last name **first names** **id#**

---

cmput 210 W2016 **asn 1** **due 3:30 Thu Feb 4**

You can work in groups, but hand in only 1 assignment per group. Hand in the answers from each page on a separate page: page 1 answers on one page (with name and id), page 2 answers on another page (with name and id), etc.

1. **keyphrase generation** A *key* for the substitution cipher is a permutation of the alphabet. One way to generate keys is the *keyphrase* method:

   - pick a keyphrase, say `doctor suess`
   - drop spaces, then after keyphrase append the alphabet, so
     `doctorsuessabcdefghijklmnopqrstuvwxyz`
   - from start of resulting string, remove duplicate letters, so
     `doctrsueabfghijklmnpqvwxyz`

   One variant is to rotate the alphabet so that it starts with the letter that follows the last keyphrase letter, so `doctorsuesstuvwxyzabcdefghijklmnopqrs`, which gives key

   `doctrsuevwxyzabfghijklmnpq`.

   Which is more secure, the original, or the variant? Why?

   (You can run program keyphrase.py to try both variations, and then use linux `tr` command to generate some ciphertext).

2. **smoothing english** Cracking English substitution ciphers is helped by the uneven distribution of letter frequency in English text. One way to smooth this distribution is to sacrifice rarer letters, e.g. j, and then use the sacrificed letters as extra symbols for homophones, e.g. eliminate j, then use both e and j as homophones for e.

   (i) Below is some smoothed text. Guess the plaintext, and guess the smoothing function (i.e. smoothing algorithm).

   ```
   simplq smoozhing introduces spxlling qrrors.

   humjns cjn ofzen corrqcz thxsx qrrors qjsier zhan

   spxcial purpose cryptogrjm crackxrs such js kuipkuip
   ```

   (ii) Below is more text. The smoothing function is the same, but now I have added a Caesar shift. Guess the original plaintext. (You can run the program caesar.py to generate all Caesar shifts.)

   cwyyjrsxq mywlsxan gsjr celcjsdedsyx wtuhc mbkmusxq rkbnob lej xyd swzyccslvo csxmh rsqr pbouehxmi nsqbtwc mkx cjsvv lo nhjomjan

   (iii) Below is ciphertext: first I smoothed the plaintext using the same function from (ii), then I used a keyphrase cipher. Crack the ciphertext. Give the plaintext and the keyphrase.

   Hint: The keyphrase is a person. The plaintext mentions a cryptolit author.

   ```
   qifms aggae pjy uyatqi rot sqaiqst djs znmst xocr rot

   alogocz cj ksakb crqos tvltuouvuoje kopryst, lqdjsn doeaggz

   snwqagoef rot hqurji oe crn fjgi lvf
   ```

last name **first names** **id#**

cmput 210 W2016 **asn 1** **due 3:30 Thu Feb 4**2

3. **index of coincidence** Here are two baskets of letters:

```
{ a, a, a, a, b, b, c, c, c, c }
{ a, a, b, b, c, c, d, d, e, e }
```

Give the index of coincidence if we pick (i) two letters from basket 1 (ii) two letters from basket 2 (iii) one letter from each basket. (iv) one letter from basket 1, and one letter from typical English. (v) two letters from typical English.

For (i-iv), show your work.

4. **cracking Vigenere** Here is some Vigenere ciphertext, with some substring frequency data below.

(i) There is only one repeated trigram: wol, with offset 96. Using Babbage's method, give possible keyword lengths.

(ii) Based on substring frequencies, which is the more likely keylength: 7 or 8? Justify briefly.

(iii) Assume the keylength is 8. Examine the freqencies of the 3rd substring. What do you think is the most likely shift character here? Justify briefly.

(iv) Either by hand, or with the help of program friedman.py, crack the cipher: give keyphrase and plaintext.

```
lrv pzvgplqz azrmio qgsdenc hwol nhcvwsn flbders jr siu

eyogceieew siktngwt kjroet assolpr ulp ogwz dkwninwpd

evjdosrrctsm rmeh jmxvzv lliscwolxie rzhzw zn dempvkps

oenvdrps cvp qjrdificsy fj mcrj hj fp tji qwmwe cqqaioic ptsrfvqd

length 7 substring frequencies
 a  b  c  d  e  f  g  h  i  j  k  l  m  n  o  p  q  r  s  t  u  v  w  x  y  z
 0  0  0  1  2  1  0  0  1  3  0  1  2  2  2  3  1  2  2  1  0  0  3  1  0  1 29 0.0357
 1  0  1  2  0  0  1  2  1  0  0  2  0  1  0  2  2  5  3  2  1  1  0  0  1  0 28 0.0485
 1  1  3  2  1  1  0  0  1  1  2  1  1  0  2  0  1  1  4  0  0  1  2  0  0  2 28 0.0358
 0  0  1  1  4  1  0  2  3  2  1  0  2  0  1  2  1  2  1  0  0  2  1  0  0  1 28 0.0383
 1  0  1  0  1  2  1  1  1  0  0  2  1  2  3  2  0  1  0  1  1  2  3  0  0  2 28 0.0307
 0  0  1  0  3  0  1  0  3  2  0  2  0  1  2  1  0  1  2  2  0  4  1  1  0  1 28 0.0434
 0  0  3  3  2  0  2  0  2  1  1  1  1  2  0  1  2  2  2  0  0  0  1  0  1  1 28 0.0332
0.0379
length 8 substring frequencies
 a  b  c  d  e  f  g  h  i  j  k  l  m  n  o  p  q  r  s  t  u  v  w  x  y  z
 1  0  2  0  0  0  0  1  0  2  0  2  1  2  1  2  1  2  2  1  0  0  2  1  1  1 25 0.0256
 0  0  0  1  0  1  0  2  2  0  1  0  1  0  2  1  3  1  5  0  0  2  3  0  0  0 25 0.0641
 0  0  0  1  0  0  3  0  1  4  0  0  1  2  5  0  0  1  0  0  0  3  0  0  1  3 25 0.0833
 1  0  1  0  0  3  0  0  1  0  2  3  1  0  0  1  1  4  2  0  0  1  4  0  0  0 25 0.0641
 0  0  1  3  3  0  0  0  0  1  0  2  0  1  1  5  0  1  1  1  0  0  0  1  0  4 25 0.0737
 0  1  1  2  2  0  0  2  4  0  0  1  1  2  0  1  0  3  2  1  0  1  0  0  0  0 24 0.0487
 0  0  4  2  3  1  2  0  1  2  1  0  1  1  1  0  1  0  0  2  2  0  0  0  0  0 24 0.0487
 1  0  1  0  5  0  0  0  3  0  0  1  1  0  0  1  1  2  2  1  0  3  2  0  0  0 24 0.0661
0.0593
```