# （算法之美）Playlist建表问题总结

## 前言

其实这本应该不是问题，就是说当不用考虑硬件成本的时候，有足够的内存空间给你使用，排序建表是分分钟的事。 但是当从企业的角度来看，特别是芯片原厂，出货量用KK算的时候，他需要考虑的一个重要的问题就是成本和利润，每颗芯片能够省1毛钱，1KK颗就省10万元，100KK就省1000万的利润，是非常可观的。 关键问题就在这里，企业为了省这一笔钱，把内RAM/ROM空间卡得很死，没有足够的余量，然而又想支持一些当初没有考虑到的项目的时候，问题就出现了。

## 问题

这属于一个MP3播放器项目，使用的芯片本来没有规划来做这个项目的产品，但是现在为了应对市场更高的要求，决定使用这颗芯片，对于这个项目的高规格这颗芯片有一个缺点————内存不足。

**需求描述**:

```
1. 支持4000首歌曲到8000首歌曲的建表排序;
2. 需要有Title, Artist, Album, Genre目录，当然每一个目录都是要建表的;
3. 每首歌曲文件需要按照ID3或者文件名排序，Title最大支持255个字节,其他只需要支持48个字节;
4. 可使用RAM空间96K，Card存储空间管够，下面所说的空间都指RAM空间;
5. 时间要求3min;
```

## 分析

支持大于4000首歌曲排序，并且要有Title,Artist,Album和Genre目录，我们知道，这几个目录是有关联性的，关联方式如下:

```
歌曲:  Title->end
专辑:  Album->Title
作者:  Artist->Album->Title
风格:  Genre->Artist->Album->Title
```

这是一种**树形结构**，每一个节点都与上一节点有关。因此需要有几个表来缓存排好的索引，以便下一属性排序的使用，一个表按照4000首歌曲算，至少需要用有一个单位为2个字节的索引表保存，因此一个表需要8K空间，此处需要8个表，共需要64K空间，并非一开始就要64K，有一些表可以复用，为什么需要8个表，后面会说明。 我们知道每一个标准的歌曲文件都具有ID3信息，不论是ID3 v1还是v2版本。当然也有非标准的音乐文件啊，此时我们给其赋默认值。并进行统一分类。 现在我们对这几种目录的排序进行一一的分析。

**1. 歌曲排序** 歌曲排序默认排序ID3信息的Title,当不存在ID3信息的时候取歌曲的文件名前255字节，每个Title需要最大支持255个字节排序。 排序最终生成的是一个`TitleIndexList[8K]`表，可以理解为`4000*2bytes`等于8K的数组，排序过程中还需要一个8K表`TmpIndexList[8K]`来缓存上一次排序的结果，因此表数据占用16K空间，可以用于实际排序的只有88K空间。

**2. 专辑排序** 专辑排序需要`TitleIndexList[8K]`表，因为每一个专辑里都有许多歌曲，不可能再重新排序一遍。专辑也需要有一个对应的`AlbumIndexList[8k]`，再加上`TmpIndexList[8K]`，剩下排序能使用的空间为80K。排完序还需要生成一份`AlbumSameObjList[8K]`用来记录有相同专辑的歌曲文件。

**3. 作者排序** 作者排序需要使用`AlbumIndexList[8k]`和`SameObjList[8K]`，再加上自己的`ArtistIndexList[8K]`和`TmpIndexList[8K]`排序能使用的空间为72K。当然他也需要一个`ArtistSameObjList[8K]`来记录相同作者的专辑。

**4. 风格排序** 风格排序需要使用`GenreIndexList[8k]`,`ArtistIndexList[8K]`,`TmpIndexList[8K]`,`AlbumSameObjList[8K]`,`ArtistSameObjList[8K]`五个表来记录和生成列表，占用40K空间，能用来排序的空间有56K。