1. Planet of the Apes

카이사르암호 문제이다.

주어진 문자열은 Fdhvdu lv krph 이고, python으로 반복문 돌리며 key값을 증가시켜 결과를 출력해봤고 사람이 읽을 수 있는 결과를 flag로 제출했다.

```
E = "Fdhvdu lv krph"

for i in range(26):
    for j in E:
        print(chr((ord(j)-i)), end='')
    print("₩n")
```

```
Fdhvdu lv krph
Ecguctkujqog
Dbftbsjtipnf
Caesarishome
B`dr`qhrgnld
A_cq_p←gq←fmkc
@^bp^o→fp→eljb
?]ao]n┤eo┤dkia
```

IxC{Caesarishome}

2. Frequency

빈도수 공격 문제이다.

Df ntc sbuc-mwcdzcw fdn dn tif ucfz fndwilg dn ntc
sbvjxch fomfninonibl sijtcw il ywbln by tiv, tif vilu wdscu
rint edwibof vcntbuf by dnndsz, ywbv ywcpoclsk
dldxkfif nb nwkilg bon dxx jbffimxc zckf, tc zlcr in rdf
gbilg nb mc d xblg dlu dwuobof jwbscff, mon rint
ucncwvildnibl dlu jdniclsc tc zlcr tc sboxu swdsz ntc
sbuc dlu olsbecw ntc tiuucl vcffdgc rintil.
lhS{vdzc_ntc_jwbm_rdf_vbbbwc_tdwucw}

주어진 문장이 짧아 알파벳 빈도 수 통계만으로는 정확한 해독이 어려울 것이라 생각이 되었고 여러가지 시도를 해보다 주어진 문제 즉, 문제 설명에 frequency라는 단어가 하나쯤은 들어가 있을 거 같다는 생각이 들어 찾아보니 **ywcpoclsk** 가 같은 알파벳 위치와 단어 길이 모두 만족하길래 frequency가 암호화 된 것이 **ywcpoclsk**라 가정하였다.

또한 flag의 시작은 IxC이니 I -> I로 x -> h로 C -> S로 암호화 되었을 것이고, 추가적으로 파이썬 코드를 돌려 단어 단위로 적게 쓰인 단어 순으로 정렬하여 출력한 결과 아래와 같았다.

```
str = 'Df ntc sbuc-mwcdzcw fdn dn tif ucfz fndwilg d
str = str.lower()
str = str.split(" ")

dic = {}
for i in str:
    if i in dic:
        dic[i] += 1
    else:
        dic[i] = 1

import operator
d1 = sorted(dic.items(), key=operator.itemgetter(1))
print(d1)
```

```
[('df', 1), ('sbuc-mwcdzcw', 1), ('fdn', 1), ('ucfz', 1), ('fndwilg', 1), ('sbvjxch', 1)
, ('fomfninonibl', 1), ('sijtcw', 1), ('il', 1), ('ywbln', 1), ('tiv,', 1), ('vilu', 1),
('wdscu', 1), ('edwibof', 1), ('vcntbuf', 1), ('dnndsz,', 1), ('ywbv', 1), ('ywcpoclsk',
1), ('dldxkfif', 1), ('nwkilg', 1), ('bon', 1), ('dxx', 1), ('jbffimxc', 1), ('zckf,', 1
), ('in', 1), ('rdf', 1), ('gbilg', 1), ('mc', 1), ('d', 1), ('xblg', 1), ('dwuobof', 1)
, ('jwbscff,', 1), ('mon', 1), ('ucncwvildnibl', 1), ('jdniclsc', 1), ('sboxu', 1), ('sw
dsz', 1), ('sbuc', 1), ('olsbecw', 1), ('tiuucl', 1), ('vcffdgc', 1), ('rintil.', 1), ('
ihs', 1), ('dn', 2), ('tif', 2), ('by', 2), ('rint', 2), ('nb', 2), ('zlcr', 2), ('tc',
3), ('dlu', 3), ('ntc', 4)]
```

이를 기반으로 가장 많이 쓰이는 단어 통계와 비교하여 ntc ->the , tc -> he로 암호화 된 것이라 추측하였다. 이렇게 대응되는 알파벳을 기반으로 http://quipqiup.com/ 에서 알아낸 대응관계를 인자로 줘서 암호문을 해독하였다.

IxC{make_the_prob_was_mooore_harder}

3. ROX

```
import base64

known_str = b'????'
flag = b'??????????????????????????????'

res = ''

key_len = len(known_str)
flag_len = len(flag)

for i in range(flag_len):
    res += chr(known_str[i%key_len] ^ flag[i])

print(base64.b64encode(res.encode()))
```

Output : Ih06Wg5RTFg0W0URGTomWQQXJh9XVQsAFg==

Flag를 암호화 시킨 known_str 값을 모르지만 flag의 시작 4글자가 IxC{ 이고 output의 결과 앞 4글자가 IxC{ 가 암호화 된 것이라고 생각하면 known_str 값을 알아내어 복호화 시킬 수 있다.

```
import base64

s = "IhO6Wg5RTFgOWOURGTomWQQXJh9XVQsAFg=="

d =  base64.b64decode(s)
d =  d.decode()

'''
flag[0] = l
flag[1] = x
flag[2] = C
flag[3] = {
'''
for i in range(0, 127):
    if(ord("l") ^ i == ord(d[0])):
        print("key[0] is :"+str(i))

for i in range(0, 126):
    if(ord("x") ^ i == ord(d[1])):
        print("key[1] is :"+str(i))

for i in range(0, 126):
    if(ord("C") ^ i == ord(d[2])):
        print("key[2] is :"+str(i))

for i in range(0, 126):
    if(ord("{") ^ i == ord(d[3])):
        print("key[3] is :"+str(i))
'''
key[0] is :107
key[1] is :101
key[2] is :121
key[3] is :33
'''
key = "key!"
flag = ""
for i in range(len(d)):
    flag += chr(ord(d[i]) ^ ord(key[i%4]))

print(flag)
```

lxC{e45y_><0r_xor_><0r!}