

بسمه تعالی

گزارش فاز سوم پروژه ب

درس الگوریتم های بیوانفورماتیک

تهیه و تنظیم : الهه بدلی (۹۸۲۰۹۰۷۲)

گزارش پیش رو شامل نحوه ی پیاده سازی و نحوه استفاده از کد می باشد.

کد فاز سوم در ادامه فاز دوم نوشته شده است.

خروجی فاز دوم پروژه به صورت زیر بود:

```
1 All_Counts ,All_Presence_Absence = phase2_main()

-> Read Genomes ...
Genomes Readed!
-> Split Genomes to Blocks ...
Blocks Created!
-> Start Count and Filter kmers ...
Kmer Count and Filtering: Done!
```

که شامل دو دیکشنری All\_Counts و All\_Presence\_Absence می باشد.

در این مرحله ابتدا دایرکتوری comparison را برای ذخیره نتایج مقایسه دو به دو ژنوم ها می سازیم

```
1 mkdir 'compration'
```

```
1 %cd 'compration'
```

```
/content/drive/My Drive/phase2Algo/compration
```

تابع `calculate_metric` وظیفه اجرای فاز سوم بخش ۱ را برعهده دارد.

```
import numpy as np
def calculate_metric(All_Counts ,All_Presence_Absence ):
    k = 1
    cmp_list = []
    similarity_dict = {}
    for genome1 in All_Counts:
        k1 = int(genome1[genome1.index('k')+1:genome1.index('_')])
        for genome2 in All_Counts:
            k2 = int(genome2[genome2.index('k')+1:genome2.index('_')])
            K = k1 * k2
            if (k1 == k2) and (K < 150) and (genome1 != genome2) and ((genome1 , genome2) and (genome2 , genome1) not in cmp_list):
                cmp_list.append((genome1 , genome2))
                cmp_list.append((genome2 , genome1))
                k+=1
                bit_blocks1 = All_Presence_Absence[genome1]
                bit_blocks2 = All_Presence_Absence[genome2]

                w, h = len(bit_blocks1), len(bit_blocks2)
                dotplot = np.array([[0 for x in range(h)] for y in range(w)] )
                for i in range(w):
                    for j in range(h):
                        dotplot[i][j] = min(bit_blocks1[i].count() , bit_blocks2[j].count())
                this_two_genomes = str(genome1+str('&')+genome2)+'.txt'
                similarity_dict.setdefault(k1 , []).append( [genome1 , genome2 , np.sum(dotplot)])
                similarity_dict.setdefault(k1 , []).append( [genome2 , genome1 , np.sum(dotplot)])

                np.savetxt(this_two_genomes , dotplot.astype(int),fmt="%i")

    return similarity_dict
```

Activate Window:  
Go to PC settings to ac

در این تابع ابتدا دو تا ژنوم با شرایط زیر انتخاب می کنیم که با دو فور این دو تا ژنوم را جدا می کنیم حال  
شروط زیر را داریم:

۱. تحت  $k$  یکسان  $k$ merهایشان جدا شده باشد.

۲. با توجه به محدودیت محاسباتی که برای  $k = 32$  و  $k = 24$  داشتیم بنابراین اطلاعات بودن یا نبودن  
 $k$ mer ها برای 5 تا از  $k$  ها را داریم بنابراین محاسبات را برای  $k$  های ۳ و ۷ و ۹ و ۱۱ انجام دادیم. برای انتخاب  
نکردن  $k = 24$  و ۳۲ شرط  $K < 150$  را گذاشتیم که  $K$  حاصل ضرب  $k_1$  و  $k_2$  هست که در بیشترین حالت یک  
ژنوم با  $k = 11$  با ژنوم دیگری با  $k = 11$  مقایسه میشود که حاصل ضربشان ۱۲۱ است اما جهت حد بالا ۱۵۰  
گذاشتیم. (استدلال پیچیده ای بود D):

۳. هر ژنوم را با خودش مقایسه نمی کنیم

۴. وقتی ژنوم ۱ با ژنوم ۲ مقایسه شد دیگر نیازی به مقایسه ی ژنوم ۲ با ژنوم ۱ نداریم بنابراین این دو تا را در  
`cmp_list` اضافه میکنیم که مقایسه اضافه صورت نگیرد.

( در کل ۲ از ۴۱ ضربدر ۵ حالت بررسی شده که میشه ۴۱۰۰ حالت مختلف).

در ادامه اطلاعات بودن یا نبودن kmer ها را برای این دو ژنوم از دیکشنری All\_presence\_Absence جدا میکنیم که حاوی اطلاعات کلیه ژنوم هاست. (با عنوان bit\_block1 و bit\_block2)

بعد یه آرایه مشابه dotplot را با صفر initialize میکنیم. در درایه های این آرایه باید نتیجه محاسبه متریک برای دو ژنوم را ذخیره کنیم.

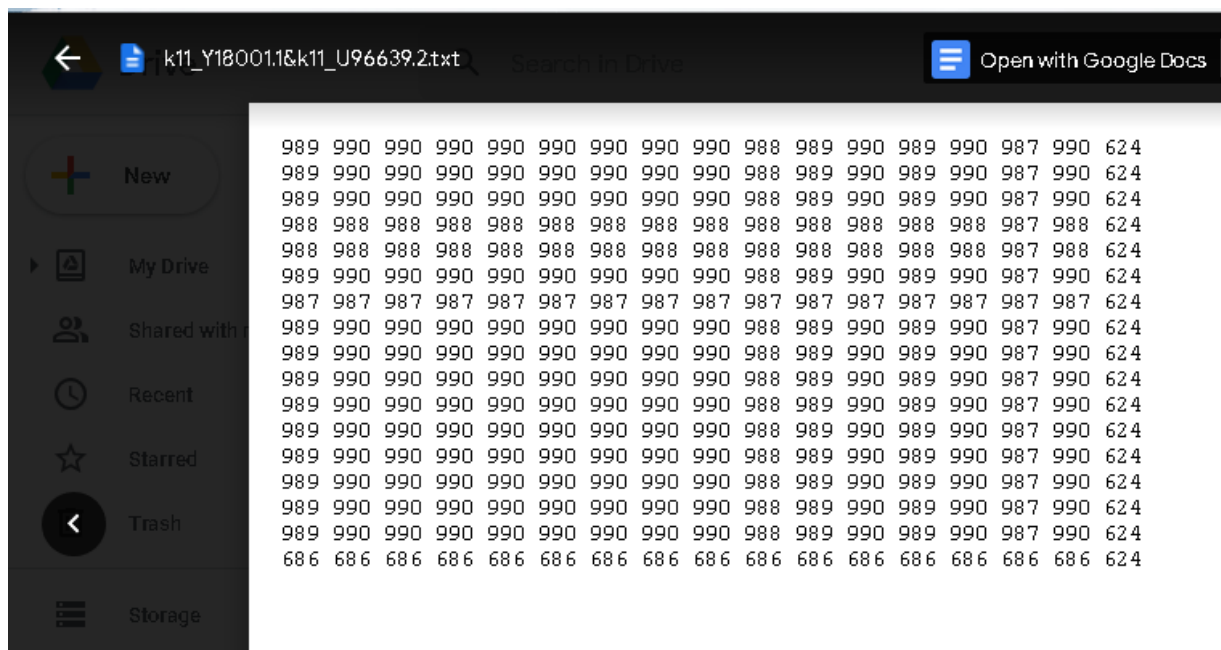
متریک من متریک 0 بود.

با توجه به اینکه من از bitarray برای نگهداری اطلاعات بودن یا نبودن kmer ها استفاده کرده بودم جاهایی که kmer بود ۱ شده بود و در صورت نبودن 0 باقی مانده بود. بنابراین ضرب درایه به درایه اطلاعات دو بلاک سپس جمع کردنشان مثل min گرفتن بین جمع ۱ های هر بلاک بوده است. D:

بعد از محاسبه دات پلات باید کل درایه های را جمع میکردیم و به عنوان معیار شباهت این دو ژنوم ذخیره میکردیم که این کار در دیکشنری similarity\_matrix انجام شده که حاوی لیست هایی هست که در هر لیست ژنوم ۱ ژنوم ۲ سپس میزان شباهت آورده شده است. similarity\_matrix حاوی اطلاعات شباهت کل دیتاست است البته بر حسب هر k ای که داریم. یعنی similarity\_matrix کلید های ۳ ۵ ۷ ۹ ۱۱ را دارد و مثلاً به ازای ۳ اطلاعات شباهت دو به دوی ژنومهایی با k ۳ را دارا می باشد.

در نهایت در فایلی با نام دو ژنوم ماتریس دات پلات شان ذخیره شده است.

برای مثال برای دو ژنوم زیر داریم :



989	990	990	990	990	990	990	990	990	990	988	989	990	989	990	987	990	624
989	990	990	990	990	990	990	990	990	990	988	989	990	989	990	987	990	624
989	990	990	990	990	990	990	990	990	990	988	989	990	989	990	987	990	624
988	988	988	988	988	988	988	988	988	988	988	988	988	988	988	987	988	624
988	988	988	988	988	988	988	988	988	988	988	988	988	988	988	987	988	624
989	990	990	990	990	990	990	990	990	990	988	989	990	989	990	987	990	624
987	987	987	987	987	987	987	987	987	987	987	987	987	987	987	987	987	624
989	990	990	990	990	990	990	990	990	990	988	989	990	989	990	987	990	624
989	990	990	990	990	990	990	990	990	990	988	989	990	989	990	987	990	624
989	990	990	990	990	990	990	990	990	990	988	989	990	989	990	987	990	624
989	990	990	990	990	990	990	990	990	990	988	989	990	989	990	987	990	624
989	990	990	990	990	990	990	990	990	990	988	989	990	989	990	987	990	624
989	990	990	990	990	990	990	990	990	990	988	989	990	989	990	987	990	624
989	990	990	990	990	990	990	990	990	990	988	989	990	989	990	987	990	624
989	990	990	990	990	990	990	990	990	990	988	989	990	989	990	987	990	624
989	990	990	990	990	990	990	990	990	990	988	989	990	989	990	987	990	624
686	686	686	686	686	686	686	686	686	686	686	686	686	686	686	686	686	624

برای استفاده از ابزار مگا احتیاج داریم که `distance_matrix` رو برای دو گونه داشته باشیم.  
بنابراین در دایرکتوری جداگانه ای اطلاعات فاصله را ذخیره میکنیم.

```
1 %cd ..
```

```
/content/drive/My Drive/phase2Algo
```

```
1 mkdir 'distances'
```

```
1 %cd 'distances'
```

```
/content/drive/My Drive/phase2Algo/distances
```

برای بدست آوردن ماتریس فاصله دو تابع زیر را داریم:

```
def dist_matrix(similarity_dict):  
    import pandas as pd  
    df = pd.DataFrame(similarity_dict)  
    df = df.pivot(index=0, columns=1, values=2).fillna(0)  
    names = df.index  
    df = np.array(df)  
    df = np.max(df) - df  
    np.fill_diagonal(df,0)  
    return np.tril(df) , names
```

```
def Create_Distance_Matrixs(similarity_dict):  
    for k in similarity_dict:  
        distance_matrix , names = dist_matrix(similarity_dict[k])  
        print("distance matrix of k =" + str(k) + " created.")  
        f_out = open(str("k"+str(k)+"_names.txt") , 'w')  
        for name in names:  
            f_out.write('#'+name+'\n')  
        f_out.close()  
        np.savetxt(str("k"+str(k)+"_txt") , distance_matrix.astype(int) , fmt="%i")
```

که ابتدای `Create_Distance_Matrixs` شروع ب کار می کند.

`Simliraty_matrix` که اطلاعات شباهت دو ژنوم است را دریافت میکند و به ازای هر `k` ماتریس فاصله کل دیتاستش را می سازد.

ساختن ماتریس فاصله دیتاست هر  $k$  ای در تابع `dist_matrix` انجام می شود. این تابع ابتدا فرمت دیکشنری را به دیتافریم تبدیل میکند سپس به `reshape` انجام میدهیم که فرمت ماتریسی داشته باشد و درایه ی  $i$ ام  $j$ ام ش شباهت  $i$  به  $j$  نوم  $j$  را نشان میدهد.

از آنجایی که برای ابزار مگا احتیاج به نام  $i$  نوم ها داشتیم این کار را در دیتافریم انجام دادیم که با ایندکس به نام ها دسترسی داشته باشیم. پس از دریافت نام های  $i$  نوم ها حال کل دیتافریم را به آرایه تبدیل می کنیم. ابزار مگا از روی ماتریس فاصله درخت را می سازد. ما تالینجا ماتریس شباهت را ساختیم. حال درایه ها را از `max` کم میکنیم تا ب ماتریس فاصله تبدیل شود.

از طرفی ابزار مگا ماتریس فاصله را به صورت پایین مثلثی دریافت میکند. بنابراین ماتریس پایین مثلثی و نام ها را برمیگردانیم و نام ها و ماتریس را در فایل های جداگانه ای ذخیره میکنیم.

البته راه بهتر این بود یک جا و خط به خط ذخیره میکردم اما چون کل ماتریس را با `np.savetxt` ذخیره کردم امکان ذخیره همزمان نام در یک فایل نبود. برای همین پس از خروجی گرفتن ها به صورت دستی نام های  $i$  نوم ها را به فایل ماتریس فاصله اضافه کردم تا به فرمت مورد پذیرش ابزار مگا در بیاید. سپس فایل `txt` را به فرمت `meg`. تبدیل کرده و درخت هر ماتریس را رسم کردم. که نتایج رسم در ۵ فایل `pdf` جداگانه در فولدر `distance` آورده شده است.

نتیجه : با بررسی درخت ها متوجه می شویم هر چقدر  $k$  بیشتر شده دقت دسته بندی بهتر شده و دو گونه مشابه تر در یک دسته قرار گرفتند مخصوصا برای ۹ و ۱۱ نتایج نزدیک ب هم هست.

فقط بخشی از درخت را در ادامه آورده ام :

