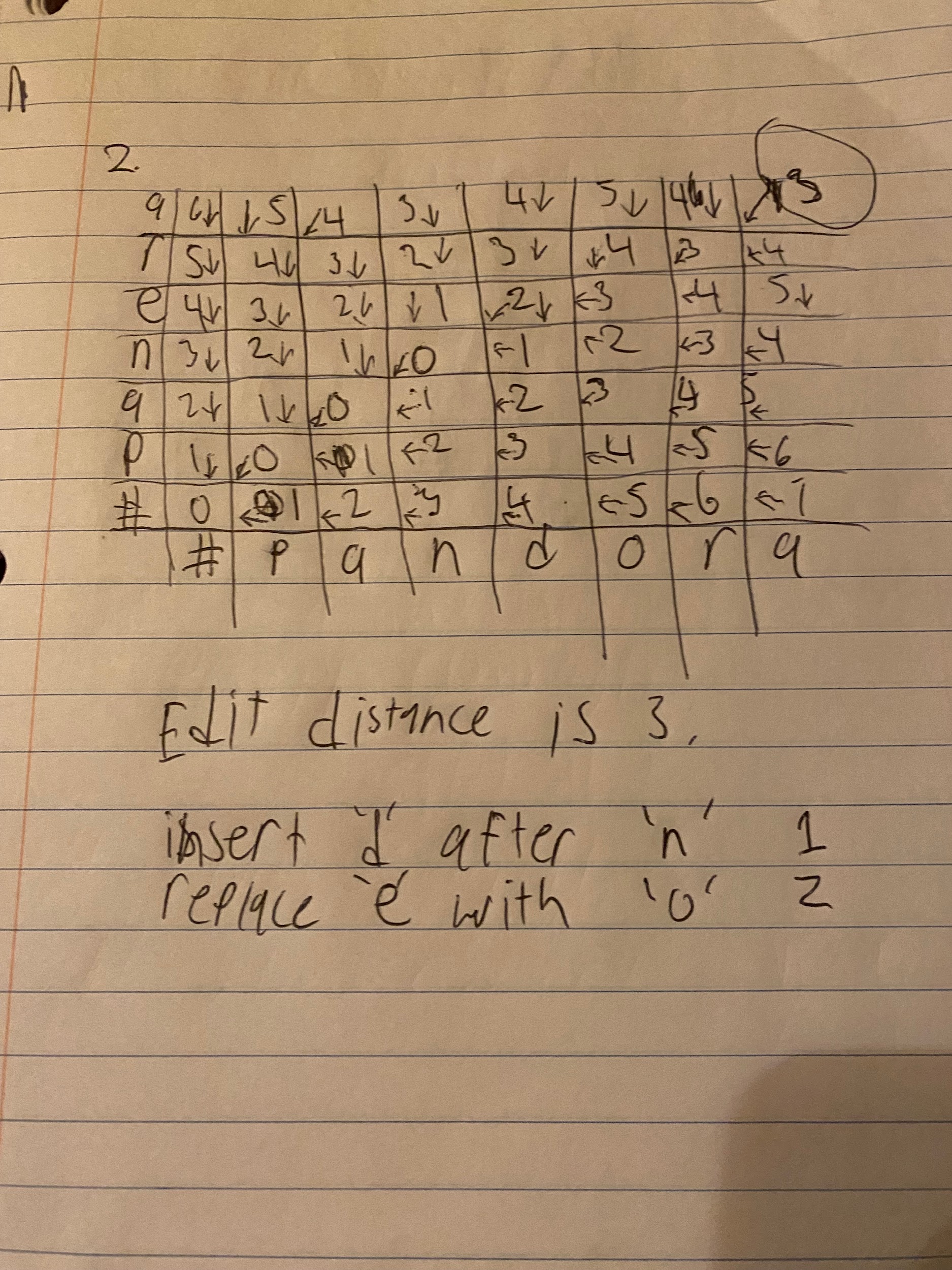
1. \b(\w+)\s+\1\b
2. ^\d+\b.\*\b\w+$
3. /

Drive -> Divers

Delete r, insert r and s

1 deletion, 2 insertion

Total Cost: 3

Drive -> brief

Substitute d with b, delete v, insert f

2 substitution, 1 deletion, 1 insertion

Total Cost: 4

Thus, drive is closer to divers

import numpy as np

from collections import defaultdict

# Given paragraph (I just tokenized and manually POS tagged for simplicity)

tagged\_corpus = [

('NVIDIA', 'NNP'), ('Corporation', 'NNP'), ('is', 'VBZ'), ('a', 'DT'), ('market', 'NN'), ('leader', 'NN'),

('among', 'IN'), ('technology', 'NN'), ('companies', 'NNS'), ('specializing', 'VBG'), ('in', 'IN'),

('graphics', 'NNS'), ('processing', 'NN'), ('units', 'NNS'), ('(', 'SYM'), ('GPUs', 'NNS'), (')', 'SYM'),

('and', 'CC'), ('artificial', 'JJ'), ('intelligence', 'NN'), ('(', 'SYM'), ('AI', 'NNP'), (')', 'SYM'),

('The', 'DT'), ('company', 'NN'), ('is', 'VBZ'), ('renowned', 'VBN'), ('for', 'IN'), ('its', 'PRP$'),

('innovative', 'JJ'), ('approaches', 'NNS'), ('to', 'TO'), ('both', 'DT'), ('hardware', 'NN'),

('and', 'CC'), ('software', 'NN'), ('solutions', 'NNS'), ('.', '.'), ('It', 'PRP'), ('reported', 'VBD'),

('strong', 'JJ'), ('earnings', 'NNS'), ('in', 'IN'), ('the', 'DT'), ('most', 'RBS'), ('recent', 'JJ'),

('quarter', 'NN'), ('.', '.'), ('It', 'PRP'), ('showcased', 'VBD'), ('robust', 'JJ'), ('revenue', 'NN'),

('growth', 'NN'), (',', ','), ('driven', 'VBN'), ('primarily', 'RB'), ('by', 'IN'), ('increasing', 'VBG'),

('demand', 'NN'), ('for', 'IN'), ('its', 'PRP$'), ('GPUs', 'NNS'), ('across', 'IN'), ('various', 'JJ'),

('sectors', 'NNS'), (',', ','), ('including', 'VBG'), ('gaming', 'NN'), ('data', 'NN'), ('centers', 'NNS'),

('and', 'CC'), ('professional', 'JJ'), ('visualization', 'NN'), ('.', '.'),

]

# Hopefully this will compute the transition probabilities

tag\_counts = defaultdict(int)

transition\_counts = defaultdict(lambda: defaultdict(int))

emission\_counts = defaultdict(lambda: defaultdict(int))

for i in range(len(tagged\_corpus) - 1):

curr\_word, curr\_tag = tagged\_corpus[i]

next\_word, next\_tag = tagged\_corpus[i + 1]

tag\_counts[curr\_tag] += 1

transition\_counts[curr\_tag][next\_tag] += 1

emission\_counts[curr\_tag][curr\_word] += 1

tag\_counts[tagged\_corpus[-1][1]] += 1 # Last word's tag

emission\_counts[tagged\_corpus[-1][1]][tagged\_corpus[-1][0]] += 1

def compute\_probabilities(counts):

probabilities = {}

for tag, next\_tags in counts.items():

total = sum(next\_tags.values())

probabilities[tag] = {k: v / total for k, v in next\_tags.items()}

return probabilities

transition\_probs = compute\_probabilities(transition\_counts)

emission\_probs = compute\_probabilities(emission\_counts)

# Viterbi Algorithm, don’t touch this

def viterbi(sentence, transition\_probs, emission\_probs, tag\_counts):

states = list(tag\_counts.keys())

V = [{}]

path = {}

# Initialize base cases

for state in states:

V[0][state] = emission\_probs[state].get(sentence[0], 1e-6) \* (tag\_counts[state] / sum(tag\_counts.values()))

path[state] = [state]

# run the algorithm??

for t in range(1, len(sentence)):

V.append({})

new\_path = {}

for curr\_state in states:

(prob, state) = max((V[t-1][prev\_state] \* transition\_probs[prev\_state].get(curr\_state, 1e-6) \*

emission\_probs[curr\_state].get(sentence[t], 1e-6), prev\_state)

for prev\_state in states)

V[t][curr\_state] = prob

new\_path[curr\_state] = path[state] + [curr\_state]

path = new\_path

# I hope this returns the best path

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

return path[state]

# Finally! Just test the sentence now

test\_sentence = ['NVIDIA', 'is', 'a', 'leader', 'among', 'technology', 'companies']

predicted\_tags = viterbi(test\_sentence, transition\_probs, emission\_probs, tag\_counts)

print("Predicted POS Tags:", list(zip(test\_sentence, predicted\_tags)))

1. 