Exercise 1

In [ ]:
```python
import hashlib
import time
import random
import string
```

In [ ]:
```python
# !wget https://raw.githubusercontent.com/danielmiessler/SecLists/master/Passwords/Common-Credentials/10k-most-comm
```

In [ ]:
```python
with open('10k-most-common.txt') as f:
    passwords = f.read().split()

print(passwords[:20])
```

```
['password', '123456', '12345678', '1234', 'qwerty', '12345', 'dragon', 'pussy', 'baseball', 'football', 'letmein',
'monkey', '696969', 'abc123', 'mustang', 'michael', 'shadow', 'master', 'jennifer', '111111']
```

In [ ]:
```python
m=hashlib.md5(b"password").hexdigest()
print(m)
m=hashlib.md5(b"Chulalongkorn").hexdigest()
print(m)
685396
m=hashlib.md5(passwords[0].encode()).hexdigest()
print(m)

finding = "d54cc1fe76f5186380a0939d2fc1723c44e8a5f7"
ch = False
```

```
5f4dcc3b5aa765d61d8327deb882cf99
46fa3b56c660faff420190c18c98a56b
5f4dcc3b5aa765d61d8327deb882cf99
```

In [ ]:
```python
def play(idx, st, now):

    if len(passwords[idx]) == st:
        # print(now)
        k = now.encode()
        m = hashlib.md5(k).hexdigest()
        if m == finding:
            print("found :", now)
            return True
        m = hashlib.sha1(k).hexdigest()
        if m == finding:
            print("found :", now)
            return True
        return False
    sub = [['o', '0'], ['l', 'i', '1']]
    key = passwords[idx][st].lower()
    # play(idx, st + 1, now + key)
    chh = True
    ch = False
    for i in sub:
        if key in i:
            chh = False
            for x in i:
                ch = ch | play(idx, st + 1, now + x)
                if x.isalpha():
                    ch = ch | play(idx, st + 1, now + x.upper())
    if chh:
        ch = ch | play(idx, st + 1, now + key)
        if key.isalpha():
            ch = ch | play(idx, st + 1, now + key.upper())

    return ch
```

In [ ]:
```python
for i in range(len(passwords)):
    if play(i, 0, ""):
        break
```

```
found : ThaiLanD
```

`Answer: ThaiLanD`

Exercise 2

In [ ]:
```python
gen = string.punctuation + string.ascii_letters + string.digits + " "
passwords = []
for i in range(1, 1024+1):
    for k in range(100):
        passwords.append(''.join(random.choices(gen, k=i)))
```

```
# print(passwords)
print(len(gen), gen)
# print(len(passwords))
```

```
95 !"#$%&'()*+,-./:;<=>?@[\]^_`{|}~abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789
```

In [ ]:
```
ma = 0
for i in passwords:
    ma = max(ma, len(i))
print(ma)
aa = [0.0 for i in range(1, ma + 2)]
cc = [0 for i in range(1, ma + 2)]
tt = [-1 for i in range(1, ma + 2)]
```

```
1024
```

In [ ]:
```
a = []
t = 0.0
def play2(idx, st, now):
    global t, aa, cc
    if len(passwords[idx]) == st:
        # print(now)
        k = now.encode()
        stt = time.time()
        m = hashlib.sha1(k).hexdigest()
        # n = hashlib.md5(k).hexdigest()
        edt = time.time()
        t += edt - stt
        # print(m, now)
        a.append((m, now))
        aa[st] += edt - stt
        cc[st] += 1
        # a.append((n, now))
        return

    sub = [['o', '0'], ['l', 'i', '1']]
    key = passwords[idx][st].lower()
    chh = True

    for i in sub:
        if key in i:
            chh = False
            for x in i:
                play2(idx, st + 1, now + x)
                if x.isalpha():
                    play2(idx, st + 1, now + x.upper())
    if chh:
        play2(idx, st + 1, now + key)
        if key.isalpha():
            play2(idx, st + 1, now + key.upper())

    # play2(idx, st + 1, now + key)

    return

st = time.time()
for i in range(len(passwords)):
    play2(i, 0, "")
ed = time.time()
```

In [ ]:
```
print("total :", len(a), "items")
print("total create time :", ed - st, "sec")
```

```
total : 13685396 items
total create time : 9.084963321685791 sec
```

`time for create is 9.08 sec and table size is 13685396 items`

Exercise 3

In [ ]:
```
print("total hash time :", t, "sec")
print("Avg time per hash :", t*1e9/len(a), "ns")
```

```
total hash time : 4.120308876037598 sec
Avg time per hash : 301.0734125660374 ns
```

In [ ]:
```
# for i in range(len(aa)):
for i in range(1, 33):
    if cc[i] != 0:
        tt[i] = aa[i] * 1e9 / cc[i]
    print(i, "bits hashing time :",  tt[i])
# print(tt)
```

```
1 bits hashing time : 372.5290298461914
2 bits hashing time : 295.3001690394987
3 bits hashing time : 289.87061288943767
4 bits hashing time : 307.4284098386072
5 bits hashing time : 331.56659942756085
6 bits hashing time : 305.86765579963486
7 bits hashing time : 312.49142980390104
8 bits hashing time : 306.7078358803413
9 bits hashing time : 308.4545423509632
10 bits hashing time : 310.85887757858427
11 bits hashing time : 298.9226400782249
12 bits hashing time : 309.92735852039556
13 bits hashing time : 316.94496781073957
14 bits hashing time : 307.5193174685965
15 bits hashing time : 308.34816550971254
16 bits hashing time : 306.10030844476637
17 bits hashing time : 305.9885338469166
18 bits hashing time : 308.1738415599718
19 bits hashing time : 294.2466165049216
20 bits hashing time : 306.9092073988106
21 bits hashing time : 321.8650817871094
22 bits hashing time : 302.7915954589844
23 bits hashing time : 295.6390380859375
24 bits hashing time : 326.6334533691406
25 bits hashing time : 302.7915954589844
26 bits hashing time : 314.7125244140625
27 bits hashing time : 333.7860107421875
28 bits hashing time : 345.7069396972656
29 bits hashing time : 369.5487976074219
30 bits hashing time : 340.9385681152344
31 bits hashing time : 348.09112548828125
32 bits hashing time : 345.7069396972656
```

`For not above 32 bytes hashing, it's take nearly time for sha1 hashing from the experiment. For 13685396 items take 4.12 secs for hashing which is around 301 ns per hash. (I use 4.12 which is only hashing time because python has too many overheads and let's assume C++ has nearly 0 overheads)`

Exercise 4

$$\text{Time For Crack} = K^N \cdot T$$

$$\begin{aligned}&\text{Where,}\\&\text{K} = \text{Number of possible characters}\\&\text{N} = \text{Length of password}\\&\text{T} = \text{Time for each hashing}\end{aligned}$$

Exercise 5

`Let's`

- `K = 95 (26: a-z), (26: A-Z), (10: 0-9), (32: special characters), (1: space)`
- `T = 301 ns`

For `4 characters password` take 95^4 * 301 ns = `around 24.5 seconds`

For `5 characters password` take 95^5 * 301 ns = `around 39 minutes`

For `6 characters password` take 95^6 * 301 ns = `around 2 and a half days`

For `7 characters password` take 95^7 * 301 ns = `around 8 months`

For `8 characters password` take 95^8 * 301 ns = `around 63 years`

For `9 characters password` take 95^9 * 301 ns = `around 6011 years`

Let's say I might not change the password to the end of my life, so I can use 8 characters password or 9 characters is plus for more security.

Exercise 6

For each password will have corresponding salt and use it along with the password to hash. So, the hash will be different for the same password.
ex. `password = "1234"` and `salt = "@#_@!"` will hash to `hash("@#_@!1234")`

1. Password that pass to hash is now longer which harder or even impossible to crack.

2. Even if the password is as same as another user, the salt will be different make rainbow table useless due to different hash for the same password.