Data Structure comparison — Load Audio

① map — fast for lookup, but if there are doubles, problems. map → vector solves, but gets complicated.

② Vector of albums w/ vector of tracks
- search by artist, album benefit
- search by track name more complicated.
- not easily applied to results/filtering from other source

History - MServ
Play buffer was 1 sec. — search of 20k records using enhanced algorithm was within this.

→ ③ Single list Vector of tracks
- Simple, potentially reusable for subfiltering of service search results.
- Not as optimal as other options in certain circumstances, but those may not be used enough to justify complexity.
- Vector is better for random selection.

④ ~~Multiset~~ unordered_multiset
- fast lookups
- doubles handled okay
- clever hashing off first few letters might work a lot of the time (but not always)
- But to work on multiple fields, there needs to be multiple maps. ~~or multiple insertion ???~~

Chosen implementation

Enhanced file Scan - Pianod

Limitations/Problems of current scanner:

- IDs assigned at random - data loss if file corrupted
- Poor behavior for compilations
- Duplicate names within album = clobber

Solutions:

→ Generate ID based on Data

- Album: Use unique ID from CDDB (ID3 metadata) if available
  Alternate: name checksum/hash.
- Artist: Check album first
  1. Lookup by CDDB ID/hash, and check album title (not lookup)
  2. If match, change album artist → compilation flag set; artist's name override in all of the album's songs.
  3. ~~Assign artist ID, from found album OR by name checksum/hash~~
  3. Assign artist ID:
     - if no album, use hash/checksum of name.
     - if album, update artist ID to hash/checksum of lowest artist's name in songs. (Provides consistency)

  (margin note: when compilation is set, get artist ... list all by concatenation)
- Songs: checksum/hash album name + title
  (Provides better uniqueness over title alone).

  If already exists
  - On initial scan → simply insert.
  - On rescan, check filename match within album's songs; if not found then insert.

  If does not exist:
  - Insert

Random biasing algorithm

Factors:
- Rating) - average user rating
- Time since last played.
- Number of plays? No, will counter/influence age

~~factor = base * rating * agefactor~~
default neutral
1 – 10

factor = (rating * rating scale) * (age * age scale)

To get all equal, we want a fixed factor for any age / rating,
- Can't use ∅ for either scale b/c that kills both.

Thus we must have
factor = (base ± adjustment) * (base ± adjustment)
such that either portion ≠ 0

Let base = 10.0

To let the factor do an range of ±10*, adjustment must be 0 -10.0

NO! Not 0.1 - 10.0; 0 - 10.0

log 1

Rating portion:

- Scale factor (user) = 1 - 100 = S
- Scale factor (algo const) = S'

$$S' = (\log(9+S) - 1) * 8.675313411 + 1$$

- Per-song multiplier frate

$$f_{rate} = S'^{\frac{\text{Rating given} - 6}{4}}$$

6 = neutral rating
4 = max - neutral

Age portion:

- ~~Song~~ average all ages in library, where known
- Individual factor = $\frac{\text{age of song}}{\text{average age}}$, max 10.

shared →

- Scale factor (user) = 1 - 100 = S
- Scale factor (algo const) = S'

~~$S' = (\log(9+S) - 1) * \_\_ + 1$~~

$$S' = (\log(9+S) - 1) * 8.675313411 + 1$$

- ~~Rating~~ Per-song multiplier f_{age}

$$f_{age} = S'^{\log(\min(10, \frac{age}{average\ age}))}$$

music library architecture - planed