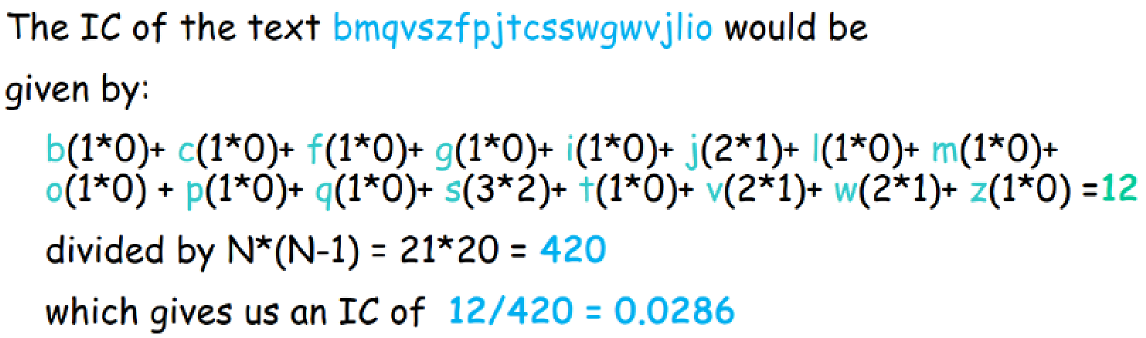
Lezione3

Breaking Vigenére cipher

Recovering the length of the key

This is done with the Friedman method:
Using the index of coincidence:
$I_c(x) = \frac{\sum_{i=1}^{26} f_i(f_i-1)}{n(n-1)} \approx \sum_{i=1}^{26} p_i^2$
Where:

- n is the length of the text.
- f_i is the frequency of the i-th letter (i.e. the number of times it occurs in a text)
- p_i is the probability of the i-th letter, $p_i = \frac{f_i}{n}$, this gives the probability that two letters, randomly chosen from a text, are the same.



The IC of the text bmqvszfpjtcsswgwvjlio would be given by:

b(1*0)+ c(1*0)+ f(1*0)+ g(1*0)+ i(1*0)+ j(2*1)+ l(1*0)+ m(1*0)+
o(1*0) + p(1*0)+ q(1*0)+ s(3*2)+ t(1*0)+ v(2*1)+ w(2*1)+ z(1*0) =12

divided by N*(N-1) = 21*20 = 420

which gives us an IC of 12/420 = 0.0286

The value of the index of coincidence can range from $\frac{1}{26}$ (if the letters have the same probability, basically random text) to 1 (single letter text e.g. AAAAA).
After computing the I_c, if the value is ≈ 0.038, we're trying to break a poly alphabetic cipher; if the value is ≈ 0.065, we're trying to break a mono alphabetic cipher. (I believe this is assuming we're using the English language).

```
m = 1
LIMIT=0.06 # this is to check that ICs are above 0.06 and thus close to
0.065
found = False
while(not found):
    sub = subciphers() # takes the m subciphertexts sub[m] obtained by
selecting one letter every m
    found = True
    for i in range(0,m): # compute the Ic of all subtexts
        if Ic(sub[i]) < LIMIT:
            # if one of the Ic is not as expected try to increase length
            found = False
            m += 1
            break
# survived the check, all Ic's are above LIMIT
output(m)
```

Once we find a value close to the original I_c, we can try to recover the original key.

Recovering the key

Using the mutual index of coincidence:
$MI_c(x,x') = \frac{\sum_{i=1}^{26} f_i f_i'}{nn'} = \sum_{i=1}^{26} p_i p_i'$
where x and x' are two subtexts (two sub ciphers basically).
Given the previous sub ciphers, we can use the mutual index of coincidence to find the shift for each letter, granting us the key.

```
key = [] # empty list
for i in range(0,m): # for any letter of the key
    k = 0      # current relative shift
    mick = 0   # maximum index so far (we start with 0)
    for j in range(0,26): # for any possible relative shift
        # compute the mutual index of coincidence between the first subcipher
        # sub[0] and the i-th subcipher shifted by j
        mic = MIc(sub[0], shift(j,sub[i]))
        if mic > mick: # if it is the biggest so far
            k = j      # we remember the relative shift
            mick = mic # ... and the new maximum
    key.append(k)      # we append to the list the shift we have found
```