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 Splitting BAM

A BAM file can be processed in parallel by conceptually dividing the file into splits (typically of a fixed, but arbitrary, number of bytes) and for each split processing alignments from the first known alignment after the split start up to the first known alignment of the next split.

A splitting BAM index is a linear index of virtual file offsets of alignment start positions. The index must contain the virtual file offset for the first alignment, and a virtual file offset for the overall length of the BAM file.[26] It does not need to contain a virtual file offset for every alignment, merely a subset. A granularity of n means that an offset is written for every n alignments.

To find the alignments 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 alignments. Otherwise, it has alignments in the range `[v1, v2)`. This method will map a set of contiguous, non-overlapping *file ranges* that cover the whole BAM file to a set of contiguous, non-overlapping *virtual file ranges* that cover the whole file.

Splitting BAM index filenames have a `.sbi` extension added to the BAM filename (so `foo.bam.sbi` is the splitting BAM index filename for `foo.bam`). Index files contain a header followed by a sorted list of virtual files offsets in ascending order.

Field	Description	Type	Value
magic	Magic string	`char[4]`	`SBI\1`
granularity	Number of alignments between offsets, or -1 if unspecified	`int32_t`	
List of offsets			
offset	Virtual file offset of the alignment	`uint64_t`	

[26] In the unlikely event the BAM file has no alignment records, the index will consist of a single entry for the overall length of the BAM file.