5.3 C source code for computing bin number and overlapping bins

The following functions compute bin numbers and overlaps for a BAI-style binning scheme with 6 levels and a minimum bin size of 2^{14} bp. See the CSI specification for generalisations of these functions designed for binning schemes with arbitrary depth and sizes.

```
/* calculate bin given an alignment covering [beg,end) (zero-based, half-closed-half-open) */
int reg2bin(int beg, int end)
{
    --end;
    if (beg>>14 == end>>14) return ((1<<15)-1)/7 + (beg>>14);
    if (beg>>17 == end>>17) return ((1<<12)-1)/7 + (beg>>17);
    if (beg>>20 == end>>20) return ((1<<9)-1)/7  + (beg>>20);
    if (beg>>23 == end>>23) return ((1<<6)-1)/7  + (beg>>23);
    if (beg>>26 == end>>26) return ((1<<3)-1)/7  + (beg>>26);
    return 0;
}
/* calculate the list of bins that may overlap with region [beg,end) (zero-based) */
#define MAX_BIN (((1<<18)-1)/7)
int reg2bins(int beg, int end, uint16_t list[MAX_BIN])
{
    int i = 0, k;
    --end;
    list[i++] = 0;
    for (k =    1 + (beg>>26); k <=    1 + (end>>26); ++k) list[i++] = k;
    for (k =    9 + (beg>>23); k <=    9 + (end>>23); ++k) list[i++] = k;
    for (k =   73 + (beg>>20); k <=   73 + (end>>20); ++k) list[i++] = k;
    for (k =  585 + (beg>>17); k <=  585 + (end>>17); ++k) list[i++] = k;
    for (k = 4681 + (beg>>14); k <= 4681 + (end>>14); ++k) list[i++] = k;
    return i;
}
```

5.4 The SBI index format for BGZF files

The SBI format is a binary file format to provide random access to records in data files that have been block compressed with BGZF.

SBI facilitates parallel processing of BGZF data files. Since records are indexed by their virtual file offset rather than position in the genome, unlike the BAI and CSI formats, SBI does not suffer from skew due to uneven distribution of records across the genome. Furthermore, SBI does not require that the data file is coordinate sorted.

SBI is a linear index that contains virtual file offsets of record start positions. The index must contain the virtual file offset for the first record, and a final sentinel virtual file offset for the position at which the next record would start if it were added to the file.[26]

The granularity of the index indicates the number of records between subsequent offsets in the index (excluding the sentinel offset). A granularity of -1 means that there is not a fixed number of records between subsequent offsets in the index.

SBI filenames have a `.sbi` extension added to the name of the file it is an index for. For example, `foo.bam.sbi` is the SBI filename for `foo.bam`. Index files contain a header followed by a sorted list of virtual file offsets in ascending order.

The main uses for the index are:

- Splitting a file for parallel processing. To find the records for a split that covers a byte range `[beg, end)` use the index to find the smallest virtual file offset, `v1`, that falls in this range, and the smallest virtual file offset, `v2`, that is greater than or equal to `end`. If `v1` does not exist, then the split has no records. Otherwise, it has records that start in the range `[v1, v2)`. This method will map a set of contiguous,

[26] In the unlikely event the data file has no records, the index will consist solely of the sentinel offset.

Field	Description	Type	Value
magic	Magic string	`char[4]`	`SBI\1`
file_length	Length of the data file in bytes	`uint64_t`	
md5	MD5 hash of the data file, or 16 \0 bytes if unspecified	`byte[16]`	
uuid	UUID for the data file, or 16 \0 bytes if unspecified	`byte[16]`	
n_records	Total number of records	`uint64_t`	
granularity	Number of records between offsets, or −1 if unspecified	`int32_t`	
n_offsets	Number of virtual file offsets	`int32_t`	
List of offsets (n=n_offsets)			
offset	Virtual file offset of the start of the record	`uint64_t`	

non-overlapping *file ranges* that cover the whole data file to a set of contiguous, non-overlapping *virtual file ranges* that cover the whole data file.

- Finding the nth record in a file. For an index with granularity g, find the virtual file offset at position $\lfloor n/g \rfloor$ in the index. Seek to the record in the data file at this position, and then read a further $n \bmod g$ records to find the desired record.