2008 ACM ICPC South Central USA Regional Programming Contest

The Extent of the Problem

Introduction:

Files on modern filesystems are not always stored contiguously. Instead, they are broken up into chunks (called *extents*), each of which may be located almost anywhere on the underlying disk. Each extent takes up one or more contiguous *blocks* on the disk, which represent the underlying physical disk structure. Files may not share blocks or extents between themselves, although there may be (and hopefully are) blocks not currently in use on the filesystem.

Despite all of the advances in modern disk technology, this disparate layout still makes for slower access than when all of a file's blocks are in order and adjacent (and therefore occupying a single extent). Because of this, many modern filesystems support *defragmentation*: the reorganizing of files on the disk so that the blocks of any given file are in order and adjacent.

With RAD's Awesome Dynamic Filesystem (commonly referred to as **RADfs**), a file can occupy any number of extents, each of which consists of at least two adjacent blocks. The first block in an extent holds metadata about that extent; the rest contain some portion of the file. Consider this simple representation of the extents and blocks occupied by a file:

```
RADfs.doc: 37-38,102-114,23-47
```

The file `RADfs.doc` currently occupies 40 blocks on the disk, even though the file itself is only 37 blocks in size. The smallest on-disk size that it could have is 38 blocks (37 plus a single metadata block), and that can only happen if the entire file is in a single extent. One potential single-extent representation of the file is:

```
RADfs.doc: 115-152
```

Here, the single metadata block (115) is followed by the 37 data blocks in a single extent.

The developers of RADfs have also developed the RADical Defragmentation Daemon, or RADDD. RADDD works with a simple two-step algorithm; despite its simplicity, it still manages to significantly reduce the number of extents consumed by files, given enough free space on the disk.

A RADDD pass works as follows:

- First Step ("to the back"):
  - For every file on the filesystem that hasn't been run through this step on this pass, sorted in **ascending** order by the **first** block it occupies on the disk:
    - Find the series of adjacent unused blocks nearest the **end** of the disk that can hold the file plus a single metadata block
    - If such a series exists:
      - Move the file to those blocks
      - Mark the original blocks as unused
    - Else do nothing to the file
- Second Step ("to the front"):
  - For every file on the filesystem that hasn't been run through this step on this pass, sorted in **descending** order by the **last** block it occupies on the disk:
    - Find the series of adjacent unused blocks nearest the **beginning** of the disk that can hold the file plus a single metadata block
    - If such a series exists:
      - Move the file to those blocks
      - Mark the original blocks as unused
    - Else do nothing to the file

After one pass on a filesystem with some free space, some files have hopefully been reduced to a single extent. Multiple passes can often defragment the filesystem even further.

Of course, it's not that simple; some files simply cannot be moved, perhaps due to being in use at the time of the run. These are marked on the filesystem as *immobile*, and are ignored by RADDD.

Given the size of a disk, the current layout of files on the disk, and a number of passes of RADDD to run, can you determine the final filesystem layout?

Input:

Input to this problem will begin with a line containing a single integer N ($1 \le N \le 100$) indicating the number of data sets. Each data set consists of the following components:

- A line containing a single integer S ($2 \le S \le 100000$) indicating the number of blocks on the particular filesystem;
- A line containing a single integer C ($1 \le C \le 100$) indicating the number of files on the filesystem;
- C lines representing the files on the filesystem, in the format "`NAME TYPE E A-B[ X-Y[ ...]]`", where:
  - *NAME* is a unique (to the dataset) identifier, consisting of at least 1 and no more than 16 lowercase letters;
  - *TYPE* is "`I`" if the file is immobile, or "`M`" otherwise;
  - E is an integer ($1 \le E \le 20$) representing the number of extents the file occupies;
  - A and B ($1 \le A, B \le S$) are, respectively, the first and last blocks of the first extent the file occupies;
  - X and Y ($1 \le X, Y \le S$) are, if present, the first and last blocks of the second extent the file occupies;
  - and so on;
- A line containing a single integer P ($1 \le P \le 100$) indicating the number of passes of RADDD that should be run.

Output:

For each data set in the output, output the heading "`DATA SET #k`" where k is 1 for the first data set, 2 for the second, and so on. On the next C lines, output the locations of the files on the disk after the RADDD passes in ascending order by the first block occupied on the disk. The format should be identical to the representation in the input; if a file occupies multiple extents, the output should be sorted in ascending order by the first block in each extent.

Sample Input:

```
2
152
1
radfsdoc M 3 37-38 102-114 23-47
1
100
4
swapfile I 3 5-10 80-95 25-50
smallfile M 2 1-4 11-14
bigfile M 2 15-24 51-60
tinyfile M 1 61-64
2
```

Sample Output:

```
DATA SET #1
radfsdoc M 1 1-38
DATA SET #2
tinyfile M 1 1-4
swapfile I 3 5-10 25-50 80-95
bigfile M 2 15-24 51-60
smallfile M 1 61-67
```

REVISED: 2008.10.16.1322