Draft BFA Specification

General notes

- The BFA specification translates the GFA specification in a binary format.
- Data types and data organization are derived from the BAM specification.
- Data types are always little-endian.
- File can be optionally compressed

Data types

- Numeric:
  - Float:
    `double`
    [f tags; B subtype f tag elements]
  - Integer:
    `[u]int[8|16|32]_t`
    [i tags; B subtype cCsSiI tag elements; containment pos field]
  - Size/Counters:
    `uint32_t`; larger than 0
    [block size; number of segments in path; lenght of variable length strings, sequences, B/H tags, cigars]
- Arrays:
  - of variable-size elements:
    length obtained by parsing element
    `uint32_t` (number of elements) + elements of variable length
    [path segment informations]
  - of fixed-size elements with predefined element type
    `uint32_t` (number of elements) + `elem_type[n_elems]`
    [record (as byte array); variable length strings (see below); H tags; cigar strings]

  - of fixed-size elements specifying the element type
    `uint32_t` (number of elements) + `char` (element type code) + `elem_type[n_elems]`
    [B tags]

- Strings:

  - Fixed-length strings:
    `char[string_length]`; string length is predefined
    not 0-terminated
    [record type; segment orientations links/containments and in paths; A tags; tag name in optional fields; type specification in optional fields; subtype in B tags]
  - Zero-delimited strings:
    `char[]`; no 0-byte before last byte
    0-terminated
    [Z and J tags]
  - Variable-length strings:
    as arrays of fixed-size elements with predefined element type;
    `uint32_t` (`string_length`) + `char[string_length]`
    not 0-terminated
    [segment/path name; segment names in links/containments]
  - CIGAR strings:
    as arrays of fixed-size elements with predefined element type;
    each element (oplen and opcode) is encoded in one `uint32_t` (see BAM specification)
    [links/containments alignments, path segment alignments]
  - Sequences:
    `uint32_t` (`seq_length`, before encoding) + `char[str_length]`;
    sequence is 4bits encoded (see BAM spec.); `str_length` = $\lceil$`seq_length`$/2\rceil$
    not 0-terminated
    [segment sequence]

- Optional fields:

  - `char[2]` name + `char[1]` datatype + datatype-dependent representation

Optional field values

The integer fields (i) of GFA can be stored in a number of integer fields with different sizes and either signed or unsigned. The application is free to choose an appriopriate type according to the value. When converting BFA to GFA all integer type optional fields shall be written as i optional fields.

A	char
i (cCsSiIlL)	[u]int[8|16|32|64]_t
f	double
Z	char[] (0-term)
J	char[] (0-term)
B	char (value_type) + uint32_t (value_len) + value_type[value_len]
H	uint32_t (value_len) + char[value_len]

Header

The header contains all tags found in all header lines of the GFA file. It supports multiple H lines (unclear if this shall be allowed or not) and multiple definitions of the same tag (also unclear if this is meant to be allowed as it is not explicitely forbidden, but it is never so in examples).

Field	Description	Type
n_header_tags	Number of header tags	uint32_t
Header fields (length: n_header_tags times)		
tag	Two-character tag	char[2]
val_type	Value type	char (AcCsSiIfZBJH)
value	Tag value	depends on val_type

Segments

To store the sequence in a separate file, use empty sequences (GFA "*") and store the filename/sequence number in optional fields. Empty sequences shall be stored as follows: `l_seq` shall be set to 0 and the field `seq` shall be skipped.

Field	Description	Type
n_segments	Number of segment records	`uint32_t (< MAXINT32T)`
Segments (length: n_segments)		
l_name	Length of segment name	`uint32_t`
name	Segment name	`char[l_name]`
l_seq	Uncompressed sequence length	`uint32_t`
seq	4-bit encoded read (see SAM spec)	`uint8_t[(l_seq+1)/2]`
n_optfields	Number of optional fields	`uint32_t`
Optional fields (length: n_header_tags times)		
tag	Two-character tag	`char[2]`
val_type	Value type	`char (AcCsSiIfZBJH)`
value	Tag value	`depends on val_type`

Links

Empty overlap shall be stored as follows: `l_cigar` shall be set to 0 and the field `cigar` shall be skipped.

Field	Description	Type
n_links	Number of link records	`uint32_t (< MAXINT32T)`
Links (length: n_links)		
from	From segment ID + 1 (orient: sign)	`int32_t`
to	To segment ID + 1 (orient: sign)	`int32_t`
l_cigar	Number of CIGAR operations	`uint32_t`
cigar	CIGAR: (see BAM spec)	`uint32_t[l_cigar]`
n_optfields	Number of optional fields	`uint32_t`
Optional fields (length: n_header_tags times)		
tag	Two-character tag	`char[2]`
val_type	Value type	`char (AcCsSiIfZBJH)`
value	Tag value	`depends on val_type`

Containments

Empty overlap shall be stored as follows: `l_cigar` shall be set to 0 and the field `cigar` shall be skipped.

Field	Description	Type
n_containments	Number of containment records	`uint32_t`
Containments (length: n_containments)		
from	From segment ID + 1 (orient: sign)	`int32_t`
to	To segment ID + 1 (orient: sign)	`int32_t`
l_cigar	Number of CIGAR operations	`uint32_t`
cigar	CIGAR: (see BAM spec)	`uint32_t[l_cigar]`
pos	Position (0-based, as in GFA)	`uint32_t`
n_optfields	Number of optional fields	`uint32_t`
Optional fields (length: n_header_tags times)		
tag	Two-character tag	`char[2]`
val_type	Value type	`char (AcCsSiIfZBJH)`
value	Tag value	`depends on val_type`

Paths

Field	Description	Type
n_paths	Number of path records	`uint32_t (< MAXINT32T)`
Paths (length: n_paths)		
l_name	Length of path name	`uint32_t`
name	Path name	`char[l_name]`
n_links	Number of links (negative if circular path)	`int32_t`
Path links (length: n_links times)		
link_id	Link ID	`uint32_t`
n_optfields	Number of optional fields	`uint32_t`
Optional fields (length: n_header_tags times)		
tag	Two-character tag	`char[2]`
val_type	Value type	`char (AcCsSiIfZBJH)`
value	Tag value	`depends on val_type`