```
# -*- coding: utf-8 -*-
"""Untitled1.ipynb

Automatically generated by Colaboratory.
Collarborated with Vibhav Joshi, vibjoshi

Original file is located at
    https://colab.research.google.com/drive/1UCLpyUeFgNL6ZnliMDAEMiu_pOlRKqVN
"""

import copy

# problem definition

states = ("R", "S")
trans = {"R": {"R": 0.65, "S": 0.35},
         "S": {"R": 0.25, "S": 0.75}}
emission = {"R": {"Y": 0.8, "N": 0.2},
            "S": {"Y": 0.2, "N": 0.8}}
initial = {"R": 0.5, "S": 0.5}
observed = ["Y", "N", "Y", "Y", "Y", "Y", "N"]
print("%40s: %s " % ("observed sequence", str(observed)))
N = len(observed)
############################
# We can find the most likely state sequence in a brute-force way by just
# trying all of them!
joint = {}


for s0 in states:
      for s1 in states:
            for s2 in states:
                  for s3 in states:
                        for s4 in states:
                              for s5 in states:
                                    for s6 in states:
                                                p0 = initial[s0] * emission[s0]
[observed[0]]
                                                p1 = p0 * trans[s0][s1] *
emission[s1][observed[1]]
                                                p2 = p1 * trans[s1][s2] *
emission[s2][observed[2]]
                                                p3 = p2 * trans[s2][s3] *
emission[s3][observed[3]]
                                                p4 = p3 * trans[s3][s4] *
emission[s4][observed[4]]
                                                p5 = p4 * trans[s4][s5] *
emission[s5][observed[5]]
                                                ans = p5 * trans[s5][s6] *
emission[s6][observed[6]]
                                                seq = str([s0, s1, s2, s3, s4, s5,
s6])
                                                joint[seq] = ans

print("%40s: %s" % ("Most likely sequence by brute force:", str(max(joint,

key=joint.get))))
```

```
def viterbi(observed, states, initial, trans, emission):
    path = {s: [] for s in states}
    current = {}
    for s in states:
        current[s] = initial[s] * emission[s][observed[0]]
    for i in range(1, len(observed)):
        lastrans = current
        current = {}
        for curr_state in states:
            probability, last_sta = max(
                ((lastrans[last_state] * trans[last_state][curr_state] *
emission[curr_state][observed[i]], last_state)
                 for last_state in states))
            current[curr_state] = probability
            path[curr_state].append(last_sta)


    probability = -1
    optimal = None
    for s in states:
        path[s].append(s)
        if current[s] > probability:
            optimal = path[s]
            probability = current[s]

    return optimal


##############################
# obviously that's a big mess, and slow -- each every day requires another nested
# loop and 2x the computation time.
# so instead, compute using Viterbi!
# Viterbi table will have two rows and N columns
V_table = {"R": [0] * N, "S": [0] * N}
# Here you'll have a loop to build up the viterbi table, left to right
# Here you'll have a loop that backtracks to find the most likely state sequence
viterbi_seq = viterbi(observed, states, initial, trans, emission)
print("%40s: %s" % ("Most likely sequence by Viterbi:", str(viterbi_seq)))
```