredients(inverse_recipes)`, die systematisch verschiedene Kombinationen von Zutaten ausprobiert und die beste Kombination zurückgibt (wenn die Suche zu lange dauert, verwenden Sie ein geeignetes Abbruchkriterium und geben die bis dahin beste Lösung zurück). Welche sind Ihre optimalen Zutaten, und wie viele verschiedene Cocktails können Sie daraus herstellen? *10 Punkte*