Dealing with Indirect Blocks in `share`

Although the `share` command should share both data blocks and indirect blocks, considering all of them together as a pool does not work. The reason for this is the fact that each block affects the numbers in the indirect block pointing to it (if the indirect block exists). Thus, each level needs to be traversed one by one. One pass considering all the data blocks together, the second pass considering all the single indirect blocks together, the third pass considering all the double indirect blocks together and a final pass considering all the triple indirect blocks together.

An example will make this clearer. Consider a file consisting of 7 blocks, and assume that the inodes only have 3 direct blocks and that indirect blocks can contain 8 entries for simplicity. The file information is in inode A, shown in Figure 1. Also consider a **copy** of this file placed in inode B shown next to it. Since the file is a copy, matching blocks contain the same data, but are different blocks: Block 17 and 30 contain the same data, 99 and 31 contain the same data, so do 19 and 32, 54 and 33 and so on.

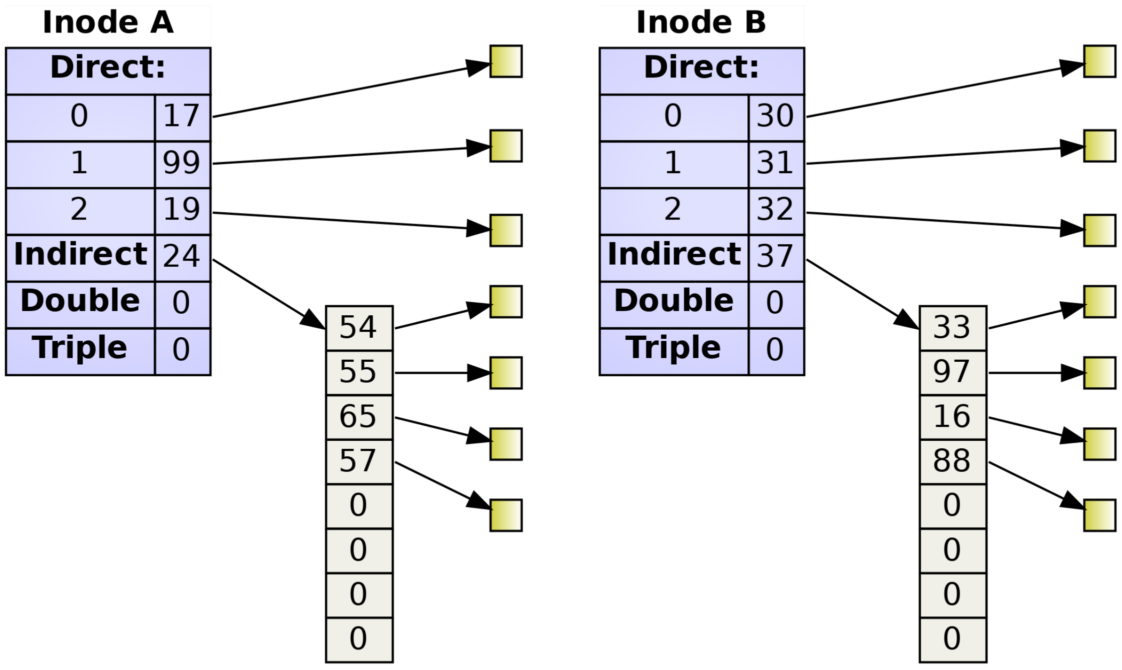

Figure 1: inode A and its copy B's block layout

Now, since these two files are copies of each other, we want `share` to be able to share them entirely, as if they had been `dup`ed. However, pooling all the blocks together would not serve us well, because the indirect blocks of inode A and B are currently different.

To solve this, we first pass over only the data blocks of the two files. During this pass, we will notice that 17 and 30 contain the same data, and thus replace 30 with 17. The same goes for 99 and 31; where 99 will be replaced by 31 (we always choose the block with the smallest number). The same goes for all the other matching data blocks. The files become as shown in Figure 2 after the first pass over the data block. The modified block pointers are marked in red.

Now that we've modified all pointers to data blocks, we can pass over them in the second pass. Our two files only have a single indirect pointer: 24 and 37. But now their contents are the same,

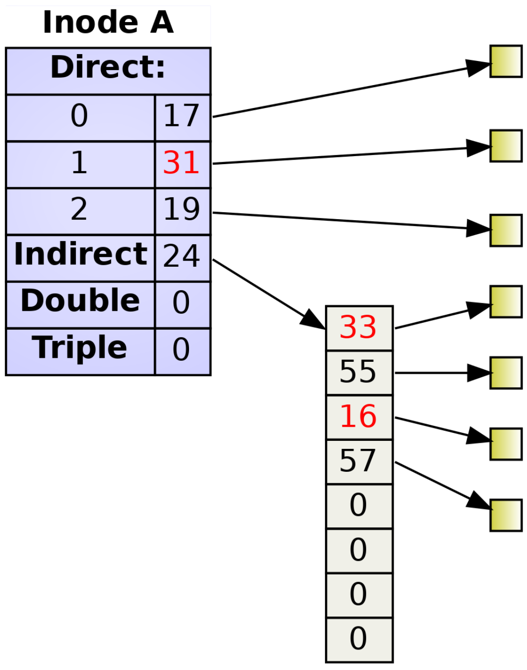

Figure 2: The block layout after the first (data) pass

and our pass will detect this and replace 37 by 24, achieving the final layout show in Figure 3, with the changes for this pass marked in blue.

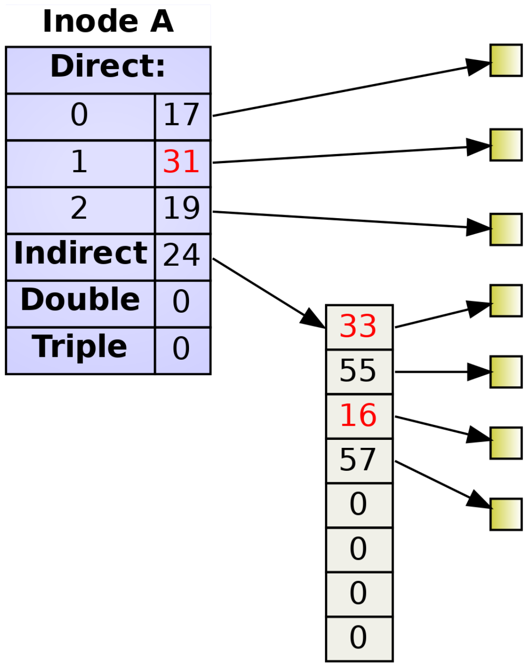

Figure 3: The block layout after the second (single indirect) pass

Since our files do not have double and triple indirects, we can stop here. Our copied files now show all the same blocks and appear as if they had been `dup`ed originally! For completeness, a correct output for this case could be the following:

```
17 17:1 30:1
31 31:1 99:1
19 19:1 32:1
33 33:1 54:1
55 55:1 97:1
16 16:1 65:1
57 57:1 88:1
24 24:1 37:1
```

Hopefully, this demonstrates the core idea well enough. In the more generic case, we need up to four passes, one for each level, because we can only check a level after it has been modified by the lower level. A file with all pointers is shown in Figure 4 and blocks at the same level are marked with the same color. Note that the file is full of holes for the sake of keeping the figure small! Obviously, sharing triple indirects only becomes meaningful when multiple files exist, so that there can be multiple triple indirects.

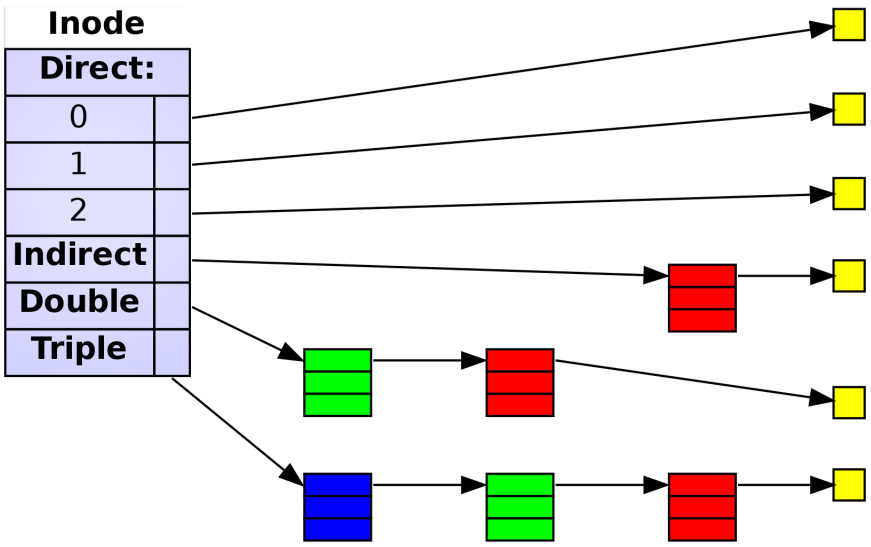

Figure 4: Hypothetical inode figure, blocks of the same pass are marked with the same color

Also, `share` is not just limited to the two file case. It should work on any given number of files: N copies of a file would make them all `dups` of each other. It should also be able find matches inside a single file (an example case could be sharing the blocks of a huge file filled with ones). Essentially, file boundaries do not matter: You can consider all the blocks at a given level from every file together.

Limitations

You may have noticed an issue with the above approach. We have been modifying indirect blocks to our heart's content, but this is an action that will corrupt the filesystem if the indirect blocks involved already have a reference count greater than one. This is because we will be modifying a possibly unrelated file when we modify the indirect block.

Solving this would require copy-on-write: Allocating a new block and copying the content in case the reference count is greater than one. You are free to implement this, however this will not be tested because it further complicates the implementation.

You are free to assume that any indirect block you will have to modify has a reference count equal to exactly one. With this assumption, you can live in a simpler universe and modify indirect blocks without a care in the world.

Best of luck! Remember that you can get a lot of partial points even if your implementation does not satisfy the time complexity requirements. Even only dealing with direct blocks will get you through a few cases.