from math import log as ln

import itertools

def allpaths(T, K):

return itertools.product(xrange(K), repeat=T)

def plus1(L): return [x+1 for x in L]

def minus1(L): return [x-1 for x in L]

###########

def logjointprob(Z, X, A, phi, prior):

Z = minus1(Z)

X = minus1(X)

return sum([ln(prior[Z[0]]), ln(phi[X[0]][Z[0]])] +

[ln(A[Z[e]][Z[e-1]]) + ln(phi[X[e]][Z[e]])

for e in xrange(1, len(Z))] +

[ln(A[2][Z[-1]])])

def exhaustive\_bestpath(X, A, phi, prior):

X = minus1(X)

T = len(X)

K = len(phi[0])

allpt = allpaths(T, K)

vals = [(logjointprob(plus1(path), plus1(X), A, phi, prior), path)

for path in allpt]

max\_value, max\_path = max(vals)

return plus1(max\_path)

def viterbi\_bestpath(X, A, phi, prior):

V = [{}] # V[t][z], DP matrix

path = {} # Path memo to avoid backtracking

states = xrange(2)

X = minus1(X)

# Base cases (t == 0)

for z in states:

V[0][z] = prior[z] \* phi[X[0]][z]

path[z] = [z]

# t > 0

for t in xrange(1, len(X)):

V.append({})

updated\_path = {}

for z in states:

(prob, state) = max([(phi[X[t]][z] \* A[z][zarg] \* V[t-1][zarg],

zarg) for zarg in states])

V[t][z] = prob

updated\_path[z] = path[state] + [z]

path = updated\_path

# Get max from last row of V

(prob, state) = max([(V[len(X)-1][z], z) for z in states])

return plus1(path[state])

if \_\_name\_\_ == '\_\_main\_\_':

from params\_and\_data import \*

print exhaustive\_bestpath(smallX, A, phi, prior)

print exhaustive\_bestpath(smallX2, A, phi, prior)

print viterbi\_bestpath(bigX, A, phi, prior)