哈尔滨工业大学计算机科学与技术学院

实验报告

课程名称：生物信息学

课程类型：专业必修课

实验项目名称： 基因组测序片段比对与映射系统设计与实现

实验题目：基因组测序片段比对与映射系统设计与实现

班级：

学号：

姓名：

|  |  |  |
| --- | --- | --- |
| 设计成绩 | 报告成绩 | 指导老师 |
|  |  |  |

1. 实验目的

给定基因组序列（大肠杆菌E.Coli）作为参考基因组，给定一定数量的来自大肠杆菌的基因组测序片段作为输入，建立算法和系统完成对基因组测序片段的比对，重构片段位置和基因组片段与参考序列的差异

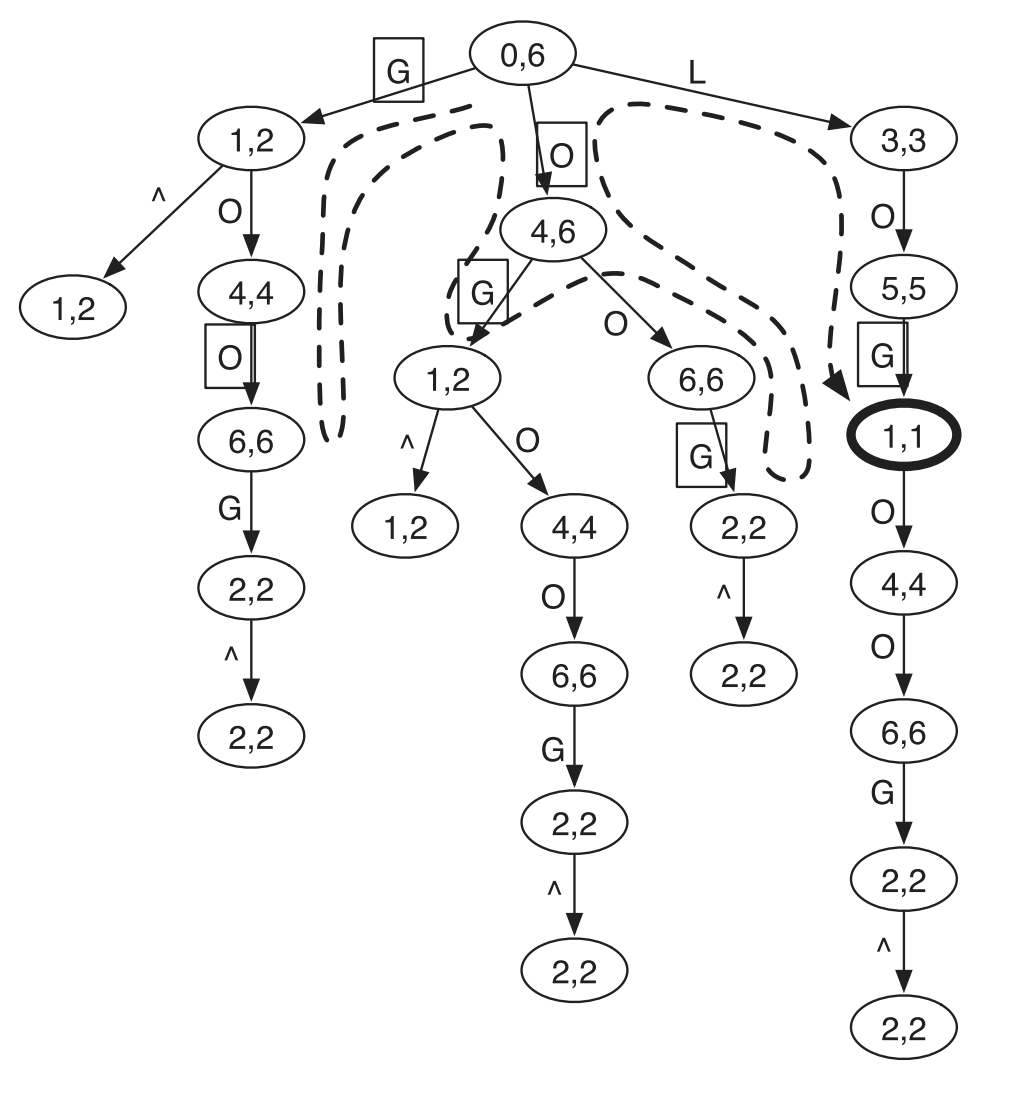
1. 实验原理

本次实验选择了BWA算法进行实现，BWA是一种基于BWT索引结构且性能十分优秀的序列比对软件，主要应用二代测序后大量短小片段与参考基因组之间的定位比对，能够以较小的空间代价和时间代价获得较为优秀的比对结果。

**算法原理：**

2.1 前缀树与字符串匹配

前缀树：字符串的prefix trie是一棵树，每条边都标有一个符号，从叶子到根的路径上的边符号的字符串连接，给出了字符串的唯一前缀。该理论也可以应用于字符串的后缀树。前缀树可用于查找某字符串W是否是该字符串的精确匹配子串，仅需要O（|W|）时间即可找到。

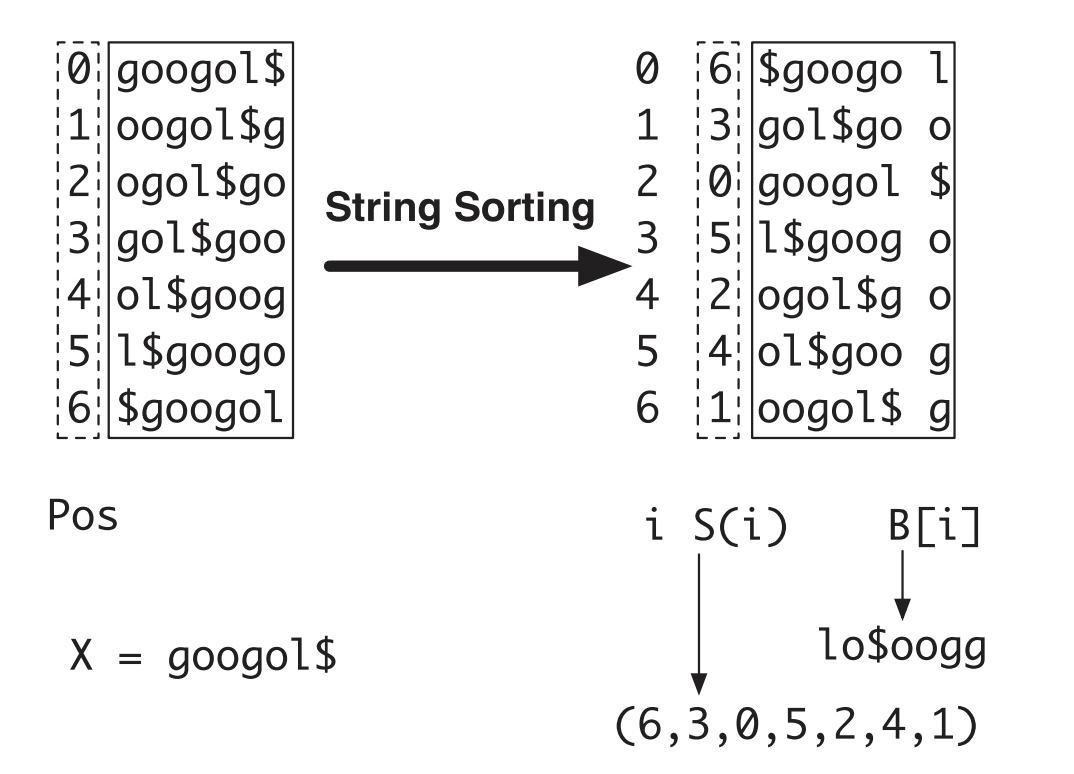


上图即为字符串GOOGOL的前缀树，图中加粗的1.1节点代表匹配上了前缀GOL

2.2 BWT数据结构索引

令∑为字母表，标志＄为字母表中最小的符号，BWT将源文本字符串置换成一种可逆的压缩变换形式。变换过程如下：

1. 将源文本T后添加标志＄得到T＄
2. 文本T’作循环移动，每次移动都将文本最后一个字符移到最前面，最后组成一个文本矩阵，如下图左侧矩阵所示
3. 对文本矩阵的字符串按照字典序进行排序，最后得到的矩阵是越小的字符串排的越靠前的矩阵，如下图右侧所示。
4. 抽取排序后的矩阵的最后一列形成字符串L，矩阵中每一列都是源文本的一个置换，L也是一个置换。该矩阵的第一列也是一个特殊的列，标记为F。排序后的矩阵可以看作源文本进行后缀排序形成的，故可用利用后缀数组的特性进行文本查找。



BWT特性：

1. 进行变换后只保存字符串L，L可以直接映射出M，故变换后占用的内存依然是和源文本一样的
2. F和L中同行的两个字符在原字符串中必然相邻，且L中的字符串必在F中的字符的前面。
3. L中会有同一个字符的多次出现2，它们在L中的顺序和F中的顺序是相同的，例如在L中第一个出现的A在F中也必然是第一个出现的A
4. L中的每一个字符都可以唯一的在F中找到，且在T中的位置是与之相对应的，同一个字符在T中的位置仍然保持不变
5. 可以通过L和F还原源字符串T

根据L和F重建源文本T：  
 令L[i]=c，r[i]为行M[i]在所有以字符c结束的行中的次序，令M中以c开始的第ri行为行M[j],那么第一列F中与L[i]相关的字符就定位在F[j]，即LF映射，LF[i]=j

1. 计算数组C[1…|∑|]，其中C[c]保存字符(＄，1…c-1在文本T中出现次数之和，C[c]+l则表示字符c在F中出现的第一个位置
2. 计算LF映射数组，LF[i]=C[L[i]]+ri,ri表示字符L[i]在i位之前出现的次数
3. 后向重建T，设s=1且T[u]=L[1]，那么对于i=u-1,。。。1,作s=LF[s]并且T[i]=L[s]

重建源字符串T需要的时间为线性复杂度

2.3 FM-index

BWA采用的索引结构是基于BWT的FM-index索引，以下作简单介绍。

FM-index是Ferragina和Manzini提出的一种全文索引结构，其空间占用大小依赖于建立索引的文本所具有的可压缩性，且压缩后的索引查询效率下降不明显。FM-index索引由两部分构成，一部分是压缩以后的BWT，一部分是用于辅助的文本信息。辅助文本信息包括压缩的OCC(r,c)函数，Occ(r,c)即在第r行前有多少个c字符。但是如果在每行都存储一个Occ(r,A/C/G/T)则会占用太多内存，但每行的occ值是固定的，故FM-index为了减小内存开销，设置每隔一定的行数存储一个Occ，即设置了一个检查点，如果查找该行的时候没有遇到checkpoint，则从最近的checkpoint开始计算该点的checkpoints。

FM-index还包括一个SA[]数组，记录BWT的第r行在参考基因组中在什么位置，且实际使用中也进行了简化，每隔固定的行数存储一次，需要的时候再进行推算即可。

以人类基因组为例，一个碱基用2bit表示，那么3G长的BWT约占空间680M，occ函数的checkpoint需要BWT大小的约14%，简化SA约占BWT的50%，故建立一个FM-index约需要1.1G.

2.4 精确搜索exact-search

字符串精确模式匹配过程包括计数和定位两个操作，计数是确定子串在文本中出现的次数，定位是确定找到的字符串在文本中的具体位置，由经过BWT变换的后缀矩阵按照字典序排序可知，如果要查找模式P，只需要P匹配文本T所有后缀的头部，而具有相同后缀1头部的串在矩阵中排列在一起，位于连续的位置上。可以使用sp和ep分别表示连续区域的起始和结束位置。那么ep-sp+1即为所有子串开头的行的个数，此时可以确定模式P在串中出现的次数和位置。

算法伪代码如下：

Exact-Search(P[1,p])

C = P[p],i=p;

Sp = C[c]+1,ep = C[c+1];

While((sp<=ep)&i>=2)) do

c=P[i-1];

sp=C[c]+Occ(c,1,sp-1)+1;

ep=C[c]+Occ(c,1,ep);

i++:

if(ep<sp) return NULL; //说明没找到这样的后缀

else return ep,sp

2.5 BWT算法优势：

相对于哈希算法，BWT算法内存占用低，效率高；可执行任意长度序列的精确匹配，时间消耗只与精确匹配的长度有关，与参考序列长度和重复程序无关；另外BWT更适合测试100bp内的短序列

2.6 BWA算法

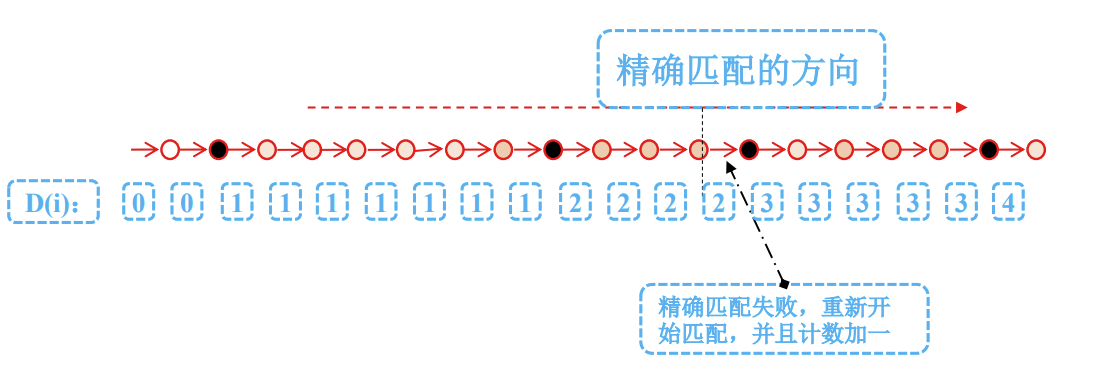
BWA实现模糊匹配是基于有界遍历/回溯的思想。字符串模式P与原序列的匹配可以有四种情况：

1. 正确，与原序列一致
2. 错误，被其他碱基替换
3. 插入：比原序列多出额外的碱基
4. 删除：与原序列相比少了碱基

|  |  |
| --- | --- |
| 情况 | 图示 |
| 正确匹配 |  |
| 碱基替换错误 |  |
| 插入错误 |  |
| 删除错误 |  |

BWA模糊匹配的思想：

首先对源文本进行反向，并求出反向的BWT用来对待匹配字符串进行前缀匹配，以求出如果待匹配序列精确匹配的情况下，匹配到第i位至少要发生多少错误。



再采用递归的思想，将序列从后到前进行匹配，分成A,C,G,T，插入，删除六种情况进行查找，除了正确匹配的情况，其他情况允许的错误数目都-1，如果积累的错误数目上限超过了预设的错误数，则该支必不可能有结果，直接剪去。

实际实现中论文作者使用了非递归的方法，利用堆来模拟了递归。

Precalculation: Calculate BWT string B for reference string X

//准备工作：计算参考基因序列X的BWT序列B

Calculate array C(·) and O(·,·) from B //计算B的C函数和Occ函数

Calculate BWT string B\* for the reverse reference

//计算参考序列的反向序列B\*的BWT序列B\*

Calculate array O\* (·,·) from B\* //计算B\*的C函数和Occ函数

Procedures:

InexactSearch(W,z) //模糊匹配序列W，最多允许z个错误

CalculateD(W) //计算序列W的D

return InexRecur(W,|W|−1,z,1,|X|−1) //返回在参考序列X的1到|X|-1范围内从W序列的第|W|-1位开始匹配的结果，最多允许z个错误

CalculateD(W) //计算D函数

k←1

l←|X|−1

z←0

for i=0 to |W|−1 do

k←C(W[i])+O\* (W[i],k−1)+1

l←C(W[i])+O\* (W[i],l)

if k >l then //如果发生了不匹配，z在该位+1

k←1 //重新开始搜索

l←|X|−1

z←z+1

D(i)←z //最后得到的是待匹配序列在第i位处有D(i)个错误的D数组

InexRecur(W,i,z,k,l) //递归模糊匹配，序列W从第i位开始匹配，最多允许z个错误，BWT范围为k，l

if z<D(i) then //如果匹配到此处已经发生的错误数超过了允许的范围

return ∅ //返回空，即减去不必要的分支

if i<0 then //如果已经将待匹配序列匹配完毕则返回结果

return {[k,l]}

I ←∅

* I ←I ∪ InexRecur(W,i−1,z−1,k,l) //此处是发生了插入错误的情况 z-1

for each b∈{A,C,G,T} do //对于ACGT每种匹配情况进行搜索

k←C(b)+O(b,k−1)+1

l←C(b)+O(b,l)

if k ≤l then //如果尚在范围内

* I ←I ∪ InexRecur(W,i,z−1,k,l) //此处发生删除错误，z-1，下一次仍然搜索当前位

if b=W[i] then //如果是正确匹配，z的数目不变，搜索下一位，其他状况都需要将z-1

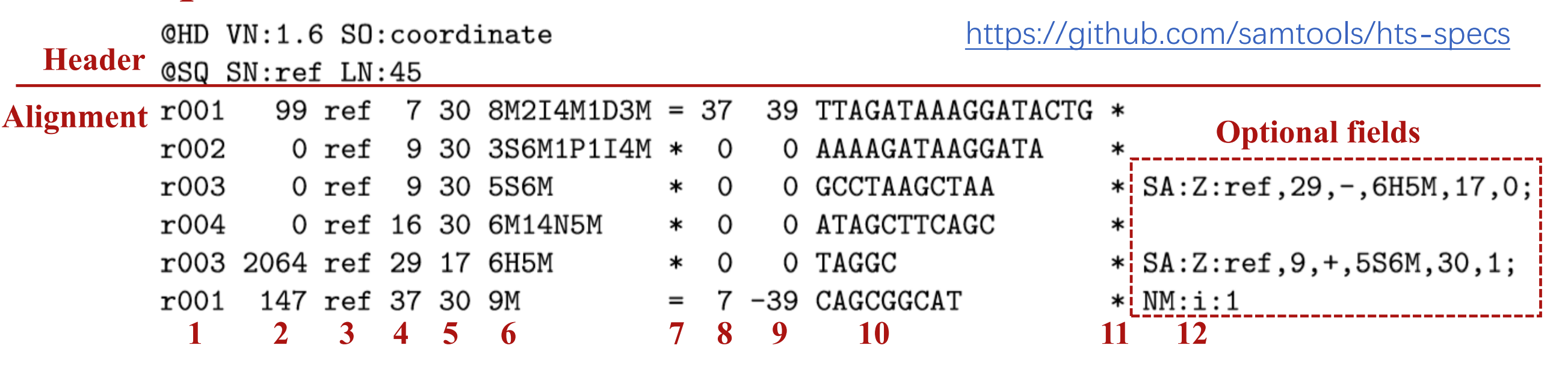
I ←I ∪ InexRecur(W,i−1,z,k,l)

else

I ←I ∪ InexRecur(W,i−1,z−1,k,l) //否则此处错配，z-1

return I

2.7 SAM格式：



这里重点介绍Alignment部分：

|  |  |  |  |
| --- | --- | --- | --- |
| Col | Field | Type | 描述 |
| 1 | QNAME | 字符串 | 比对片段编号以及read name |
| 2 | FLAG | 整型数int | 比对情况的代表值，也叫位标识，每种数字代表一种情况(最终的flag是各种情况的总和) |
| 3 | RNAME | 字符串 | 比对上的染色体 |
| 4 | POS | 整型数int | 比对到染色体的具体位置 |
| 5 | MAPQ | 整型数int | Mapping的质量值 |
| 6 | CIGAR | 字符串 | 表示的匹配状态 |
| 7 | RNEXT | 字符串 | Pair read另外一半比对的参考基因组，若无则为\* |
| 8 | PNEXT | 整型数int | Pair read另外一半比对上的参考基因组坐标，若无则为0 |
| 9 | TLEN | 整形数int | 表示pairread比对到基因组上后上游第一个碱基到下游最后一个碱基的距离，用正负区分上下游 |
| 10 | SEQ | 字符串 | Read序列信息 |
| 11 | QUAL | 字符串 | 比对质量信息 |

FLAG信息:

|  |  |
| --- | --- |
| FLAG值 | 含义 |
| 1 | read是pair中的一条（read表示本条read，mate表示pair中的另一条read） |
| 2 | pair一正一负完美的比对上 |
| 4 | 这条read没有比对上 |
| 8 | materead没有比对上 |
| 16 | 这条read是反向比对到参考序列的 |
| 32 | Matepair能反向比对上 |
| 64 | 这条read是read1 |
| 128 | 这条read是read2 |
| 256 | 该read是次优比对结果 |
| 512 | 比对质量不合格 |
| 1024 | 由pcr或者测序错误产生的重复read |
| 2048 | 补充匹配的read |

CIGAR值

|  |  |
| --- | --- |
| 标记 | 含义 |
| M | match或 mismatch； |
| I | 插入错误 |
| D | 删除错误 |
| N | 跳过该段区域 |
| S | soft clipping（被剪切的序列存在于序列中）； |
| H | hard clipping（被剪切的序列不存在于序列中）； |
| P | padding； |
| = | match |
| X | mismatch |

1. 测试结果及分析

3.1 BWT的实现:

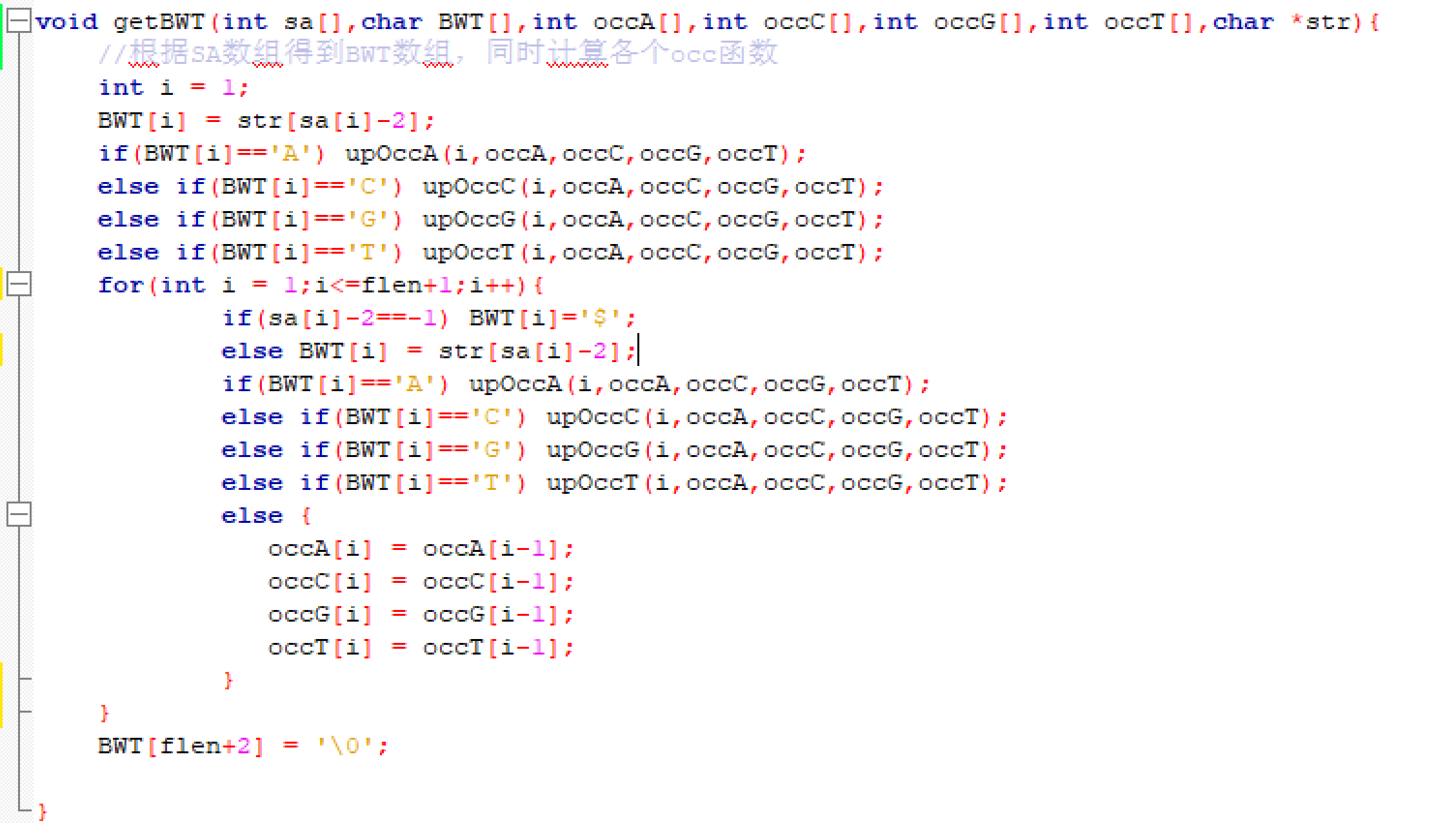
由于实验一已经实现了用CSA算法得到基因组的后缀数组，故BWT序列只需要由公式

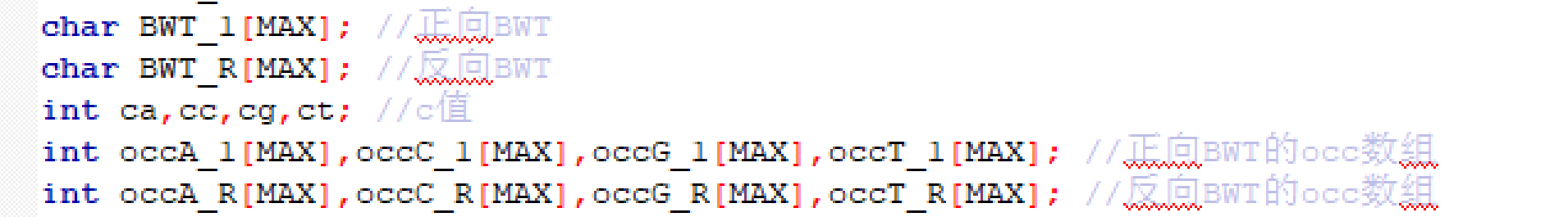
W[i]=T[SA[i]-1] if SA[i]>0

W[i]=$ if SA[i]=0

在常数时间内即可得到序列的BWT序列W

在BWT构建过程中可以同时构建Occ数组，C数组由lc，rc计算即可得到。



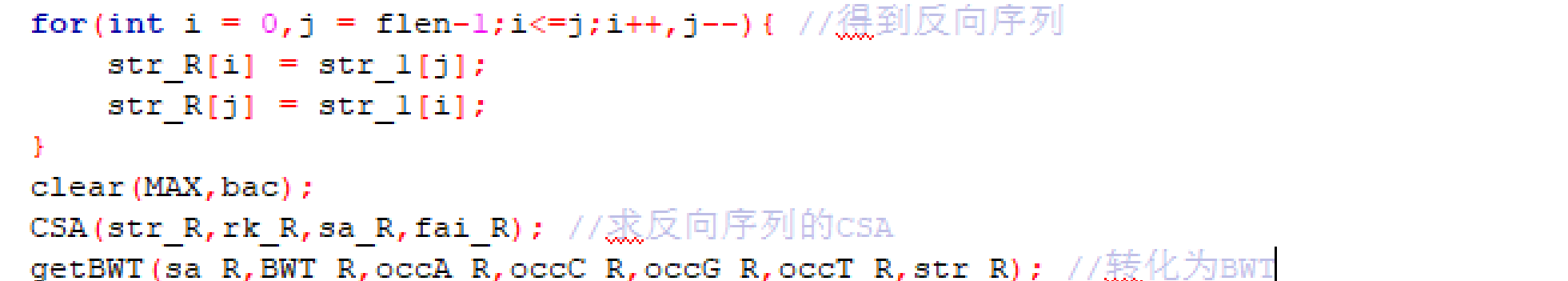


存储BWT，OCC，C等的数据结构

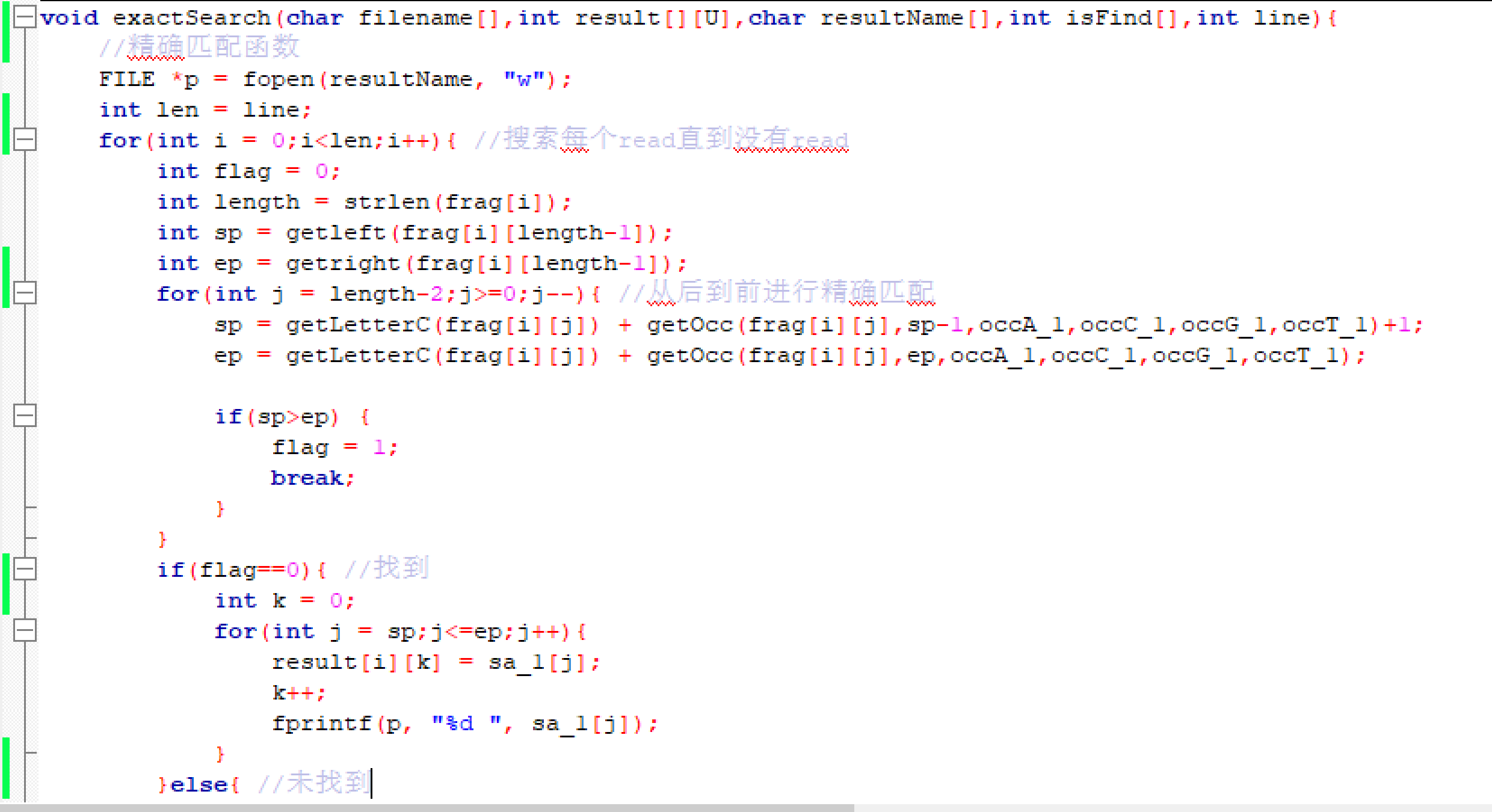
为了加快搜索速度，且待生成序列不是太长，故没有像FM-index一样设置checkpoint而是选择用空间换时间，存储了每个occ

* 1. BWA的实现:

首先求反向BWT，即将序列反向得到B\*再求该反向序列的BWT序列W\*，同时求出其OCC\*

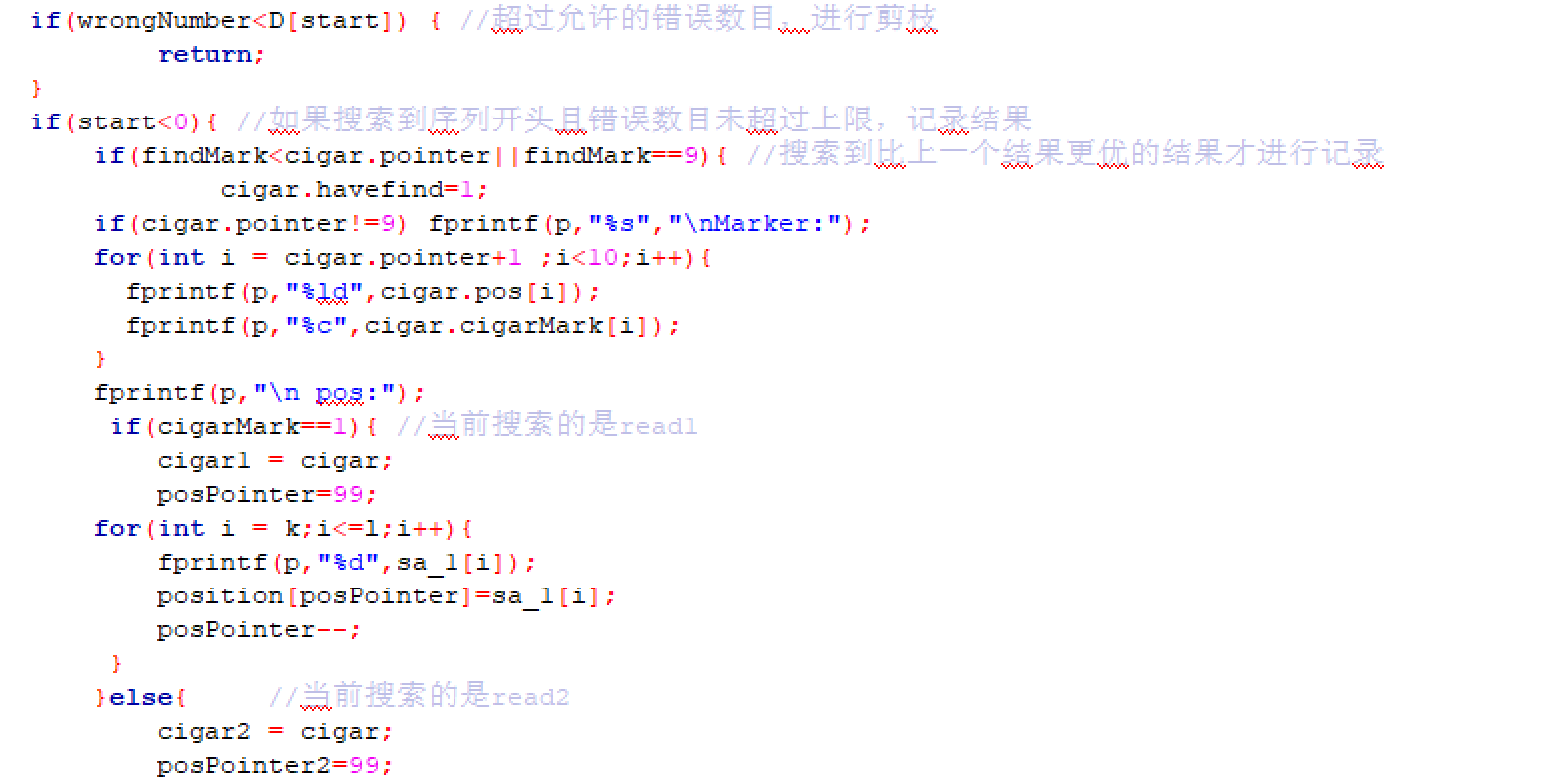


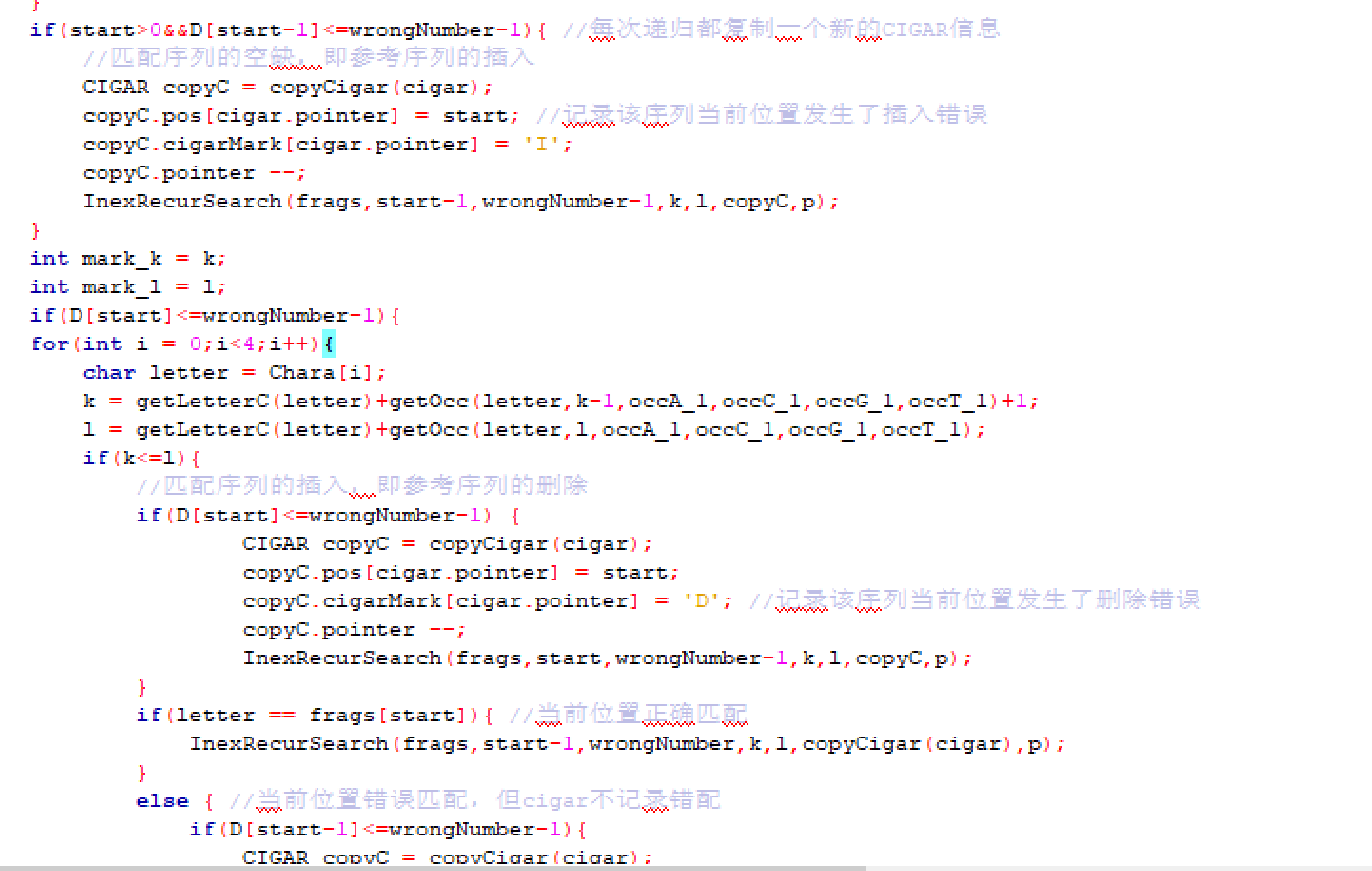
精确匹配：直接根据精确匹配算法在正向BWT中搜索，当sp>ep时查找失败，sp<=ep且搜索到头时精确匹配成功



模糊匹配:

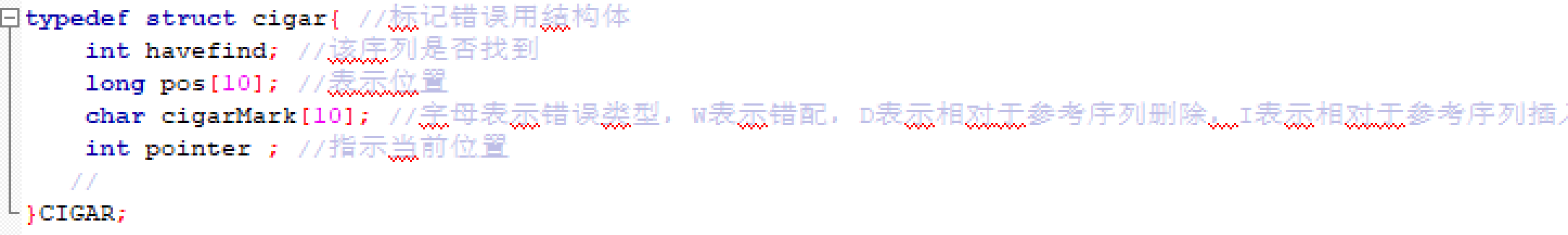
根据模糊匹配算法进行递归搜索，每次进入新递归的时候都复制一个CIGAR进行信息记录



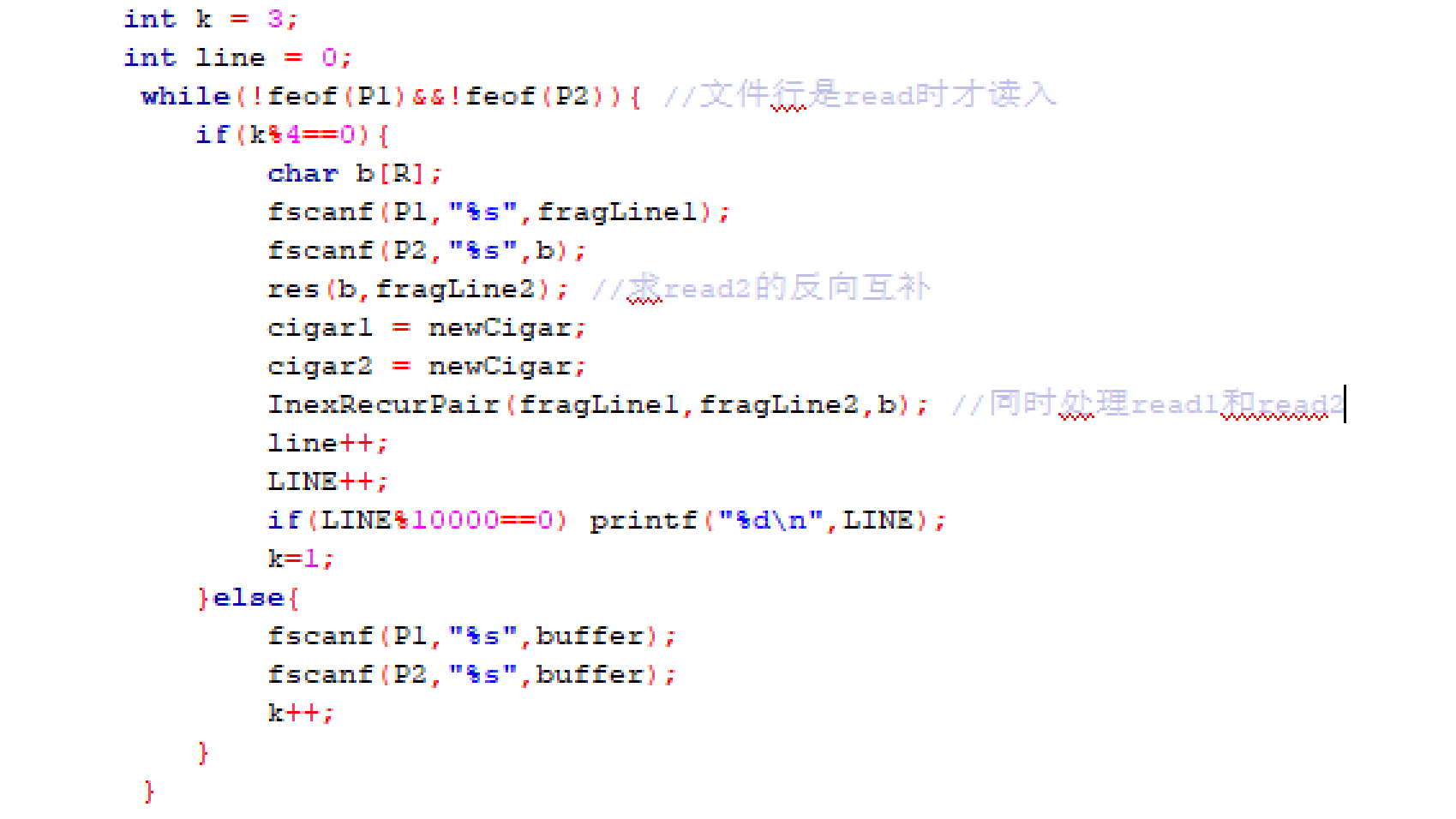


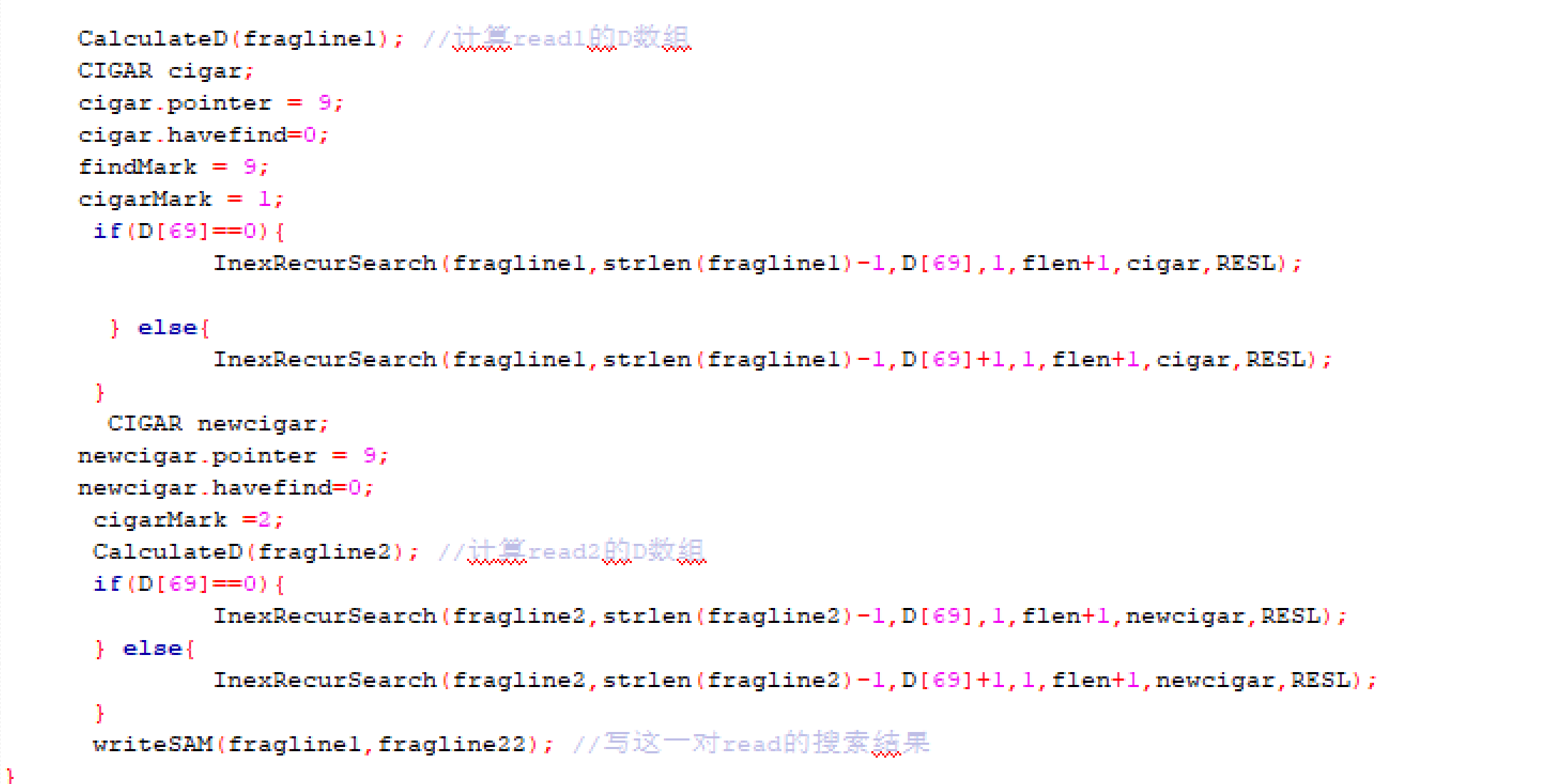
其中每次搜索的Z是根据该序列的最大D值决定。

CIGAR数据结构用于记录搜索过程中的信息



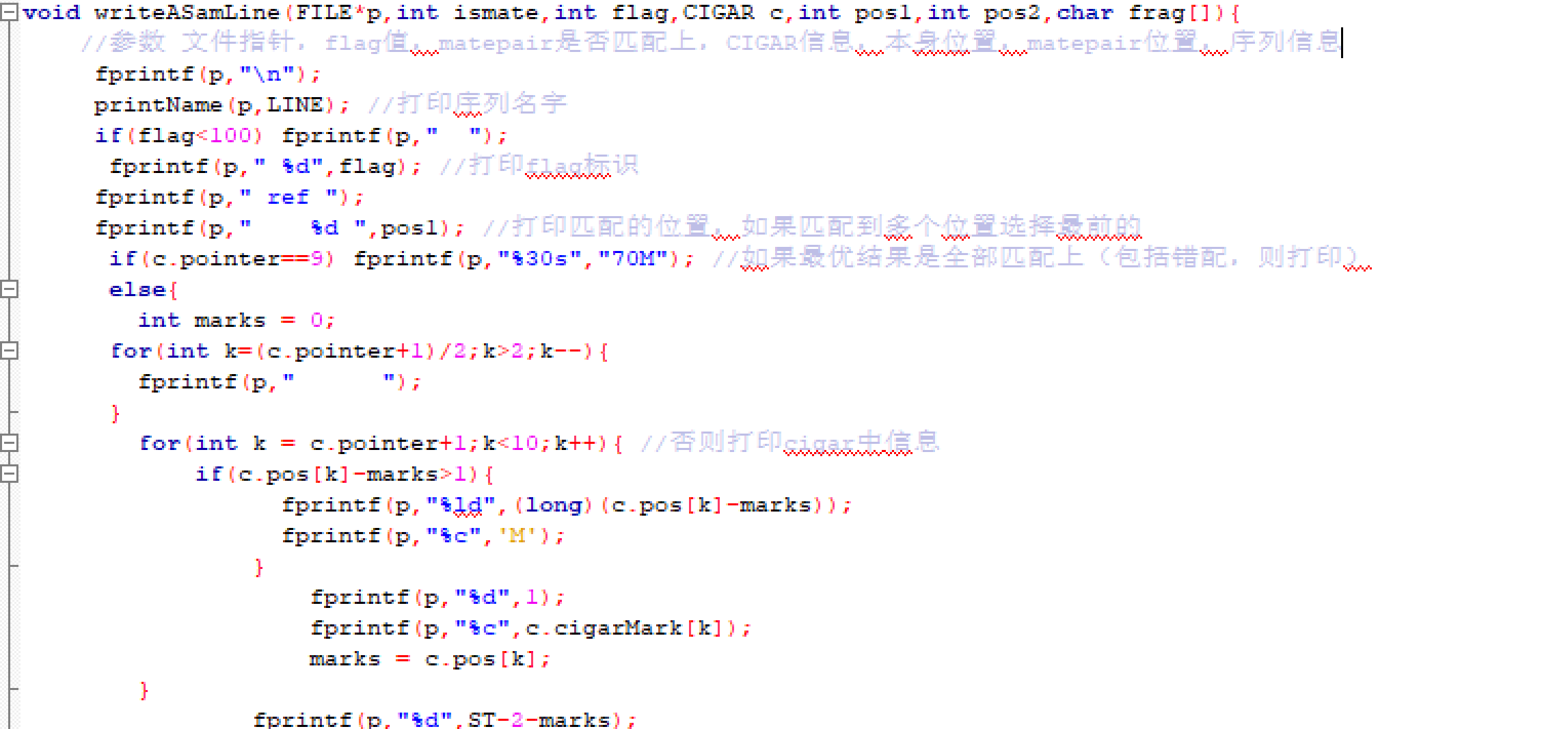
一开始尝试将文件1的所有序列和文件2的序列反向全部读入字符数组再逐个进行查找，后来发现这样输出结果文件的时候非常不方便，改为两个文件同时读取，每次都是两个文件同时读入read1和read2进行查找：

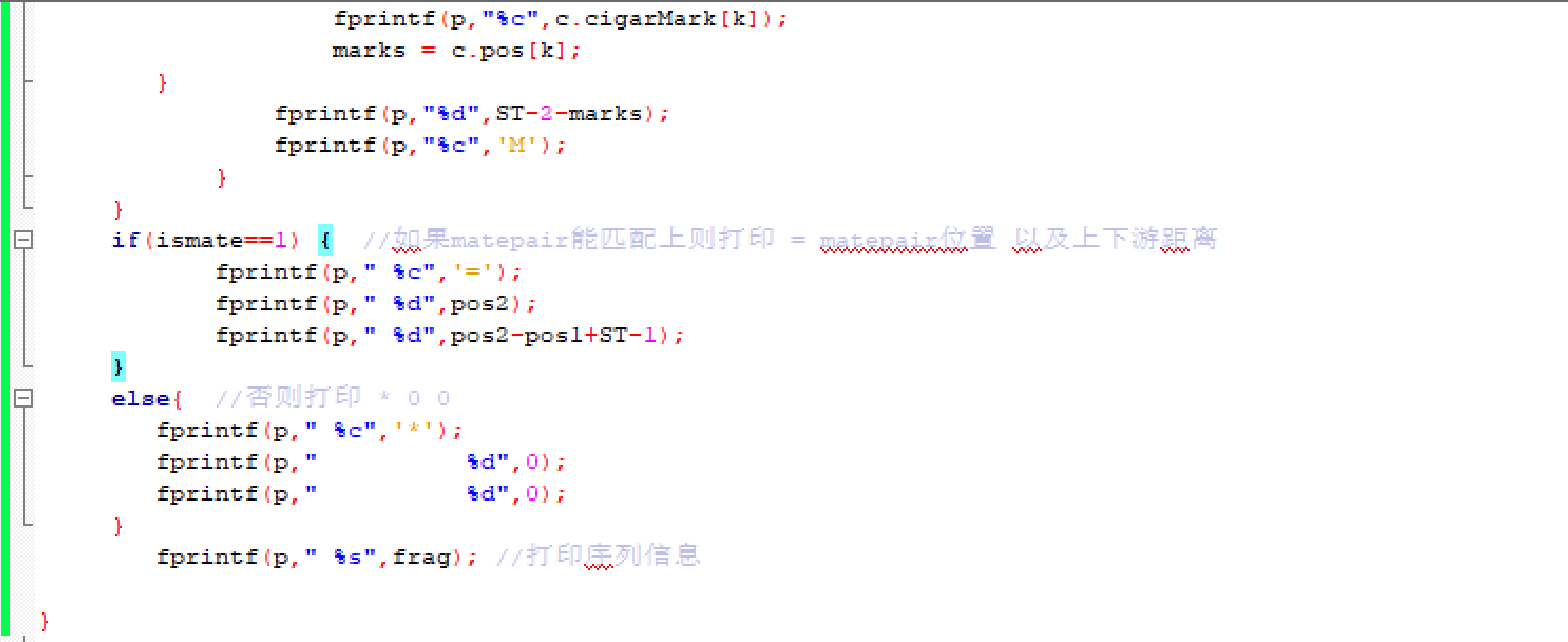




* 1. SAM格式模拟:

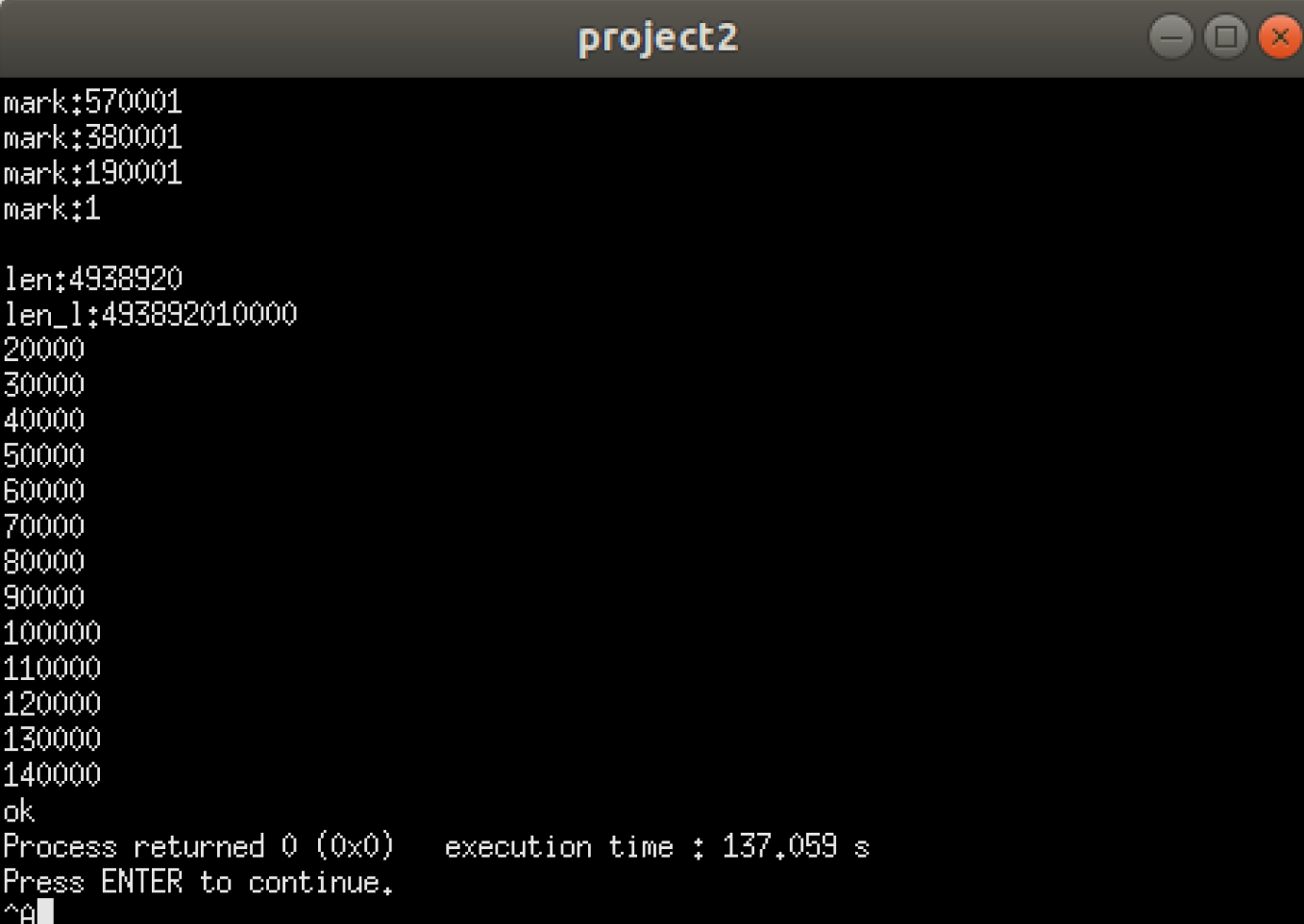
由于不知道如何生成SAM文件，使输出的结果对SAM格式进行了简单模拟，





结果分析：

在linux系统下，构建正反两个BWT共需不到20s，精确搜索几乎不消耗时间，但是能完全精确匹配上的序列较少；但可能由于模糊搜索一些支未减去加上递归搜索，导致搜索需要约2min时间，实际上应该仍然有优化空间。



本实验在约500w个碱基中搜索约14w对70碱基长的read pairs，结果以成对的搜索信息输出，模仿SAM格式打印信息,基本上除了编辑距离过大的序列，其他序列都能模糊匹配到参考基因组，编辑距离0-5不等。

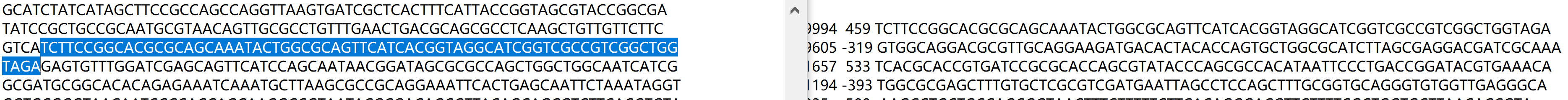
输出结果举例:



如图，标注了序列编号，若两个序列变化相同则上一个是read1，下一个是read2；flag值，（eg若flag=99，说明该序列是read1，且两个序列都匹配上）；参考序列位置；CIGAR值（eg 70M表示序列能匹配上，包括错配和正确匹配）,RNEXT值（=表示matepair能匹配上，否则为\*）;matepair匹配上的位置；TLEN值；序列文本信息。（由于对序列质量值的计算不太了解，故省略）

根据匹配结果和原文件，匹配正确率较高。

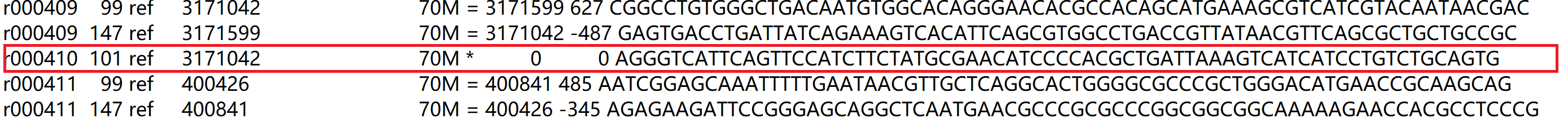
举例，第一对read的read1匹配结果：



能够在输出结果的匹配位置找到该序列；且如果read2也能反向匹配到，几乎都在read1上/下游300-500范围内；由于序列索引构建的正确性上一次已经得到验证，故搜索的正确性也能得到保障。绝大多数序列都能正确/错误匹配上一个位点，少部分序列能匹配上多个位点，少部分序列能插入/删除匹配上，极少数序列匹配不上。

其他匹配情况举例:

1.只匹配上一条read:



可以看到flag值不同且TLEN和PNEXT也都为0

2.发生了插入错误：

待比对的read：



参考序列相对应位置：

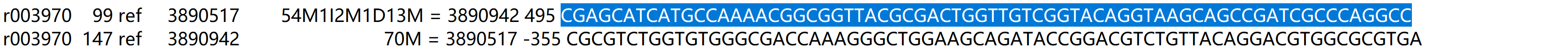


|  |  |
| --- | --- |
| Read | GCTGAATTTGCCTGGGATCTGTTTACCGCCTGGCAGCCCGCTGTCGCGCCGTCGAAAGACGAGTTGGGCG |
| Ref | GCTGAATTTGCCTGGGATCTGTTTACCGCCTGGCAGACCGCTGGCGCGCCGTCGAAAGAI GAGTTGGGCG |

可以看到Read相对于参考基因组插入了一个C

3.发生了删除错误

待比对的read：



参考基因组序列匹配位置:



|  |  |
| --- | --- |
| Read | CGAGCATCATGCCAAAACGGCGGTTACGCGACTGGTTGTCGGTACAGGTAAGCADCCAGATCGCCCAGCCC |
| Ref | CGAGCATCATGCCAAAACGGCGGTTACGCGACTGGTTGTCGGTACAGGTAAGCAGCCIGATCGCCCAGGCC |

可以看到前59碱基匹配，之后相对于参考基因组有一个删除，匹配2个碱基以后又有一个插入；此后13个碱基能匹配上（包括一个错配）

另外，匹配的精确度，正确率越高，时间就会越久。

1. 经验体会

由于实验一能在较短时间内得出正确的序列结果，故实验二选择了BWA算法进行序列比对，并在CSA的基础上得到了BWT；了解算法思想和具体流程以后首先进行了精确匹配的实现，再进行模糊匹配的实现；在这个过程中非常感谢老师的帮助，使我改进了代码并且更正确和深刻的理解了算法的思想。每当算法实现出现问题的时候，大多数时候应该是对算法的理解出现了问题，首先应该返回去读论文，常常会有所收获。

此次实验工作量还是较大的，只能说还好上次实验结果比较好，一定程度上减少了此次的工作量；此外本次实验还有许多可以改进的地方，例如index可以占用更少的空间，模糊搜索还有更多剪支的余地，时间复杂度还能再降低；SAM格式还需要更多的学习；希望自己能在以后的学习中完善这些不足，逐渐进步，如果有明显缺陷恳请老师进行批评指正。

1. 附录：源代码（带注释）

#include <stdlib.h>

#include <math.h>

#include<string.h>

#include<stdio.h>

#define MAX 5000000

#define N 5000000

#define M 400000

#define S 200000

#define L 150000

#define R 100

#define U 1000

//#define Z 3

#define ST 71

char start\_string[S];

int n,m,num;

int sa\_1[N],rk\_1[N],bac[N],y[N],tmp[N];//sa[i]数组表示后缀排名为i的位置，rk[i]表示后缀[i..n]的排名

int sa\_R[N],rk\_R[N];

int copy\_rank[N],copy\_fai[N];

int la,ra,lc,rc,lg,rg,lt,rt;

int order[M];

char string\_part[M] ;

int fai\_1[N];

int fai\_R[N];

char str\_1[MAX];

char str\_R[N];

char BWT\_1[MAX]; //正向BWT

char BWT\_R[MAX]; //反向BWT

int ca,cc,cg,ct; //c值

int occA\_1[MAX],occC\_1[MAX],occG\_1[MAX],occT\_1[MAX]; //正向BWT的occ数组

int occA\_R[MAX],occC\_R[MAX],occG\_R[MAX],occT\_R[MAX]; //反向BWT的occ数组

double l;

char frag[L][R];

int flen;

int f[N];

int orderNumber[N];

char buffer[R];

int last;

int array\_len;

int D[ST];

char Chara[5] ={'A','C','G','T','\0'};

char charTable[256];

int pointer = 2;

int LC[256];

int RC[256];

int CC[256];

int findMark;

char fragLine1[R];

char fragLine2[R];

FILE \*P1;

FILE \*P2;

FILE \*RES;

FILE \*RESL;

int LINE;

int cigarMark;

int FINDMARK;

typedef struct cigar{ //标记错误用结构体

int havefind; //该序列是否找到

long pos[10]; //表示位置

char cigarMark[10]; //字母表示错误类型，W表示错配，D表示相对于参考序列删除，I表示相对于参考序列插入

int pointer ; //指示当前位置

//

}CIGAR;

CIGAR cigar1; //当前搜索的序列1的cigar

CIGAR cigar2; //当前搜索的序列2的cigar

CIGAR newCigar; //空cigar

long position[R]; //记录当前搜索序列read1的位置信息

int posPointer;

long position2[R]; //记录当前搜索序列read2的位置信息

int posPointer2;

void writeSamLine(FILE\*p,int is1,int is2,int flag,int flag2,char frag1[],char frag2[]);

CIGAR copyCigar(CIGAR oldcigar){ //每次分支，都需要对上一个CIGAR进行拷贝

CIGAR newcigar;

newcigar.pointer = oldcigar.pointer;

for(int i = 9;i>oldcigar.pointer;i--){

newcigar.pos[i] = oldcigar.pos[i];

newcigar.cigarMark[i] = oldcigar.cigarMark[i];

}

// newcigar.posPointer = oldcigar.posPointer;

//newcigar.havefind = oldcigar.havefind;

return newcigar;

}

void printCigar(CIGAR cigar){

printf("\n");

printf("\n");

if(cigar.pointer==2) printf("Prefect Match");

else{

for(int i = cigar.pointer+1;i<3;i++){

printf("%ld",cigar.pos[i]);

printf("%c",cigar.cigarMark[i]);

}

}

printf("\n");

}

void Fai(int SA[],int rank[],int nn,int fai[]){

//printf("\nFai");

fai[1]=rank[1];

for(int i = 2;i<=nn;i++){

fai[i]=rank[SA[i]+1];

}

}

void getsa(char \*s,int rk[],int sa[]){

//rk[i]表示位置i的第一关键字(排名)

//二元组(rk[i],rk[i+k])

//初始化基数排序(单字符，其实是单元)

num = 0;

m=122;

for(int i=1;i<=n;i++)

{

bac[rk[i]=s[i]]++;//bac[i]表示第一关键字小于i的个数

}

for(int i=2;i<=m;i++)

{

bac[i]+=bac[i-1];//故就先逐个统计，再求前缀和，相当于用桶计数

}

for(int i=n;i>=1;i--) {sa[bac[rk[i]]--]=i;//在第二关键字有序的情况下，对位置按照第一关键字的bac数组，逐个附排名

}

for(int k=1;k<=n;k<<=1){

num=0;//排名的数量,初始化为单个字符的ASCLL码上限,m=ascll('z')=122

//y[i]表示第二关键字排名为i的二元组的第一关键字位置(i)

for(int i=n-k+1;i<=n;i++)y[++num]=i;//没有第二关键字的排名最靠前

for(int i=1;i<=n;i++)if(sa[i]>k)y[++num]=sa[i]-k;//利用已有的sa\_(k/2)数组，rk靠前的且有第一关键字的，将第一关键字位置sa\_(k/2)[i]-k放入y

//在上一轮中，rk\_k数组已经求好了，倍增求解,

//接下来利用基数排序，求出sa\_k

for(int i=1;i<=m;i++)bac[i]=0;

for(int i=1;i<=n;i++)bac[rk[i]]++;//bac[i]表示第一关键字小于i的个数

for(int i=2;i<=m;i++)bac[i]+=bac[i-1];//故就先逐个统计，再求前缀和，相当于用桶计数

for(int i=n;i>=1;i--)sa[bac[rk[y[i]]]--]=y[i];//在第二关键字有序的情况下，对位置按照第一关键字的bac数组，逐个附排名

memcpy(tmp,rk,sizeof(tmp));//用tmp存一下rk\_(k/2)，在求rk\_k时仍会用到

//其实就是利用sa\_k和rk\_(k/2)数组求rk\_k

rk[sa[1]]=1;num=1;

for(int i=2;i<=n;i++){

if(tmp[sa[i]]==tmp[sa[i-1]]&&tmp[sa[i]+k]==tmp[sa[i-1]+k])rk[sa[i]]=num;

else rk[sa[i]]=++num;

}

if(num==n)break;//已经有了n种排名

m=num;//m表示排名的数量（在桶排中为值域）

}

}

void copy(int\*a,int\*copys,int k){ //将字符串a拷贝k个字符给copys

for(int i = 1;i<=k;i++){

copys[i] = a[i];

}

}

int getleft(char letter){ //返回字符对应的l

/\* if(letter=='A') return la;

if(letter=='C') return lc;

if(letter=='G') return lg;

if(letter=='T') return lt;

else return -1;\*/

return LC[letter];

}

int getright(char letter){ //返回字符对应的R

/\* if(letter=='A') return ra;

if(letter=='C') return rc;

if(letter=='G') return rg;

if(letter=='T') return rt;

else return -1;\*/

return RC[letter];

}

int getNextOrder(int orderLast,char letter,int fai[]){

int lastorder=0;

int left\_side = getleft(letter);

int right\_side = getright(letter);

// int flag = 0;

//printf("\nletter:%c lc:%d lr:%d ",letter,left\_side,right\_side);

/\*for(int i = left\_side;i<=right\_side;i++){

printf("%d ",fai[i]);

}\*/

/\*for(int j = right\_side;j>=left\_side;j--){

if(fai[j]<=orderLast){

lastorder = j;

flag = 1;

}

if(flag==1) break;

}

if(flag==0) lastorder = left\_side-1;\*/

if(left\_side>right\_side) return right\_side;

if(fai[left\_side]>orderLast) lastorder = left\_side-1;

else if(fai[left\_side]==orderLast) lastorder = left\_side;

else if(fai[right\_side]<=orderLast) lastorder = right\_side;

else{

int m = (right\_side+left\_side)/2;

//printf("orderLast:%d\n",orderLast);

while(1){

//printf("%d,%d,%d\n",left\_side,m,right\_side);

//printf("%d,%d,%d\n",fai[left\_side],fai[m],fai[right\_side]);

if(fai[m]<=orderLast&&(fai[m+1<right\_side?m+1:right\_side]>orderLast||left\_side==right\_side)){

lastorder = m;

break;

}else if(fai[m]>orderLast){

right\_side = m;

m = (right\_side+left\_side)/2;

}else{

left\_side = m;

m = (right\_side+left\_side)/2;

}

}

}

//printf("lastorder:%d\n",lastorder);

return lastorder;

}

void clear(int a,int array[]){

for(int i = 0;i<a;i++){

array[i] = 0;

}

}

void getSA(int nn,int sa[],int rk[]){

//printf("%d %d\n",i,rk[i]);

//printf("\nsa:");

for(int i = 1;i<=nn;i++){

sa[rk[i]] = i;

//printf("%d %d\n",i,rk[i]);

}

}

void updateLR(int lens,char\* str){

//printf("\n++++++++++++++++++++++");

//printf("len:%d\n",lens);

int ra1=1,rc1=0,rg1=0,rt1=0;

int flagA = 0,flagC = 0,flagG = 0,flagT=0;

for(int i = lens;str[i]!='\0';i++){

if(str[i]=='A'){

if(flagA==0) {

flagA = 2;

}

ra1++;

}

if(str[i]=='C'){

if(flagC==0){

//lc1++;

flagC=1;

}

rc1++;

}

if(str[i]=='G'){

if(flagG==0){

//lg1++;

flagG=1;

}

rg1++;

}

if(str[i]=='T'){

if(flagT==0){

//lt1=1;

flagT=1;

}

rt1++;

}

}

la = flagA; ra = ra1; lc = ra+flagC; rc = rc1+ra; lg = rc+flagG; rg = rc+rg1;lt=flagT+rg; rt = rt1+rg;

if(flagA==0) la = ra+1;

if(flagC==0) lc = rc+1;

if(flagG==0) lg = rg+1;

if(flagT==0) lt = rt+1;

LC['A']=la;LC['C']=lc;LC['G']=lg;LC['T']=lt;

RC['A']=ra;RC['C']=rc;RC['G']=rg;RC['T']=rt;

}

void dealrank(int start,int length,int sa[],int rk[]){

for(int i = 1,j= 1;i<=length;i++){

if(sa[i]<=start){

sa[j++] = sa[i];

}

}

for(int i = 1;i<=start;i++){

rk[sa[i]] = i;

}

}

void merge(int start,int len,int array\_long,int rk[],int sa[],int fai[],char \*str){

copy(rk,copy\_rank,flen-start-len+3);

int orderx = rk[1];

//printf("orderx:%d\n",orderx);

int length = flen-start+1;

n = len\*2>length?flen-start+1:len\*2;

strncpy(string\_part+1,str+start-1,n);

// printf("\nstrncpy");

//string\_part[flen-start] = '\0';

string\_part[n+1]='\0';

clear(flen-start,orderNumber);

//printf("\nclean");

order[len] = getNextOrder(orderx,string\_part[len],fai);

//printf("\ngetNextOrder");

orderNumber[order[len]]++;

for(int i = len-1;i>=1;i--){

order[i]=getNextOrder(order[i+1],string\_part[i],fai);

orderNumber[order[i]]++;

}

clear((128>n)?128:n,bac);

//printf("\nclean");

n = strlen(string\_part+1);

getsa(string\_part,rk,sa);

//printf("\ngetsa");

dealrank(len,n,sa,rk);

//printf("\ndealrank");

int smaller = 0;

for(int i = 1;i<=flen-start-len+2;i++){

smaller += orderNumber[i-1];

f[i] = i+smaller;

}

// printf("\nsmaller");

for(int i = 1;i<=flen-start-len+2;i++){

rk[i+len] = f[copy\_rank[i]];

}

// printf("\nf");

//求g函数

for(int i = 1;i<=len;i++){

//printf("%d,%d,%c\n",order[i],suffix[i]-1,string\_part[i]);

rk[i] = order[i]+ rk[i];

//if(string\_part[i]>=mark\_letter) rk[i]=rk[i]-1;

}

// printf("\ng");

getSA(flen-start+2,sa,rk);

}

void upOccA(int i,int occA[],int occC[],int occG[],int occT[]){

occA[i] = occA[i-1]+1;

occC[i] = occC[i-1];

occG[i] = occG[i-1];

occT[i] = occT[i-1];

}

void upOccC(int i,int occA[],int occC[],int occG[],int occT[]){

occA[i] = occA[i-1];

occC[i] = occC[i-1]+1;

occG[i] = occG[i-1];

occT[i] = occT[i-1];

}

void upOccG(int i,int occA[],int occC[],int occG[],int occT[]){

occA[i] = occA[i-1];

occC[i] = occC[i-1];

occG[i] = occG[i-1]+1;

occT[i] = occT[i-1];

}

void upOccT(int i,int occA[],int occC[],int occG[],int occT[]){

occA[i] = occA[i-1];

occC[i] = occC[i-1];

occG[i] = occG[i-1];

occT[i] = occT[i-1]+1;

}

void getBWT(int sa[],char BWT[],int occA[],int occC[],int occG[],int occT[],char \*str){

//根据SA数组得到BWT数组，同时计算各个occ函数

int i = 1;

BWT[i] = str[sa[i]-2];

if(BWT[i]=='A') upOccA(i,occA,occC,occG,occT);

else if(BWT[i]=='C') upOccC(i,occA,occC,occG,occT);

else if(BWT[i]=='G') upOccG(i,occA,occC,occG,occT);

else if(BWT[i]=='T') upOccT(i,occA,occC,occG,occT);

for(int i = 1;i<=flen+1;i++){

if(sa[i]-2==-1) BWT[i]='$';

else BWT[i] = str[sa[i]-2];

if(BWT[i]=='A') upOccA(i,occA,occC,occG,occT);

else if(BWT[i]=='C') upOccC(i,occA,occC,occG,occT);

else if(BWT[i]=='G') upOccG(i,occA,occC,occG,occT);

else if(BWT[i]=='T') upOccT(i,occA,occC,occG,occT);

else {

occA[i] = occA[i-1];

occC[i] = occC[i-1];

occG[i] = occG[i-1];

occT[i] = occT[i-1];

}

}

BWT[flen+2] = '\0';

}

int getLetterC(char l){

if(l=='A') return ca;

else if(l=='C') return cc;

else if(l=='G') return cg;

else if(l=='T') return ct;

else return 0;

}

int getOcc(char l,int k,int occA[],int occC[],int occG[],int occT[]){

if(l=='A') return occA[k];

else if(l=='C') return occC[k];

else if(l=='G') return occG[k];

else if(l=='T') return occT[k];

else return 0;

}

void CalculateD(char frags[]){ //计算D函数

int k = 2,l = flen -1,z = 0;

for(int i = 0;i<strlen(frags);i++){

k = getLetterC(frags[i])+getOcc(frags[i],k-1,occA\_R,occC\_R,occG\_R,occT\_R)+1;

l = getLetterC(frags[i])+getOcc(frags[i],l,occA\_R,occC\_R,occG\_R,occT\_R);

if(k>l){

k = 2;

l = flen - 1;

z = z +1;

}

D[i] = z;

}

}

void InexRecurSearch(char frags[],int start,int wrongNumber,int k,int l,CIGAR cigar,FILE \*p){

if(wrongNumber<D[start]) { //超过允许的错误数目，进行剪枝

return;

}

if(start<0){ //如果搜索到序列开头且错误数目未超过上限，记录结果

if(findMark<cigar.pointer||FINDMARK==0){

//搜索到比上一个结果更优的结果才进行记录

FINDMARK=1;

cigar.havefind=1;

if(cigar.pointer!=9) fprintf(p,"%s","\nMarker:");

for(int i = cigar.pointer+1 ;i<10;i++){

fprintf(p,"%ld",cigar.pos[i]);

fprintf(p,"%c",cigar.cigarMark[i]);

}

fprintf(p,"\n pos:");

if(cigarMark==1){ //当前搜索的是read1

cigar1 = cigar;

posPointer=99;

findMark=cigar1.pointer;

for(int i = k;i<=l;i++){

fprintf(p,"%d",sa\_1[i]);

position[posPointer]=sa\_1[i];

posPointer--;

}

}else{ //当前搜索的是read2

cigar2 = cigar;

posPointer2=99;

findMark=cigar2.pointer;

for(int i = k;i<=l;i++){

fprintf(p,"%d",sa\_1[i]);

position2[posPointer2]=sa\_1[i];

posPointer2--;

}

}

}

return ;

}

if(start>0&&D[start-1]<=wrongNumber-1){ //每次递归都复制一个新的CIGAR信息

//匹配序列的空缺，即参考序列的插入

CIGAR copyC = copyCigar(cigar);

copyC.pos[cigar.pointer] = start; //记录该序列当前位置发生了插入错误

copyC.cigarMark[cigar.pointer] = 'I';

copyC.pointer --;

InexRecurSearch(frags,start-1,wrongNumber-1,k,l,copyC,p);

}

int mark\_k = k;

int mark\_l = l;

if(D[start]<=wrongNumber-1){

for(int i = 0;i<4;i++){

char letter = Chara[i];

k = getLetterC(letter)+getOcc(letter,k-1,occA\_1,occC\_1,occG\_1,occT\_1)+1;

l = getLetterC(letter)+getOcc(letter,l,occA\_1,occC\_1,occG\_1,occT\_1);

if(k<=l){

//匹配序列的插入，即参考序列的删除

if(D[start]<=wrongNumber-1) {

CIGAR copyC = copyCigar(cigar);

copyC.pos[cigar.pointer] = start;

copyC.cigarMark[cigar.pointer] = 'D'; //记录该序列当前位置发生了删除错误

copyC.pointer --;

InexRecurSearch(frags,start,wrongNumber-1,k,l,copyC,p);

}

if(letter == frags[start]){ //当前位置正确匹配

InexRecurSearch(frags,start-1,wrongNumber,k,l,copyCigar(cigar),p);

}

else { //当前位置错误匹配，但cigar不记录错配

if(D[start-1]<=wrongNumber-1){

CIGAR copyC = copyCigar(cigar);

InexRecurSearch(frags,start-1,wrongNumber-1,k,l,copyC,p);

}

}

}

k = mark\_k;

l = mark\_l;

}

}else{

char letter = frags[start];

k = getLetterC(letter)+getOcc(letter,k-1,occA\_1,occC\_1,occG\_1,occT\_1)+1;

l = getLetterC(letter)+getOcc(letter,l,occA\_1,occC\_1,occG\_1,occT\_1);

if(k<=l){

InexRecurSearch(frags,start-1,wrongNumber,k,l,copyCigar(cigar),p);

}

k = mark\_k;

l = mark\_l;

}

}

void res(char \*rs,char \*rs2){ //求反向互补序列

int j=0;

for(int i=69;i>j;i--,j++){

rs2[j]= charTable[rs[i]];

rs2[i]=charTable[rs[j]];

}

}

void writeSAM(char fragline1[],char fragline2[]){ //匹配情况不同时，flag也不同

int flag1, flag2;

if(cigar1.havefind==1&&cigar2.havefind==1){

flag1=99, flag2=147;

writeSamLine(RES,1,1,flag1,flag2,fragline1,fragline2);

}else if(cigar1.havefind==1&&cigar2.havefind==0){

flag1=73; flag2=133;

writeSamLine(RES,1,0,flag1,flag2,fragline1,fragline2);

}else if(cigar1.havefind==0&&cigar2.havefind==1){

flag1=101; flag2=153;

writeSamLine(RES,1,0,flag1,flag2,fragline1,fragline2);

}

}

void InexRecurPair(char fragline1[],char fragline2[],char fragline22[]){

CalculateD(fragline1); //计算read1的D数组

CIGAR cigar;

cigar.pointer = 9;

cigar.havefind=0;

findMark = 9;

cigarMark = 1;

FINDMARK=0;

if(D[69]==0){

InexRecurSearch(fragline1,strlen(fragline1)-1,D[69],1,flen+1,cigar,RESL);

} else{

InexRecurSearch(fragline1,strlen(fragline1)-1,D[69]+1,1,flen+1,cigar,RESL);

}

CIGAR newcigar;

newcigar.pointer = 9;

newcigar.havefind=0;

cigarMark =2;

CalculateD(fragline2); //计算read2的D数组

findMark = 9;

FINDMARK=0;

if(D[69]==0){

InexRecurSearch(fragline2,strlen(fragline2)-1,D[69],1,flen+1,newcigar,RESL);

} else{

InexRecurSearch(fragline2,strlen(fragline2)-1,D[69]+1,1,flen+1,newcigar,RESL);

}

writeSAM(fragline1,fragline22); //写这一对read的搜索结果

}

void ReadFileInX(){

newCigar.havefind=0;

newCigar.pointer=9;

P1 = fopen("Ecoli1.txt","r");

P2 = fopen("Ecoli2.txt","r");

RES = fopen("SAM.txt","w");

RESL = fopen("result.txt","w");

LINE=0;

if(P1==NULL||P2==NULL){

printf("no");

exit(1);

}else{

int k = 3;

int line = 0;

while(!feof(P1)&&!feof(P2)){ //文件行是read时才读入

if(k%4==0){

char b[R];

fscanf(P1,"%s",fragLine1);

fscanf(P2,"%s",b);

res(b,fragLine2); //求read2的反向互补

cigar1 = newCigar;

cigar2 = newCigar;

InexRecurPair(fragLine1,fragLine2,b); //同时处理read1和read2

line++;

LINE++;

if(LINE%10000==0) printf("%d\n",LINE);

k=1;

}else{

fscanf(P1,"%s",buffer);

fscanf(P2,"%s",buffer);

k++;

}

}

fclose(P1);

fclose(P2);

fclose(RES);

fclose(RESL);

}

}

void exactSearch(char filename[],int result[][U],char resultName[],int isFind[],int line){

//精确匹配函数

FILE \*p = fopen(resultName, "w");

int len = line;

for(int i = 0;i<len;i++){ //搜索每个read直到没有read

int flag = 0;

int length = strlen(frag[i]);

int sp = getleft(frag[i][length-1]);

int ep = getright(frag[i][length-1]);

for(int j = length-2;j>=0;j--){ //从后到前进行精确匹配

sp = getLetterC(frag[i][j]) + getOcc(frag[i][j],sp-1,occA\_1,occC\_1,occG\_1,occT\_1)+1;

ep = getLetterC(frag[i][j]) + getOcc(frag[i][j],ep,occA\_1,occC\_1,occG\_1,occT\_1);

if(sp>ep) {

flag = 1;

break;

}

}

if(flag==0){ //找到

int k = 0;

for(int j = sp;j<=ep;j++){

result[i][k] = sa\_1[j];

k++;

fprintf(p, "%d ", sa\_1[j]);

}

}else{ //未找到

result[i][0] = -1;

fprintf(p, "%d ", -1);

}

fprintf(p,"\n");

}

fclose(p);

}

void CSA(char \*str,int rk[],int sa[],int fai[]){

//printf("array\_len:%d\n",array\_len);

strncpy(start\_string+1,str+(flen-last),last);

start\_string[last+1]='$';

start\_string[last+2] = '\0';

n = strlen(start\_string+1);

// printf("n:%d\n",n);

getsa(start\_string,rk,sa);

// printf("getsa\n");

updateLR(flen-last,str);

// printf("updateL\n");

Fai(sa,rk,last+1,fai);

n = array\_len+1;

int mark = flen - last - array\_len+1; //从字符串str的mark处开始截取长度为array\_len的串并合并

merge(mark,array\_len,last,rk,sa,fai,str); //第一次合并只拷贝长度为last的数组

int lens = flen-mark;

Fai(sa,rk,lens+2,fai);

updateLR(mark-1,str);

mark = mark - array\_len;

do{

printf("mark:%d\n",mark);

if(mark<0) break;

merge(mark,array\_len,flen-mark-array\_len+1,rk,sa,fai,str);

lens = flen-mark;

Fai(sa,rk,lens+2,fai);

updateLR(mark-1,str);

mark = mark - array\_len;

}while(mark-array\_len>=-array\_len);

Fai(sa,rk,n,fai);

}

void printName(FILE \*p,int i){

fprintf(p,"%s","r");

if(i<10) fprintf(p,"%s","00000");

else if(i<100) fprintf(p,"%s","0000");

else if(i<1000) fprintf(p,"%s","000");

else if(i<10000) fprintf(p,"%s","00");

else if(i<100000) fprintf(p,"%s","0");

fprintf(p,"%d",i);

fprintf(p," ");

}

void writeASamLine(FILE\*p,int ismate,int flag,CIGAR c,int pos1,int pos2,char frag[]){

//参数 文件指针，flag值，matepair是否匹配上，CIGAR信息，本身位置，matepair位置，序列信息

fprintf(p,"\n");

printName(p,LINE); //打印序列名字

if(flag<100) fprintf(p," ");

fprintf(p," %d",flag); //打印flag标识

fprintf(p," ref ");

fprintf(p," %d ",pos1); //打印匹配的位置，如果匹配到多个位置选择最前的

if(c.pointer==9) fprintf(p,"%30s","70M"); //如果最优结果是全部匹配上（包括错配，则打印）

else{

int marks = 0;

for(int k=(c.pointer+1)/2;k>2;k--){

fprintf(p," ");

}

for(int k = c.pointer+1;k<10;k++){ //否则打印cigar中信息

if(c.pos[k]-marks>1){

fprintf(p,"%ld",(long)(c.pos[k]-marks));

fprintf(p,"%c",'M');

}

fprintf(p,"%d",1);

fprintf(p,"%c",c.cigarMark[k]);

marks = c.pos[k];

}

fprintf(p,"%d",ST-2-marks);

fprintf(p,"%c",'M');

}

if(ismate==1) { //如果matepair能匹配上则打印 = matepair位置 以及上下游距离

fprintf(p," %c",'=');

fprintf(p," %d",pos2);

fprintf(p," %d",pos2-pos1+ST-1);

}

else{ //否则打印 \* 0 0

fprintf(p," %c",'\*');

fprintf(p," %d",0);

fprintf(p," %d",0);

}

fprintf(p," %s",frag); //打印序列信息

}

void writeSamLine(FILE\*p,int is1,int is2,int flag,int flag2,char frag1[],char frag2[]){

int pos1=0;

int pos2=0;

if(is1==1) pos1= position[posPointer+1];

if(is2==1) pos2 = position2[posPointer2+1];

if(is1==1&&is2==1){

writeASamLine(p,is2,flag,cigar1,pos1,pos2,frag1);

writeASamLine(p,is1,flag2,cigar2,pos2,pos1,frag2);

}else if(is1==0&&is2==1){

writeASamLine(p,is1,flag2,cigar2,pos2,pos1,frag2);

}else if(is1==1&&is2==0){

writeASamLine(p,is2,flag,cigar1,pos1,pos2,frag1);

}

}

int main()

{

FILE \*fp;

charTable['A']='T';

charTable['C']='G';

charTable['G']='C';

charTable['T']='A';

fp = fopen("DNA.txt","r");

if(fp==NULL){

printf("no");

exit(1);

}else{

int k = 1;

for(int i = 0;i<5;i++){

fscanf(fp,"%s",buffer);

}

while(!feof(fp)){

fscanf(fp,"%s",str\_1+k);

k = strlen(str\_1);

}

flen = k;

}

fclose(fp);

array\_len = 190000;

last = flen%array\_len;

printf("arraylen:%d\n",array\_len);

printf("last:%d\n",last);

CSA(str\_1,rk\_1,sa\_1,fai\_1);

FILE \*fp3;

fp3 = fopen("RK.txt","w");

for(int i = 1;i<=flen+1;i++){

fprintf(fp3, "%d\n", rk\_1[i]);

}

fclose(fp3);

getBWT(sa\_1,BWT\_1,occA\_1,occC\_1,occG\_1,occT\_1,str\_1);

ca = la -1;

cc = ra==lc?ra:lc -1;

cg = rc==lg?rc:lg -1;

ct = rg==lt?rg:lt -1;

for(int i = 0,j = flen-1;i<=j;i++,j--){ //得到反向序列

str\_R[i] = str\_1[j];

str\_R[j] = str\_1[i];

}

clear(MAX,bac);

CSA(str\_R,rk\_R,sa\_R,fai\_R); //求反向序列的CSA

getBWT(sa\_R,BWT\_R,occA\_R,occC\_R,occG\_R,occT\_R,str\_R); //转化为BWT

printf("\nlen:%d",strlen(str\_R));

printf("\nlen\_l:%d",strlen(str\_1));

FILE \*fp4;

fp4 = fopen("RK\_R.txt","w");

for(int i = 1;i<=flen+1;i++){

fprintf(fp4, "%d\n", rk\_R[i]);

}

fclose(fp4);

ReadFileInX();

str\_1[flen] = '$';

printf("ok");

return 0;

}