

In the name of God



استاد : دکتر هراتی

دانشجو : توحید حقیقی سیس

شماره دانشجویی : 830598021

موضوع : تمرین سوم

تمرین اول :

دانشگاه تصمیم گرفته برای حمایت از دانشجویان در روزهای همه‌گیری ویروس، با همکاری چند آپراتور تلفن همراه به تعداد محدودی از آنها ۱۰۰ گیگابایت اینترنت هدیه اختصاص بدهد. دانشجویان باید در روز معین و در بازه دو ساعته تعیین شده به سامانه ثبت نام مراجعه و با درج کد ملی و شماره تلفن خود ثبت نام کنند.

سامانه باید قادر باشد تک‌تک کد ملی‌های ورودی را در پایگاه داده خود جستجو کرده و نتیجه را اعلام کند. برای پیاده سازی فرایند جستجو از **فیلتر بلوم**^۱ استفاده شده است. دیتاست `uniquemelli.csv` شامل کدهای ملی دانشجویان در اختیار شما قرار گرفته است.

خبر رسیده است که تعدادی از دانشجویان دانشگاه‌های دیگر با اطلاع از اینکه سامانه از فیلتر بلوم استفاده می‌کند تصمیم گرفته‌اند که شانس خود را برای ثبت نام امتحان کنند.

فیلتر بلوم به طور کلی به این صورت است که ما در ابتدا یک راییه تمام صفر با طول دلخواه در نظر میگیریم و یه تعداد تابع هش در این تمرین هدف تعیین کردن تعداد تابع های هش مناسب و طول راییه اولیه مناسب است من این تمرین را با پایتون در محیط جوضیتر پیاده سازی کردم که مرحله به مرحله توضیح میدهم .

- پیاده سازی به صورت کلاینت سرور : در این قسمت من داده ها را تک تک در راییه شامل یک ها تولید و چک میکنم تا ببینم این کد ملی در لیست دانش جویان دانشگاه تهران هست یا نه .
هر داده ای از سمت سرور گرفته شود اول هش شده و در راییه اولیه چک میشود تا ببینیم به خانه ای هش میشود که 1 است اگر یک بود الگوریتم ما این فرد را به صورت اشتباه تشخیص داده و False positive رخ میدهد .
- برای مشاهده موارد بالا از ابتدا باید پیاده سازی را توضیح دهم .
- مرحله اول : خواندن داده ها از دیتاست و ساختن راییه ای که شامل یک ها است .

```
In [341]: import numpy as np
import pandas as pd
```

```
In [342]: import random
```

```
In [343]: with open('uniquemelli.csv') as f:
lines = f.readlines()
```

```
In [344]: lines = [int(i[:-1]) for i in lines]
```

در تابع بالا دیتاست موجود را خوانده و به صورت لیست اعداد نگه میدارم و خروجی این قسمت به صورت زیر است .

```
In [5]: lines
```

```
Out[5]: [123505,
123522,
123540,
123605,
123610,
123634,
123635,
123703,
123718,
123794,
123860,
123935,
123942,
123960,
123997,
124030,
124105,
124187,
124197,
```

در قسمت بعد تولید آرایه از روی دیتاست موجود را با تابع `fill_table` انجام میدهم اما برای اینکه این تابع را توضیح دهم قبل از آن باید تابع هش که در این تابع انجام شده را توضیح دهم .

```
def get_hash_function(hash_range):
    r = random.randint(1, 100000)
    def hash(x):
        nonlocal r
        return (r * x + (r + 1)) % hash_range
    return hash
```

ما در این تابع یک عدد رندوم درست کرده و از روی آن تابع $ax+b$ را تولید کرده و خود func را برمیگردانیم. این قسمت برای این است که چون در روش بلوم فیلتر لیستی از هش ها لازم است که یک شماره ملی را با همه آن ها هش کرده و در آن خانه ها یک قرار دهیم و برای تشخیص و یا جستجو کد ملی نیز اول آن عدد را با تمام هش ها هش کرده اگر در همه آن ها به 1 رسید پس در ارایه قرار دارد.

```
def fill_table(student_list, hash_functions, hash_table):
    for student in student_list:
        for h in hash_functions:
            index = h(student)
            hash_table[index] = 1
```

در تابع بالا ارایه را به وسیله تابع هش قسمت قبل پر میکنیم ولی یک مورد را باید دقت کنیم که هر چه قدر ارایه اولیه بیشتر باشد تعداد 1 های موجود در آن به نسبت کمتر میشود چون خانه های زیادی دارد و هش ها به خانه های متفاوت میتواند اعداد را هش کند و الگوریتم جستجو ما دقیق تر میشود اما باید زیاد هم بزرگ نباشد که بی دلیل حافظه بگیرد در قسمت آخر این سوال من با طول ارایه مختلف و تعداد هش های مختلف تست کردم و به صورت نموداری آن ها را نمایش داده ام.

```
def is_in(element, array, hash_functions):
    for h in hash_functions:
        index = h(element)
        if(array[index]==False):
            return False
    return True
```

تابع بالا برای بررسی این که یک عدد که برای دانشجویان خارج از دانشگاه تهران گرفتیم به صورت اشتباهی میتواند ثبت نام کند یا نه منظور از اشتباهی عدد رندومی که داده در تمام هش ها به یک خانه میوفتد یا نه.

```
def count_false_positive(nonstudent_list, hash_functions, hash_table):
    false_postive = 0
    for nonstudent in nonstudent_list:
        if(is_in(nonstudent, hash_table, hash_functions)):
            false_postive += 1
    return false_postive
```

در این سوال یک مفهوم False positive است یعنی کد ملی اختیاری که زدیم جزو لیست کد ملی ها نیست ولی به اشتباه روش ما ان را جزو ان کد ملی ها میگیرد .

```
nonstudent_list = [random.randint(100000, 1000000) for i in range(1000)]

student_list = lines
```

در تابع بالا لیست داده های رندوم که در زیر ایجاد کردیم بنابر صورت سوال 1000 تا ایجاد کردیم . الگوریتم ما ان کد ملی را میگه جزو لیست هست یا نه برای محاسبه ان با تمام هش ها ان را بررسی و باید در تمام هش ها به مقدار 1 برسد .

```
def bloom_filter(student_list, nonstudent_list, table_size, hash_num):
    hash_functions = [get_hash_function(table_size) for _ in range(hash_num)]
    hash_table = np.zeros(table_size, dtype = bool)
    fill_table(student_list, hash_functions, hash_table)
    return count_false_positive(nonstudent_list, hash_functions, hash_table)
```

به صورت کلی تابع فیلتر بلوم من این مراحل را انجام میدهد و توابع بالا که توضیح دادم را فراخوانی میکند .
من توابع بالا رو بر اساس یکی از قسمت های سوال با تعداد هش متفاوت و طول ارایه متفاوت اجرا کردم و نمودار ان ها را در زیر قرار دادم .

```
bf = lambda x, y: bloom_filter(student_list, nonstudent_list, x, y)
```

```
x = [i for i in range(1, 20)]
```

```
y = [bf(500000, j) for j in x]
```

```
import matplotlib  
from matplotlib import pyplot as plt
```

```
plt.plot(x, y)
```

کد رسم نمودار به صورت بالا است و در زیر اجرا و خروجی آن را قرار می‌دهم .

قسمت اول با تعداد هش 15 طول آرایه را تغییر می‌دهم از 5000 تا 1000000 :

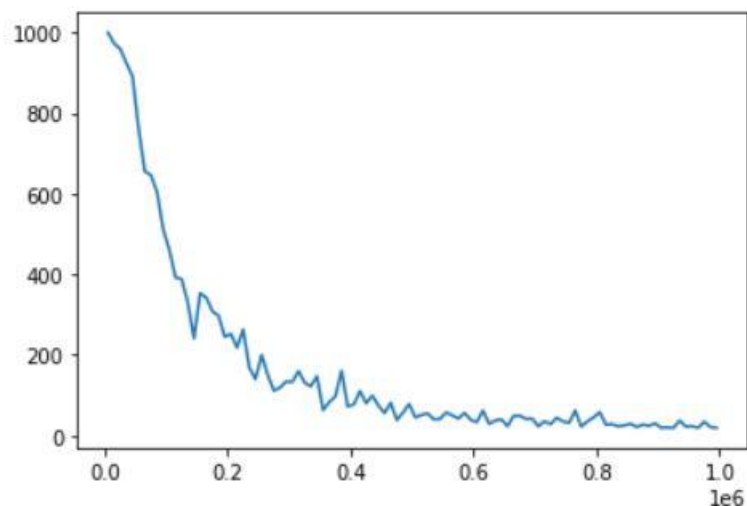
```
In [32]: x = [i for i in range(5000, 1000000, 10000)]
```

```
In [33]: y = [bf(j, 15) for j in x]
```

```
In [34]: import matplotlib  
from matplotlib import pyplot as plt
```

```
In [35]: plt.plot(x, y)
```

```
Out[35]: [<matplotlib.lines.Line2D at 0x111e3508>]
```



شکل بالا خیلی واضح نشان میدهد که هر چه طول آرایه اولیه بیشتر شود تعداد افرادی که به صورت رندوم بخواهند وارد سیستم شوند کمتر میشود یعنی خطای **false positive** کمتر میشود.

قسمت دوم طول آرایه را ثابت میگیریم و تعداد هش ها را تغییر میدهم.

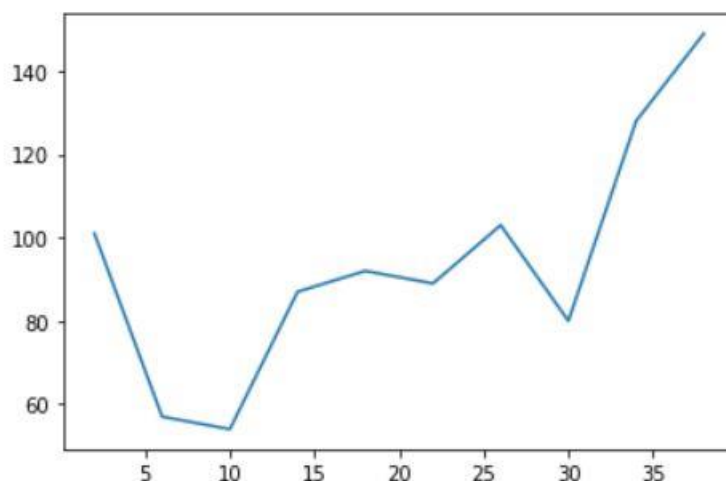
```
In [38]: x = [i for i in range(2,40,4)]
```

```
In [39]: y = [bf(500000, j) for j in x]
```

```
In [40]: import matplotlib
from matplotlib import pyplot as plt
```

```
In [41]: plt.plot(x, y)
```

```
Out[41]: [<matplotlib.lines.Line2D at 0x11168280>]
```



بر اساس شکل بالا هر چه تعداد هش بالا بره الگوریتم بهتر کار نمیکنه در اعدادی مثلا 10 در این مثال میبینیم که این عدد به کمترین مقدار خود رسیده است یعنی **false positive** در کمترین مقدار خود است.

- آیا همه دانشجویان دانشگاه تهران موفق به ثبت نام میشوند؟ جواب این سوال بله است چون این روش False negative ندارد یعنی افرادی که در دیتاست قرار دارند و با هش های مختلف در جدول قرار گرفته اند اگر با همان شماره ملی دوباره بخواهند وارد شوند هش همان است و به راحتی میتوانند ثبت نام کنند پس جواب این سوال بله هست.
- برای اینکه 0.01 دانشجویان دانشگاه های دیگر بتوانند ثبت نام کنند.

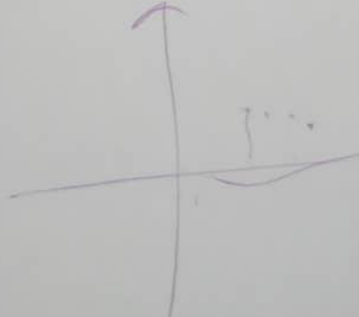
• حداقل ۰.۰۱ از دانشجویان دانشگاه های دیگر موفق به ثبت نام شوند.

$k = \text{تعداد هش}$
 $m = \text{تعداد مجموعه ارایه}$
 $n = \text{طول ارایه}$

$$\left(1 - e^{-k \frac{m}{n}}\right)^k = 0.01$$

$m = 20000$

$\left(1 - e^{-\frac{10 \times 20000}{n}}\right)^{10} = 0.01$



یبق شکل بالا باید حساب شود با چه اعدادی ان تساوی بر قرار است با محاسبات من یکی از نقاطی که در ان صدق میکند.

تعداد هش : 10

طول ارایه : 400000

چند فرمول زیر نیز برای محاسبه k و n استفاده میشود :

Optimum number of hash functions: The number of hash functions k must be a positive integer. If m is size of bit array and n is number of elements to be inserted, then k can be calculated as :

$$k = \frac{m}{n} \ln 2$$

Size of Bit Array: If expected number of elements n is known and desired false positive probability is p then the size of bit array m can be calculated as :

$$m = -\frac{n \ln P}{(\ln 2)^2}$$

که با محاسبه $m=20000$ در این صورت با فرمول پایین $n=2070$ به دست می آید.

و $k = 10 * 0.69 = 6.9$ به دست می آید .

تمرین دوم :

۲- (طراحی و پیاده سازی الگوریتم)

دیتاست این تمرین (digikala.xlsx) شامل اطلاعات سفارشات فروشگاه دیجی کالا در یکی از روزهای سال ۲۰۱۸ میلادی است. با استفاده از الگوریتم **Flajolet-Martin** تعداد استانهای منحصر به فرد در این دیتاست را تخمین بزنید.

همچنین دیتاست دیگری نیز (urls.csv) در اختیار شما قرار گرفته است. این دیتاست شامل urlهایی است که کاربران در یک موتور جستجو کلیک کرده‌اند. با استفاده از الگوریتم **Flajolet-Martin** تعداد urlهای منحصر به فرد در این دیتاست را تخمین بزنید.

برای حل این قسمت نیز از روش هش استفاده میکنیم به این صورت که به اعداد باینری هش میکنیم و تعداد صفرها را از آخر شمارش میکنیم اگر تعداد صفرهای آن برابر بود به یک نفر اشاره میکند .

برای هر دو دیتاست زیر این الگوریتم را اجرا میکنیم برای این که شمایه کلاینت – سروری بدهد اعداد را تک تک وارد و بزرگترین سول داده شده را نگه میداریم .

دیتاست اول: **digikala.xlsx** چون این دیتاست از نوع اکسل است به صورت زیر خوانده میشود .

```
import pandas as pd
import hashlib
import random
import openpyxl

wb = openpyxl.load_workbook("digikala.xlsx")
ws = wb['Sheet1']
```

کد بالا فایل اکسل را میخواند و در **WS** قرار میدهد .

```
In [84]: import pandas as pd
import hashlib
import random
import os

ALL_VALUES=[]
for i in range(20):
    ZeroCount_list=[]
    salt=os.urandom(64)
    for row in ws.rows:
        hashcode=hashlib.pbkdf2_hmac("sha1",row[6].value.encode('utf-8'),salt,1,8)
        a=int.from_bytes(hashcode,"big")
        b=bin(a)[2:]
        ZeroCount_list.append(len(b)-len(b.rstrip("0")))
    ALL_VALUES.append(max(ZeroCount_list))
ALL_VALUES

Out[84]: [4, 11, 4, 7, 5, 5, 6, 8, 5, 4, 9, 6, 4, 7, 5, 4, 6, 5, 5, 2]
```

کد بالا به این صورت عمل میکند که هر خط را پردازش میکند و هش آن را میگیرد و تعداد صفرهای آخر آن را شمارش میکند و در لیست **ZeroCount_list** قرار میدهد هر موقع بخواهیم تعداد خانه های مورد نیاز را بدانیم از آن لیست **max** میگیریم .

در دیتاست اول **max** برابر 5 است یعنی به دو به دو به پنج خانه نیاز داریم .

این روش را باید چند بار تکرار کنیم و از اعداد به دست آمده میانگین یا میانه بگیریم .

دی دیتاست دوم :

دیتاست دوم چون CSV است از پانداس برای خواندن ان استفاده میکنیم .

```
import pandas as pd

data=pd.read_csv("urls.csv")
print(data)
```

بعد از خواندن ان به ترتیب شروع به خواندن میکنیم و به تابع هش پاس میدهیم ان نیز ان ها را هش کرده و تعداد صفرهای ان را میشمارد و در لیست قرار میدهد .

```
In [83]: import pandas as pd
import hashlib
import random
import os

ALL_VALUES=[]
for i in range(20):
    ZeroCount_list_url=[]
    salt=os.urandom(64)
    data_table=pd.DataFrame(data)
    for row in range(len(data_table)):
        s=data_table.iloc[row].urls
        hashcode=hashlib.pbkdf2_hmac("sha1",s.encode('utf-8'),salt,1,8)
        a=int.from_bytes(hashcode,"big")
        b=bin(a)[2:]
        ZeroCount_list_url.append(len(b)-len(b.rstrip("0")))
    ALL_VALUES.append(max(ZeroCount_list_url))
ALL_VALUES

Out[83]: [14,
17,
12,
17,
12,
13,
11,
12,
13,
16,
14,
12,
15,
13,
14,
12,
15,
11,
13,
13]
```

در کد بالا نمایان است که با پانداس ارایه را خوانده و تک تک تابع flajolet_martin را روی ان اجرا میکنیم و در نهایت اگر بخواهیم تعداد r را بدست اوریم از ان ارایه max را نگه میداریم .

در دیتاست دوم این عدد 12 بدست میآید که برابر است با 2^{12}

این روش را باید چند بار تکرار کنیم و از اعداد به دست آمده میانگین یا میانه بگیریم .

یک کلاس کلی برای این روش نوشتم که در زیر قرار میدهم که کد نویسی تمیز در آن رعایت شده است .

```
import hashlib
import os
from collections import defaultdict, namedtuple
import pandas as pd

class Flajolet_Martin:
    def __init__(self, data):
        self.data = data

    def Find_Zero_Count(self, salt=b''):
        ZeroCount_list = []
        for i in range(len(self.data)):
            s = self.data.iloc[i].person
            hashcode = hashlib.pbkdf2_hmac("sha1", s.encode('utf-8'), salt, 1, 8)
            a = int.from_bytes(hashcode, "big")
            b = bin(a)[2:]
            ZeroCount_list.append(len(b) - len(b.rstrip("0")))
        return ZeroCount_list

    def Find_Max_ZeroCount(self, list_zero_count):
        return max(list_zero_count)

    def Find_Salt(self):
        return os.urandom(64)

    def List_Hash_Flajolat_Martin(self):
        flajolat_data_list = defaultdict(list)
        for i in range(10):
            salt = self.Find_Salt()
            list_zero_count = self.Find_Zero_Count(salt)
            print(max(list_zero_count))
            flajolat_data_list["hash_function"].append(salt)
            flajolat_data_list["hash_group"].append(i//100)
            flajolat_data_list["tail_length"].append(max(list_zero_count))

        return flajolat_data_list

    def Calculate_UserCount_Stimate(self, listhash):
        hash_df = pd.DataFrame(listhash)
        return hash_df.groupby("hash_group").mean().median()
```

من برای هر سوال 20 تا تابع هش متفاوت را اجرا کردم و خروجی هر یک را در زیر آن نشان دادم همان طور که میبینید اکثراً در یک بازه عددی خروجی میدهد ما میتوانیم با میانگین گرفتن آن مقدار بهتر آن را بدست آوریم برای مثال در اولی لیست به صورت زیر است .

```
[4, 11, 4, 7, 5, 5, 6, 8, 5, 4, 9, 6, 4, 7, 5, 4, 6, 5, 5, 2]
```

```
Out[84]: [4, 11, 4, 7, 5, 5, 6, 8, 5, 4, 9, 6, 4, 7, 5, 4, 6, 5, 5, 2]
```

```
In [86]: sum(ALL_VALUES)/20
```

```
Out[86]: 5.6
```

میانگین آن همان طور که میبینید 5.6 میباشد پس تعداد 2^5 تا مقدار Distinct در این آرایه وجود دارد.

[Source In My Github](#)