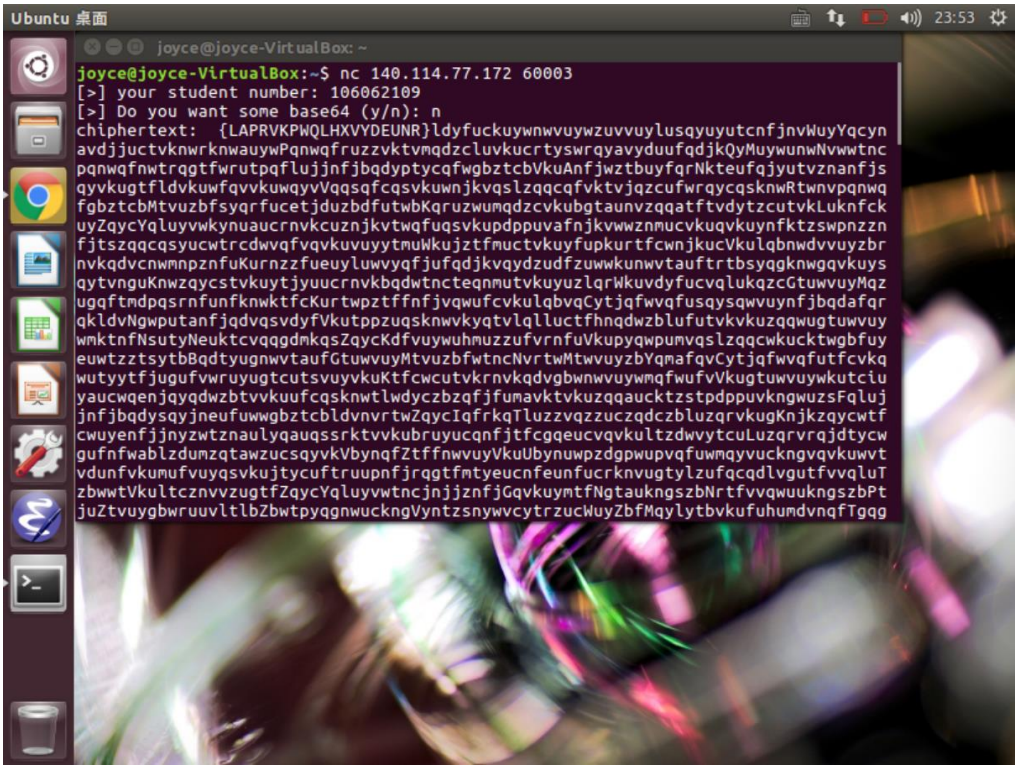
HW2_substitution cipher

● Compile environment

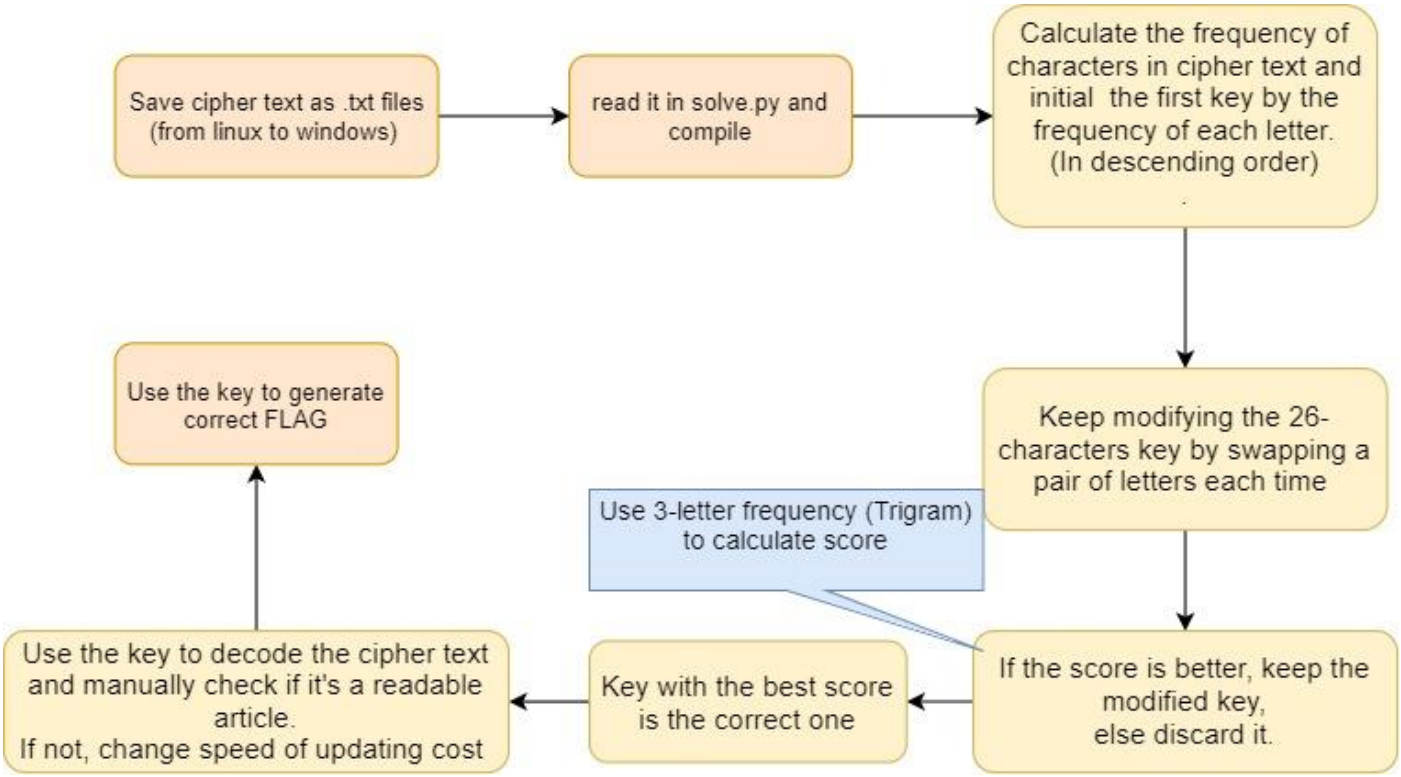
1. Ubuntu virtual box to get the cipher materials



2. Use python 3.6 on my PC with operating system of windows (jupyter notebook)

● Implement process and explanation (Base256)

Main Flowchart :



Detail concepts(decribing code):

```
with open('textfile/ciphertext1.txt') as f:
    cipher = f.read()
    count = 0
    for c in cipher:
        if c.isupper():
            count +=1
    flagg = cipher[1:20]
with open('three_no_spaces_freqs.txt') as ff:
    triscores = {}
    # type(ff) <class '_io.TextIOWrapper'>
    # type(ff.read()) <class 'str'>
    for item in ff:
        (word, score) = item.split()
        triscores[word] = float(score)
```

Open the ciphertext.txt file and create the FLAG between {}

Open the 3-letter-frequency file

https://github.com/juesato/ciphersolver/blob/master/DataGen/3_no_spaces_freqs.txt

and using dict() to create a lookup table

```
def count_frequency(ciphertext):

    LETTERS = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ'
    letters = 'abcdefghijklmnopqrstuvwxyz'
    letterCount_big = {'A': 0, 'B': 0, 'C': 0, 'D': 0, 'E': 0,
     'H': 0, 'I': 0, 'J': 0, 'K': 0, 'L': 0, 'M': 0, 'N': 0, 'O
    letterCount_small = {'a': 0, 'b': 0, 'c': 0, 'd': 0, 'e': 0
     'h': 0, 'i': 0, 'j': 0, 'k': 0, 'l': 0, 'm': 0, 'n': 0, 'o
    for letter in ciphertext:
        if letter in LETTERS:
            letterCount_big[letter] += 1
        elif letter in letters:
            letterCount_small[letter] += 1

    return letterCount_big, letterCount_small
```

Count letters frequency (In uppercase and lowercase)

But afterwards I realize that uppercase is useless (too few of them) so I don't use it anymore.

```
def frequency_table(fre_big, fre_small):

    new_table_big = dict()
    new_table_small = dict()
    new_fre_big = dict()
    new_fre_small = dict()

    table_big = {'E': 12.70, 'T': 9.06, 'A': 8.17, 'O': 7.51, 'I': 6.97, 'N': 6
    'W': 2.36, 'F': 2.23, 'G': 2.02, 'Y': 1.97, 'P': 1.93, 'B': 1.29, 'V': 0.98

    table_small = {'a': 8.167, 'b': 1.492, 'c': 2.782, 'd': 4.253, 'e': 12.702,
    'n': 6.749, 'o': 7.507, 'p': 1.929, 'q': 0.095, 'r': 5.987, 's': 6.327, 't'

    new_table_big = sorted(table_big, key=table_big.get, reverse= True)
    new_table_small = sorted(table_small, key=table_small.get, reverse= True)
    new_fre_big = sorted(fre_big, key=fre_big.get, reverse= True)
    new_fre_small = sorted(fre_small, key=fre_small.get, reverse= True)

    return new_fre_small, new_fre_big
```

Create a frequency table after consuming whole ciphertext.txt.

Sort the frequency in descending order and return

(I only use "new_fre_small" in future steps, others are redundant)

```
def create_first_key(new_fre_small):
    guesskey = ""
    for c in new_fre_small:
        guesskey += c
    return guesskey
```

Use new_fre_small to initial my first key

```
def tri_freq_score(guesskey): # score the higher the better

    guess_plaintext, match = create_article(cipher, guesskey)
    totalcost = 0

    for i in range(len(guess_plaintext)-2):
        curstr = guess_plaintext[i:i+3]
        if triscores.get(curstr):
            totalcost += triscores[curstr]
        else:
            totalcost -= 90.0
    return totalcost
```

Calculate the score of each new-created key.

First, apply the key in decoding the ciphertext, count the 3-letter-word frequency in plaintext,

If we can't find words in lookup table, decrease the score, otherwise increase it.

Finally return the total cost of this key.

BTW: I think for other articles, we only need to modify the speed of adjusting totalcost (marked above) to update the cost. (But I can use same number in this assignment, although with different ciphertexts)

```python
def key_modify(guesskey):
    key_score = tri_freq_score(guesskey)
    Swap = list(itertools.combinations(range(26), 2))
    item = 0
    while len(Swap) > 0:
        new_key = swap(guesskey, Swap[item][0], Swap[item][1])
        new_score = tri_freq_score(new_key)
        if new_score > key_score:
            key_score = new_score
            guesskey = new_key
            item = 0
        else:
            Swap.pop(item)
        item += 1
        if item >= len(Swap):
            item = 0

    for i in range(0,2):
        Swap = list(itertools.combinations(range(26), 2))
        for item in range(0,len(Swap)):
            new_key = swap(guesskey, Swap[item][0], Swap[item][1])
            new_score = tri_freq_score(new_key)
            if new_score > key_score:
                key_score = new_score
                guesskey = new_key
    return guesskey, key_score
```

If the counts are small, it means that the plaintext might not be so “realistic”, we need to change the key

```python
def create_article(article, key):

    original_article = ""
    match = list()
    table = {'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'}

    for i,j in zip(table, key):
        new = (i,j)
        match.append(new)

    for c in article:
        if c == '{' or c == '}':
            continue
        for pair in match:
            if pair[1] == c:
                original_article = original_article + pair[0]

    return original_article, match
```

In convenience, I zipped the key with a,b,c,d,….z as a lookup table, and use the table(match) to decode the cipher text into plain text.

```python
def main():
    letterCount_big, letterCount_small = count_frequency(cipher)
    new_fre_small, new_fre_big = frequency_table(letterCount_big, letterCount_small)

    guesskey = create_first_key(new_fre_small)
    key, score = key_modify(guesskey)
    # print(score) -43526.37898300028
    flag, match = create_article(flagg.lower(), key)
    print("{",flag,"}")
    # {} ->set is not subscriptable use list() -> [] to let elements iterate in order
    table = ['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']

    final = ""
    for k in table:
        for i,j in match:
            if i == k:
                final += j
    print("key=",final)

    article, match = create_article(cipher, key)
    print(article)

if __name__ == "__main__":
    main()
```

1. Generate lowercase letter frequency.
2. Use it to create first key.
3. Use the first key to decode the cipher text and continuously modify the key until we get the best score.

4. Use the key to generate correct FLAG and plaintext

Use ciphertext1 as an example:

Results(I skip all upper letters since they are minority, won't influence the result):

```
{ bkpwthpsobxztruveiw }
key= tlmcusjkniazgfqpoywvderhbx
burnedhersistersletterbeforereadingiterodriktuggedathiswhiskersoisonwellthatcouldbethedwarfsworktrueenoughrerseistssaidpoisonis
awomansweaponbeggingyourpardonsmyladyheingslayernowhavenogreatlikingforthemanbuthesnotthesortoofondofthesightofbloodonthatgolde
nswordofhisasitpoisonmyladyatelynfrownedvaguelyuneasyowelsecouldtheymakeitlookanaturaldeathehindherordobertshriekedwithdelighta
soneofthepuppetknightsslicedtheotherinhalfspillingafloodofredsawdustontotheterraceheglancedathernephewandsighedheboyisutterlywi
thoutdisciplineewillneverbestrongenoughtoruleunlessheistakenawayfromhismotherforatimeislordfatheragreedwithyousaidavoiceatherel
bowheturnedtobeholdaesterolemonacupofwineinhishandewasplanningtosendtheboytoragonstoneforfosteringyouknowohbutmspeakingoutoftur
nheappleofhisthroatbobbedanxiouslybeneaththeloosemaesterschainfearvehadtoomuchofordurtersexcellentwineheprospectofbloodshedhasm
ynervesallafrayouaremistakenaesteratelynsaidtwasasterlyocknotragonstoneandthosearrangementsweremadeaftertheandsdeathwithoutmysi
stersconsenthemaestersheadjerkedsovigorouslyattheendofhisabsurdlylongneckthathelookedhalfapuppethimselfobeggingyourforgivenessm
yladybutitwasordonwhobelltolledloudlybelowthemighlordsandservinggirlsalikebrokeoffwhattheyweredoingandmovedtothebalustradeelowt
woguardsmeninskybluecloaksledforthyrionannisterheyriesplumpseptonescortedhimtothestatueinthecenterofthegardenaweepingwomancarve
dinveinedwhitemarblenodoubtmeanttobelyssahebadlittlemanordobertsaidgigglingothercanmakehimflywanttoseehimflyageatermysweetbabyy
sapromisedhimrialfirstdrawlederynorbraythenexecutionmomentlaterthetwochampionsappearedfromoppositesidesofthegardenheknightwasat
tendedbytwoyoungsquiresthesellswordbytheyriesmasteratarmserardisgenwassteelfromheadtoheelencasedinheavyplatearmorovermailandpad
dedsurcoatargecircularrondelsenameledcreamandblueinthemoonandfalconsigilofouserrynprotectedthevulnerablejunctureofarmandbreasts
```

	ABCDEFGFIJKLMNOPQRSTUVWXYZ
KEY	TLMCUSJKNIAZGFQPOYWVDERHBX (by ciphertext1.txt)

You can try other ciphertexts provided in textfile to check my FLAG
(I use 4 ciphertexts in textfile to make sure my FLAG is correct)
FLAG : {BKPWTHPSOBXZTRUVEIW}

● Implement process and explanation (Base64)
(write in bonus.py file)

```
def create_first_key():
    key = list(range(26))
    random.shuffle(key)
    return key
```

Since there's no need to calculate the frequency of letters, so I change the way of generating first key by just randomly initialize it.

```python
def create_article(article, key): # ord -> num

    original_article = ""

    for x in article:
        if (ord(x) >= 65) & (ord(x) <= 90):
            original_article += chr(65 + key[ord(x)-65])
        elif (ord(x) >= 97) & (ord(x) <= 122):
            original_article += chr(97 + key[ord(x)-97])
        else:
            original_article += x

    return original_article
```

I change the way of decoding the article, since I use lowercase letters to decode article in Base256 version, but it didn't solve the Base64 version properly since there're many numbers and uppercase characters. Therefore, I trace the serve.py code and found out the ciphertext generating method and just "copy" it as the decoding method.
(BTW, I think this way is much more elegant than the way I used before.)

```python
def isBase64(ciphertext):
    try:
        temp_text = ciphertext.encode()
        return base64.b64encode(base64.b64decode(temp_text)) == ciphertext.encode()
    except Exception:
        return False
```

It's the function to determine if the text is Base64 or not.

```python
def tri_freq_score(guesskey): # score the higher the better

    totalcost = 0
    plaintext = create_article(cipher, guesskey)

    for i in range(int(len(plaintext)/4)):
        if isBase64(plaintext[i*4:i*4+4]):
            try:
                curstr = plaintext[i*4:i*4+4].encode()
                curstr = base64.b64decode(curstr) #decode base64 to base256
                curstr = curstr.decode()
                totalcost += triscores[curstr]
            except:
                totalcost -= 50.0
        else:
            totalcost -= 50.0

    return totalcost
```

It's very same as the version base256, only need to do more examination.
We need to check if the current plaintext is base64 (check four at a time whether can be 8bit encoded (6*4 == 8*3)), if do, we check the frequency and assign the score by the same three-letter-frequency file. Each fall, I decrease the cost by 50 at a time.

```python
def main():

    guesskey = create_first_key()
    key, score = key_modify(guesskey)
    print(score)

    ch = [0]*26
    for i in key:
        ch[key[i]] = chr(i+97).upper()
    print("key=", "".join(ch))

    article = create_article(cipher, key)
    print(base64.b64decode(article))
```

Finally, we can create the key and corresponding original article.
(Some modification base on Base256 version, much more clearer.)

```
-30055.44955599998
key= CJSRWYVUPOAXQDKFETMIZHGLNB
b'{BOTQFGBWXBQKGJOXAMQ}tohishorseandbrokeintoagallopracingdownthekingsroadasiftooutrunhisdoubtsJonwasnotafraidofdeathbuthedid
notwanttodielikethattrussedandboundandbeheadedlikeacommonbrigandIfhemustperishletitbewithaswordinhishandfightinghisfatherskil
lersHewasnotrueStarkhadneverbeenonebuthecoulddielikeoneLetthemsaythatEddardStarkhadfatheredfoursonsnotthreeGhostkeptpacewitht
hemforalmosthalfamileredtonguelollingfromhismouthManandhorsealikeloweredtheirheadsasheaskedthemareformorespeedThewolfslowedst
oppedwatchinghiseyesglowingredinthemoonlightHevanishedbehindbutJonknewhewouldfollowathisownpaceScatteredlightsflickeredthroug
hthetreesaheadofhimonbothsidesoftheroadMolesTownAdogbarkedasherodethroughandheheardamulesraucoushawfromthestablebutotherwiset
hevillagewasstillHereandtheretheglowofhearthfiresshonethroughshutteredwindowsleakingbetweenwoodenslatsbutonlyafewMolesTownwas
biggerthanitseemedbutthreequartersofitwasunderthegroundindeepwarmcellarsconnectedbyamazeoftunnelsEventhewhorehousewasdownther
enothingonthesurfacebutawoodenshacknobiggerthanaprivywitharedlanternhungoverthedoorOntheWallhedheardmencallthewhoresburiedtre
asuresHewonderedwhetheranyofhisbrothersinblackweredownAGAMEOFFHRONEStheretonightminingThatwasoathbreakingtooyetnooneseemedtoc
arePageNotuntilhewaswellbeyondthevillagedidJonslowagainBythenbothheandthemareweredampwithsweatHedismountedshiveringhisburnedh
andachingAbankofmeltingsnowlayunderthetreesbrightinthemoonlightwatertricklingofftoformsmallshallowpoolsJonsquattedandbroughth
ishandstogethercuppingtherunoffbetweenhisfingersThesnowmeltwasicycoldHedrankandsplashedsomeonhisfaceuntilhischeekstingledHisf
ingerswerethrobbingworsethantheyhadindaysandhisheadwaspoundingtooIamdoingtherightthinghetoldhimselfsowhydoIfeelsobadThehorsew
aswelllatheredsoJontooktheleadandwalkedherforawhileTheroadwasscarcelywideenoughfortworiderstopassabreastitssurfacecutbytinyst
reamsandlitteredwithstoneThatrunhadbeentrulystupidaninvitationtoabrokenneckJonwonderedwhathadgottenintohimWasheinsuchagreatru
shtodieOffinthetreesthedistantscreamofsomefrightenedanimalmadehimlookupHismarewhinniednervouslyHadhiswolffoundsomepreyHecuppe
dhishandsaroundhismouthGhostheshoutedGhosttomeTheonlyanswerwasarushofwingsbehindhimasanowltookflightFrowningJoncontinuedonhis
```

ABCDEFGFIJKLMNOPQRSTUVWXYZ
KEY CJSRWYVUPOAXQDKFETMIZHGLNB (by bonus2.txt)

FLAG : {BOTQFGBWXBQKGJOXAMQ}

You can try other ciphertexts provided in textfile to check my FLAG
(I use bonus1, bonus2 in textfile to make sure my FLAG is correct)

Reference:
https://github.com/juesato/cipher-solver/blob/master/decrypt_no_spaces.cpp