**算法分析与设计大作业实验报告**

**孟妍廷 2015202009**

1. **实验目的和要求**

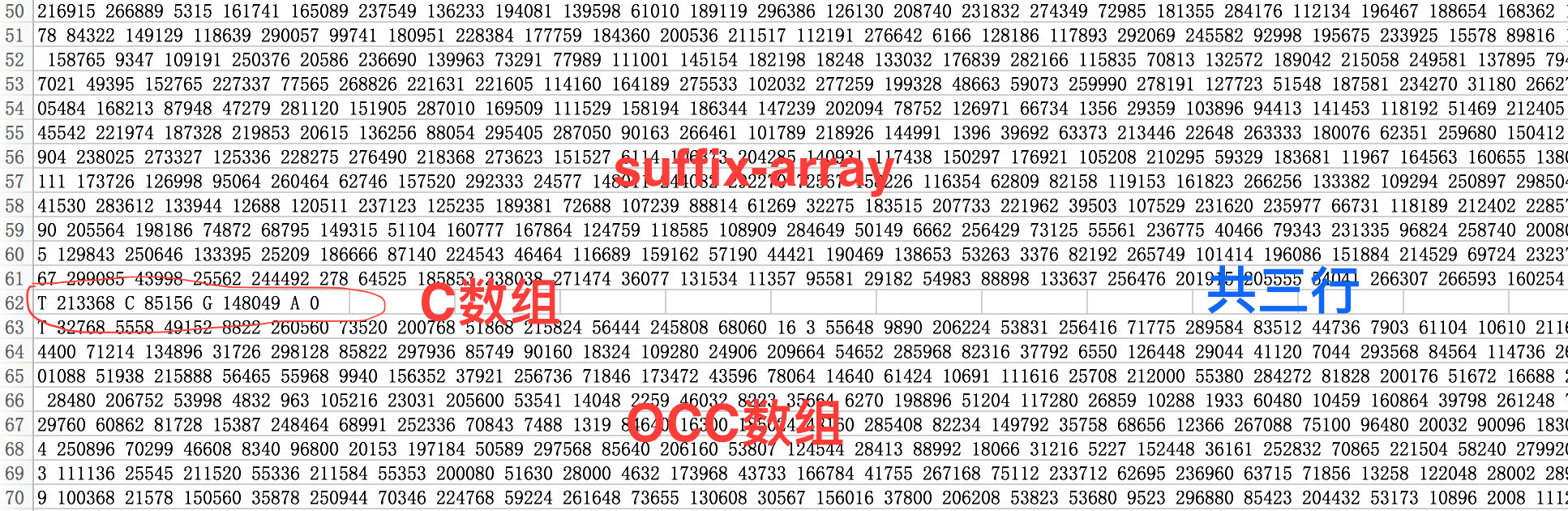
1.实验目的：用O(m)时间复杂度找出一个长度为m的短字符串在一个长度为n的长字符串中的精确匹配，其中，m远小于n。

2.实验要求：要求基于BWT压缩和FM索引技术的序列匹配，用Burrows-Wheeler transform算法解决该问题。实验提供三个数量级为900M的长串和三个取自长串前200000个字符的短串。

1. **算法简述**

**前提：**

由于长串的数量级是900M，一次取出的计算量太大并且产生的索引文件占用的存储空间过大，因此每一次从长串中取300000个字符来生成索引，并保证两次取出的字符有200000个字符的重叠。同时，每300000个字符生成的索引存在一个文件中，也就是说由于每次有100000个字符是未被之前的字符串所覆盖的因此共生成9000个索引文件。索引文件的结构如下：



**1.第一步：建立BWT结构——不生成矩阵创建索引：**

若按照严格的Burrows-Wheeler transform 算法建立FM索引，首先需要将长串取出，每次左移一位生成新的字符串。因此需要O(n^2)的矩阵来存储，十分浪费空间。故我选择了不生成矩阵创建索引的方法：

首先将长度为n的长串S复制一遍生成长度为2n的字符串SS，然后建立一个长为n的int型数组A，使得A[i]=i。接下来对A数组做快速排序，但是将PARTITION中的比较函数修改为：对于当前比较值x,y，比较S.substr(x, n)和S.substr(y, n)的字典序。这样就不需要存储矩阵，缩小了空间复杂度，最后排序后的A数组就是FM-index需要的suffix-array。

**2.第二步：利用suffix-array建立C和OCC并写入本地**

可知suffix-array的值i在原始长串中取到的字符就是就是左移i次后的字符串的最后一个字符，而i-1在原始长串中取到的字符就是就是左移i次后的字符串的第一个字符。因此利用suffix-array即可还原出左移过程中的第一列和最后一列数组，同时也可以计算出C和OCC。

为了节约运算时间和存储空间，对OCC数组进行部分存储，每隔16个位置计算一次OCC，即计OCC[X,16]，OCC[X,32],OCC[X,48]....之后查询时先找离得最近的存储在计算真实值。

把C和OCC以及suffix-array数组写入索引文件，每个占一行，方便之后读取。

**3.第三步：进行精确匹配**

首先读取索引文件，还原出suffix-array，C和OCC。

由于最后一列的第 i 个字符在第一列第 i 个字符前面，故对于待匹配的短串，从后往前进行匹配。先找最后一个字符，利用C确定这个字符在第一列中出现的范围，在这个范围中根据OCC找最后一列是倒数第二个字符x的位置，由于我的OCC此时是每隔16个位置存储一个，因此先找到最接近的位置，再以这个位置为上界计算真实值。得到真实值count之后，得到以倒数第一个字符和倒数第二个字符为开头，倒数第三个字符结尾的位置为count+C[x]。匹配之后得到一个字符串，查看该字符串与待匹配的短串是否相同，若不同，则说明找到的倒数第二个字符的位置不正确（由于满足第一个字符是短串最后一个字符，最后一个字符是短串倒数第二个字符的位置并不唯一），此时回溯，找一个新的位置重新匹配，直到得到的字符串与待匹配短串相同，输出它在长串中的位置，匹配成功。

1. **时间空间复杂度分析（具体到算法中每一步的复杂度）**

**设短串长度为m，长串为n**

**第一部分：生成索引**

**读入 时间复杂度O(1),空间复杂度O(300000)(相当于 O(m),3000000<<900M)**

**快速排序时间复杂度 O(mlgm),空间复杂度O(2m)**

**计算C时间复杂度O(m),空间复杂度O(8)(四中字符+四个数量)**

**计算OCC时间复杂度O(m),空间复杂度O(4+300000/16\*4) =O(75004)**

**将索引存到本地时间复杂度O(m)，空间复杂度O(1)**

**第二部分：精确匹配**

**读取索引时间复杂度O(1),空间复杂度O(300000+8+75004)=O(m)**

**还原suffix-array时间复杂度O(1),空间复杂度O(m)**

**还原C时间复杂度O(1),空间复杂度O(8)**

**还原OCC 时间复杂度O(m),空间复杂度O(75004)**

**找短串最后一个字符的范围时间复杂度O(1),空间复杂度O(m)**

**确定满足倒数第二个字符在最后一列的位置时间复杂度O(m),空间复杂度O(m)**

**精确匹配时间复杂度:单次O(m),回溯O(m^2),空间复杂度O(m)**

**四、实验源代码**

**生成索引：**

**import os**

**#python的sorted函数直接能够实现将字符串按照字典序排序**

**def Move\_To\_Right(S):**

**SS=S#记录初始状态**

**#result=[]**

**#result.append(SS)**

**result={}**

**result[SS]=0**

**#suffix\_array.append(0)**

**for j in range(len(SS)-1):**

**temp = SS[0]**

**SS = SS[1:] + temp**

**#result.append(SS)**

**#suffix\_array.append(j+1)**

**result[SS]=j+1**

**return result**

**#同时也要注意空间复杂度**

**#一开始想的是根据results修改suffix\_array，时间复杂度较大**

**def BWT():**

**suffix\_array=[]**

**results = []**

**result = Move\_To\_Right(S)**

**results = sorted(result.keys())#但是要根据results修改suffix\_array**

**for i in results:**

**suffix\_array.append(result[i])**

**#print(results,suffix\_array)**

**result.clear()**

**results.clear()**

**return suffix\_array**

**#不生成矩阵创建索引**

**def exchange(A,a,b):**

**temp = A[a]**

**A[a]=A[b]**

**A[b]=temp**

**def QUICKSORT(S,suffix\_array,p,r):**

**if p<r:**

**q=PARTITION(S,suffix\_array,p,r)**

**QUICKSORT(S,suffix\_array,p,q-1)**

**QUICKSORT(S,suffix\_array,q+1,r)**

**def PARTITION(S,suffix\_array,p,r):**

**n=len(suffix\_array)**

**x=suffix\_array[r]**

**i=p-1**

**for j in range(p,r):**

**if S[suffix\_array[j]:suffix\_array[j]+n-1]<S[x:x+n-1]:**

**i = i+1**

**exchange(suffix\_array,i,j)**

**exchange(suffix\_array,i+1,r)**

**return i+1**

**def BWT1(SS):**

**suffix\_array=[]**

**for i in range(int(len(SS)/2)):**

**suffix\_array.append(i)**

**QUICKSORT(SS,suffix\_array,0,len(suffix\_array)-1)**

**print("FM-index:",suffix\_array)**

**return suffix\_array**

**#建立C和OCC数组——当前想法有两层循环**

**#根据suffix\_array就可以得到F和L**

**#默认只有AGCT四种字符**

**def Build\_Structure(S):**

**SS = S+S**

**F=[]**

**L=[]**

**suffix\_array=[]**

**suffix\_array=BWT1(SS)**

**for i in suffix\_array:**

**F.append(S[i])**

**L.append(S[i-1])**

**#print(F,L)**

**C={}#C表示的是在第一列的字符串,比字符X小的有多少个**

**OCC={}#OCC的的元素OCC[X][i]表示的意思是在最后一列字符串中,前i个字符里有几个字符X**

**#首先求C**

**counta=0#记录agct个数**

**countg=0**

**countc=0**

**#countt=0**

**for i in F:**

**if i == 'A':**

**counta += 1**

**elif i == 'C':**

**countc += 1**

**elif i == 'G':**

**countg += 1**

**C['A']=0**

**C['C']=counta**

**C['G']=counta+countc**

**C['T']=counta+countc+countg**

**#接下来求OCC**

**OCC['A']={}**

**OCC['G']={}**

**OCC['C']={}**

**OCC['T']={}**

**i=0**

**counta=0#记录agct个数**

**countg=0**

**countc=0**

**countt=0**

**while i < len(L):**

**#i=i+16**

**for j in L[i:i+16]:**

**if j=='A':**

**counta += 1**

**elif j=='G':**

**countg += 1**

**elif j=='C':**

**countc += 1**

**elif j=='T':**

**countt += 1**

**i=i+16**

**OCC['A'][i]=counta**

**OCC['G'][i]=countg**

**OCC['C'][i]=countc**

**OCC['T'][i]=countt**

**F.clear()**

**L.clear()**

**return suffix\_array,C,OCC**

**def main():**

**file = open('s.txt','r')**

**file.read(10)#前面10个是记录长度的和制表符**

**begin=100010**

**ff=1**

**S=file.read(300000)**

**while S:**

**file.seek(begin,0)**

**begin += 100000**

**print(S)**

**if str(0xFF) in S:**

**S.remove(str(0xFF))**

**suffix\_array,C,OCC=Build\_Structure(S)**

**filename = str(ff)+'.txt'**

**f = open(filename,'w')**

**for i in suffix\_array:**

**f.write(str(i)+' ')**

**f.write('\n')**

**for i in C:**

**f.write(i+' '+str(C[i])+' ')**

**f.write('\n')**

**for i in OCC:**

**f.write(i+' ')**

**for j in OCC[i]:**

**f.write(str(j)+' '+str(OCC[i][j])+' ')**

**f.write('\n')**

**ff += 1**

**f.close()**

**S=file.read(300000)**

**file.close()**

**if \_\_name\_\_ == "\_\_main\_\_":**

**main()**

**精确匹配：  
#根据存入本地的索引生成精确匹配**

**#注意OCC是每隔16个存一个，最后一个可能超出了字符串的长度，先找到最近的位置再精确匹配**

**import re**

**import math**

**import datetime**

**def FM\_Reduction(line):#还原FM-index**

**suffix\_array = re.split(r' ',line)**

**#print(suffix\_array[0])**

**suffix\_array.remove('\n')**

**print(len(suffix\_array))**

**#suffix\_array.remove('')**

**return suffix\_array**

**def C\_Reduction(line):#还原C**

**C={}**

**temp = re.split(r' ',line)**

**i=0**

**while i<len(temp)-1:**

**C[temp[i]]=int(temp[i+1])**

**i += 2**

**temp.clear()**

**return C**

**def OCC\_Reduction(line):#还原OCC**

**OCC={}**

**temp = re.split(r' ',line)**

**length = 2\*int(300000/16) #37500**

**#print(length)**

**i = 0**

**while i<len(temp)-1:**

**OCC[temp[i]]={}**

**#print(temp[i])**

**j=i**

**i += 1**

**while i<=j+length:**

**OCC[temp[j]][int(temp[i])]=int(temp[i+1])**

**i += 2**

**i = j+length**

**i += 1**

**temp.clear()**

**return OCC**

**def PRECISION(beg,sit,C,suffix\_array,ff,aim):#根据最近的位置来计算精确的OCC**

**f = open(r"s.txt")**

**f.read(10)**

**f.seek((ff-1)\*100000,1)**

**S = f.read(300000)**

**f.close()**

**L=[]**

**for i in suffix\_array:#取出的suffix是str类的**

**if i is not '\n':**

**L.append(S[int(i)-1])**

**#print(F,L)**

**res = -1**

**print("PRECISION-beg",beg)**

**for i in range(beg,sit):**

**if L[i]==aim:**

**res=i#找到第一个即可**

**break**

**print("PRECISION-res",res)**

**count = 0**

**for i in range(0,res):#求在他之前有几个他**

**if L[i] == aim:**

**count += 1**

**count = C[aim]+count**

**print("PRECISION",aim,count)**

**return count,L[count]**

**def Match\_Second(res,ff,cur,S,R,suffix\_array,C):**

**f = open(r"s.txt")**

**f.read(10)**

**f.seek((ff-1)\*100000,1)**

**SS = f.read(300000)**

**f.close()**

**aim = S[cur]**

**L=[]**

**R.append(aim)**

**for i in suffix\_array:#取出的suffix是str类的**

**L.append(SS[int(i)-1])**

**#print(F,L)**

**count = 0**

**for i in range(0,res):#求在他之前有几个他**

**if L[i] == aim:**

**count += 1**

**print(res,aim,cur,count,C[aim])**

**#print(aim,count)**

**L.clear()**

**#print(aim,res,cur)**

**count = C[aim]+count**

**print(count)**

**print("当前位置",count,suffix\_array[count])**

**if cur == 0:**

**print("该子串取自原长串的第",suffix\_array[count],"位置")**

**return**

**Match\_Second(count,ff,cur-1,S,R,suffix\_array,C)**

**def Match\_First(S,C,OCC,suffix\_array,ff):#匹配**

**R=[]**

**cur = len(S)-1**

**#print(S[0])**

**rangea=[0,C['C']]**

**rangec=[C['C'],C['G']]**

**rangeg=[C['G'],C['T']]**

**ranget=[C['T'],300000]**

**r = []**

**if S[cur] == 'A':**

**r = rangea**

**elif S[cur] == 'C':**

**r = rangec**

**elif S[cur] == 'G':**

**r = rangeg**

**elif S[cur] == 'T':**

**r = ranget**

**flag = False #判断是否匹配成功的指示变量**

**i=0**

**while i < r[1]:**

**if i == 0:**

**sit = math.ceil((i+1)/16)\*16**

**else:**

**sit = math.ceil(i/16)\*16**

**i=i+17**

**if OCC[S[cur-1]][sit]>0:#要进一步精确**

**print("sit",S[cur-1],sit,OCC[S[cur-1]][sit])**

**beg = sit-16**

**while beg < sit:**

**print("beg",beg,"cur",cur)**

**res,judge=PRECISION(beg,sit,C,suffix\_array,ff,S[cur-1])**

**if res == -1:**

**continue**

**if(judge==S[cur-2]):**

**Match\_Second(res,ff,cur-2,S,R,suffix\_array,C)**

**R.reverse()**

**R.append(S[198])**

**R.append(S[199])**

**#print(R)**

**R=''.join(R)**

**if R == S:**

**print("匹配成功")**

**print(R)**

**return**

**else:**

**beg += 1**

**print(R)**

**R=[]**

**else:**

**beg += 1**

**return**

**def main():**

**#S一行200个**

**S = 'TGCAATTTCTACTATAGTTACTCATTAAGTTATCTAGTAAATTACCATGTAATGTATAAGCTGTGGAATTAATTCTCAGTTACATAGCATTCACCATGTAATTACTAAACAGTCCCCATCATTACAGCTTTCTAGCCTATTATGAAAAGACCAGAAACATAATTTCAATTACAAAGCCCCTGTTGGAACCGAAAGGGTTT'**

**ff=1**

**filename=str(ff)+".txt"**

**f = open(filename,"r")**

**line = f.readline()**

**suffix\_array = FM\_Reduction(line)**

**for i in range(len(suffix\_array)):**

**if int(suffix\_array[i]) == 0:**

**print(i)**

**line = f.readline()**

**C = C\_Reduction(line)**

**line = f.readline()**

**OCC=OCC\_Reduction(line)**

**f.close()**

**Match\_First(S,C,OCC,suffix\_array,ff)**

**if \_\_name\_\_ == "\_\_main\_\_":**

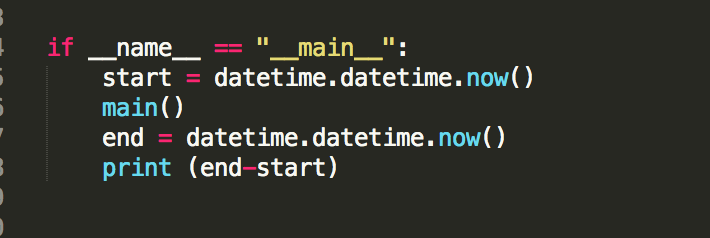
**start = datetime.datetime.now()**

**main()**

**end = datetime.datetime.now()**

**print (end-start)**

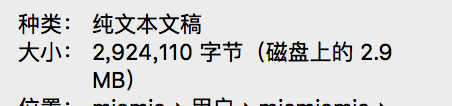
**计算时间：**

****

**五、实验结果（存储空间、查询时间以及结果截图）**

**1.存储空间：**

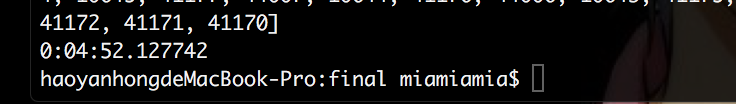
单个部分存储的索引：

****

共有9000个文件，因此总的空间约为30G

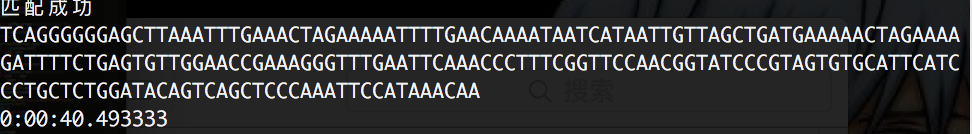
**2.查询时间：**

生成索引时间：

****

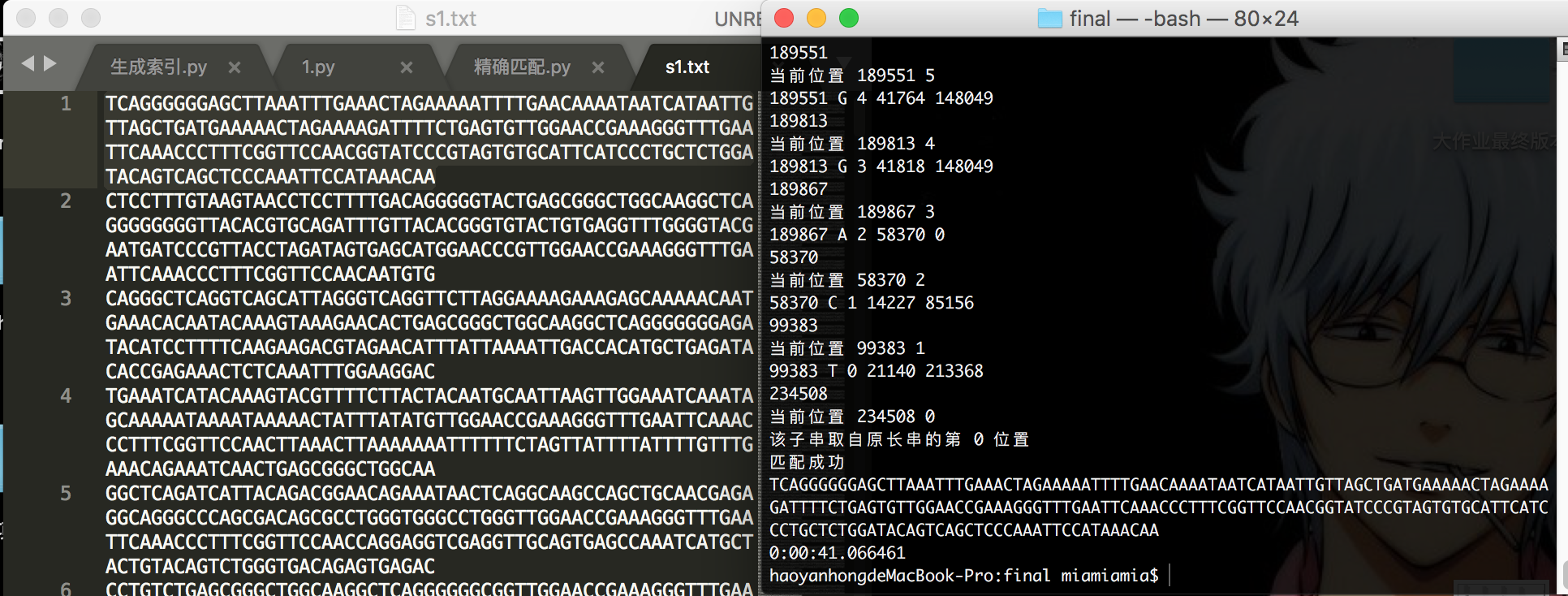
4分52秒

精确匹配时间：

****

40秒

**3.结果截图**

匹配短串第1行——结果为从位置0开始

匹配短串第10行——结果为从位置200\*(10-1)=1800开始



匹配短串第77行——结果为从位置200\*(77-1)=15200开始



匹配短串第401行——结果为从位置200\*(401-1)=8000开始

