ICS 35.040
CCS L 71

团　　体　　标　　准

T/AI 133.1—2025

信息技术 基因数据压缩 第1部分：测序数据

Information technology — Genomic data compression —

Part 1: sequencing data

2025－11－19 发布　　2025－11－19 实施

中关村视听产业技术创新联盟　发　布

目　　次

前　　言 II
引　　言 III
1 范围 1
2 规范性引用文件 1
3 术语和定义 1
4 缩略语 3
5 约定 3
5.1 概述 3
5.2 算术运算符 3
5.3 逻辑运算符 4
5.4 关系运算符 4
5.5 赋值 4
6 位流语法规则 4
6.1 描述方法 4
6.2 描述符 6
6.3 函数 6
7 测序数据编码 7
7.1 编码框架 7
7.2 位流的语法和语义 19
8 解码 36
8.1 解码流程 36
8.2 头部码流解码 36
8.3 尾部码流解码 36
8.4 压缩数据码流解码 36
附　录　A　（资料性）　简并碱基规则 43
附　录　B　（规范性）　序列识别码数字组分解码详细步骤 44
附　录　C　（规范性）　差分位置及错配碱基还原过程 45
附　录　D　（规范性）　质量分数编码上下文模型建立 46

前　言

本文件按照GB/T 1.1—2020《标准化工作导则 第1部分：标准化文件的结构和起草规则》的规定起草。

本文件是T/AI 133《信息技术 基因数据压缩》的第1部分。T/AI 133已经发布以下部分：

——第1部分：测序数据。

请注意本文本的某些内容可能涉及专利。本文件的发布机构不承担识别这些专利的责任。

本文件由数字音视频编解码技术标准工作组提出。

本文件由中关村视听产业技术创新联盟归口。

本文件起草单位：深圳大学、鹏城实验室、深圳华大生命科学研究院、哈尔滨工业大学、西安电子科技大学、深圳华大智造科技股份有限公司、华为技术有限公司、香港城市大学。

本文件主要起草人：朱泽轩、张勇、徐讯、刘贤明、牛毅、陈毓新、李胜康、王荣杰、王诗淇、谢少辉、马明明、周倩、江俊君、孙杰、黎宇翔、古圣昌、郭良顺、谢寅龙、曾文君、魏晓峰、单日强、刘元盛、曾湘祥、黄铁军、高文。

引　言

T/AI 133《信息技术 基因数据压缩》是为适应基因测序、基因序列比对、分析、数据传输等应用中对基因数据压缩的需求而制定的。T/AI 133的编制目的是建立统一的基因数据压缩编解码规范，促进不同平台和系统间的互操作与高效传输。T/AI 133采用一系列技术来达到基因数据的高效率压缩编码包括基于输入性参考序列的碱基序列压缩编码、基于自组装参考序列的碱基压缩编码、基于自适应上下文的质量分数压缩编码等，拟由以下五个部分构成：

——第1部分：测序数据。目的在于确立适应于基因测序数据中序列识别码、碱基序列、质量分数等的压缩编解码。

——第2部分：比对数据。目的在于确立基因数据比对结果中参考基因组序列、高通量碱基序列、比对位置、错配信息等的压缩编解码。

——第3部分：分析结果数据。目的在于确立基因测序数据变异检测结果中参考序列、变异位置、变异结果等的压缩编解码。

——第4部分：参考软件。目的在于提供T/AI 133第1、2、3等部分所定义工具集的无歧义、可执行标准实现，是编码器是否符合标准的一致性参考。

——第5部分：符合性测试。目的在于定义如何测试验证编码位流和解码器是否满足T/AI 133所规定的要求。

信息技术 基因数据压缩
第1部分：测序数据

1 范围

本文件给出对高通量测序产生的FASTQ和FASTA数据的压缩数据描述，包括基因组序列、序列识别码、测序碱基序列、质量分数等的编码和解码。

本文件适用于基因测序数据产生、分析、传输等过程中涉及数据压缩的相关应用。

2 规范性引用文件

下列文件中的内容通过文中的规范性引用而构成本文件必不可少的条款。其中，注日期的引用文件，仅该日期对应的版本适用于本文件；不注日期的引用文件，其最新版本（包括所有的修改单）适用于本文件。

GB/T 29859—2013 生物信息学术语

GB/T 35890—2018 高通量测序数据序列格式规范

GB/T 1988—1998 信息技术 信息交换用七位编码字符集

3 术语和定义

下列术语和定义适用于本文件。

3.1

参考基因组 reference genome

通过对一个或多个个体的DNA进行测序而组装得到的一组核酸序列。

注：参考基因组是一个物种的理想个体生物中一组基因的代表实例。

3.2

测序 sequencing

测定氨基酸或者核苷酸序列的过程。

[来源：GB/T 29859—2013,2.4]

3.3

测序序列 reads

高通量测序平台产生的序列片段，包含序列识别码、碱基序列和质量分数信息。

[来源：GB/T 35890—2018,3.2，有修改]

3.4

错配 mismatch

序列比对过程中碱基序列和参考基因组或参考序列不匹配的碱基。

3.5

EBML 元素 EBML elements

EBML的基本单元，包含元素ID、元素数据大小和元素数据三部分。

3.6

反向互补 reverse complement

将核酸序列中每个碱基替换为其互补碱基并将所得序列反向排列的操作。

3.7

FASTA 格式 FASTA format

基于文本的、保存生物序列(可以是DNA序列或蛋白质序列)的、每行表示一条序列的标准格式。

3.8

FASTQ 格式 FASTQ format

基于文本的、保存生物测序序列的、每四行表示一条序列的标准格式。

[来源：GB/T 35890—2018, 3.9，有修改]

3.9

简并碱基 ambiguous bases

用于表示可能为多种碱基的符号。

注：具体对应规则见附录A。

3.10

碱基序列 base sequence

测序片段中记录碱基排列的字符串，

注：碱基序列中每个碱基使用大写字母（A、T、C、G和N）或小写字母（a、t、c、g和n）表示，其中字母A和a表示腺嘌呤，字母T和t表示胸腺嘧啶，字母C和c表示胞嘧啶，字母G和g表示鸟嘌呤，字母N和n表示未测定的碱基。

[来源：GB/T 35890—2018, 3.6]

3.11

魔数 magic number

一段用于在码流头部或尾部识别编码类型或格式的预定义字符序列。

3.12

长读长测序 long-read sequencing

对单个DNA分子直接进行测序的技术。

注：单分子测序技术在测序时不需要经过聚合酶链式反应扩增。

3.13

序列比对 sequence alignment

将碱基序列和参考基因组或者参考序列进行字符串匹配的过程。

3.14

序列识别码 sequence identifier

用于识别一段生物序列的具有唯一性的字符串。

注：对于FASTA序列来说，序列识别码对应每条序列的首行，一般为染色体信息；对于FASTQ格式来说，序列识别码对应每条测序序列的首行，一般为测序片段属性信息如ID、测序平台、读长等。

[来源：GB/T 35890—2018, 3.5]

3.15

质量分数 quality score

FASTQ数据格式中每一条测序序列的第四行，与第二行碱基序列的等长的字符串，每个字符代表对应碱基的测序质量。

注：常见的质量分数体系为Phred+33和Phred+64。其中，Phred+33体系质量分数0对应GB/T 1988—1998中规定的信息交换码33，Phred+64体系质量分数0对应GB/T 1988—1998中规定的信息交换码64。

[来源：GB/T 35890—2018, 3.8,有修改]

3.16

自组装参考序列 self-assembled reference sequence

不借助外部参考基因组，通过寻找原始碱基序列之间的重叠组装得到的参考序列。

4 缩略语

下列缩略语适用于本文件。

ACO 自适应编码顺序（Adaptive Coding Order）
AMGC 基于匹配的自适应基因压缩(Adaptive Match-based Genomic Compression)
BSC 块排序压缩器（Block Sorting Compressor）
CRC32 循环冗余校验（Cyclic Redundancy Check 32）
DNA 脱氧核糖核酸（Deoxyribonucleic Acid）
EBML 可扩展二进制元语言（Extensible Binary Meta Language）
GenBank 美国国家生物技术信息中心基因序列数据库（National Center for Biotechnology Information Genetic Sequence Database）
GZ GNU压缩（GNU zip）
ID 标识号（Identifier）
LZMA Lempel-Ziv-Markov链算法（Lempel-Ziv-Markov Chain-algorithm）
MD5 消息摘要算法第五版（Message-Digest Algorithm 5）
METC 基于最高位平面和流道列上下文的序列识别码编码（Meta Encoding based on Max Bit Plane and Tile Context）
RefSeq 美国国家生物技术信息中心参考序列数据库（National Center for Biotechnology Information Reference Sequence Database）
VINT 可变长整数（Variable-length Integer）
XXH3 非加密哈希算法（xxHASH-XXH3）

5 约定

5.1 概述

本部分中使用的数学运算符和优先级与C语言使用的基本一致，除特别说明外，约定编号和计数从0开始。

5.2 算术运算符

算术运算符描述见表 1。

表 1 算术运算符描述

算术运算符	描述
+	加法运算
-	减法运算（二元运算符）或取反（一元前缀运算符）
×	乘法运算
a^b	幂运算，表示a的b次幂。也可表示上标
/	整除运算，结果截断取整
$\frac{a}{b}$	除法运算，不做截断或四舍五入

5.3 逻辑运算符

逻辑运算符描述见表 2。

表 2 逻辑运算符描述

逻辑运算符	描述
a && b	a和b之间的与逻辑运算
a || b	a和b之间的或逻辑运算
!	逻辑非运算

5.4 关系运算符

关系运算符描述见表 3。

表 3 关系运算符描述

关系运算符	描述
>	大于
>=	大于或等于
<	小于
<=	小于或等于
==	等于
!=	不等于

5.5 赋值

赋值运算符描述见表 4。

表 4 赋值运算符描述

赋值运算符	描述
=	赋值运算符
++	递增，x++相当于$x = x + 1$。当用于数组下标时，在自加运算前先求变量值
--	递减，x--相当于$x = x - 1$。当用于数组下标时，在自减运算前先求变量值
+=	自加指定值，例如x += 3相当于$x = x + 3$，x += (-3)相当于$x = x + (-3)$
-=	自减指定值，例如x -= 3相当于$x = x - 3$,，x -= (-3)相当于$x = x - (-3)$

6 位流语法规则

6.1 描述方法

位流语法描述方法类似C语言。位流的语法元素使用粗体字表示，每个语法元素通过名字（用下划线间隔的英文字母组，所有字母都是小写）、语法和语义来描述。语法表和正文中语法元素的值用常规字体表示。语法元素值的助记符和变量值的助记符与它们的值之间的关系在正文中说明。助记符由一个或多个使用下划线分隔的字母组表示，每个字母组以大写字母开始，也可包括多个大写字母。条件语句中0表示FALSE，非0表示TRUE。

语法表描述所有符合本部分的位流语法的超集，附加的语法限制在相关小节中说明。

语法描述的伪代码示例见表 5。当语法元素出现时，表示从位流中读一个数据单元。

表 5 语法描述的伪代码

码流描述	描述符
/*语句是一个语法元素的描述符，或者说明语法元素的存在、类型和数值，下面给出两个例子。*/	
syntax_element	u(2)
conditioning statement	
/*花括号括起来的语句组是复合语句，在功能上视作单个语句。*/	
{	
Statement	
Statement	
…	
}	
/*“while”语句测试condition是否为TRUE，如果为TRUE，则重复执行循环体，直到condition不为TRUE。*/	
while (condition)	
Statement	
/*“do … while”语句先执行循环体一次，然后测试condition是否为TRUE，如果为TRUE，则重复执行循环体，直到condition不为TRUE。*/	
Do	
Statement	
while (condition)	
/*“if … else”语句首先测试condition，如果为TRUE，则执行primary语句，否则执行alternative语句。如果alternative语句不需要执行，结构的“else”部分和相关的alternative语句可忽略。*/	
if (condition)	
primary statement	
Else	
alternative statement	
/*“for”语句首先执行initial语句，然后测试condition，如果conditon为TRUE，则重复执行primary语句和subsequent语句直到condition不为TRUE。*/	
for (initial statement; condition; subsequent statement)	
primary statement	

解析过程和解码过程用文字和类似C语言的伪代码描述。

6.2 描述符

描述符表示不同语法元素的解析过程，描述符说明见表 6。

表 6 描述符说明

描述符	说明
u(*n*)	*n*位无符号整数。如果*n*是'v'，其位数由其他语法元素值确定。
uc(*n*)	*n*位无符号字符单元。如果*n*是'v'，其位数由其他语法元素值确定。
vi	变长无符号整数。数据结构包含三部分即前缀、终止位和实际记录数值。前缀为连续*k*个'0'表示数据长度为*k*+1个字节，终止位为第一个'1'，实际记录数值为终止位后面7*k*+7位。 **示例：** "01000000 00000001"前缀1个0表示数据长度为2个字节，实际记录数值为"0000000 00000001"。
eb	由两个vi和一个uc组成的EBML元素。第一个vi实际记录数值*n*表示该EBML元素ID，第二个vi实际记录数值*m*表示后续uc的长度为*m*字节，最后uc(8*m*)表示该EBML元素实际记录数值。 **示例：** "10000011 10000010 00010010 00110100"第一vi数值为"10000011"，其实际记录数值为"0000011"即该EBML元素ID为3。第二个vi数值为"10000010"，其实际记录数值为"0000010"即表示后续uc数值长度为2字节，该EBML元素实际记录数值为"0001001000110100"。
ac(*k*,*n*)	阶数为*k*，取值范围为*n*的自适应算术编码器编码码流，使用前序*k*个字符作为上文对当前字符概率进行预测。如果*k*或*n*是'v'，其数值由其他语法元素值确定。
rc(*k*,*n*)	阶数为*k*，取值范围为*n*的自适应区间编码器编码码流，使用其前序*k*个字符作为上文对当前字符概率进行预测。如果*k*或*n*是'v'，其数值由其他语法元素值确定。
src(*n*)	多字段混合自适应区间编码器编码码流，阶数为0，取值范围为*n*。多个字段按预定顺序依次使用对应区间编码编入同一码流。不同字段可使用不同*n*值。
qs(*n*)	用*n*阶上下文模型驱动的多值算术编码器对一个字符的编码码流，*n*阶上下文模型的构建见7.1.6.5 。
lz	LZMA编码码流，按LZMA解码器解码
bs	采用块排序压缩器BSC编码的码流

6.3 函数

6.3.1 概述

函数由函数名及左右圆括号内的参数构成，函数也可没有参数。

6.3.2 num_chars(str)

返回字符串str中字符的个数。

6.3.3 num_tokens(str)

以所有非字母（按GB/T 1988—1998中规定的信息交换码从65到90以及97到122）或非数字（按GB/T 1988—1998中规定的信息交换码从48到57）的字符为间隔符将字符串str进行间隔，返回间隔后子串数量。

示例：

str=“@ERR966765.1 HS12_14113:3:2308:5268:18887#4/1”,间隔后得到10个子串 ["ERR966765", "1", "HS12", "14113", "3", "2308", "5268", "18887", "4", "1"]，函数返回10。

6.3.4 token(str, i)

以所有非字母（按GB/T 1988—1998中规定的信息交换码从65到90及97到122）或非数字（按GB/T 1988—1998中规定的信息交换码从48到57）的字符为间隔符将字符串str进行间隔，返回间隔后第*i*个子串（从0开始计数）。

示例：

str=“@ERR966765.1 HS12_14113:3:2308:5268:18887#4/1”,间隔后得到10个子串 ["ERR966765", "1", "HS12", "14113", "3", "2308", "5268", "18887", "4", "1"]， token(str，2)返回"HS12"。

6.3.5 is_number(str)

判断字符串str是否由全数字字符构成，是则返回TRUE，否则返回FALSE。

6.3.6 next_ebml_id()

返回码流当前位置后续EBML元素ID（EBML元素说明见表 6中eb描述符说明）。

6.3.7 ebml_value(e)

返回EBML元素e的实际记录数值（EBML元素说明见表 6中eb描述符说明）。

6.3.8 ebml_length(e)

返回EBML元素e的长度，即第二个vi的实际数值（EBML元素说明见表 6中eb描述符说明）。

6.3.9 cardinality (str)

返回二进制码流str中‘1’的个数。

7 测序数据编码

7.1 编码框架

7.1.1 概述

测序数据编码主要由三部分构成，包括头部码流、压缩数据码流和尾部码流。其中头部码流主要包含码流基础信息和压缩基础信息。压缩数据码流针对FASTQ和FASTA数据格式，主要包括压FASTQ数据分块压缩码流和FASTA整体压缩数据码流等。尾部码流主要包括数据块统计信息、压缩方法参数、自定义信息等。码流概况见图 1。

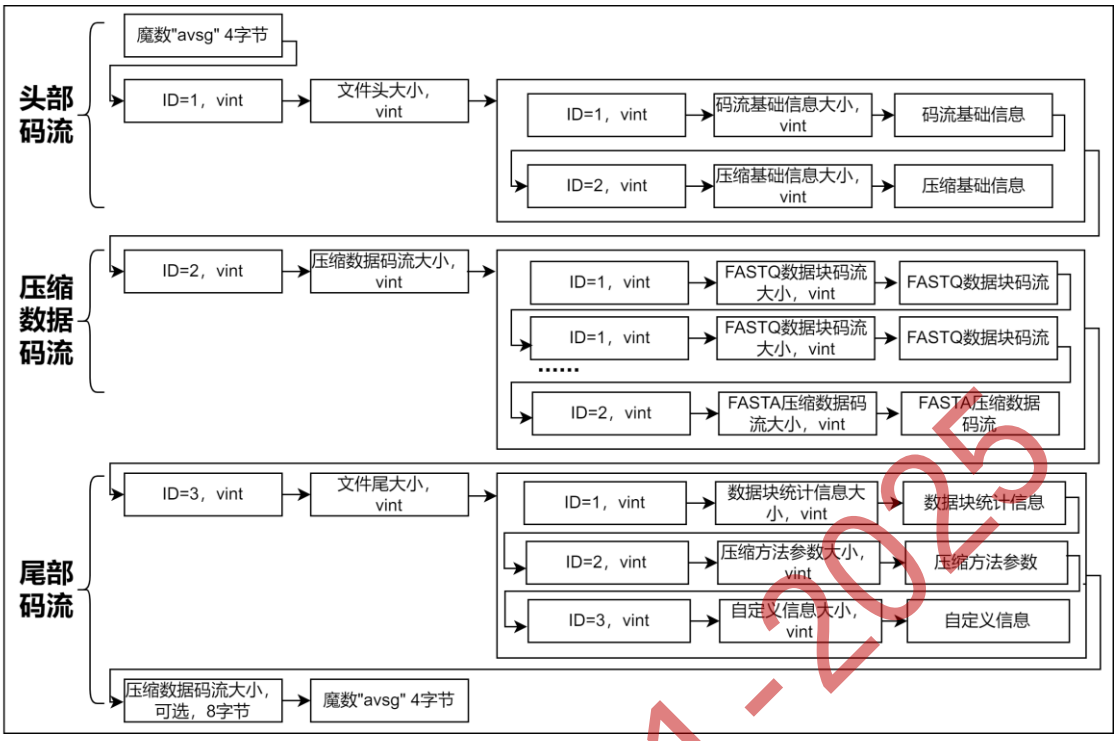

图 1 码流概况

7.1.2 系统编码

7.1.2.1 头部码流

7.1.2.1.1 概述

头部码流主要记录编解码必需的基础信息包括码流基础信息和压缩基础信息。

7.1.2.1.2 码流基础信息

码流基础信息的内容依次包含标准类型、标准版本号、编码器标识符、原始文件名、原始文本字节大小、原始GZ字节大小共六类信息，码流基础信息码流结构见图 2。

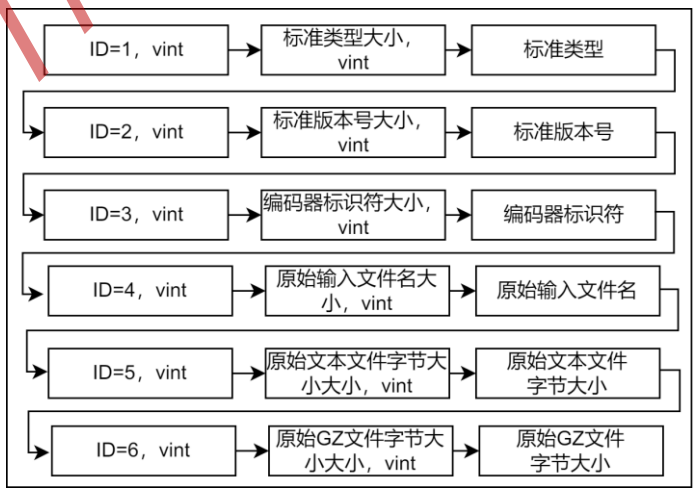

图 2 码流基础信息码流结构

7.1.2.1.3 压缩基础信息

压缩基础信息的内容依次包含原始输入数据类型、原始输入测序序列第三行是否单字符（针对FASTQ数据）、是否长读长序列数据、校验算法、原始输入数据校验码、压缩校验码、FASTQ数据统计信息，压缩基础信息码流结构见图 3。

图 3 压缩基础信息码流结构

7.1.2.2 压缩数据码流

7.1.2.2.1 概述

压缩数据码流可编码FASTQ或FASTA数据。FASTQ数据采用分块编码，FASTA数据采用整体编码。

7.1.2.2.2 FASTQ 数据块编码

单个FASTQ数据块结构见图 4，包括数据块基础信息、序列识别码校验码、序列识别码数据码流（编码规则详见7.1.3 ）、序列长度校验码、序列长度数据码流（编码规则详见7.1.4 ）、碱基序列校验码、碱基序列数据码流（编码规则详见7.1.5 ）、质量分数校验码、质量分数数据码流（编码规则详见7.1.6 ）和自定义信息。

7.1.2.2.3 FASTA 数据编码

FASTA数据码流包括RefSeqID、GenBankID、序列识别码校验码、序列识别码码流、碱基序列大小写标记验证码、碱基序列大小写标记、碱基序列校验码、碱基序列码流（编码规则详见7.1.7 ）和自定义信息。FASTA数据码流结构见图 5。

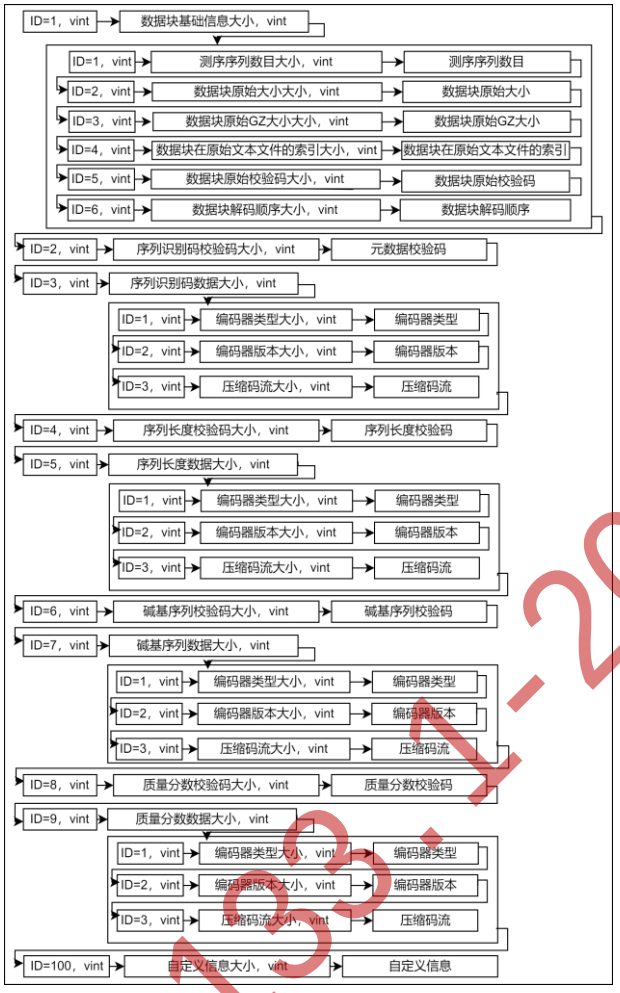

图 4 FASTQ 数据块码流结构

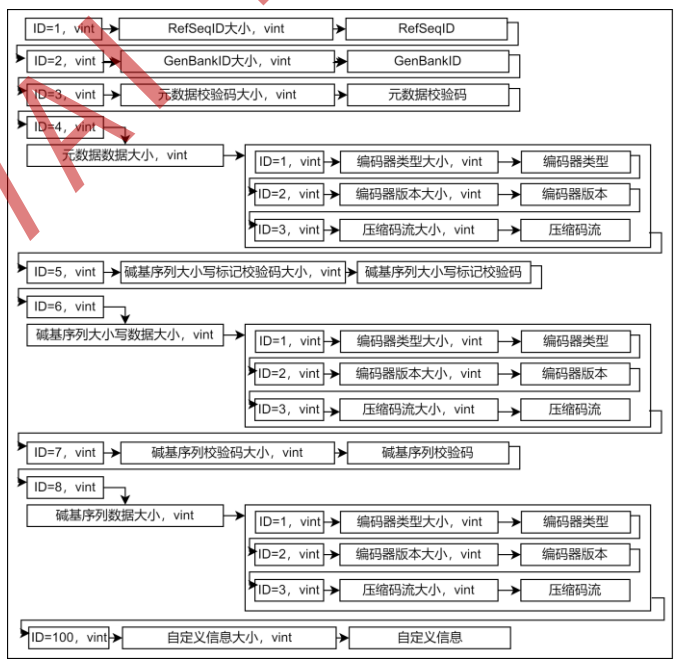

图 5 FASTA 数据码流结构

7.1.2.3 尾部码流

7.1.2.3.1 概述

尾部码流记录编码统计信息包括数据块统计信息、压缩方法参数及其他自定义信息。尾部码流数据结构见图 6。

图 6 尾部数据码流结构

7.1.2.3.2 数据块统计信息

数据块统计信息包括数据块总数、表示单个数据块字节大小的位数、最大数据块原始大小、各数据块的原始大小列表、各数据块的压缩后大小列表、表示单个偏移量的位数、各数据块原始起始偏移量列表和各数据块在压缩码流中起始偏移量列表。

7.1.2.3.3 压缩方法参数

压缩方法参数包括序列识别码压缩全局参数、碱基序列长度压缩全局参数、碱基序列压缩全局参数、质量分数压缩全局参数。其中碱基序列压缩全局参数又包括算术码阶数、是否使用有参压缩、RefSeqID、GeneBankID、参考序列文件名、参考序列内容校验码、参考序列压缩编码方法、参考序列压缩编码方法版本、参考序列压缩码流和质量分数压缩全局参数。

7.1.2.3.4 自定义信息

自定义信息由编码器自行定义，兼容其他类型编码器。

7.1.3 FASTQ 序列识别码编码规则

7.1.3.1 概述

FASTQ序列识别码为记录测序信息的文本，通常具有特定格式，编码可以采用LZMA、基于窗口划分编码或基于最高位平面嵌入和流道列上下文编码方法。

7.1.3.2 基于窗口划分编码

7.1.3.2.1 概述

基于窗口划分的编码以符号（所有不为字母或数字的字符）为分隔标志把首个序列识别码拆分成多个组分，根据每个组分的特点，对后续序列识别码的各个组分进行针对性编码。对于相邻的序列识别码，若各个位置上的组分特征相同，则归为同一序列识别码块，若存在差异，则被归入不同序列识别码块，编码包括两个内容即序列识别码块信息和序列识别码块。

7.1.3.2.2 序列识别码块信息

序列识别码块信息以8字节为一个单位，假设有n个序列识别码块，则序列识别码块信息有$8n$字节。每个块信息前4个字节表示对应块内序列识别码数量，后4个字节表示本序列识别码块首个序列识别码长度。

7.1.3.2.3 序列识别码块

各序列识别码块前后直接拼接。单个序列识别码块中的各组分的格式都与首个序列识别码的各组分格式相同。如首个序列识别码的组分个数为M，则组分数据由M+1个编码组成即首个序列识别码和M个组分。根据组分格式，组分可以分为数字组分和非数字组分。所有序列识别码在进行组分划分后，相同顺位的组分都为纯数字组分，该组分即为数字组分，否则为非数字组分。

首个序列识别码作为其他序列识别码编码参照需要对逐个字符进行编码，字符取值范围为所有GB/T 1988—1998中规定的信息交换码字符（区间大小等于128）。

对于纯数字组分，根据以下判定细分为四类：

a）若组分长度大于 8，或出现前缀 0，则将其作为非数字组分处理；

b）若组分长度小于等于 8，且其前一个组分内容为“length”，则将其作为序列长度处理，无需额外编码，解码时可通过序列长度恢复；

c）若非前两种情况，且其前一个字符为“/”，且组分长度为 1，则认定为单端测序尾部信息，值恒定为 1，无需额外编码；

d）若非前三种情况，则用以下两个部分进行记录：

1）差值类型:记录当前序列识别码该组分与上一个序列识别码该组分的差值的类型。类型 0 表示二者差值为 0；1 表示二者差值为 1；2 表示二者差值为[2,4096]区间内的整数；3 表示二者差值为[-4096,-1]区间内的整数。

2）差值：记录当前序列识别码组分与上一个序列识别码该组分的差值绝对值（不为 0 或 1 时在该编码中记录，否则在差值类型的编码中记录）。

字母或字母数字混合的组分都属于非数字组分，非数字组分编码包含以下两个部分：

——标识符：取值范围为 4。其中 0 表示该组分内容与首个序列识别码对应组分相同；1 表示该组分内容与首个序列识别码对应组分长度相同但内容不同；2 表示当前组分长度大于首个序列识

别码对应组分；3 表示当前组分长度小于首个序列识别码对应组分。该标识符用范围为 4 的编码器编码。当取 2 和 3 时，会编码额外的长度差值。当取 2 时，会增加额外编码码流。

——不相同字符：取值范围为 64，包括 52 个字母、10 个数字和 2 个预留的终止符。

7.1.3.3 基于最高位平面嵌入和流道列上下文的编码

7.1.3.3.1 概述

在基于窗口划分的编码基础上，划分各组分具有明确的物理意义。其中第4组分通常表示流道列编号，第6和第7组分分别表示测序时的物理坐标X和Y。流道列和物理坐标具有乱序性，针对这三组分可采用最高位平面嵌入和流道列上下文的编码方法。

7.1.3.3.2 流道列和物理坐标编码

基于最高位平面嵌入和流道列上下文的编码过程如下：

a) 流道列数量是可枚举的，共有 96 种。将流道列数值千位数和百位数做为上下文，将十位数和个位数联合起来采用 4 位组合 16 值算术编码。流道列的十位数和个位数联合后的取值范围为[1,16]；

b) 根据流道列数据对物理坐标 X 和 Y 进行分组，分组方式为基于流道列十位数和个位数的值为依据（即 0~16），分为 16 组。同时，为捕捉 X 和 Y 分组后的线性信息，采用组内的第 n+1 个组分的数值与第 n 个组分的数值做差值的方式处理数据，将差值作为信息采用 4 位组合 16 值算术编码。

c) 对于分组后的 X 和 Y 差值数据，计算差值的最高位平面，将最高位平面嵌入 4 位组合 16 值算术编码，采用联合编码形式（见图 7）。同时对于剩余位平面部分，将最高位平面用作上下文，捕捉 X 和 Y 坐标潜在的排序信息，使用 4 位组合 16 值算术编码器编码；

d) 由步骤 a)和 b）得到 X 和 Y 上下文模型的序号的计算公式：序号 = 组号×位平面×最高位平面。最终的上下文个数为 16×10×16=2560 个，位平面预设的最大值为 10，但实际上位平面的范围分布在[3,6]之间，因此实际使用的上下文应该是 1024 个，编码器根据上下文模型序号进行遍历编码。

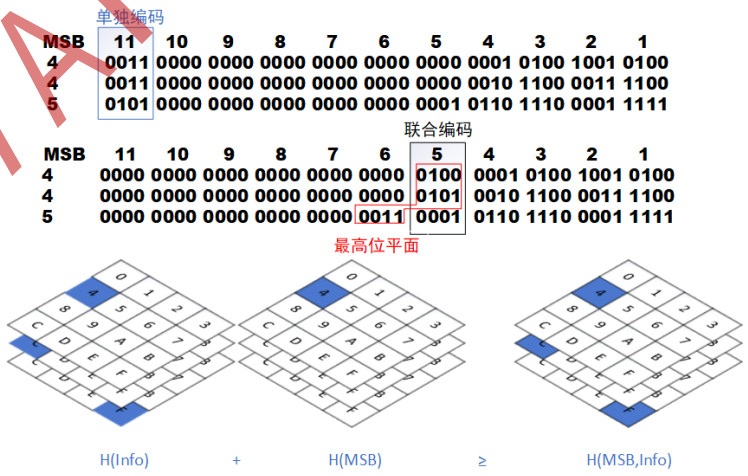

图 7 最高位平面嵌入联合编码示意图

7.1.4 FASTQ 序列长度编码规则

7.1.4.1 概述

FASTQ序列长度编码记录各个碱基序列长度（质量分数串长度相同）。将各个序列长度与其前一个长度进行比较。若编码数据为短读长序列，则每个短读长度采用等长标识符和两个字节（末字节和次末字节）表示；若编码数据为长读长序列，则每个短读长度采用等长标识符和四个字节（末字节、次末字节、次高字节和最高字节）表示。

7.1.4.2 等长标识符

表示当前序列是否与上一条序列等长的布尔值。

7.1.4.3 末字节

若当前序列与上一条序列不等长，编码当前序列长度最低8位的字节。

7.1.4.4 次末字节

若当前序列与上一条序列不等长，编码当前序列长度次低8位的字节。

7.1.4.5 次高字节

若当前序列与上一条序列不等长，且序列为长读长，编码当前序列长度次高8位的字节。

7.1.4.6 最高字节

若当前序列与上一条序列不等长，且序列为长读长，编码当前序列长度最高8位的字节。

7.1.5 FASTQ 碱基序列编码规则

7.1.5.1 概述

FASTQ碱基序列包括碱基字符（’A’，’C’，’G’，’T’）或部分简并碱基字符（详见附录A），其编码可以直接采用LZMA或者基于参考序列比对进行编码，通过记录比对结果和错配信息（只考虑替换错误）代替原始序列达到压缩效果，参考序列可以源自外部输入或者自组装。

7.1.5.2 简并碱基编码

碱基序列中若存在简并碱基可对其进行提取单独编码。简并碱基对应质量分数普遍偏低，可将简并碱基在低质量碱基中的位置编码为码流，包括简并碱基存在标识符、简并碱基字符、简并碱基最大质量分数、低质量碱基中非简并碱基个数、非简并碱基间的简并碱基数量。

7.1.5.3 基于输入性参考序列的容错配编码

7.1.5.3.1 概述

基于输入性参考序列的碱基序列编码包含序列在参考序列上的比对信息，包括比对位置、比对方向、错配个数、错配位置、错配字符。存在简并碱基的位置已单独记录，默认为比对成功。

7.1.5.3.2 比对位置

序列比对到参考序列上的位置信息可直接记录，若正向比对则为5’端所在位置，若反向互补比对则为3’端所在位置，最大取值范围为参考序列长度。

比对位置也可以采用基于上下文的位平面比对位置编码，碱基测序中存在一定的局部性，即相邻测序序列大概率来自于参考序列的临近片段，在每个编码块内部，比对位置并不是均匀分布在参考序列的所有区域，可利用高位作为上下文建模低位的条件概率，对比对位置进行编码即 METC 编码：

a) 删除连续相同的比对位置,连续出现几个相同的比对位置仅需保留一个。将每个原始数据中每个比对位置是否与前一个相同用一串二进制序列表示；

b) 使用中值滤波差分处理去重复后的比对位置，差分被减数被选择为前三个比对位置的中间值；

c) 将差分结果编码为位平面，这些平面分为最高有效位和剩余位。最高有效位使用 32 值算术编码器直接编码，剩余位部分最高有效位和每个位所在的位平面在算术编码器编码时用作上下文。

7.1.5.3.3 比对方向

比对方向表示序列是匹配到正向参考序列的具体位置，还是匹配到其反向互补序列上。

7.1.5.3.4 基于二叉树分段匹配的碱基比对

碱基序列的前半段测序质量较高，而后半段较低，可以采用分段匹配的方法。比对不上的测序序列会被分段后再比对，直到记录比对信息所需的空间大于直接将子测序序列送入熵编码器编码所占空间，具体过程如下：

a) 预设一个长度下限;

b) 每输入一个测序序列，对该测序序列进行比对，如果成功比对上则比对结束;

c) 如果未比对上且该测序序列的长度大于长度下限，将该测序二等分，两个子序列分别重复 b)，直到长度小于长度下限。

7.1.5.3.5 错配个数

错配个数表示当前序列比对到参考序列的比对位置后，两个子串之间的差异碱基个数。

7.1.5.3.6 错配位置

错配位置直接记录错配出现在序列上的位置，也可以采用基于自适应二值编码的错配位置编码，将每条碱基序列的每个碱基是否错配当作二值信号进行编码，利用高阶二值上下文建模测序序列中错配趋势的相关性。将碱基是否错配用一串和测序序列等长的二值信号进行编码，用0代表对应位置的碱基匹配上，1代表出错。在顺序编码二值信号时，用当前位置的前10个碱基的出错个数来量化上下文，作为编码字符的概率。

7.1.5.3.7 错配字符

记录错配字符，四种碱基每种可错配成其他三种。

7.1.5.3.8 无比对编码

若某序列无法与输入性参考序列比对成功，则用*k*阶区间编码进行逐个碱基字符的编码。编码中包含额外标记，记录各个序列是采用有比对还是无比对压缩编码。

7.1.5.4 基于自组装参考序列的碱基编码

7.1.5.4.1 概述

采用本编码方式首先采样部分原始碱基数据自组装得到参考序列，再将其余碱基序列比对到参考序列实现编码，编码信息包括自组装参考序列、比对方向、错配碱基及其数量和位置、碱基序列在参考序列的位置或偏移。

7.1.5.4.2 自组装参考序列

自组装参考序列在测序数据种具有尽可能高的代表性，由重复率高的测序碱基序列组成。自组装参考序列采用‘A’,‘C’,‘G’,‘T’的字符数组的形式保存。自组装参考序列代表多条前后重叠的碱基序列。自组装参考碱基压缩编码见图 8。

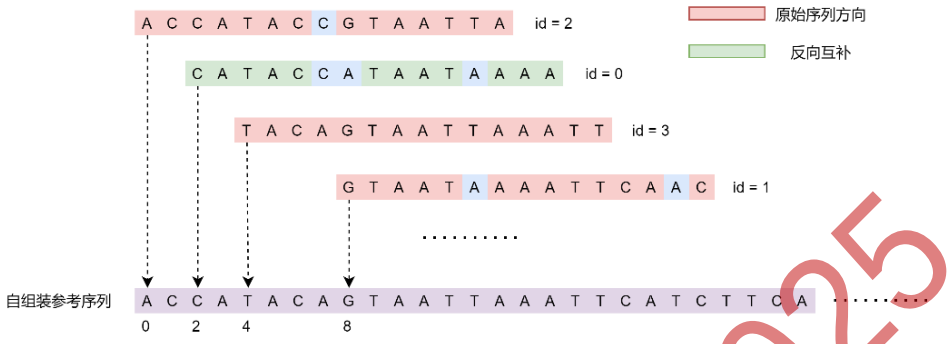

图 8 自组装参考碱基压缩编码

7.1.5.4.3 编码顺序

自组装参考碱基编码支持两种编码顺序即重排序列和保留序列顺序。重排序列模式按照碱基序列在参考序列的比对先后顺序进行编码，以图 8为例，ID为2的碱基序列第一个进行编码，ID为0的碱基序列第二个进行编码，ID为3的碱基序列第三个进行编码，依次类推。保留序列顺序的模式按照碱基序列在原始文件中的先后顺序进行编码，ID为0的碱基序列第一个进行编码，ID为1的碱基序列第二个进行编码，ID为2的碱基序列第三个进行编码，依次类推。

7.1.5.4.4 碱基序列比对方向

碱基序列比对到参考序列的方向可以是原始序列方向，也可以是其反向互补方向，在编码时只考虑其中一种情况并做标记。

7.1.5.4.5 错配碱基

碱基序列与自组装参考序列的比对只考虑碱基替换（即错配）的情况。每个错配碱基采用8位无符号字符保存，简并碱基以错配碱基处理，所有错配碱基组成连续的字符数组。

示例：

以图 8为例，ID为2的碱基序列上的错配碱基为C（自组装参考序列上对应的碱基为A），ID为0的碱基序列上的错配碱基为C、A、A，ID为1的碱基序列上的错配碱基为A、A。

7.1.5.4.6 错配碱基数量

每条碱基序列的错配碱基数量，所有碱基序列的错配碱基数量组成连续的整数数组。

7.1.5.4.7 错配碱基位置

记录碱基序列的错配碱基位置（错配碱基相对碱基序列的位置），每条碱基序列上的错配碱基位置根据其数量的不同分配到不同的数组。

示例：

以图 8为例，ID为2的碱基序列上的错配碱基位置只有7，将位置7按编码顺序放在错配碱基数量为1的错配碱基位置数组中；ID为0的碱基序列上的错配碱基位置有5、6和11，将位置5、6和11按编码顺序放在错配碱基数量为3的错配碱基位置数组中，依次类推。

7.1.5.4.8 碱基序列位置

记录碱基序列在自组装参考序列中的比对位置。对于重排序列顺序的模式，碱基序列位置是一个有序的整数数组，进一步执行差分编码。

示例：

以图 8为例，如按照重排序列的编码顺序，ID为2、0、3、1的碱基序列在自组装参考序列的起始位置分别为0、2、4、8，进行差分编码后得到的位置数组为0、2、2、4；如果按照碱基序列在原始文件的先后顺序编码，ID为0、1、2、3的碱基序列在自组装参考序列的起始位置分别为2、8、0、4，不需要其他处理。

7.1.6 FASTQ 质量分数编码规则

7.1.6.1 概述

FASTQ质量分数采用ACO压缩编码，编码包括碱基标识符、均值标识符、质量分数范围、质量分数位流等。

7.1.6.2 碱基标识符

质量分数代表对应碱基测序结果的正确程度，碱基和质量分数之间有很强的相关性。碱基标识符记录是否采用碱基对质量分数编码模型。当选择采用碱基时，读取每一个质量分数对应的碱基参与构建上下文信息对质量分数进行编解码。

7.1.6.3 均值标识符

每条测序序列的质量分数之间存在相关性，可以通过统计并计算每条测序序列的质量分数平均值来建立混合上下文模型。当选择计算均值时，保存每条质量分数序列的均值并在构建混合上下文模型过程中读取每一条测序序列对应的均值并作为上下文信息。

7.1.6.4 质量分数范围

假设*P*为测序碱基被测序软件识别错误的概率，测序软件根据打分公式计算出该碱基错误率对应的质量分数，错误率*P*越低，则碱基的质量分数得分越高。不同的评分软件有不同的质量分数计算公式，比较常用的有两种格式，第一种是早期测序软件Solexa使用的质量分数计算公式：

$$Q_{Solexa} = -10\log 10\frac{P}{1-P} \quad \cdots\cdots (1)$$

式中：

Q_{Solexa}——测序软件 Solexa 产生的质量分数值，取值范围为 [-5,62];

P ——测序错误率，取值范围[0,1];

另一种是Sanger软件使用的打分公式：

$$Q_{sanger} = -10\log_{10} P \quad \cdots\cdots (2)$$

式中：

Q_{Sanger} ——测序软件Sanger产生的质量分数值，取值范围为 [0,93];

P ——测序错误率，取值范围[0,1];

通过这两种公式计算的质量分数在P较小时几乎一致，P越大则差异愈加明显。为能使用GB/T 1988-1998中规定的信息交换码中可打印的字符范围表示质量分数，软件会选择一个偏移量，例如Q_{Solexa}加上

偏移量64即Phred+64体系，最终范围是[59,126]；Q_{Sanger}加上偏移量33即Phred+33体系，最终范围是[33,126]。

示例：假设一碱基错误概率为 0.1%，则按照 Sanger 软件算出的 Q_{Sanger} 值为 30，对应 Phred+33 体系转换后按 GB/T 1988—1998 中规定的信息交换码值为 66，对应字符为‘?’。

不同质量分数范围影响着编码模型的大小。编码时，计算当前编码块中质量分数的最大值和最小值，根据最大值和最小值确定质量分数范围，进而确定模型大小。将最大值和最小值记做质量分数范围数据流进行压缩编码。

7.1.6.5 上下文模型

对每一个需要编码的质量分数字符构建对应的混合上下文模型，首先选取编码质量分数的四阶上下文模型并进行量化，然后根据是否包含碱基和均值构建适当的混合上下文模型。

示例：

混合上下文模型示例见图 9，蓝色方块表示当前要编码的质量分数字符，绿色方块表示当前编码质量分数字符的前四个字符则q1=35， q2=34， q3=38，q4=35，红色方块表示与当前编码质量分数字符所对应的前两个碱基’A’和’C’，黄色方块表示当前编码质量分数字符所在行的均值M。

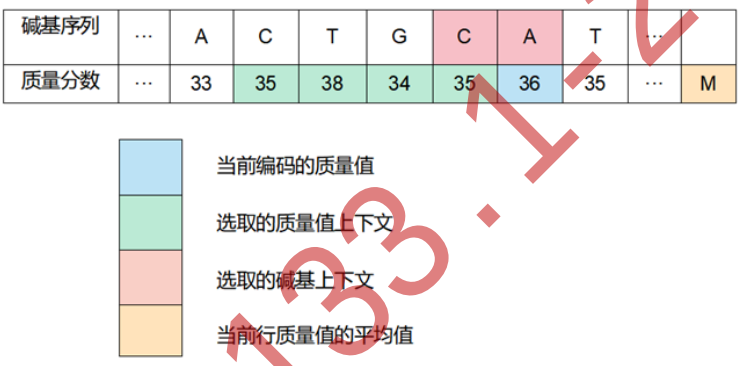

图 9 混合上下文模型

对三种不同的上下文信息进行联合建模，采取分布优化的策略达到最优效果。将质量分数较小的符号合并在一起，以同一字符的表现形式进行编码压缩，在其内部进行二次编码，对具体的字符展开编码。

7.1.6.6 遍历方式

确定好上下文模型之后，可采用常规按行遍历（见图 10左图））或者按列蛇形遍历（见图 10右图）的方式获取数据间的相关性。若采用按列遍历方式，首先向下遍历考虑更多的是行之间的相关性，字符浮动不会过于太大，在编码完红色字符之后没有再从第二列头部重新开始，而是尾尾相连，头头相接。

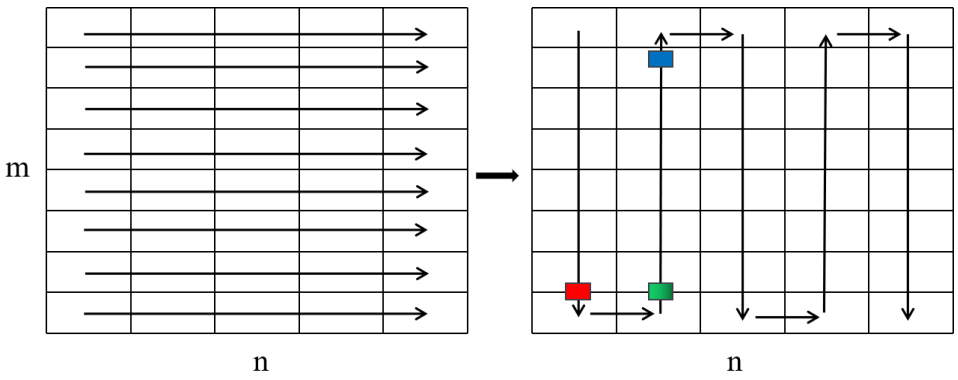

图 10 常规按行遍历方式和按列蛇形遍历方法

7.1.7 FASTA 数据编码规则

7.1.7.1 概述

FASTA数据包含序列识别码和碱基序列，序列识别码直接用LZMA编码。碱基序列如有大小写区分，使用大小写标记进行记录，之后统一转化为大写字符再采用LZMA或者区间编码直接编码或者采用基于最大匹配的有参考编码。

7.1.7.2 基于最大匹配的有参考碱基编码

若存在参考序列，可按照下列步骤查找目标序列与参考序列的最大匹配并进行碱基编码：

a) 对目标序列按预先设定步长s_1采样，将采样到的k长子串存储到布隆过滤器中；

b) 对参考序列进行步长为s_2的采样，此处s_1和s_2互质，在参考序列上采样到的子串在布隆过滤器中查找，如果查找到，则将该子串存入哈希表，否则丢弃；

c) 再一次采样目标序列，并将采样到的子串与哈希表中的子串匹配，找到则在目标序列和参考序列上左右延伸确定最大精确匹配的长度（允许不超过3个碱基错配）；

d) 得到最大精确匹配后，对匹配进行编码分别记录匹配位置、匹配长度和错配内容。

示例：

FASTA碱基最大匹配编码示例见图 11，下划线部分为目标序列与参考序列的最大匹配，中间有两个位置的碱基不匹配。目标序列最终编码为{(18,25),TA,18}，最后一个精确匹配部分不需要记录起始位置。

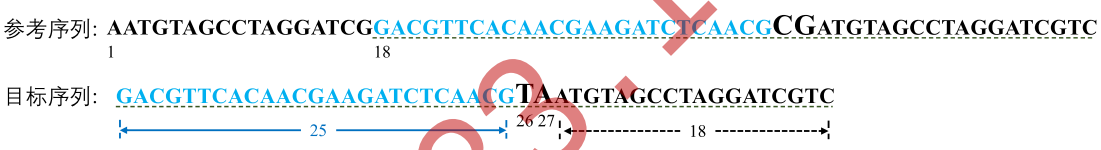

图 11 FASTA 碱基最大匹配编码

7.2 位流的语法和语义

7.2.1 系统编码语法

系统编码描述见表7。

表 7 系统编码描述

系统编码描述	说明	描述
avsg_stream() {	-	-
header_magic_num	头部魔数，用4个字节记录的“avsg”字符串信息，用于识别码流是否为本文件规定的码流。	uc(32)
header_id	数值1，表示接下来内容为头部码流。	vi
header_len	头部码流的长度。	vi
header_stream()	-	-
comp_data_id	数值2，表示接下来内容为压缩数据。	vi
comp_data_len	压缩数据码流的长度（头部），如果数值为0，则长度存放于尾部。	vi
comp_data_stream()	-	-
tailer_id	数值3，表示接下来内容为尾部码流。	vi
tailer_len	尾部码流的长度。	vi
tailer_stream()	-	-

表 7 系统编码描述（续）

系统编码描述	说明	描述
if (comp_data_len==0)	-	-
comp_data_len_alt	表示压缩数据码流的长度（尾部）。	u(64)
tailer_magic_num	尾部魔数，用4个字节记录的“avsg”字符串信息，用于识别码流是否为本文件规定的码流。	uc(32)
}	-	-

7.2.2 头部码流语法

7.2.2.1 头部码流描述

头部码流描述见表 8。

表 8 头部码流描述

头部码流描述	说明	描述
header_stream () {	-	-
basic_id	数值为1，表示接下来内容为基础信息。	vi
basic_len	基础信息长度。	vi
basic_stream()	-	-
comp_info_id	数值为2，表示接下来内容为压缩信息。	vi
comp_info_len	压缩信息长度。	vi
comp_info_stream()	-	-
}	-	-

7.2.2.2 码流基础信息描述

码流基础信息描述见表 9。

表 9 码流基础信息描述

码流基础信息描述	说明	描述
basic_stream () {	-	-
std_type	ID为1，元素内容为标准类型。如内容为 ‘fq’，表示码流为基因测序数据FASTQ；如为字符串‘fa’，表示码流为基因组数据FASTA。	eb
std_version	ID为2，元素内容为当前被编码数据对应的标准版本号。	eb
encoder_id	ID为3，元素内容表示编码器标识符。如使用参考软件写入两个字节的字符串‘ar’，使用其他编码软件可自定义。	eb
if (next_ebml_id()==4)	-	-
raw_name	ID为4，元素内容为原始输入FASTA/FASTQ文件名。	eb
if (next_ebml_id()==5)	-	-
raw_textbyte	ID为5，元素内容为原始文本字节大小，表示编码时原始输入文本的字数。	eb
if (next_ebml_id()==6)	-	-
raw_gzbyte	ID为6，元素内容为原始GZ字节大小，表示原始输入文件为GZ格式。	eb
}	-	-

7.2.2.3 压缩基础信息描述

压缩基础信息描述见表 10。

表 10 压缩基础信息描述

压缩基础信息描述	说明	描述
comp_info_stream () {	-	-
if (next_ebml_id()==1)	-	-
rawfile_type	ID为1，元素内容为原始输入文件类型。0表示文本输入，1表示输入为GZ格式，2表示输入为管道。	eb
plussign_only	ID为2，元素内容表示FASTQ中每条测序记录的第三行是否只包含单一字符‘+’，0表示不是，1表示是。	eb
longseq	ID为3，元素内容表示原始输入是否为长读长测序数据，0表示不是，1表示是。	eb
checkalgo	ID为4，元素内容表示本压缩码流中使用的各个校验码的校验算法，0表示MD5，1表示CRC32，2表示XXH3。	eb
if (next_ebml_id()==5)	-	-
rawtext_check	ID为5，元素内容表示原始输入的文本内容的校验码。	eb
if (next_ebml_id()==6)	-	-
rawcomp_check	ID为6，元素内容表示输入GZ数据的校验码。	eb
if (next_ebml_id()==7)	-	-
fq_stats	ID为7，元素内容为FASTQ数据统计信息。	eb
}	-	-

7.2.3 压缩数据码流语法

7.2.3.1 压缩数据码流描述

压缩数据码流描述见表 11。

表 11 压缩数据码流描述

压缩数据码流描述	说明	描述
comp_data_stream () {	-	-
if(ebml_value(std_type)== 'fq')	-	-
for(i=0; i<ebml_value(block_num); i++) {	-	-
read_block_indiv_id	数值为1表示接下来的内容为单个FASTQ数据块。	vi
read_block_indiv_len	单个FASTQ数据块码流长度。	vi
read_block_indiv_stream()	-	-
}	-	-
if(ebml_value(std_type)== 'fa'){		
fasta_id	数值为2表示接下来内容为FASTA压缩数据码流。	vi
fasta_len	FASTA压缩数据码流长度。	vi
fasta_stream()	-	-
}	-	-
}	-	-

7.2.3.2 FASTQ 数据码流语法

7.2.3.2.1 FASTQ 数据块码流描述

FASTQ数据块码流描述见表 12。

表 12 FASTQ 数据块码流描述

FASTQ数据块码流描述	说明	描述
read_block_indiv_stream () {	-	-
block_basic_id	数值为1表示接下来内容为数据块基础信息。	vi
block_basic_len	数据块基础信息码流的长度。	vi
block_basic_stream()	-	-
if (next_ebml_id()==2)	-	-
block_seqid_check	ID为2，元素内容为序列识别码校验码。	eb
block_seqid_id	数值为3表示接下来内容为序列识别码。	vi
block_seqid_len	序列识别码码流的长度。	vi
block_seqid_stream()	-	-
if (next_ebml_id()==4)	-	-
block_seqlen_check	ID为4，元素内容为序列长度校验码。	eb
block_seqlen_id	数值为5表示接下来内容为序列长度。	vi
block_seqlen_len	序列长度码流长度。	vi
block_seqlen_stream()	-	-
if (next_ebml_id()==6)	-	-
block_seq_check	ID为6，元素内容为碱基序列校验码。	eb
block_seq_id	数值为7表示接下来内容为碱基序列。	vi
block_seq_len	碱基序列码流的长度。	vi
block_seq_stream()	-	-
if (next_ebml_id()==8)	-	-
block_qual_check	ID为8，元素内容为质量分数校验码。	eb
block_qual_id	数值为9表示接下来内容为质量分数。	vi
block_qual_len	质量分数码流长度。	vi
block_qual_stream()	-	-
if (next_ebml_id()==100)	-	
block_encode_custom	ID为100，自定义信息，由编码器写入。	eb
}	-	-

7.2.3.2.2 数据块基础信息码流描述

数据块基础信息码流描述见表 13。

表 13 数据块基础信息码流描述

数据块基础信息码流描述	说明	描述
block_basic_stream () {	-	-
if (next_ebml_id()==1)	-	-
block_readcnt	ID为1，元素内容记录当前数据块的序列数目。	eb

表 13 数据块基础信息码流描述（续）

数据块基础信息码流描述	说明	描述
if (next_ebml_id()==2)	-	-
block_textbyte	ID为2，元素内容记录当前数据块的原始文本大小。	eb
if (next_ebml_id()==3)	-	-
block_gzbyte	ID为3，元素内容记录当前数据块的原始GZ大小。	eb
if (next_ebml_id()==4)	-	-
block_textoffset	ID为4，元素内容记录当前数据块在原始文本中的偏移量。	eb
if (next_ebml_id()==5)	-	-
block_textcheck	ID为5，元素内容记录当前数据块内容校验码。	eb
block_order	ID为6，元素内容记录当前数据块中各存储码流的解码顺序。0表示解码先后顺序为序列识别码-序列长度-碱基序列文本-质量分数文本，1表示解码先后顺序为序列识别码-序列长度-质量分数文本-碱基序列文本。	eb
}	-	-

7.2.3.2.3 序列识别码码流语法

序列识别码码流描述见表 14。

表 14 序列识别码码流描述

序列识别码码流描述	说明	描述
block_id_stream () {	-	-
seqid_encoder	ID为1，元素内容为序列识别码的编码器类型。0表示LZMA编码；1表示基于窗口划分的编码； 2表示基于最高位平面嵌入和流道列的上下文方法编码。	eb
seqid_encoder_ver	ID为2，元素内容为序列识别码编码器版本。	eb
seqid_data_id	数值为3表示接下来内容为序列识别码编码码流。	vi
seqid_data_len	序列识别码编码码流长度。	vi
if(ebml_value(id_encoder)==0)	-	-
seqid_data_lzma	LZMA编码码流	lz
else{	-	-
seqid_chunk_id	数值为1表示接下来内容为序列识别码块内容。	vi
seqid_chunk_len	序列识别码块内容码流长度。	vi
seqid_chunk_stream()	-	-
vec_format	ID为2，元素内容为各序列识别码块序列数目与首序列识别码长度，以8个字节为一单位，每个单位前4个字节存储当前序列识别码块序列数目，后4个字节存储当前序列识别码块首个序列识别码的长度	eb
}	-	-
}	-	-

序列识别码块码流描述见表 15。

表 15 序列识别码块码流描述

序列识别码块码流描述	说明	描述符
seqid_chunk_stream(){	-	-
if (ebml_value(seqid_encoder_ver) == 1){	-	-
for (i = 0; i<ebml_length(vec_format)/8; ++i){	-	-
first_seqid	首个序列识别码码流。	src(128)
for (i = 0; i<num_tokens(first_seqid); ++i){	-	-
if(is_number(token(first_seqid,i))){	-	-
diff_type	记录当前序列识别码该数字组分与上一个序列识别码同组分的差值的类型：0表示二者差值为0；1表示二者差值为1；2表示二者差值大于1；3表示二者差值小于0。	src(4)
diff_scale_n_len	差异值绝对值表示为16进制时的串长度。	src(16)
for (i = 0; i<16; ++i)	-	-
diff_scale_n_num	16进制表示的差异值绝对值。	src(16)
}	-	-
else{	-	-
if(ebml_value(seqid_encoder)== 1){	-	-
char_flag	该组分与首个序列识别码对应组分对比标识符：0表示内容相同；1表示内容不同但长度相同；2表示当前组分长度更大；3表示当前组分长度更小。	src(4)
len_diff	当前组分长度与首个序列识别码对应组分长度差值绝对值。	src(12)
for (j = 0;j<char_flag>2? char_flag-len_diff : char_flag+len_diff; ++j)	-	-
diff_char	差异标识符，不为0时，编码当前组分的每位字符。	src(64)
}	-	-
else{	-	-
tile	流道列，上限为6*16=96。	ac(0，96)
spilt_x	X列坐标。	ac(0，16)
spilt_y	Y列坐标	ac(0，16)
for (j = 0;j<16;++j) {	-	-
next_msb	为下一组的指示最高有效位，以及和正常需要编码的数字一起做联合编码。	ac(0，16)
context_diff_char	分组过后的差异标识符码流，取值范围为16。	ac(0，16)
}	-	-
}	-	-
}	-	-
}	-	-
}	-	-
}	-	-
}	-	-
注：本表中所有src(n)共享同一码流		

7.2.3.2.4 序列长度码流语法

序列长度码流描述见表 16。

表 16 序列长度码流描述

序列长度码流描述	说明	描述
block_seqlen_stream () {	-	-
seqlen_encoder	ID为1，元素内容为序列长度的编码器类型，0表示 LZMA编码；1表示字节间隔编码。	eb
seqlen_encoder_ver	ID为2，元素内容为序列长度的编码器版本。	eb
seqlen_data_id	值为3表示接下来内容为序列长度编码码流。	vi
seqlen_data_len	序列长度编码码流长度。	vi
seqlen_data_stream()	-	-
}	-	-

序列长度数据码流描述见表17。

表 17 序列长度数据码流描述

序列长度数据码流描述	说明	描述符
seqlen_data_stream() {	-	-
if(ebml_value(seqlen_encoder)==1 && ebml_value(seqlen_encoder_ver)==1){		
seqlen_same	序列长度相等标识符，0表示当前序列长度与上一条序列长度相等，否则为1。第一个序列seqlen_same置1。	src(2)
seqlen_1byte	序列长度首字节信息，序列长度采用小端序表示为2个字节，内存地址从低到高，第一个字节的编码	src(256)
seqlen_2byte	序列长度次字节信息，序列长度采用小端序表示为2个字节，内存地址从低到高，第二个字节的编码	src(256)
if (ebml_value(longseq)) {	-	-
seqlen_3byte	序列长度次末字节信息，序列长度采用小端序表示为4个字节，内存地址从低到高，第三个字节的编码	src(256)
seqlen_4byte	序列长度末字节信息，序列长度采用小端序表示为4个字节，内存地址从低到高，第四个字节的编码	src(256)
}	-	-
}		
}	-	-
注：本表中所有src(n)共享同一码流		

7.2.3.2.5 碱基序列码流语法

碱基序列码流描述见表18。

表 18 碱基序列码流描述

碱基序列码流描述	说明	描述
block_seq_stream () {	-	-
seq_encoder	ID为1，元素内容为碱基序列的编码器类型。 0表示LZMA编码；1表示LZMA编码，简并碱基单独编码； 2表示容错配的序列比对编码；3表示容错配的序列比对编码，简并碱基单独编码；4表示使用AMGC优化的容错配比对编码；5表示使用AMGC优化的容错配比对编码，简并碱基单独编码； 6表示自组装参考比对编码。	eb
seq_encoder_ver	ID为2，元素内容为碱基序列的编码器版本。	eb
seq_data_id	值为3表示接下来内容为碱基序列编码码流。	vi
seq_data_len	碱基序列编码码流长度。	vi
if(ebml_value(seq_encoder)==0)	-	-
seq_data_lzma	使用LZMA编码的序列。	lz
if(ebml_value(seq_encoder) == 1 || ebml_value(seq_encoder) == 3 || ebml_value(seq_encoder) == 5){	-	-
ambi_stream()	-	-
if(ebml_value(seq_encoder)==1)	-	-
seq_bidata_lzma	将碱基每位碱基转化为2位再使用LZMA编码器编码的序列（简并碱基用‘A’替换）。	lz
if(ebml_value(seq_encoder)>=2 && ebml_value(seq_encoder)<=5)	-	-
seq_align_stream()	-	-
}	-	-
if(ebml_value(seq_encoder)==6)	-	-
seq_assem_stream()	-	-
}		

简并碱基码流描述见表19。

表 19 简并碱基码流描述

简并碱基码流描述	说明	描述符
ambi_stream () {	-	-
ambi_base_flag_id	值为1表示接下来内容为简并碱基存在标识符。	vi
ambi_base_flag_len	简并碱基存在标识符码流长度。	vi
if (ebml_value(seq_encoder_ver)==1){	-	-
ambi_base_flag_num	ambi_base_flag处理的字符数。	u(32)
ambi_base_flag	简并碱基存在标识符，0表示该碱基序列不包含简并碱基，1表示包含简并碱基。	rc(0，2)
}	-	-

表 19 简并碱基码流描述（续）

简并碱基码流描述	说明	描述符
ambi_base_char_id	值为2表示接下来内容为简并碱基字符。	vi
ambi_base_char_len	简并碱基字符码流长度。	vi
if (ebml_value(seq_encoder_ver)==1){	-	-
ambi_base_char_num	ambi_base_char处理的字符数。	u(32)
ambi_base_char	简并碱基字符，编码范围为11（见附录A）。	rc(0，11)
}	-	-
ambi_max_qual_id	值为3表示接下来内容为简并碱基最大质量分数。	vi
ambi_max_qual_len	简并碱基最大质量分数码流长度。	vi
if (ebml_value(seq_encoder_ver)==1){	-	-
ambi_max_qual_num	ambi_max_qual处理的字符数。	u(32)
ambi_max_qual	含简并碱基的碱基序列上简并碱基对应的最大质量分数，编码范围为95。	rc(0，95)
}	-	-
ambi_lowqual_normalcount_id	值为4表示接下来内容为低质量分数碱基中非简并碱基个数。	vi
ambi_lowqual_normalcount_len	低质量分数碱基中非简并碱基个数码流长度。	vi
ambi_lowqual_normalcount_stream()	-	-
ambi_lowqual_normalintv_id	值为5表示接下来内容为非简并碱基间的简并碱基数量	vi
ambi_lowqual_normalintv_len	非简并碱基间的简并碱基数量码流长度	vi
ambi_lowqual_normalintv_stream()	-	-
}	-	-

低质量分数碱基中非简并碱基个数码流描述见表20。

表 20 低质量分数碱基中非简并碱基个数码流描述

低质量分数碱基中非简并碱基个数码流描述	说明	描述符
ambi_lowqual_normalcount_stream() {	-	-
if (ebml_value(seq_encoder_ver)==1){	-	-
ambi_normal_num	低质量分数碱基中非简并碱基个数。	u(32)
ambi_normal_num_n_len	低质量分数碱基中非简并碱基个数表示为2进制串的长读。	src(64)
for(i = 0;i< 64;++i) {	-	-
ambi_normal_num_n_num	低质量分数碱基中非简并碱基个数的2进制位。	src(2)
}	-	-
}	-	-
}	-	-
注：本表中所有src(n)共享同一码流		

非简并碱基间的简并碱基数量码流描述见表21。

表 21 非简并碱基间的简并碱基数量码流描述

非简并碱基间的简并碱基数量码流描述	说明	描述符
ambi_lowqual_normalintv_stream () {	-	-
if (ebml_value(seq_encoder_ver)==1){	-	-
ambi_normal_intv	非简并碱基间的简并碱基数量。	u(32)
ambi_normal_intv_n_len	非简并碱基间的简并碱基数量表示为2进制串的长度。	src(64)
for (i= 0;i<64;++i) {	-	
ambi_normal_intv_n_num	非简并碱基间的简并碱基数量的2进制位。	src(2)
}	-	-
}	-	-
}	-	-
注：本表中所有src(n)共享同一码流		

基于输入性参考序列比对编码码流描述见表22。

表 22 基于输入性参考序列比对编码码流描述

基于输入性参考序列比对编码码流描述	说明	描述符
seq_align_stream() {	-	-
if(ref_bool){	-	-
align_seq_stream()	-	-
}	-	-
unalignseq_id	值为4表示接下来内容为比对失败的序列。	vi
unalignseq_len	比对失败序列码流长度。	vi
unalignseq_stream	比对失败序列，用范围为4的*k*阶区间编码器编码,简并碱基统一使用字符‘A’代替，*k*由rc_order确定。	rc(v，4)
}	-	-

比对信息描述见表23。

表 23 比对信息描述

比对信息描述	说明	描述符
align_seq_stream () {	-	-
align_flag_id	值为1表示接下来内容为比对标记符。	vi
align_flag_len	比对标记符码流长度。	vi
if (ebml_value(seq_encoder_ver)==1){	-	-
align_flag_stream_num	align_flag_stream编码的字符数。	u(32)
align_flag_stream	比对标记符，0为比对失败，1为比对成功。	rc(0，2)
}		
align_pos_id	值为2表示接下来内容为比对位置。	vi
align_pos_len	比对位置码流长度。	vi
if(ebml_value(seq_encoder)==2){	-	-
align_pos_plain_stream_num	比对位置个数。	u(32)

表 23 比对信息描述（续）

比对信息描述	说明	描述符
align_pos_plain_stream	比对位置二进制表示，位数与参考序列长度的二进制一致（不足高位补0）。	rc(0，2)
}	-	-
if(ebml_value(seq_encoder)==3)	-	-
align_pos_amgc_stream()	-	-
align_strand_id	值为3表示接下来内容为比对方向。	vi
align_strand_len	比对方向码流长度。	vi
if (ebml_value(seq_encoder_ver)==1){	-	-
align_strand_stream_num	比对方向个数。	u(32)
align_strand_stream	比对方向， 0为正向比对，1为反向互补比对。	rc(0，2)
mis_num_id	值为4表示接下来内容为错配个数。	vi
mis_num_len	错配个数码流长度。	vi
if (ebml_value(seq_encoder_ver)==1){	-	-
mis_num_stream_num	mis_num_stream处理字符个数	u(32)
mis_num_stream	错配个数二进制表示，位数与*N*+1的二进制一致（*N*为允许最大错配数，不足高位补0）。	rc(0，2)
}	-	-
mis_pos_id	值为5表示接下来内容为错配位置。	vi
mis_pos_len	错配位置码流长度。	vi
if(ebml_value(seq_encoder_ver)==1){	-	-
mis_pos_stream_num	错配位置个数。	u(32)
mis_pos_stream	错配位置二进制，位数与*N*+1的二进制一致（*N*为序列剩余长度，不足高位补0）。	rc(0，2)
}	-	-
mis_base_id	值为6表示接下来内容为错配碱基字符。	vi
mis_base_len	错配碱基字符码流长度。	vi
if (ebml_value(seq_encoder_ver)==1){	-	-
mis_base_stream_num	错配碱基字符数量。	u(32)
mis_base_stream	错配碱基字符，对于ACGT任一字符，错配情况均为4种（另三种字符与N）。	rc(0，4)
}	-	-
}	-	-

基于AMGC比对位置编码码流描述见表 24。

表 24 基于 AMGC 比对位置编码码流描述

基于AMGC比对位置编码码流描述	说明	描述符
align_pos_amgc_stream () {	-	-
first_mapped_pos	参考序列上第一个比对位置。	u(32)

表 24 基于 AMGC 比对位置编码码流描述（续）

基于AMGC比对位置编码码流描述	说明	描述符
for(i=1;i<cardinality(align_flag_stream);++i) {	-	-
is_equal_to_previous	当前碱基是否和前一碱基相等，1为相同，0为不同。	u(1)
mapped_pos_sign	比对位置差的符号。	u(1)
most_signifi_bits	二进制表示的位置差左起0的个数即最高有效位。	u(8)
bits_left	所有位置差去掉最左边的0后即剩余的位。	u(1)
}		
}		

基于自组装参考碱基序列编码码流描述见表 25

表 25 基于自组装参考碱基序列编码码流描述

基于自组装参考碱基序列编码码流描述	说明	描述符
seq_assem_stream() {	-	-
reads_count	碱基序列总数。	u(32)
order_preserving_flag	保留顺序压缩标志，1为保留顺序，0为重排序。	u(1)
reference_length	自组装的参考序列长度。	u(64)
self_assem_reference	使用LZMA编码的自组装参考序列。	lz
reads_direction	序列比对方向，每一位对应一条碱基序列比对方向，1表示反向互补，0表示原始序列方向。	rc(0，256)
mismatch_count	错配碱基数量，每个8位无符号整数储存对应一条碱基序列在参考序列上误匹配的碱基个数。	rc(0，256)
mismatch_base_size	错配碱基总数。	u(32)
mismatch_base	错配碱基，每个字节表示一个碱基序列在参考序列上匹配错误的碱基。	rc(0，256)
max_mismatch_count	最大错配碱基数量，8位无符号整数表示每条碱基序列允许的最大错配碱基个数。	u(8)
for (i = 1; i <= max_mismatch_count; ++i) {	-	-
mismatch_pos_size	错配碱基位置数组的大小，表示当碱基序列错配碱基个数为*i*时，错配碱基位置总数。	u(32)
mismatch_pos	错配碱基位置，表示当碱基序列的错配碱基个数为*i*时，每个错配碱基在碱基序列中的位置。	rc(0，256)
}	-	-
if (order_preserving_flag)	-	-
reads_pos	保留序列顺序模式的碱基序列位置，按照碱基序列在FASTQ文件的先后顺序储存每条碱基序列在自组装参考序列的起始位置。	rc(0，256)
else	-	-
reads_offset	碱基序列位置偏移，即每条碱基序列在自组装参考序列的位置偏移量，通过差分得到，起始位置为0。	rc(0，256)
}	-	-

7.2.3.2.6 质量分数码流语法

质量分数码流描述见表26。

表 26 质量分数码流描述

质量分数码流描述	说明	描述
block_qual_stream () {	-	-
qual_encoder	ID为1，元素内容为质量分数的编码器类型。0表示LZMA编码；1表示ACO编码。	eb
qual_encoder_ver	ID为2，元素内容为质量分数的编码器版本。	eb
qual_data_id	值为2表示接下来内容为质量分数压缩码流。	vi
qual_data_len	质量分数压缩码流长度。	vi
if(ebml_value(qual_encoder)==0)		
qual_data_lzma	LZMA编码器编码质量分数。	lz
else	-	-
qual_data_stream()	-	-
}	-	-

基于ACO编码的质量分数码流描述见表27。

表 27 基于 ACO 编码的质量分数码流描述

基于ACO编码的质量分数序列描述	说明	描述符
qual_data_stream() {	-	-
quality_count	块中质量分数字符总数。	u(32)
traverse_order	块内遍历顺序。1为按行扫描；0为按列扫描。	u(1)
base_flag	碱基标识。1表示采用包含碱基的上下文压缩模型；0表示采用不包含碱基的上下文压缩模型。	u(1)
mean_value_flag	均值标识。1表示采用包含均值的上下文压缩模型；0表示采用不包含均值的上下文压缩模型。	u(1)
max_value	表示当前压缩块中质量分数字符的最大值。	u(8)
min_value	当前压缩块中质量分数字符的最小值。	u(8)
reads_number	当前压缩块测序序列总个数，8位无符号整数。表示当前压缩块中测序序列的总个数。	u(8)
for (i = 0; i < quality_count; ++i)	-	-
quality_value	当前编码质量分数字符。	qs(7)
if (mean_value_flag==1)	-	-
for (i = 0; i < reads_count; ++i)	-	-
mean_value[i]	质量分数所在序列的质量分数平均值。	u(8)
}	-	-

7.2.3.3 FASTA 数据码流语法

7.2.3.3.1 FASTA 数据码流描述

FASTA数据码流描述见表28。

表 28 FASTA 数据码流描述

FASTA数据码流描述	说明	描述
fasta_stream () {	-	-
if (next_ebml_id()==1)	-	-
refseq_id	ID为1，元素内容为RefSeq的数据ID。	eb
if (next_ebml_id()==2)	-	-
genebank_id	ID为2，元素内容为GeneBank的数据ID。	eb
if (next_ebml_id()==3)	-	-
fa_seqid_check	ID为3，元素内容为序列识别码文本校验码。	eb
fa_seqid_id	值为4表示接下来内容为序列识别码。	vi
fa_seqid_len	序列识别码码流的长度。	vi
fa_seqid_lzma	序列识别码LZMA编码码流。	lz
if (next_ebml_id()==5)	-	-
fa_case_check	ID为5，元素内容为碱基序列大小写标记校验码。	eb
fa_case_id	值为6表示接下来内容为碱基序列大小写标记。	vi
fa_case_len	碱基序列大小写标记码流的长度。	vi
fa_case_stream()	-	-
if (next_ebml_id()==7)	-	-
fa_seq_check	ID为7，元素内容为碱基序列文本校验码。	eb
fa_seq_id	值为8表示接下来内容为碱基序列。	vi
fa_seq_len	碱基序列码流的长度。	vi
fa_seq_stream()	-	-
if (next_ebml_id()==100)	-	-
fa_encode_custom	ID为100，内容为自定义信息。	eb
}		

7.2.3.3.2 FASTA 碱基序列大小写标记码流描述

FASTA 碱基序列大小写标记码流描述见表 29。

表 29 FASTA 碱基序列大小写标记码流描述

FASTA 碱基序列大小写标记码流描述	说明	描述符
fa_case_stream() {	-	-
fa_case_encoder	ID 为 1，元素内容为碱基序列大小写标记的编码器类型。0 为 LZMA 编码；1 为区间编码。	eb
fa_case_encoder_ver	ID 为 2，元素内容为碱基序列大小写标记的编码器版本。	eb
fa_case_data_id	值为 3 表示接下来内容为碱基序列大小写标记编码码流。	vi
fa_case_data_len	碱基序列大小写标记编码码流长度。	vi
if(ebml_value(fa_case_encoder)==0)	-	-
fa_case_data_lzma	LZMA 编码码流	lz

表 29 FASTA 碱基序列大小写标记码流描述（续）

FASTA 碱基序列大小写标记码流描述	说明	描述符
if(ebml_value(fa_case_encoder)==1 && ebml_value(fa_case_encoder_ver)==1)	-	-
fa_case_data_rc_num	记录 fa_case_data_rc 的字符数	u(64)
fa_case_data_rc	大小写标记区间编码器编码码流，阶数由 rc_order 确定	rc(v,5)
}	-	-

7.2.3.3.3 FASTA 碱基序列码流描述

FASTA 碱基序列码流描述见表 30。

表 30 FASTA 碱基序列码流描述

FASTA 碱基序列码流描述	说明	描述符
fa_seq_stream() {	-	-
fa_seq_encoder	ID 为 1，元素内容为碱基序列的编码器类型。0 表示 LZMA 编 1 表示区间编码；2 表示基于 BSC 的最大匹配编码。	eb
fa_seq_encoder_ver	ID 为 2，元素内容为碱基序列的编码器版本。	eb
fa_seq_data_id	值为 3 表示接下来内容为碱基序列编码码流。	vi
fa_seq_data_len	碱基序列编码码流长度。	vi
if(ebml_value(fa_seq_encoder)==0)	-	-
fa_seq_data_lzma	LZMA 编码码流。	lz
if(ebml_value(fa_seq_encoder)==1 && ebml_value(fa_seq_encoder_ver)==1)	-	-
fa_seq_data_rc_num	记录 fa_seq_data_rc 的字符数	u(64)
fa_seq_data_rc	碱基序列区间编码器编码码流，阶数由 rc_order 确定	rc(v,5)
if(ebml_value(fa_seq_encoder)==2)	-	-
fa_seq_data_mm	最大匹配编码码流	bs
}	-	-
注：所有碱基符号统一转换成大写字母处理		

7.2.4 尾部码流语法

7.2.4.1 尾部码流描述

尾部码流描述见表31。

表 31 尾部码流描述

尾部码流描述	说明	描述
tailer_stream () {	-	-
if(ebml_value(std_type)== 'fq'){	-	-
read_block_info_id	值为1表示接下来内容为数据块信息。	vi
read_block_info_len	数据块信息码流长度。	vi
read_block_info_stream()	-	-
}	-	-

表 31 尾部码流描述（续）

尾部码流描述	说明	描述
encoder_param_id	值为2表示接下来内容为编码器参数。	vi
encoder_param_len	编码器参数码流长度	vi
encoder_param_stream()	-	-
if (next_ebml_id()==100)	-	-
encoder_custom	ID为100，表示数据块自定义信息。	eb
}	-	-

7.2.4.2 数据块统计信息码流描述

数据块统计信息码流描述见表32。

表 32 数据块统计信息码流描述

数据块信息码流描述	说明	描述
read_block_info_stream () {	-	-
block_num	ID为1，元素内容为数据块总数。	eb
size_bitunit	ID为2，元素内容为表示单个数据块字节大小的位数。	eb
block_maxbyte	ID为3，元素内容为最大数据块文本字节大小。	eb
block_textbyte	ID为4，元素内容为各数据块的原始文本字节大小列表，元素内容长度 = size_bitunit × 数据块总数。	eb
block_compbyte	ID为5，元素内容为各数据块的压缩后字节大小列表，元素内容长度 = size_bitunit × 数据块总数。	eb
offset_bitunit	ID为6，元素内容为表示单个偏移量的位数。	eb
block_textoffset	ID为7，元素内容为各数据块在原始文本起始偏移量列表，元素内容长度 = offset_bitunit × 数据块总数。	eb
block_compoffset	ID为8，元素内容为各数据块在压缩后起始偏移量列表，元内容素长度 = offset_bitunit × 数据块总数。	eb
}	-	-

7.2.4.3 压缩方法参数码流描述

压缩方法参数码流描述见表33。

表 33 压缩方法参数码流描述

压缩方法参数码流描述	说明	描述
encoder_param_stream () {	-	-
if (next_ebml_id()==1)	-	-
param_seqid	ID为1，元素内容为序列识别码编码的全局参数。	eb
if (next_ebml_id()==2)	-	-
param_seqlen	ID为2，元素内容为序列长度编码的全局参数。	eb
if (next_ebml_id()==3)	-	-
param_seq_id	值为3表示接下来内容为碱基序列编码的全局参数。	vi
param_seq_len	碱基序列编码全局参数码流长度。	vi

表 33 压缩方法参数码流描述（续）

压缩方法参数码流描述	说明	描述
param_seq_stream()	-	
if (next_ebml_id()==4)	-	-
param_qual	ID为4，元素内容为质量分数编码的全局参数。	eb
}	-	-

7.2.4.4 碱基序列编码全局参数码流描述

碱基序列编码全局参数码流描述见表34。

表 34 碱基序列编码全局参数码流描述

碱基序列编码全局参数码流描述	说明	描述
param_seq_stream () {	-	-
rc_order	ID为1，记录区间编码的阶数。	eb
if(ebml_value(seq_encoder)>=2 && ebml_value(seq_encoder)<=5){	-	-
ref_bool	ID为2，记录是否使用输入性参考序列进行压缩。	eb
refseq_id	ID为3，元素内容为参考序列的RefSeq数据ID，实际存储时可以是参考序列首条染色体的序列识别码。	eb
genebank_id	ID为4，元素内容为参考序列的GeneBank数据ID，实际存储时可以是参考序列首条染色体的序列识别码。	eb
ref_name	ID为5，元素内容为参考序列文件名。	eb
ref_checksum	ID为6，元素内容为参考序列校验码。	eb
if (next_ebml_id()==7)	-	-
ref_encoder	ID为7，元素内容为参考序列编码器类型。0表示不保存参考序列；1表示LZMA编码；2表示区间编码。	eb
if (next_ebml_id()==8)	-	-
ref_encoder_ver	ID为8，元素内容为参考序列的编码器版本。	eb
if (next_ebml_id()==9){	-	-
ref_data_id	ID为9，表示接下来内容为参考序列码流。	vi
ref_data_len	参考序列码流长度。	vi
if(ebml_value(ref_encoder)==1)	-	-
ref_data_lzma	采用LZMA编码的参考序列码流。	lz
if(ebml_value(ref_encoder)==2 && ebml_value(ref_encoder_ver)==1){	-	-
ref_data_rc_num	记录ref_data_rc中的字符个数。	u(64)
ref_data_rc	参考序列编码码流，阶数由rc_order确定	rc(v,5)
}	-	-
}	-	-
}	-	-
}		

8 解码

8.1 解码流程

测序数据解码依次解码头部流码、尾部流码和压缩数据码流。

8.2 头部码流解码

头部码流解码过程如下：

a) 按表 7 描述读取头部魔数 header_magic_num，判断其内容是否为“avsg”，若否，则不解码；
b) 按表 8、表 9 和表 10 描述解析获得码流基础信息和编码信息。

8.3 尾部码流解码

尾部码流解码过程如下：

a) 按表 7 描述获取码流最后 4 字节中的魔数 tailer_magic_num，判断其内容是否为“avsg”，若否，则不进行解码;
b) 按表 7 描述解析 header_len 和 comp_data_len 获得文件头和编码码流长度，若 comp_data_len 不为 0，则按其数值跳转至尾部码流；否则在文件结尾倒数第 12 至倒数第 4 字节解析得到 comp_data_len_alt，按照 comp_data_len_alt 数值跳转至尾部码流；
c) 按表 32、表 33 和表 34 描述解析获得数据块信息、编码器参数和参考序列信息，供后续其他部分解码使用，其中自定义信息可以定义使用外部编码器。

8.4 压缩数据码流解码

8.4.1 FASTQ 压缩数据码流解码

8.4.1.1 概述

在前述头部码流的解码结果中，若码流基础信息中的标准类型 std_type 为’fq’，则表示编码为 FASTQ 数据。根据尾部码流记录中的各数据块偏移量，以压缩数据码流的内容为起始，计算得知各个数据块的起始位置，逐个跳转到各个数据块，基于尾部码流描述依次对数据块基础信息、序列识别码、碱基序列、质量分数进行解码。

8.4.1.2 数据块基础信息解码

数据块基础信息解码过程如下：

a) 按表 12 描述解析获得数据块基础信息长度；
b) 按表 13 描述解析获得测序序列数目、数据块原始文本大小等等信息，按照数据长度的数值读取相应的字节数解码出数据。

8.4.1.3 序列识别码解码

8.4.1.3.1 概述

按表 14描述解析序列识别码压缩码流获得编码器类型、编码器版本。若编码器类型为0则采用LZMA解码；否则按表 14和表 15描述解析序列识别码码流。其中若编码器类型为1，则数据采用基于窗口划分的编码方法，按照8.4.1.3.2 解码；若编码器类型为2，则数据采用基于最高位平面嵌入和流道列上下文方法编码，按照8.4.1.3.3 解码。

8.4.1.3.2 基于窗口划分的解码

按表 14 描述解析 vec_format 得到各个序列识别码块的序列数目与首个序列识别码长度，并初始化多字段混合自适应区间编码器编码码流，对 seqid_chunk_stream 循环进行 8.4.1.3.2.2 中规定的组分解码，直到码流结尾，即完成所有序列识别码块的解码。

按表 15描述解析得到当前序列识别码块的首个序列识别码first_seqid，以非字母或非数字字符为间隔符对首个原数据进行划分，得到组分数量和类型，按照相同的组分数量和类型对后续序列识别码逐一解码：

a) 数字组分解码过程如下：
   1) 采用范围为4的0阶区间解码器对结果码流进行解码得到当前差值类型diff_type；
   2) 采用范围为16的0阶区间解码器对结果码流解码得到当前diff_scale_n_len；
   3) 采用范围为16的0阶区间解码器对结果码流解码得到各个diff_scale_n_num，并组合得到原始差异值，再结合diff_type，得到原始数值，具体步骤应符合附录B的规定；
   4) 重复以上过程直到结果码流解码完毕。
b) 非数字组分解码过程如下：
   1) 获取首个序列识别码first_seqid在当前组分的字符串，记录为temp_char;
   2) 采用范围为4的0阶区间解码器对结果码流解码得到当前char_flag；
   3) 如果char_flag为0，直接将temp_char作为当前序列识别码的当前组分结果；
   4) 如果char_flag为1，从结果码流中解码出长度为num_chars（temp_char）的字符串；
   5) 如果char_flag为2，解析出len_diff，再从结果码流中解码出长度为num_chars（temp_char）+len_diff的字符串；
   6) 如果char_flag为3，解析出len_diff，再从结果码流中解码出长度为num_chars（temp_char）-len_diff的字符串；
   7) 重复以上过程直至结果码流解码完毕；

各个组分都完成各自的解码后，根据首个序列识别码first_seqid得到各组分间的间隔符，各组分解码后依次连接对应间隔符拼接即得到原始序列识别码。

8.4.1.3.3 基于最高位平面嵌入和流道列上下文的解码

基于最高位平面嵌入和流道列上下文的序列识别码编码中，流道、X 和 Y 坐标之外的组分编码与基于窗口划分的编码方法相同，其解码参照 8.4.1.3.2 ，流道列、X 和 Y 坐标组分解码如下：

a) 按照表 15描述，流道列、X和Y坐标组分解码过程如下：
   1) 采用范围为96的0阶算术解码器对结果码流解码tile字段得到各位组装成流道；
   2) 采用范围为16的0阶算术解码器解码next_msb得到指示数字最高位平面位；
   3) 结合next_msb和tile，由next_msb获知当前组分长度，由tile获知当前码流所属的组，并用范围为16的0阶算术解码器，对结果码流进行解码，直到数字元素数量等于当前组分长度next_msb，解码结束，得到当前组分的字符串；
   4) 重复以上过程直至结果码流解码完毕；

各个组分都完成各自的解码后，按照8.4.1.3.2 方法汇总。

8.4.1.3.4 校验

按照表 12描述，若存在ID为2的EBML元素block_seqid_check，其内容为序列识别码校验码，采用表 10描述中的校验算法和此校验码对序列识别码解码结果进行校验。

8.4.1.4 序列长度解码

8.4.1.4.1 概述

按表 12描述解析block_seqlen_id和block_seqlen_len得到ID和数据长度，再按照表 16描述依次读取编码器类型、编码器版本。若编码器类型为0则采用LZMA解码，否则继续解析序列长度压缩码流。

8.4.1.4.2 序列长度码流解码

按照表17描述并按下列步骤对序列长度码流进行解码：

a) 从码流中解码出seqlen_same，若seqlen_same值为0，则当前序列长度与前一序列长度相同,执行c）步骤。若seqlen_same值为1，按照b）步骤等到当前序列长度；

b) 从码流中解码出seqlen_1byte和seqlen_2byte，根据头部码流解码获得压缩信息确定是否为长读数据。若为长读数据，从码流中解码出seqlen_3byte和seqlen_4byte，由此拼接得序列长度；

c) 重复步骤,直到码流解码完毕，即得到所有序列长度；

8.4.1.4.3 检验

按表 12描述，若存在ID为4的EBML元素block_seqlen_check，其内容为序列长度数据校验码，采用表 10描述中的校验算法和此校验码对序列长度数据解码结果进行校验。

8.4.1.5 碱基序列解码

8.4.1.5.1 概述

按照表 12描述解析block_seq_id和block_seq_len得到ID和碱基序列数据长度，再按照数据长度数值读取表18中编码器类型seq_encoder和版本seq_encoder_ver。具体解码方式如下：

——若编码器类型为0，直接采用LZMA对码流seq_data_lzma进行解码；

——若编码器类型为1，说明数据包含简并碱基编码和LZMA编码，则先对简并碱基码流进行解码，再对seq_data_lzma进行解码；

——若编码器类型为2，说明数据是容错配的序列比对和编码得到，按照8.4.1.5.3 中的a)解码；

——若编码器类型为3，说明数据是容错配的序列比对和编码，且简并碱基编码单独编码，则先对简并碱基码流进行解码再按照8.4.1.5.3 中的a)项解码；

——若编码器类型为4，说明数据是经AMGC优化的容错配编码，按照8.4.1.5.3 中的b)解码；

——若编码器类型为5，说明数据是经AMGC优化的容错配编码，且简并碱基编码单独编码，则先对简并碱基码流进行解码再按照8.4.1.5.3 中的b)解码；

——若编码器类型为6，数据采用自组装参考压缩编码，则采用8.4.1.5.4 进行解码。

另外，参考序列编码于尾部码流，包括参考序列内容校验码、参考序列的压缩算法类型、参考序列的压缩算法版本、参考序列压缩码流，按照8.3 进行解码。

8.4.1.5.2 简并碱基解码

简并碱基解码过程如下：

a) 按照表 19 描述定位存在标识符 ambi_base_flag、简并碱基字符 ambi_base_char、简并碱基最大质量分数 ambi_max_qual、低质量碱基中非简并碱基个数 ambi_lowqual_normalcount_stream()、非简并碱基间的简并碱基数量 ambi_lowqual_normalintv_stream()这五个码流；

b) 分别初始化范围为 2、11、95 的区间解码器，根据总字符数 ambi_base_flag_num，分别解码前三个码流，得到各自原始数组；

c) 对于 ambi_lowqual_normalcount_stream()，先按照表 20 描述解码 1 个 4 字节无符号整数得到 ambi_normal_num，然后初始化多字段混合自适应区间解码器，再按表 21 描述，交替解码出

ambi_normal_num_n_len和ambi_normal_num_n_num，直到完成解码的ambi_normal_num_n_len元素个数等于ambi_normal_num，解码完毕，将ambi_normal_num_n_len和ambi_normal_num_n_num组合为原始的低质量分数碱基中非简并碱基个数数组；

d) 对于ambi_lowqual_normalintv_stream()，先按照表21描述解码1个4字节无符号整数得到ambi_normal_intv，然后初始化多字段混合自适应区间解码器，再按表22中的定义，交替解码出ambi_normal_intv_n_len和ambi_normal_intv_n_num，直到完成解码的ambi_normal_intv_n_len个数等于ambi_normal_intv，解码完毕，将ambi_normal_intv_n_len和ambi_normal_intv_n_num组合为原始的非简并碱基间的简并碱基数量数组；

e) 结合解码结果，对于含简并碱基的各个碱基序列，在获取其质量分数后，将各个简并碱基替换到对应的原始位置上。

8.4.1.5.3 基于输入性参考碱基序列解码

基于输入性参考序列的碱基序列编码数据解码时，根据序列是否比对成功及比对位置存储方式有下列解码方法：

a) 比对成功且比对位置直接存储解码过程如下：

1) 根据表34描述获取参考序列，首先判断ref_bool是否使用输入性参考序列，若是，则继续判断ref_encoder字段是否存在，若存在，则根据ref_id和ref_name获取外部参考序列，并使用ref_checksum校验码验证。若ref_encoder不存在，根据相应解码器解压参考序列数据码流获得参考序列。

2) 按照表23描述解码得到比对位置 align_pos_plain_stream或align_pos_amgc_stream()、比对方向 align_strand_stream、错配个数 mis_num_stream、错配位置 mis_pos_stream、错配字符 mis_base_stream五个码流；

3) 如果编码器seq_encoder的ID值为1或者2，则初始化范围为2的区间解码器解析align_pos_plain_stream，得到二进制串，根据参考序列的长度，取能够满足2n大于参考序列长度的最小n，每n个位为一个align_pos_plain_stream的初始元素的二进制串，还原得到各个align_pos；否则，采用8.4.1.5.3b）中的位置序列解码方法。

4) 初始化范围为2的解码器解码align_strand_stream，得到原始数组；

5) 初始化范围为2的解码器解码mis_num_stream，得到二进制串，根据参考序列的长度，取能够满足2n> max_mismatch_count的最小n，每n个位为一个比对位置的初始元素的二进制串，还原得到各个比对位置；

6) 初始化范围为2的解码器解码mis_pos_stream，得到二进制串，设当前错配所可能出现的子串长度为L（若错配为当前序列第一个错配，则L为当前序列总长，否则，L为当前序列总长与上一个错配所在位置的差值），取能够满足2n>L的最小n，获取n个位为当前错配位置的二进制串，还原得到初始值；

7) 初始化范围为4的解码器解码mis_base_stream，得到四进制串，根据参考序列上的子串在该位置上的碱基，获知原始碱基（若参考序列上碱基为A，则0123对应GCTN；若参考序列上碱基为C，则0123对应GATN；若参考序列上碱基为G，则0123对应CATN；若参考序列上碱基为T，则0123对应GCAN）；

8) 结合解码结果，对于某条碱基序列来说，会访问参考序列的当前比对位置，截取相应子串，根据当前比对方向，决定是否将子串进行反向互补处理，然后若当前错配个数大于0，根据当前各错配位置，将对应位置的碱基根据错配字符替换成实际字符。重复该过程，直到恢复所有碱基序列。

b) 比对成功且采用AMGC编码比对位置的解码过程如下：

1) 按照表 24描述，根据cardinality(align_flag_stream)获取比对位置总数ref_pos_count，申请字节数为ref_pos_count*4的内存空间表示为32位整形数组refpos；
2) 申请类型为布尔型，长度为ref_pos_count的数组is_equal，从码流中解码出is_equal_to_previous的值；
3) 申请类型为布尔型，长度为ref_pos_count的数组sign，从码流中解码出mapped_pos_sign；
4) 申请类型为整型，长度为ref_pos_count的数组msb，从码流中解码出most_signifi_bits的值；
5) 申请类型为布尔型，长度为ref_pos_count*32的数组bitleft，从码流中解码出bits_left的值；
6) 遍历msb数组，根据msb[i]的值确定从bitleft中取出对应数量的位，恢复出位置差diff_pos，将diff_pos暂时存储在refpos数组中；
7) 根据sign的值，给每个diff_pos的值加上正负号；
8) 解码出first_mapped_pos存放于 refpos数组，令p_d指向diff_pos的第一个值，根据is_equal，由diff_pos得到最终的ref_pos。当is_equal[i]为1，则refpos[i+1] = refpos[i]，否则refpos[i+1] = diff_pos[p_d++]。

c) 比对不成功的碱基序列，初始化范围为4的解码器，对表22中unalignseq_stream进行解码，得到无参考编码的碱基序列。

8.4.1.5.4 基于自组装参考碱基序列解码

按照表 25描述依次解码匹配信息、错配碱基位置、碱基序列位置、碱基序列和参考序列：

a) 匹配信息解码过程如下：
1) 按照表 25描述解析32位无符号整数得到碱基序列总数reads_count；
2) 按照表25描述解析1位无符号整数获取保留顺序压缩标志 order_preserving_flag；
3) 按照表25描述解析64位无符号整数得到自组装参考序列长度 reference_length；
4) 根据 reference_length 解码下个数据块得到自组装参考序列 self_assem_reference（8位无符号字符数组）；
5) 根据 reads_count解析下个数据块得到碱基序列链方向reads_direction （1位无符号整数数组）；
6) 根据 reads_count解析下个数据块得到每个碱基序列的错配碱基数量mismatch_count（8位无符号整数数组）；
7) 按照表25描述解析32位无符号整数得到错配碱基总数 mismatch_base_size；
8) 根据 mismatch_base_size 解析下个数据块得到错配碱基mismatch_base （8位无符号字符数组）；

b) 错配碱基位置解码过程如下：
1) 按照表 25描述解析8位无符号整数得到最大错配碱基数量 max_mismatch_count
2) 按照下面的过程迭代 max_mismatch_count 次得到错配碱基位置列表 mismatch_pos，该列表包含max_mismatch_count个数组：对于第i次迭代，首先按照表26描述解析32位无符号整数得到碱基序列错配碱基个数为i时错配碱基位置的总数 mismatch_pos_size[i]；然后根据该轮迭代得到的 mismatch_pos_size[i] 解析下个数据块得到错配碱基位置数组mismatch_pos[i] (16位无符号整数数组)。

c) 碱基序列位置解码过程如下：
1) 如果 order_preserving_flag 为 1，根据 reads_count 解析下个数据块得到保留序列顺序模式下（按FASTQ文件中的先后顺序）的碱基序列位置数组 reads_pos（64位无符号整数数组）

2) 否则，根据 reads_count 解析下个数据块得到重排序列模式下的碱基序列位置偏移reads_offset（16位无符号整数数组），然后通过逆向差分编码还原碱基序列在参考序列的位置数组 reads_pos，过程应符合附录C.1的规定。

d) 碱基序列解码过程如下：

碱基序列的解码中初始化碱基序列数组seq，数组大小为 reads_count，初始化错配碱基数组下标 mismatch_base_idx 为 0；初始化错配碱基位置数组下标 mismatch_pos_idx 为全0数组，该数组大小为max_mismatch_count + 1。

按照下面的计算过程迭代reads_count次得到每一条碱基序列字符串，第*i*轮迭代的过程如下：

1) 根据碱基序列在自组装参考序列的位置reads_pos[i] 以及碱基序列的长度在自组装参考序列截取相应的子串作为seq[i]的初始值；

2) 碱基序列的错配碱基个数mis_cnt为mismatch_count[i]。如果mis_cnt大于0，需要还原碱基序列上发生错配的碱基，还原过程应符合附录C.2的规定。

e) 参考序列解码过程如下：

按照表34描述解析param_seq_stream，若存在ID为9的元素，说明数据中存在参考序列，然后继续解析ID为7和8的元素，分别是压缩算法类型和压缩算法版本，若压缩算法类型为0，说明参考序列由LZMA编码得到，则使用LZMA进行解码，若压缩算法类型为1，说明参考序列由算术码编码得到，则使用算术码进行解码。

8.4.1.5.5 校验

按表 12描述，若存在ID为6的EBML元素block_seq_check，其内容为碱基序列数据校验码，采用表10中定义的校验算法和此校验码对碱基序列数据解码结果进行校验。按照表34描述，若存在ID为6的EBML元素ref_checksum，其内容为参考序列校验码，采用表 10中定义的校验算法和此校验码对参考序列数据解码结果进行校验。

8.4.1.6 质量分数解码

8.4.1.6.1 概述

按照表 12描述解析获得质量分数压缩码流block_qual_stream()。对于质量分数压缩码流，按照表26描述解析qual_encoder获取编码器类型和版本，若编码器类型为0，采用LZMA对质量分数码流进行解码；若编码器类型为1，说明数据使用基于ACO的编码，按照下列步骤解码并重复至解码完毕。

8.4.1.6.2 基于 ACO 编码的质量分数解码

质量分数分块的解码过程如下：

a) 按照表27描述，获取质量分数总数quality_count，申请字节数为quality_count的内存空间表示为32位整型数组qs;

b) 构建的混合上下文模型，依次解码quality_count个质量分数并存放于qs，质量分数编码上下文模型建立应符合附录D规定；

c) 按照遍历顺序traverse_order截断qs输出质量分数序列：如traverse_order为1，根据表 12中block_seqlen_id字段定义的序列长度按顺序依次截断qs并添加换行符；如traverse_order为0，申请reads_number个内存空间用于存储相应的质量分数序列，空间大小根据中block_seqlen_id字段定义的序列长度确定，按7.1.6.6 定义的列扫描顺序遍历qs依次填入相应质量分数序列空间并加上换行符。

8.4.1.6.3 校验

按表 12描述，若存在ID为8的EBML元素block_qual_check，其内容为质量分数数据校验码，按照表10描述中的校验方法和此校验码对质量分数数据解码结果进行校验。

8.4.2 FASTA 压缩数据码流解码

8.4.2.1 概述

在头部码流的解码结果中，若码流基础信息中的标准类型std_type为’fa’，则表示编码为FASTA数据。按照表28描述解析获得FASTA压缩数据码流中序列识别码、碱基序列大小写标记、碱基序列码流长度信息，在FASTA压缩数据码流中逐个解析上述各个部分。

8.4.2.2 FASTA 各数据流解码

FASTA数据编码各数据流解码过程如下：

——序列识别码解码：用 LZMA 直接解码；

——碱基序列大小写标记解码：按表 29 描述解析 fa_case_encoder 的值，若值为 0，采用 LZMA 对 fa_case_lzma 进行解码获得碱基大小写标记数据；若值为 1，采用区间编码解码器对 fa_case_rc 进行解码，解码结果为二进制数组，记录各个碱基为大写或是小写；

——碱基序列解码：按照表 30 描述解析 fa_seq_encoder 的值，若值为 0，采用 LZMA 对 fa_seq_lzma 进行解码获得碱基数据；若值为 1，采用区间编码解码器对 fa_seq_rc 进行解码；若值为 2，使用 BSC 编码解码器对 fa_seq_mm 进行解码获得基于最大匹配的有参考编码字符串，根据 e)中参考序列解码方法得到参考序列。利用参考序列和 7.1.7.2 获得 FASTA 碱基序列。

8.4.2.3 校验

按表28描述，若存在ID为3的EBML元素fa_seqid_check，其内容为序列识别码校验码，按照表 10描述的校验算法和此校验码对序列识别码解码结果进行校验。若存在ID为5的EBML元素fa_case_check，其内容为碱基序列大小写标记校验码，按照表 10描述的校验算法和此校验码对碱基序列大小写标记解码结果进行校验。若存在ID为7的EBML元素fa_seq_check，其内容为碱基序列校验码，按照表 10描述的校验算法和此校验码对碱基序列解码结果进行校验。

附 录 A
（资料性）
简并碱基规则

简并碱基按照国际纯化学和应用化学联合会IUPAC（International Union of Pure and Applied Chemistry）规则见表A.1。

表A.1 简并碱基对照表

简并碱基	对应碱基
A	A
C	C
G	G
T	T
R	A或G
Y	C或T
S	G或T
W	A或T
K	G或T
M	A或C
B	C或G或T
D	A或G或T
H	A或C或T
V	A或C或G
N	任何碱基
.或-	空格

附 录 B
（规范性）
序列识别码数字组分解码详细步骤

本附录给出序列识别码中纯数字组分处理详细步骤：

a) 获取首个序列识别码在当前组分的数字，记录为 last_num;

b) 获取 diff_type 在当前下标的数字。如果为 0，则当前数字等于 last_num；如果为 1，则当前数字等于 last_num+1；这两种情况则进入步骤 d)。如果为 2，则当前数字大于 last_num 且差值大于 1；如果为 3，则当前数字小于 last_num；这两种情况进入步骤 c)。

c) 获取 diff_scale_n_len 在当前下标的数字，然后从 diff_scale_n_num 中当前下标往后获取相应位数的数字，即为当前的差值绝对值。如果 diff_type 的当前下标数字为 2，则当前数字等于 last_num 与差值绝对值的数字之和，否则，当前数字等于 last_num 与差值绝对值之差。

d) 如果首个序列识别码在该组分的数字存在前缀 0 的情况，则初始化范围为 10 的解码器解码得知当前的前缀 0 个数，加到当前数字前作为前缀。当前数字作为这一轮的结果输出。

e) 如果该组分码流已处理完毕，则结束；否则，将当前数字赋值给 last_num，回到 b)。

附 录 C
（规范性）
差分位置及错配碱基还原过程

C.1 逆向差分编码还原

逆向差分编码还原的过程如下：

a) 设第一个序列位置 reads_pos[0] 为偏移量 reads_offset[0];
b) 第i个（i>1）序列位置 reads_pos[i] 为其前一个序列位置 reads_pos[i-1] 加上相应的偏移量 reads_offset[i]；
c) 重复b)直至所有序列位置被还原。

C.2 碱基错配还原

碱基错配还原过程步骤如下：

a) 将 seq[i]上位置为 mismatch_pos[mis_cnt][mismatch_pos_idx[mis_cnt] + j] 的碱基替换为 mismatch_base[mismatch_base_idx +j]，其中 j 代表第 j 轮碱基替换，总共需要迭代 mis_cnt 次；
b) 更新下标 mismatch_pos_idx[mis_cnt] += mis_cnt，mismatch_base_idx += mis_cnt；
c) 如果碱基序列的比对方向 reads_direction[i] 等于 1，计算碱基序列 seq[i]的反向互补序列。

附 录 D
（规范性）
质量分数编码上下文模型建立

本附录给出质量分数编码上下文模型建立过程：

a) 按照表27定义获取碱基标识符base_flag值，如果其值为1获取与当前质量分数字符对应的前两个碱基字符base_value[*i*]和base_value[*i*-1]；

b) 获取均值标识符mean_value_flag的值，如果其值为1，根据表27定义获取当前质量分数字符所在行的质量分数均值mean_value[*i*]；

c) 获取当前质量分数分块字符最大值max_value和最小值min_value，确定模型范围；

d) 从qs中获取质量分数字符的前四个字符q1=qs[*i*-1]，q2= qs[*i*-2]，q3= qs[*i*-3]和q4 =qs[*i*-4]；

e) 按照下列方式建立上下文模型：

——如果base_flag的值为1且mean_value_flag为1，则根据q1、q2、q3、q4、base_value[i]、base_value[i-1]和mean_value，对熵编码上下文模型统一量化建模；

——如果base_flag的值为1且mean_value_flag为0，则根据q1、q2、q3、q4、base_value[i]、和base_value[i-1]，对熵编码上下文模型统一量化建模；

——如果base_flag的值为0且mean_value_flag为1，则根据q1、q2、q3、q4和mean_value，对熵编码上下文模型统一量化建模；

——如果base_flag的值为0且mean_value_flag为0，则根据q1、q2、q3和q4对熵编码上下文模型统一量化建模。