In the name of God

استاد : دکتر هراتی

دانشجو : توحید حقیقی سیس

شماره دانشجویی : 830598021

موضوع : تمرین دوم

## تمرین اول :

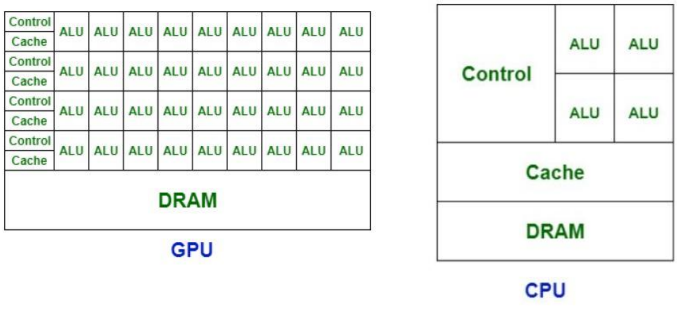
## تفاوت CPU و GPU :

CPU(واحد پردازش مرکزی) اغلب مغز کامپیوتر نامیده می‌شود اما به طور فزاینده ای این مغز توسط بخش دیگری از کامپیوتر ارتقاء می یابد – این بخش GPU(واحد پردازش گرافیکی) نام دارد که به نوعی روح کامپیوتر محسوب می‌شود.

قابلیت های پیشرفته GPU بدواً در تصویرسازی بازی‌های سه بعدی مورد استفاده قرار می‌گرفت. اما اکنون این قابلیت‌ها به طور گسترده تر به منظور شتاب بخشیدن به عملیات محاسباتی به کار گرفته می‌شود از جمله مدل سازی مالی، تحقیقات علمی پیشرفته و همچنین اکتشافات نفت و گاز .

در مقاله اخیر نشریه BusinessWeek، قابلیت‌های منحصر به فرد GPU چنین توصیف شده است: پردازنده‌های گرافیکی (GPU) برای انجام سریع عملیاتی یکسان، روی بسته‌های عظیم داده، بهینه سازی شده‌اند بر خلاف ریز پردازنده‌ها (CPU) که همه منظوره هستند و همه جای رایانه سرک می‌کشند.

- **Gpu خیلی سریع تر از Cpu است .**
- **CPU برنامه نویسی parallel رو ساپورت نمیکند درحالی که GPU ساپورت میکند .**
- **Gpu برای کارهای گرافیکی و بازی ها ساخته شده است**




- Gpu میتواند هزاران پردازش را در کسری از ثانیه انجام دهد .
- GPU کاربرد زیادی در پردازش های سنگین هوش مصنوعی و دیپ لرنینگ دارد .

## تفاوت Nvidia and Intel :

شرکت بزرگ آمریکایی در زمینه ساخت کارت‌های ویدئویی و گرافیکی رایانه‌های شخصی و کنسول بازی از قبیل xboxو پلی‌استیشن 3 فعالیت‌های به خصوصی دارد .این شرکت در سال 1993 میلادی به‌وسیله **jen-Hsun Huang**تأسیس شد، وی همچنین مدیرعامل NVIDIA نیز است.

**INTEL** : بزرگ‌ترین شرکت سازنده تجهیزات رایانه است که در زمینه تولید سخت‌افزارهای رایانه و تلفن همراه، با تمرکز بر **مادربرد**، کارت شبکه، چیپست، بلوتوث و حافظه‌های فلش، همچنین انواع ریزپردازنده‌ها، مدارهای مجتمع، واحدهای پردازش گرافیکی و سامانه‌های نهفته فعالیت می‌کند.
مقایسه قطعات شرکت AMD با شرکت NVIDIA و INTEL یکی از قدیمی‌ترین و جنجالی‌ترین مقایسه‌ها است. در سرتاسر دنیا طرفداران این ۳ شرکت وجود دارد که با خرید قطعات قوی و گران‌قیمت این شرکت‌ها، قدرت دستگاه خود را به بیشترین حد ممکن ارتقاء می‌دهند و قطعات این شرکت‌ها را به نمایش می‌گذارند اما مسئله اصلی این است که کدام‌یک از این قطعات و محصولات برای چه‌کار و چه برنامه ای مناسب می‌باشد، برای پاسخ به این سؤال باید قدرت و سرعت این قطعات را بررسی کنیم.

## خلاصه مقاله :

**روش های مدرنی برای توسعه سیستم های مبتنی بر ماشین لرنینگ انجام میشود برای مثال استفاده از سیستم های خیلی قوی و GPU های شرکت Nvidia که از قدرت خیلی زیادی برای توسعه سیستم های دیپ لرنینگ به کار میروند .**

**در این مقاله یک روش جدید معرفی کرده که به سخت افزار وابسته نیست و بدون سیستم های خیلی قوی هم میتوان پردازش های سنگین ماشین لرنینگ را انجام داد و از حالت عادی X20 برابر سریع تر است.**

برای یافتن شباهت بین متن ها و یا دسته بندی متون بر اساس شباهت و خیلی کاربرد دیگر ما داده های خیلی زیادی با درصد **sparcity** خیلی بالا داریم برای مثال یک میلیون فایل را بر اساس کلمات مشابه در نظر میگیریم پردازش این جور سوالات خیلی سنگین است چون داده ها زیاد است .

الگوریتم **LSH** اومده و این **sparticity** را از بین میبرد و کل فایل ها را دیگر لازم به تحلیل نیستیم و فقط هش های ان را تحلیل میکینیم برای این کار معیار های شباهت زیادی وجود دارد یکی شباهت ژاکارد اسا که در تمرین از ان استفاده کردم و دیگری شباهت **Hamming** و یا شباهت **cosine** است که نام های مختلفی در این مقاله دارند یکی **simhash** و دیگری **DWTA** است .

این گونه عمل میشود که داده ها را به **Batch** هایی تقسیم میکنیم و به صورت موازی ان ها را پردازش میکنیم و نتایج را در یک جا مجتمع میکنیم و پاسخ را پیدا میکنیم .

چند تا از مثال هایی که میتوان از این روش پیدا کرد **Vanilla sampling** و **TopK sampling** و **Hard Thresholding** است و از تکنیک **SLIDE** نیز برای تقسیم داده ها استفاده کرده است .

## تمرین 2 :

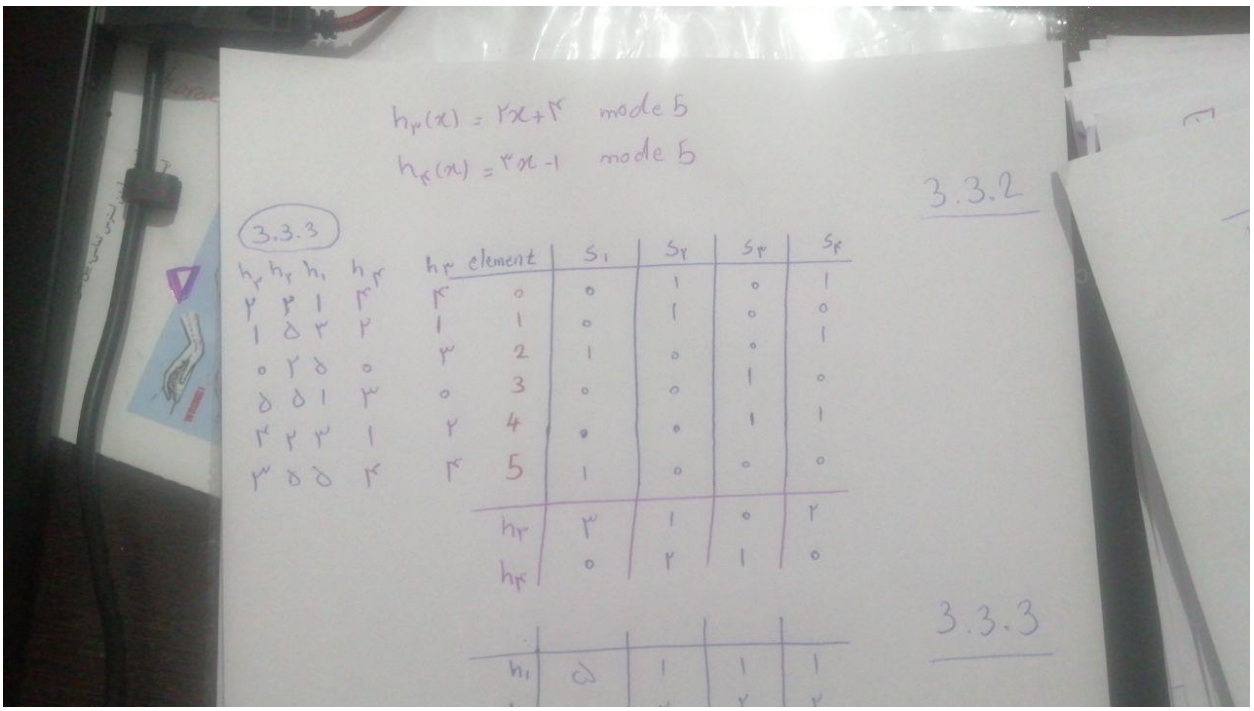
**Exercise 3.3.2 :** Using the data from Fig. 3.4, add to the signatures of the columns the values of the following hash functions:

(a) $h_3(x) = 2x + 4 \mod 5$.

(b) $h_4(x) = 3x - 1 \mod 5$.

| *Element* | $S_1$ | $S_2$ | $S_3$ | $S_4$ |
|---|---|---|---|---|
| 0 | 0 | 1 | 0 | 1 |
| 1 | 0 | 1 | 0 | 0 |
| 2 | 1 | 0 | 0 | 1 |
| 3 | 0 | 0 | 1 | 0 |
| 4 | 0 | 0 | 1 | 1 |
| 5 | 1 | 0 | 0 | 0 |

Figure 3.6: Matrix for Exercise 3.3.3



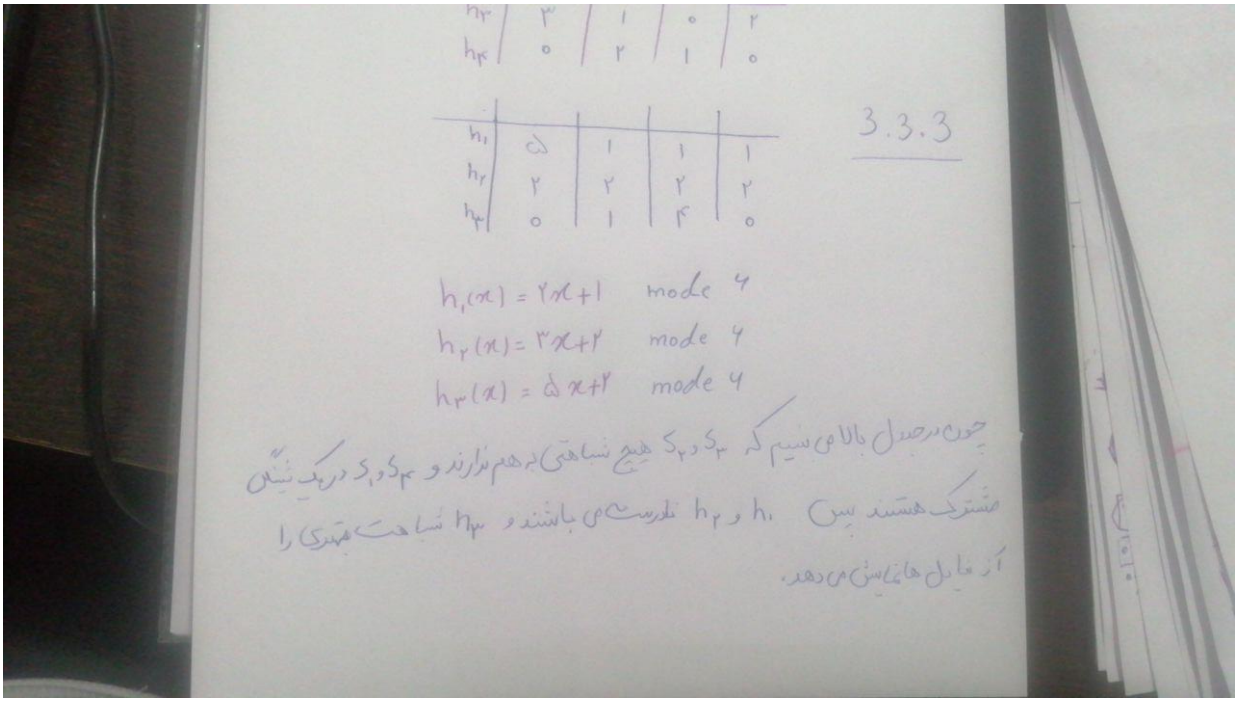
**Exercise 3.3.3:** In Fig. 3.6 is a matrix with six rows.

(a) Compute the minhash signature for each column if we use the following three hash functions: $h_1(x) = 2x + 1 \mod 6$; $h_2(x) = 3x + 2 \mod 6$; $h_3(x) = 5x + 2 \mod 6$.

(b) Which of these hash functions are true permutations?



(c) How close are the estimated Jaccard similarities for the six pairs of columns to the true Jaccard similarities?



قسمت سوم :

شباهت ژاکارد برابر اشتراک تقسیم بر اجتماع است اگر بخواهیم ژاکارد جدول اصلی را پیدا کنیم و بعد با شباهت hash function مقایسه کنیم به صورت زیر خواهد بود :

Jacard(S1,S2)=0/4=0

Jacard(S1,S3)=0/4=0

Jacard(S1,S4)=1/4

Jacard(S2,S3)=0

Jacard(S3,S4)=1/4

همانطور که میبینیم در ماتریس اصلی فقط بین s1,s4 و s3,s4 شباهت وجود دارد و بقیه به هم شبیه نیستند.

هش های h4,h3 شباهت بین s1,s4 را خوب نشان میدهند و هش های h1,h2 شباهت s3,s4 را خوب نشان میدهند ولی هر 2 چندان دقیق نیستند .

## تمرین سوم : پیاده سازی

- همه داکیومنت ها به شینگل تبدیل شوند. علائم نگارشی به فاصله تبدیل شده و فاصله های متوالی تبدیل به یک فاصله شوند.

این قسمت را به صورت زیر پیاده کردم که لیستی از مجموعه ای از Set ها در نظر گرفتم ولی چرا Set چون اجتماع و اشتراک گرفتن روی لیست Set بسیار سریع است و به سرعت این کار را برای شباهت ژاکارد پیدا کردن میتوان انجام داد .

من از شینگل هایی با طول 9 استفاده کردم و از کتابخانه hashlib برای هش کردن مقادیر استفاده کردم .

و برای تبدیل علایم نگارشی از رگولار اکسپرشن استفاده کردم Regular Expression و فواصل اضافه را حذف کردم .

---

In [1]:
```
from hashlib import sha1
import re
```

In [2]:
```
a = int(sha1("as".encode("ASCII")).hexdigest(), 16) % (10**32)
```

In [99]:
```
k = 9
```

In [116]:
```
shingles_superset = set()
docs_set_list = []
```

In [ ]:
```
for i in range(1,435):
    with open('docs/{doc_name}'.format(doc_name=i)) as fp:
        doc = fp.read()
        doc = re.sub(r"[\*\"“”\n\\…\+\-\/\=\(\)‘•:\[\]\|’;]", " ", str(doc))
        doc = re.sub(r"[ ]+", " ", doc)
        shingles_set = set()
        if (k<=len(doc)):
            for i in range(0, (len(doc)-k+1)):
                shingle = doc[i:i+k]
                shingle_hash = int(sha1(shingle.encode("ASCII")).hexdigest(), 16) % (10**32)
                shingles_set.add(shingle_hash)
                shingles_superset.add(shingle_hash)
    docs_set_list.append(shingles_set)
```

برای توضیح بیشتر کد بالا کد پایین را با کامنت گذاری در کد بالا قرار دادم .

```
[ ]: # در این قسمت چون 435 تا داکیومنت داریم از این فور استفاده کردم که ستون های جدول شینگل را تشکیل دهد
for i in range(1,435):
    # در این قسمت داکیومنتن ها را باز کرده و محتوای ان را میخوانیم
    with open('docs/{doc_name}'.format(doc_name=i)) as fp:
        doc = fp.read()

        # در این قسمت با رگولار اکسپرشن علایم نگارشی و فاصله های اضافه را حذف میکنیم
        doc = re.sub(r"[\*\"“\n\\…\+\-\/\=\(\)‘•:\[\]\|’;]", " ", str(doc))
        doc = re.sub(r"[ ]+", " ", doc)

        #در این قسمت به ازای هر داکیومنت که در اینجا ستون است یک مجموعه ست در نظر میگیریم
        shingles_set = set()

        #اگر طول شینگل از داکیومنت کمتر باشد در نظر نمیگیریم
        if (k<=len(doc)):

            # در این قسمت کل داکیومنت را با شینگل های 9 تایی پیمایش میکنیم و شینگل ها را استخراج میکنیم
            for i in range(0, (len(doc)-k+1)):
                shingle = doc[i:i+k]

                # برای این که نگه داری کلمه در حافظه فضای زیادی میبرد برای استفاده کمتر از حافظه ان را هش میکنم
                shingle_hash = int(sha1(shingle.encode("ASCII")).hexdigest(), 16) % (10**32)

                shingles_set.add(shingle_hash)
                shingles_superset.add(shingle_hash)

    docs_set_list.append(shingles_set)
```

- ماتریس شینگل-داکیومنت را بسازید

برای این قسمت و افزایش سرعت ان یک روش را انتخاب کردم در زیر توضیح میدهم .

اول با numpy یک ماتریس تمام صفر با تعداد ستون تعداد داکیومنت ها و تعداد سطر تعداد شینگل ها در نظر میگیرم .

```
In [104]: import numpy as np
          from bisect import bisect

In [105]: a = np.zeros(shape=(len(shingles_superset), 435))

In [107]: a.shape
Out[107]: (26386, 435)
```

شینگل ها رو از روی هش های ان مرتب کردم با تابع Sort تا بتوانم از کتابخانه bisect استفاده کنم که سرعت جستجو خیلی زیادی روی مجموعه دارد .

```
In [ ]: shingles_order = list(shingles_superset)
        shingles_order.sort()
```

```
In [108]: # به ازای هر داکیومنت و مجموعه شینگل های داخل ان این فور را انجام میدهم تا
          # در صورت وجود ان شینگل مقدار صفر را به یک تبدیل کند
          for j, doc_set in enumerate(docs_set_list):
              for shingle in doc_set:
                  i = bisect(shingles_order, shingle) - 1
                  a[i, j] = 1
```

این قسمت سرعت فوق العاده بالایی دارد هنگام ارایه بیشتر توضیح میدهم استراتژی این قسمتم رو .

- با توجه به بخش 3.3.5 کتاب، ماتریس MinHash Signature را بسازید .

در این قسمت باید این طوری عمل میکردیم که تعدادی تابع برای جابه جایی index سطر در نظر بگیریم .

من یک تابع نوشتم و این تابع رو 100 بار در کد اجرا میکنم تا 100 تابع مختلف به من بدهد .

این ساخت تابع روش های مختلفی دارد من یکی از ام ها که بتواند ax+b را طوری بسازد که a و b با هم سازگار باشند و شرایطی که در کتاب گفته بینشون برقرار باشد .

```
In [81]: def hash_row_number(hash_id, x):
             """
                 returns permutation's hash
             """
             hash_range = 26386
             if (hash_id ==0):
                 return x
             else:
                 return (x*hash_id + hash_id - 1) % hash_range
             return x
```

و بعد با numpy دوباره یک ماتریس میسازم این بار همه بی نهایت که به تعداد hash permutation سطر و به تعداد داکیومنت ها ستون داشته باشد .

```
In [6]: import numpy as np
        b = np.matrix(np.ones((100, 435)) * np.inf)
```

```
In [7]: b.shape
Out[7]: (100, 435)
```

بعد کد های زیر رو نوشتم که min hash signature را انجام میدهد در داخل کد ها توضیحش رو هم قرار دادم و ماتریس تشکیل شده در بالا را با مقادیر ان پر کردم .

```
In [*]: # صد تا تابع هش برای پرمیوتیشن مختلف
        for i in range(0,100):
            #یک عدد رندوم برای پاس دادن به تابع تولید تابع هش
            random_hash_number = random.randint(1,100)
            # min hash در اینجا ماتریس
            # Signiture را تولید میکنم
            for m in range(0,435):
                for j in range(0,26386):
                    l = hash_row_number(random_hash_number,j);
                    if(a[j,m]==1):
                        # اگر از مقدار قبلی کمتر بود جای گزین کن
                        if(b[i,m]>=l):
                            b[i,m]=l
                            # را تکرار نکن iteration اگر صفر شد چون کمتر از اون نمیتواند مقدار بگیرد
                            if(l==0):
                                break;
            print(b[i,:])
```

```
[[127.  71.  31.   0. 143.  74.  20.  77.   0.  38.  56.  69.  28.  71.
  127. 127.  77. 116.  77. 159. 116.   1. 484. 113.  11. 104.  76.  80.
    8. 139.  24.  53. 110.  24.  33.  72.  24.  54.  63.  58.  60.  76.
   24.  34.  24. 135.  35.  12.  24.  15.  15.  80.  80.  83. 101. 243.
   72. 114.  24.  72. 275.  32.  32.  58.  53.  45.  24.  58.   9.  24.
   15.  93.  99.  26.  72.  15.  40.  80.   6.  34.  58. 374.  80. 114.
   26.  24.   8. 169.  26.  24.  90.  37. 138.  26.  53.  37.   6.  32.
   24.  26. 107.   3.  24.  89.  24.  60. 208.  24.  16. 329.  63.   6.
  327.  21.  76.   3.   5.  72. 142. 153. 110.  15.  26.  67.  37.  43.
   80.  37.  12.   6. 150.   3. 103.   3.  26.  35.  15.  99.  15. 390.
   30.  19.  10.  63.  24.   9.  67.  26. 140.  73.  35.  22.  58.  99.
```

چون این تابع برای ساخت 100 هش و ماتریس min hash من ان را بعد از یک بار اجرا به صورت numpy در یک فایل ذخیره کرده ام تا در قسمت های بعد بتونم روش کار کنم .

```
In [41]: np.save('signiturehash', b)

In [43]: c=np.load('signiturehash.npy')
```

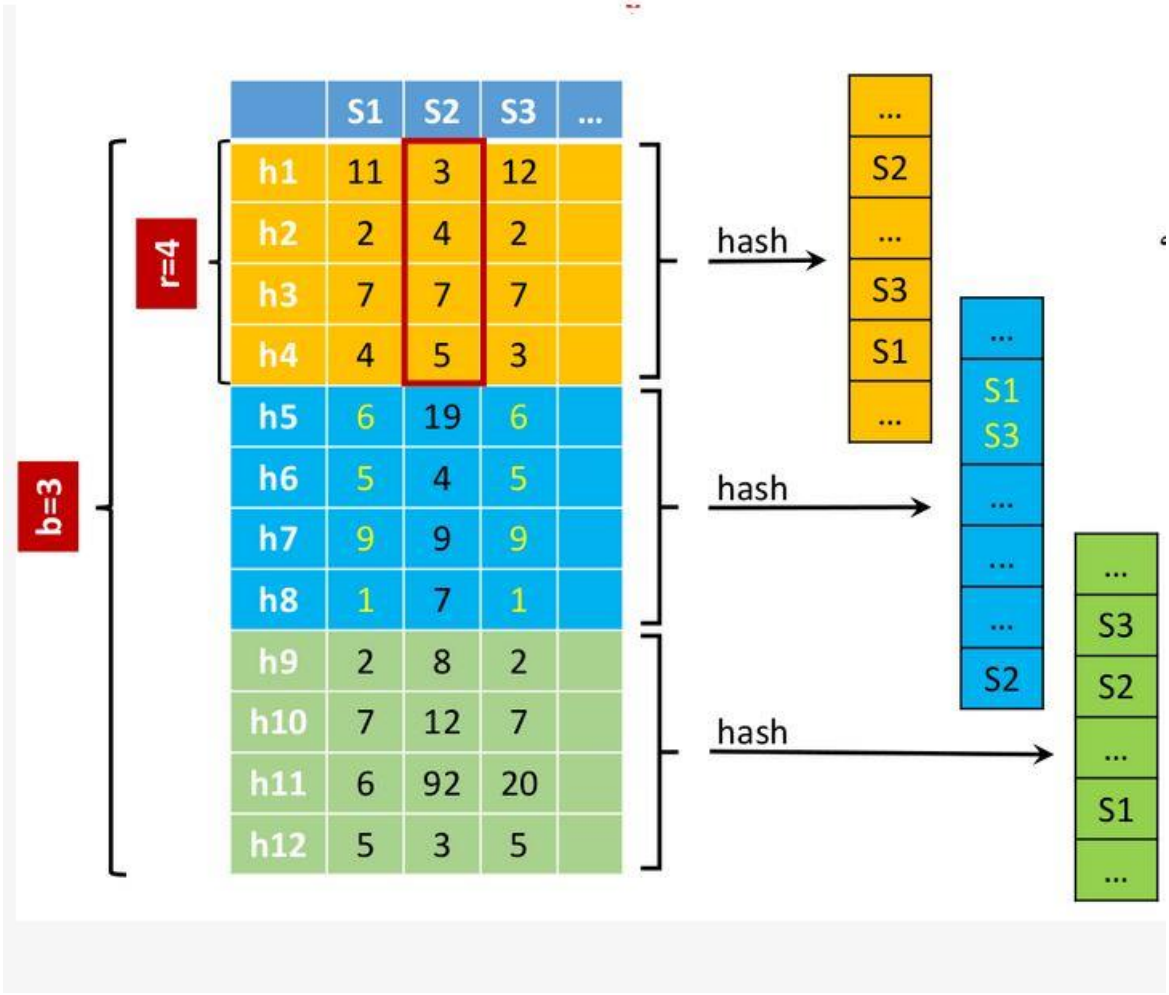
برای محاسبه شباهت ماتریس hash permutation از کد زیر استفاده میکنیم

```
In [68]: #similarity ruie matrix hash ermutation
similarity=0.3
hash_permutation_similarity=[]
for j in range(0,435):
    for k in range(j+1,435):
        count_similar=0
        for permutation in range(0,100):
            if(c[permutation,j]==c[permutation,k]):
                count_similar+=1
        final_similar=count_similar/100
        if(final_similar >= similarity):
            print(str(j) + " and "+ str(k) +" is similarity up to 0.3")
```

```
0 and 14 is similarity up to 0.3
0 and 15 is similarity up to 0.3
1 and 4 is similarity up to 0.3
14 and 15 is similarity up to 0.3
16 and 18 is similarity up to 0.3
17 and 20 is similarity up to 0.3
26 and 53 is similarity up to 0.3
26 and 380 is similarity up to 0.3
53 and 355 is similarity up to 0.3
59 and 75 is similarity up to 0.3
```

و در قسمت اخر برای این که این الگوریتم را که تا اینجا انجام داده دقیق تر شود از LSH استفاده میکنیم .

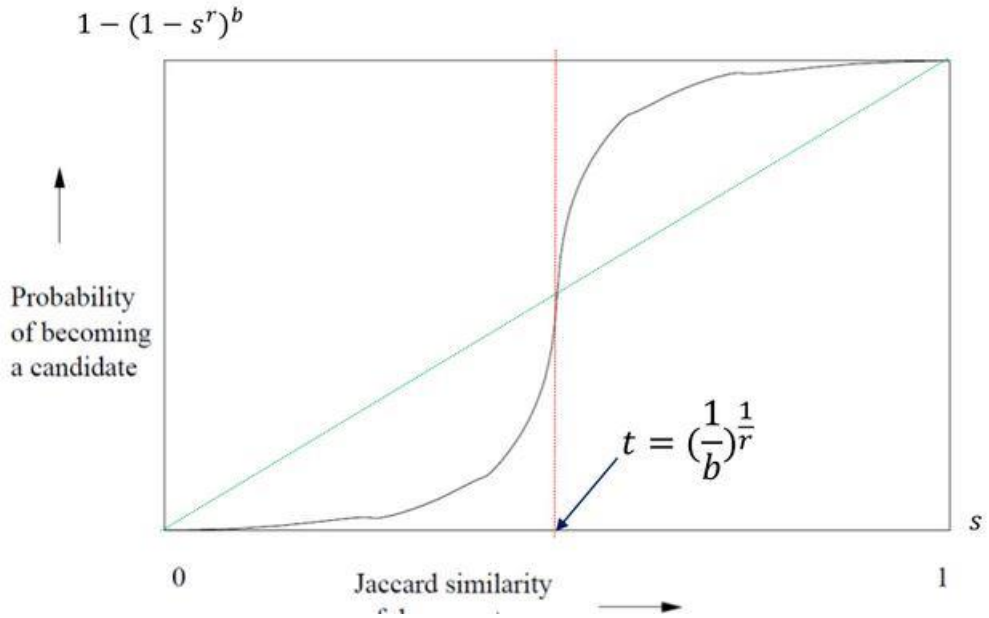
در این روش ما باید ماتریس هش ها را به b بخش r تایی تقسیم کنیم برای مثال شکل زیر نشان دهنده هش 12 تایی است ولی در مثال ما چون 100 هش داریم تقسیم بندی متفاوت است .




طبق شکل بالا من میام هر جدول هش رو به 20 قسمت هر قسمت 5 تایی تقسیم میکنم طبق فرمول LSH موارد زیر به دست می آید .

| $s$ | $1-(1-s^r)^b$ | |
|---|---|---|
| .2 | .006 | |
| .3 | .047 | |
| .4 | .186 | r= 5 |
| .5 | .470 | b= 20 |
| .6 | .802 | |
| .7 | .975 | |
| .8 | .9996 | |

اگر r,b را اعداد بالا بگیریم احتمال کاندید شدن مثل جدول بالا میشود و میبینیم که اگر b را داشته باشیم و میخواهیم که مجموعه ای که شباهت ان ها از t بیشتر باشد کاندید میشود .




بستگی به ما دارد اگر بخواهیم شباهت ما از یک treshould بیشتر باشد انتخاب میکنیم .

با جایگذاری در فرمول بالا t مقدار 0.54 را میدهد پس در مثال ما شباهت هر چه قدر از 0.54 بیشتر باشد انتخاب میشود .

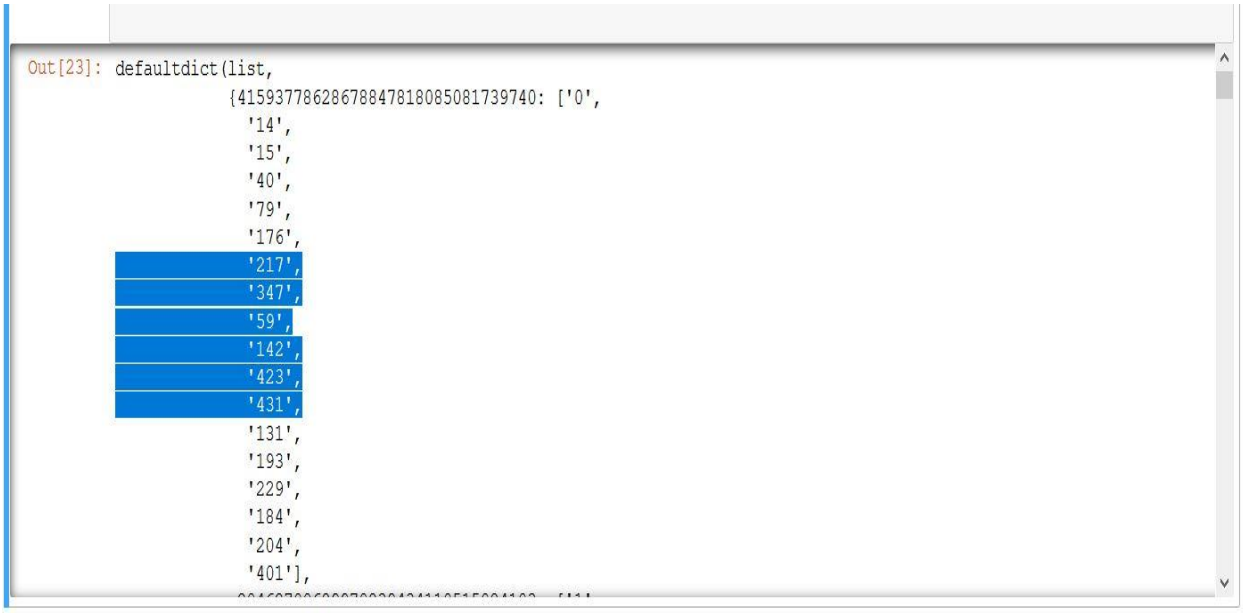
```
In [15]: #LSH Algorithm
         import numpy as np
         from hashlib import sha1
         from collections import defaultdict
         c=np.load('signiturehash.npy')
```

```
In [23]: r=5
         b=20
         # مقدار پایین با فرمول t به دست امده است
         similarity_threshould=0.54
         list_setof_similarity=defaultdict(list)
         c_split_list=np.split(c, 20)
         for list_b in c_split_list:
             for i in range(0,435):
                 count_sum=0
                 for f in list_b[:,i]:
                     count_sum+=(f)
                 h_count_sum=str(count_sum)
                 lsh_hash = int(sha1(h_count_sum.encode("ASCII")).hexdigest(), 16) % (10**32)
                 list_setof_similarity[lsh_hash].append(str(i))


         list_setof_similarity
```

کد بالا نشان میدهد چه داکیومنت هایی با هم مشابه هستند چون هش ان ها در یک خانه قرار گرفته است .

نتیجه نمونه به صورت زیر است :




- الگوریتم خود را با الگوریتم Brute Force برای یافتن داکیومنتهای مشابه مقایسه کنید

الگوریتم Brute force به این صورت عمل میکند که کل 435 داکیومنت رو از روی کل شینگل ها شباهت ژاکارد را حساب میکند و این محاسبه خیلی زمان بر است نسبت به این که ما 100 نمونه شبیه ساز هش را مقایسه کنیم .

مقایسه 65000 شینگل کجا و مقایسه 100 شینگل کجا ولی هر چه تعداد hash ها بیشتر باشد به جواب اصلی که روش Bruteforce هست نزدیک تر است

```
In [58]: #Brute force
similarity=0.3
brute_force_similarity=[]
for j in range(0,435):
    for k in range(j+1,435):
        set_one=set(docs_set_list[i])
        set_two=set(docs_set_list[j])
        num_similarity = len(set_one&set_two)/len(set_one|set_two)
        if(num_similarity >= similarity):
            print(str(j) + " and "+ str(k) +" is similarity up to 0.3")
```

```
336 and 337is similarity up to 0.3
336 and 338is similarity up to 0.3
336 and 339is similarity up to 0.3
336 and 340is similarity up to 0.3
336 and 341is similarity up to 0.3
336 and 342is similarity up to 0.3
336 and 343is similarity up to 0.3
336 and 344is similarity up to 0.3
336 and 345is similarity up to 0.3
336 and 346is similarity up to 0.3
336 and 347is similarity up to 0.3
336 and 348is similarity up to 0.3
336 and 349is similarity up to 0.3
336 and 350is similarity up to 0.3
336 and 351is similarity up to 0.3
336 and 352is similarity up to 0.3
336 and 353is similarity up to 0.3
336 and 354is similarity up to 0.3
336 and 355is similarity up to 0.3
336 and 356is similarity up to 0.3
336 and 357is similarity up to 0.3
336 and 358is similarity up to 0.3
```

الگوریتم brute force به صورت بالا است من برای نمونه شباهت های بالای 0.3 را چاپ کرده ام .

- میدانیم که معمولا طول شینگلها را 9در نظر میگیرند. اگر از شینگلهای بزرگتر یا کوچکتر استفاده کنیم چه تغییری در نتیجه ایجاد میشود.

هر چه شینگل بزرگ تر شود تعداد کل شینگل ها کاهش پیدا کرده و شباهت دقیق تر میشود و هر چه تعداد شینگل ها کاهش پیدا کند تعداد شینگل ها زیاد شده و چون شباهت بین متون در کلمات با طول کم بیشتر میشود دقت ان کمتر میشود .

تابع نوشته شده برای شینگل به طول 9 است میتوانم هر عددی بدهم اما در اخر هم به نتیجه بالا میرسم .

چون سرعت تولید ماتریس hash خیلی کم است .

- نتایج اجرای الگوریتم خود را با تعداد متفاوت توابع هَش (یا همان طول Signature(برای ساخت ماتریسMinHash Signature گزارش کنید

هر چه تعداد min hash signature بیشتر باشد شبیه سازی دقیق تر است ما در این مثال این عدد را 100 در نظر گرفتیم هر چه قدر کمتر از 100 باشد دقت این روش کمتر میشود .

این الگوریتم را با 10 تا min hash signature اجرا کردم و دقت بسیار کمی داد پس برای این که این روش به خوبی جواب دهد باید تعداد minhash signature را افزایش دهیم .