Haluk Dogan 66063720
ELEC865 — Spring 2014 **Homework 1** Due: January 31, 2014

Problem 3

(a)

$$\begin{aligned}
H &= -\sum_{i=1}^{4} P(i)\log_2 P(i) \\
&= -\left(\frac{1}{4}\log_2\frac{1}{4}+\frac{1}{4}\log_2\frac{1}{4}+\frac{1}{4}\log_2\frac{1}{4}+\frac{1}{4}\log_2\frac{1}{4}\right) \\
&= -4\left(\frac{1}{4}\log_2\frac{1}{4}\right) \\
&= -1(\log_2 1-\log_2 4) \\
&= -1(0-\log_2 2^2) \\
&= -1(0-2\log_2 2) \\
&= -1(-2) \\
&= 2
\end{aligned}$$

(b)

$$\begin{aligned}
H &= -\sum_{i=1}^{4} P(i)\log_2 P(i) \\
&= -\left(\frac{1}{2}\log_2\frac{1}{2}+\frac{1}{4}\log_2\frac{1}{4}+\frac{1}{8}\log_2\frac{1}{8}+\frac{1}{8}\log_2\frac{1}{8}\right) \\
&= -\left(\frac{1}{2}(\log_2 1-\log_2 2)+\frac{1}{4}(\log_2 1-\log_2 4)+\frac{1}{8}(\log_2 1-\log_2 8)+\frac{1}{8}(\log_2 1-\log_2 8)\right) \\
&= -\left(\frac{1}{2}(\log_2 1-\log_2 2)+\frac{1}{4}(\log_2 1-\log_2 2^2)+\frac{1}{8}(\log_2 1-\log_2 2^3)+\frac{1}{8}\left(\log_2 1-\log_2 2^3\right)\right) \\
&= -\left(\frac{1}{2}(\log_2 1-\log_2 2)+\frac{1}{4}(\log_2 1-2\log_2 2)+\frac{1}{8}(\log_2 1-3\log_2 2)+\frac{1}{8}(\log_2 1-3\log_2 2)\right) \\
&= -\left(-\frac{1}{2}-\frac{2}{4}-\frac{3}{8}-\frac{3}{8}\right) \\
&= \frac{14}{8} \\
&= 1.75
\end{aligned}$$

(c)

$$\begin{aligned} H &= -\sum_{i=1}^{4} P(i) \log_2 P(i) \\ &= -\left(0.505 \log_2 0.505 + \frac{1}{4} \log_2 \frac{1}{4} + \frac{1}{8} \log_2 \frac{1}{8} + 0.12 \log_2 0.12\right) \\ &= 1.74 \end{aligned}$$

Problem 7

(a) Generation of words by random model is the fastest method among all four different methods. However, a few of generated words are in the English language. Since random model doesn't give us much information about relations of letters, this is result is plausible.

(b) This proposed model improves the achievement of generation words with randomness by incorporating cumulative distribution function (CDF). CDF values give us insight how many of each letter is occurred. CDF values express the likelihood of the occurrence of a given letter, so the higher probability is the more occurrence of letters. In the following Table 1 shows us CDF values of each letter.

(c) Although CDF values two letters are high, it doesn't mean that they occur one after the other. Single-letter context model gives more information than two preceding models. In this model, we determine how many times letter i comes after letter j to be able to obtain more meaningful outcomes. Indeed, its result is much better than the result of CDF. In order to start generation of words, we first pick a random letter by satisfying the given condition ($F_X(x_k - 1) \leq r < F_X(x_k)$). Then we choose the next letter according to new CDF values which contain the information about the number of occurrence letter i comes after letter j.

(d) Two-letter context model has the same idea as single-letter counterpart. As expected, it gives more information than the single-letter context model. For instance, obtaining a letter i after jj has the lower probability than obtaining a letter i after j. The drawback of this model is its running time performance which is much slower than of single-letter context model. However, its achievement rate is higher than of single-letter context model. To be able to start word generation, we first pick 2 letters from the CDF model. We didn't pick the second letter from the single-letter context model, because we want to compare their achievement by taking the CDF model as a base. Since some letters i's are never occurred after jj's, we restricted the generation process with n trials. If 4-letter word can not be generated after n trials, we start the generation process from the scratch.

In the following Table 2 shows the number of words that are in the English language that each model could find. In Table 3 shows their running time in seconds.

Letter	Cumulative Frequency	Cumulative Relative Frequency	Frequency	Relative Frequency
a	852	0.099	852	0.099
b	1099	0.128	247	0.029
c	1350	0.157	251	0.029
d	1689	0.196	339	0.039
e	2538	0.295	849	0.099
f	2731	0.318	193	0.022
g	2944	0.342	213	0.025
h	3214	0.374	270	0.031
i	3704	0.431	490	0.057
j	3758	0.437	54	0.006
k	3989	0.464	231	0.027
l	4583	0.533	594	0.069
m	4869	0.566	286	0.033
n	5330	0.620	461	0.054
o	5988	0.697	658	0.077
p	6263	0.729	275	0.032
q	6272	0.730	9	0.001
r	6769	0.787	497	0.058
s	7210	0.839	441	0.051
t	7694	0.895	484	0.056
u	8042	0.936	348	0.040
v	8150	0.948	108	0.013
w	8339	0.970	189	0.022
x	8372	0.974	33	0.004
y	8553	0.995	181	0.021
z	8596	1.000	43	0.005

Table 1: Cumulative Frequencies of English Alphabet.

Random	CDF Model	Single Letter	Two Letter
2	11	26	35

Table 2: Achievement Rates Different Methods To Obtain An English Word In Terms Of Number Of Words.

Random	CDF Model	Single Letter	Two Letter
0.0004	0.0107	5.89	499.63

Table 3: Runtime Performance Of Different Methods.

Problem 8

(a) Consider C = $\{0, 01, 11, 111\}$. Applying the Sardinas-Patterson algorithm we get:

$$C_1 = \{1\}$$
$$C_2 = \{11\}$$
$$C_2 \cap C = \{11\}$$

So the code C is not uniquely decodable.

(b) Consider C = $\{0, 01, 110, 111\}$. Applying the Sardinas-Patterson algorithm we get:

$$C_1 = \{1\}$$
$$C_2 = \{10, 11\}$$
$$C_3 = \{0\}$$
$$C_3 \cap C = \{0\}$$

So the code C is not uniquely decodable.

(c) Consider C = $\{0, 10, 110, 111\}$. Applying the Sardinas-Patterson algorithm we get:

$$C_1 = \{\}$$
$$C_3 \cap C = \{\}$$

So the code C is uniquely decodable.

(d) Consider C = $\{1, 10, 110, 111\}$. Applying the Sardinas-Patterson algorithm we get:

$$C_1 = \{0, 10, 11\}$$
$$C_2 = \{1\}$$
$$C_2 \cap C = \{1\}$$

So the code C is not uniquely decodable.

Problem 7 - Source Code

```
import random
import string
import re
import time

from bisect import bisect
from collections import defaultdict
from itertools import product
```

```
from sys import argv
from fileinput import input

def randomStr(l):
        return ''.join(random.choice(string.ascii_lowercase)
          ↪ for x in range(l))

def randomWordSet(n, l):
        return "-".join(randomStr(l) for x in range(n))

def occurrenceTable(file):
        freq_table = defaultdict(lambda : 0)
        total_char = 0.0

        with open(file) as f:
                for word in f:
                        word = word.strip().lower()
                        freq_table.update((c, freq_table[c] +
                          ↪ word.count(c)) for c in set(word)
                          ↪ )
                        total_char += len(word)

        result = []
        last = 0.0
        for k,v in sorted(freq_table.items()):
                result.append((k, last + v, (last + v) /
                  ↪ total_char, v, v / total_char))
                last += v

        return result

def pickLetter(occurrence_table):
        breakpoints = []
        for e in occurrence_table:
                breakpoints.append(e[2])

        r = random.random()
        idx = bisect(breakpoints, r)
        if breakpoints[idx-1] <= r and r < breakpoints[idx]:
                return string.ascii_lowercase[idx]

def generateWord(n, occurrence_table):
        result = []
```

```
        for i in range(n):
                word = ""
                while len(word) != 4:
                        letter = pickLetter(occurrence_table)
                        if  letter != None:
                                word += letter

                result.append(word)

        return '-'.join(result)

def singleLetterContext(file):
        n = len(string.ascii_lowercase)
        occurrence_table = [[0.0 for x in xrange(n)] for x in
           ↪ xrange(n)]

        with open(file) as f:
                text = f.read().strip().lower()

        contents = input(file)

        pattern = "%s(%s+)"
        for i in string.ascii_lowercase:
                i_idx = string.ascii_lowercase.index(i)
                match1 = re.findall("(?=%s(.))" % i, text)
                total_char = float(len(match1))
                last = 0.0
                for j in string.ascii_lowercase:
                        j_idx = string.ascii_lowercase.index(j
                           ↪ )
                        match2 = re.findall(pattern % (i, j),
                           ↪ text)
                        if total_char != 0:
                                occurrence_table[i_idx][j_idx]
                                   ↪  = (len(match2) + last) /
                                   ↪  total_char
                                last += len(match2)

        word = ''
        while word == '':
                tmp = pickLetter(occurrenceTable(file))
                if tmp != None:
                        word += tmp
```

```
        while len(word) != 4:
                breakpoints = occurrence_table[string.
                    ↪ ascii_lowercase.index(word[-1])]
                r = random.random()
                idx = bisect(breakpoints, r)
                if idx == len(string.ascii_lowercase):
                        pass
                elif breakpoints[idx-1] <= r and r <
                    ↪ breakpoints[idx]:
                        word += string.ascii_lowercase[idx]

        return word

def callSingleLetterContext(file, n):
        result = []
        for i in range(n):
                result.append(singleLetterContext(file))

        return '-'.join(result)


def twoLetterContext(file, trial):
        n = len(string.ascii_lowercase)
        occurrence_table = [[0.0 for x in xrange(n)] for x in
            ↪ xrange(n*n)]

        with open(file) as f:
                text = f.read().strip().lower()
                total_char = float(len(text.replace('\n', ''))
                    ↪ )

        contents = input(file)

        pattern = "(?=(%s)(%s*))"
        cnt = 0
        for i in string.ascii_lowercase:
                for j in string.ascii_lowercase:
                        match1 = re.findall("(?=%s(.))" % (i+j
                            ↪ ), text)
                        total_char = float(len(match1))
                        last = 0.0
```

```
                    for z in string.ascii_lowercase:
                            z_idx = string.ascii_lowercase
                               ↪ .index(z)
                            match2 = re.findall(pattern %
                               ↪ (i+j, z), text)
                            match2 = filter(lambda x: x !=
                               ↪  '', map(lambda x: x[1],
                               ↪ match2))
                            if total_char != 0:
                                    occurrence_table[cnt][
                                       ↪ z_idx] = (len(
                                       ↪ match2) + last) /
                                       ↪  total_char
                                    last += len(match2)
                    cnt+=1

alph = list(product(string.ascii_lowercase, string.
   ↪ ascii_lowercase))
alph = map(lambda x: ''.join(x), alph)

word = ''
tried = 0

while tried < trial:
        while len(word) < 2:
                tmp = pickLetter(occurrenceTable(file)
                   ↪ )
                if tmp != None:
                        word += tmp

        breakpoints = occurrence_table[alph.index(word
           ↪ [-2:])]
        r = random.random()
        idx = bisect(breakpoints, r)
        if idx == len(string.ascii_lowercase) or idx
           ↪ == 0:
                tried += 1
        elif breakpoints[idx-1] <= r and r <
           ↪ breakpoints[idx]:
                word += string.ascii_lowercase[idx]

        if len(word) == 4:
                return word
```

```
                if tried >= trial:
                        tried = 0
                        word = ''

def callTwoLetterContext(file, n, k):
        result = []
        for i in range(n):
                result.append(twoLetterContext(file, k))

        return '-'.join(result)


def pretty_print(occurrence_table):
        print "%s \t %s \t %s \t %s \t %s" % ("Letter", "
          ↪ Cumulative Frequency", \
                "Cumulative Relative Frequency", "Frequency",
                  ↪ "Relative Frequency")

        for e in occurrence_table:
                print "%s \t %d \t %0.3f \t %d \t %0.3f \t" \
                % (e[0], e[1], e[2], e[3], e[4])

if __name__ == "__main__":

        start_time = time.time()
        print "Random Model:"
        print "============="
        print randomWordSet(100, 4)
        print time.time() - start_time, "seconds"

        start_time = time.time()
        print "CDF Model:"
        print "=========="
        occurrence_table = occurrenceTable(argv[1])
        print generateWord(100, occurrence_table)
        print time.time() - start_time, "seconds"


        start_time = time.time()
        print "Single Letter Context:"
        print "======================"
        print callSingleLetterContext(argv[1], 100)
```

```
        print time.time() - start_time, "seconds"

        start_time = time.time()
        print "Two Letter Context:"
        print "======================"
        print callTwoLetterContext(argv[1], 100, 10)
        print time.time() - start_time, "seconds"
```

1 Appendix

Source code can be found in this URL: `https://github.com/haluk/data-compression`

```
Random Model:
=============
ghlh-leis-vnqs-qmsd-wjzl-pihk-xcmj-vowp-urdl-qpru-tpny
apwl-upkx-paud-kzpl-puci-cadr-eovb-lxwe-letp-pcwv-xted
bzko-dfsz-nhnk-gxir-mjet-ueby-bmna-pirn-llcw-dvrt-ydmh
snlx-bpqs-tzlq-gkwy-czdz-hwqs-jnsl-exnz-dfkb-kduv-ypkf
bfjn-ooiq-jfbq-kpoq-lles-dwxe-qyct-toby-aztw-dsry-onrb
lvwb-nodv-cysy-bpsc-ihap-zqco-wqwu-khaj-qbqh-ckon-lkfh
ucdh-nvht-rsid-hmkw-jygn-ugks-hwip-afim-ircw-tkvn-meob
jvgb-ublj-dlbl-pnyp-ogbv-qeou-sfmf-nkzs-lmhg-sjra-oequ
honm-nxtz-atpm-ihdi-qoin-jbbq-pusp-vicl-cszw-cpvf-uusq-xlxy

CDF Model:
==========
zldg-geth-lloe-lveg-gdey-ddbi-ello-rolz-glfi-rine-tenr
tycg-sbos-jexu-ytos-uuli-koeu-skim-nfpl-slhl-hlst-sytn
lhzp-myep-hehu-pfmh-enrw-enel-obnr-idui-revl-lpcw-duks
ului-koos-dooe-mvus-endg-swje-drfs-ofet-ckcn-nolt-wnyr
illy-sohe-uers-rvst-hste-wymt-ulnu-sopl-delf-seot-bsed
dhog-eolr-oosf-euut-lrke-lplt-snsb-mcrr-irkw-slkl-lgol
eyko-eoro-eylr-elsv-otnu-upps-elcs-jove-cpdl-sicd-lcsc
sdrn-uuss-byer-oeqe-tcwt-uxlu-ptse-csmk-reud-npoc-fuft
dpwo-elst-gkil-etti-xfel-stnn-meet-elst-ttum-lomu-remi-wfee

Single Letter Context:
======================
loon-umpi-orch-jull-icus-pere-tthe-ling-oled-geri-regy
slel-rynk-irok-drtc-bilo-este-cesn-cken-geld-teto-lffo
yesh-pelo-sken-nthr-soke-mees-olol-peco-icok-ulep-rysh
```

```
ingr-exon-ptei-oflu-crur-hope-heto-summ-ssnt-yono-obog
celf-ucok-rive-dile-onem-urip-euti-yprk-bese-sito-poel
ckir-leli-troo-nere-egoi-stol-etim-dese-skic-eren-yoro
hele-eenk-kint-iloo-liso-blte-otey-selt-bero-eyee-ufos
esck-onon-lued-shol-oubu-pool-ypol-ntur-eelk-odoc-tusl
okew-woor-uter-sene-romb-thol-lyso-ussi-ebur-okef-nyun-mond

Two Letter Context:
===================
user-iski-edgy-iwic-cmen-eudy-shor-weet-twiv-dodd-ilto
lend-ebbe-bree-selt-hunt-yell-orty-iusy-iush-swis-elly
iwit-psoo-sudo-ueto-ooke-elso-ient-oope-styx-esti-hird
goft-ieve-eire-ecky-oeye-uedg-rmyr-gwel-rewy-gild-ynne
pres-yero-klur-uspi-nump-seep-drun-ooke-eont-ocky-eyst
estu-polk-inyx-coin-herr-xity-tone-crew-cobe-lure-clil
feen-ucho-iker-eude-hyde-effy-uell-bome-yste-loss-plee
serf-lpso-uxen-oilt-emon-fnix-uezr-sure-eong-eily-nill
ldye-ment-frop-oree-ieub-ikee-sluk-ntem-flux-oped-edir-klum
```