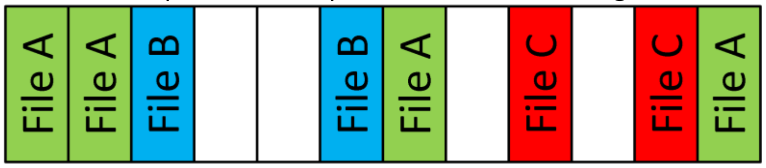
CUSEC Coding Competition

CUSEC 2012 JANUARY 19 to 21

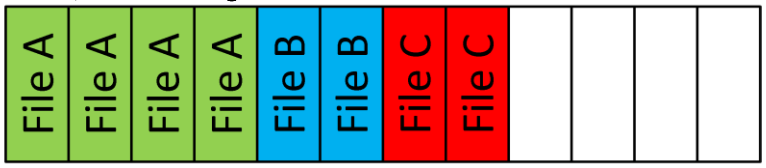
Problem Description

Defragmentation is the process of reducing the amount of file system fragmentation, which occurs when files are broken into many non-contiguous pieces. During defragmentation, files and free space are reordered into contiguous areas. This improves the disk performance, since files can be read sequentially.

For example, if the white blocks represent free space, then data on a fragmented disk might look like:



And after defragmentation, the data might look like:



Goal

The goal of this problem is to design a defragmentation algorithm for a fictional file system.

Input

The initial state of the fragmented disk is represented through a text file containing numbers.
Each line in the file represents a block on the disk:

- The address of the block is represented implicitly by the line number, starting with the first line in the file, representing address `0`
- Numbers in the file are whitespace separated (space, tab) and may contain leading and trailing white spaces. Numbers may also be padded with leading zeros. Your program should handle the tuple “`0 1`” just as well as “`      00    01      `“
- The first number of each line is the file ID
- The second number of each line is the sequence number of that block within the file, starting at 0
- For example, “1234 2” represents the second block of file 1234

There are two special file IDs:

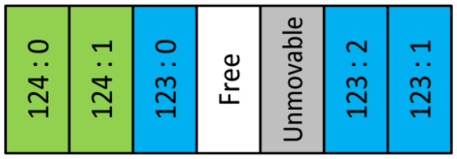
- Free space, represented by file ID **`0`**, is space that blocks can be moved onto.
- Unmovable blocks, represented by file ID **`-1`**, are blocks that must remain at their original location because they are in use by the operating system.

Sequence numbers for free space and unmovable blocks can be ignored.

All other file IDs are valid, and can be moved according to the constraints described in the Constraints section.

For example, the text file on the left would correspond to the fragmented disk on the right:

```
124 0
124 1
123 0
  0 0
 -1 0
123 2
123 1
```



Output

Your program should output a text file containing a list of moves. A move can span consecutive blocks (like memcpy). Each move must contain the **source begin address**, **source end address** and **destination address**. Once again, these numbers are whitespace separate, following the same rules as the disk file format.

- The **source begin address** in inclusive and marks the start of the source range.
- The **source end address** is exclusive and marks the end of the source range
- The **destination** is the new location of the series of blocks

Example of Moves

Move the block at address 3 to address 4:

```
3    4   4
```

Move the blocks between addresses `0` and 9 to address `10`:

```
0    9   10
```

Move them back:

```
10   19   0
```

Data can **only** be moved onto free space. For example, to swap the contents of blocks A and B, one must first move the contents of one of the blocks to an address containing free space.

Constraints

The following constraints must be observed. **Failing validation will lead to disqualification**.

- The output of your program must match the given output format.
- Moves must be **valid**
  - The destination of a move must always be **free space**, and must always have enough space to fit **all** blocks
  - Unmovable blocks cannot be moved
  - Moves must be within the valid range of the disk
- There is no temporary space available aside from the free space on the disk
- Your program must run in a ***reasonable*** amount of time (i.e. approximately less than 5 minutes on a modern computer [ex. dual-core 2.0Ghz with 2Gb RAM])

Scoring

The programs will be scored based on a weighted function containing

- **The average size of contiguous free space ranges - 25%**
    - A contiguous free space range is a sequence of blocks containing only free space
    - A good program would have a **high** average size of contiguous free space ranges
    - The average is biased to give a higher score to disks containing large ranges of free space. For example, consider two disks containing two ranges of free space each: 6-1 and 4-3. These disks have the same average free space range size. The first disk will get a higher score because it has a larger range of 6 free blocks.
- **The average number of seeks required to read files - 75%**
    - A seek represents moving the head of the hard disk to read a different part of a file
    - In our simplified disk, this is represented as a discontinuity in the file when reading it sequentially.
    - In the input file example, file #123 needs to make 3 seeks: 1 seeks to move the head to the sequence 1, another seek to move the head to sequence 2, and a final more seek to reach sequence 3 (since it is behind and our disk only reads forward). File #124 only requires 1 seek though, because sequence 1 and sequence 2 are adjacent and the disk can read them together
    - A perfectly contiguous file will have only a single seek (to reach sequence 1), so the **lowest** average number of seeks is best

This scoring function is normalized so that a score of 0.0 is the worst possible state of the disk, where all files and free space fragmented completely. A score of 10,000.0 is a perfectly defragmented disk, with all files being readable with a single seek and free space in one contiguous block.

Note: Some disks with immovable blocks will be impossible to perfectly defrag.

In the case of a tie, we will also consider the number of moves the program used to defragment the disk. The winner would be the program that has the **least** number of moves.

We encourage you to run the scoring function on your output disks to see how your algorithm stacks up with our test disks. Visit http://2012.cusec.net/competition to download a zip file containing:

- A set of test disks
- A command-line random disk generator written in Python
- A command-line move execution and scoring program written in Python
- Unit tests for the scoring program

The Python programs were written against Python 2.7.1

Scoring Program

Usage: `python defrag_scorer.py [-h] --disk DISK --moves MOVES [--output OUTPUT]`

- `--disk` specifies a path to a disk file
- `--moves` specifies a path to a moves file (the output of your program)
- `--output` is optional and will print out the final defragmented disk in the same file format as the input. If this parameter is not set, the final disk state will be printed to the console

The program will print an exception if an invalid move is encountered. It will display which line of the move file the error was found and the state of the disk at the time of the error. If you can run the scorer without an exception, your move file is valid and meets all our constraints.

Generation Program

Usage:
```
python disk_generator.py [-h] --size SIZE [--pctfree PCTFREE]
                         [--pctimmovable PCTIMMOVABLE]
                         [--minfilesize MINFILESIZE]
                         [--maxfilesize MAXFILESIZE] [--output OUTPUT]
```

- `--size` specifies the size of the disk (in blocks) to generate
- `--pctfree` is the optional percentage of free blocks (from 0 to 100). Defaults to 50
- `--pctimmovable` is the optional percentage of immovable blocks (from 0 to 100). Defaults to 10
- `--minfilesize` is the optional minimum length of a file on disk. Defaults to 1
- `--maxfilesize` is the optional maximum length of a file on disk. Defaults to SIZE
- `--output` is the output filename. If no filename is specified, the disk will be printed in the console.

Disk Visualizer

We have written a web application to help visualize the contents of the disk files as well as interface with the scoring program. You can find this application at http://cusec2012.appspot.com/

Bugs

If you find bugs with any of these programs, please contact us. We have provided unit tests for the scorer as an additional guarantee, but it's still possible some bugs will have made it through. Please provide a reproducible test case or patch when you contact us and we will try to correct it before the submission deadline is up.

Hints

Developing a trivial solution is not too difficult; therefore we suggest that you use an iterative development approach: get a basic solution working, and then tweak your algorithms to maximize your score. Here are some things to think about:

- Strategy for dealing with unmovable blocks
- Strategy for ordering/placing files
- Strategy for moving blocks

Submission

To submit your program, send an email to cusec.competition@gmail.com with your source code and detailed instructions needed to compile and run it. **If the judges cannot run your code, it will be disqualified**. Please also include a brief description of your algorithm along with the **name, school and email** of all of your team members.

When the email has been received, you will receive an automatic confirmation response.

Prizes

Winning teams will have the choice between gift certificates from **either** Future Shop or Newegg.ca

1st place	$300
2nd place	$200
3rd place	$100