Detecting Square Shuffles
GURU Competition, 2018

1 Introduction

We call a string $\mathbf{w}$ a *shuffle* of two strings $\mathbf{x}$ and $\mathbf{y}$ over an alphabet Σ if and only if there are (possibly empty) strings u_i and v_i such that $\mathbf{x} = u_1 u_2 \cdots u_k$, $\mathbf{y} = v_1 v_2 \cdots v_k$, and $\mathbf{w} = u_1 v_1 u_2 v_2 \cdots u_k v_k$. In other words, $\mathbf{w}$ is a shuffle if and only if $\mathbf{w}$ is a permutation of the concatenation of $\mathbf{x}$ and $\mathbf{y}$ such that the order of the elements in $\mathbf{x}$ and the order of the elements in $\mathbf{y}$ are preserved. Intuitively, the concept of a shuffle corresponds to the action of shuffling two decks of cards via the typical riffle method.

As an example, consider $\mathbf{x} = [x_1, x_2, x_3]$ and $\mathbf{y} = [y_1, y_2]$. Then:

$\mathbf{z} = [x_1, x_2, y_1, y_2, x_3]$ is a shuffle of $\mathbf{x}$ and $\mathbf{y}$.

$\mathbf{z}' = [x_1, x_2, y_1, x_3]$ is NOT a shuffle of $\mathbf{x}$ and $\mathbf{y}$.

$\mathbf{z}'' = [x_1, x_2, y_2, y_1, x_3]$ is NOT a shuffle of $\mathbf{x}$ and $\mathbf{y}$.

Shuffles are also called *merges* or *interleavings*.

We denote the set of all possible shuffles of two strings $\mathbf{x}$ and $\mathbf{y}$ as $\mathbf{x} \odot \mathbf{y}$. Thus if a string $\mathbf{w}$ is a shuffle of $\mathbf{x}$ and $\mathbf{y}$, we say $\mathbf{w} \in \mathbf{x} \odot \mathbf{y}$; and if $\mathbf{w}$ is NOT a shuffle of $\mathbf{x}$ and $\mathbf{y}$, we say $\mathbf{w} \notin \mathbf{x} \odot \mathbf{y}$.

We define a *square shuffle* as a string $\mathbf{w}$ where for some string $\mathbf{u}$, we have $\mathbf{w} \in \mathbf{u} \odot \mathbf{u}$; that is, $\mathbf{w}$ is a shuffle of some string $\mathbf{u}$ with itself.

2 Problem Statement

Create a program that, given a string $\mathbf{w}$ as input, determines whether or not $\mathbf{w}$ is a square shuffle.

3 Proposed Pseudocode

```
input: a string ws
output: Bool

1  if len(ws) is odd, DO
2      return False
3  else, DO
4      indws <- [0 .. len(ws)-1]
5      for each possible combination of len(ws)/2 elements from indws, DO
6          xs <- [], ys <- []
7          for each i in indws, DO
8              if i is an element of the combination, DO
9                  xs = xs ++ [ ws[i] ]
10             else, DO
11                 ys = ys ++ [ ws[i] ]
12             if xs == ys, DO
13                 return True, xs
14 return False
```

4 Sample Solution in Python

```
# not very fast when len(ws) > 25ish
from itertools import combinations

1  def squareCheck(ws):
2      if len(ws)%2 == 0:
3          indws = list(range(len(ws)))
4          for combo in combinations(indws, int(len(ws)/2)):
5              xs = []
6              ys = []
7              for ind in indws:
8                  if ind in set(combo):
9                      xs.append(ws[ind])
10                 else:
11                     ys.append(ws[ind])
12             if xs == ys:
13                 return True, xs
14     return False
```

5 Sample I/O

```
Detects lists of odd length
"0" FALSE
"sugarsuga" FALSE

Detects lists concatenated with themselves
"primeprime" TRUE
"fortunefortune" TRUE

Detects simple interleaavings
"bbaananan ana babatt" TRUE
"nunuttttererbubuttttterer" TRUE

Detects very small nonsquare shuffles
"01" FALSE
"10" FALSE

Detects very small square shuffles
"00" TRUE
"01" TRUE

Robust against repeated characters
"0111100011011110001" TRUE
"1110111110011111101" TRUE
"1101111111110111101" FALSE
"nnuutttteebrurtterbutter" FALSE
```