1. Examine the file in.largefile, and then run the simulator with flag -f in.largefile and -L 4. The latter sets the large-file exception to 4 blocks. What will the resulting allocation look like? Run with -c to check.

```
2017310367@swui:~$ chmod u+x ffs.py
2017310367@swui:~$ ./ffs.py -f in.largefile -c

num_groups:       10
inodes_per_group: 10
blocks_per_group: 30

free data blocks: 259 (of 300)
free inodes:      98 (of 100)

spread inodes?    False
spread data?      False
contig alloc:     1

      0000000000 0000000000 1111111111 2222222222
      0123456789 0123456789 0123456789 0123456789

group inodes     data
    0 /a-------- /aaaaaaaaa aaaaaaaaaa aaaaaaaaaa
    1 ---------- aaaaaaaaaa a--------- ----------
    2 ---------- ---------- ---------- ----------
    3 ---------- ---------- ---------- ----------
    4 ---------- ---------- ---------- ----------
    5 ---------- ---------- ---------- ----------
    6 ---------- ---------- ---------- ----------
    7 ---------- ---------- ---------- ----------
    8 ---------- ---------- ---------- ----------
    9 ---------- ---------- ---------- ----------

2017310367@swui:~$
```

flag -f 를 주었을 때의 결과이다. A의 값이 크기 때문에 하나의 블록 그룹에 저장되고 남은 데이터들은 그 다음 그룹에 저장된 것을 확인할 수 있다.

```
num_groups:       10
nodes_per_group: 10
locks_per_group: 30

free data blocks: 259 (of 300)
free inodes:      98 (of 100)

spread inodes?    False
spread data?      False
contig alloc:     1

      0000000000 0000000000 1111111111 2222222222
      0123456789 0123456789 0123456789 0123456789

group inodes     data
    0 /a-------- /aaaa------ ---------- ----------
    1 ---------- aaaa------ ---------- ----------
    2 ---------- aaaa------ ---------- ----------
    3 ---------- aaaa------ ---------- ----------
    4 ---------- aaaa------ ---------- ----------
    5 ---------- aaaa------ ---------- ----------
    6 ---------- aaaa------ ---------- ----------
    7 ---------- aaaa------ ---------- ----------
    8 ---------- aaaa------ ---------- ----------
    9 ---------- aaaa------ ---------- ----------
```

만약 -L 4로 4개씩 블록에 데이터를 담도록 지정한다면 이런 식으로 나누어 데이터를 저장한다.

2. Now run with -L 30. What do you expect to see? Once again, turn on -c to see if you were right. You can also use -S to see exactly which blocks were allocated to the file /a.

```
2017310367@swui:~$ ./ffs.py -f in.largefile -c -L 30

num_groups:       10
inodes_per_group: 10
blocks_per_group: 30

free data blocks: 259 (of 300)
free inodes:      98 (of 100)

spread inodes?    False
spread data?      False
contig alloc:     1

      0000000000 0000000000 1111111111 2222222222
      0123456789 0123456789 0123456789 0123456789

group inodes     data
    0 /a-------- /aaaaaaaaa aaaaaaaaaa aaaaaaaaaa
    1 ---------- aaaaaaaaaa a---------- ----------
    2 ---------- ---------- ---------- ----------
    3 ---------- ---------- ---------- ----------
    4 ---------- ---------- ---------- ----------
    5 ---------- ---------- ---------- ----------
    6 ---------- ---------- ---------- ----------
    7 ---------- ---------- ---------- ----------
    8 ---------- ---------- ---------- ----------
    9 ---------- ---------- ---------- ----------
```

-L 30 플러그를 주면 30개씩 블록에 담는다.

```
2017310367@swui:~$ ./ffs.py -f in.largefile -c -S -L 30

num_groups:       10
inodes_per_group: 10
blocks_per_group: 30

free data blocks: 259 (of 300)
free inodes:      98 (of 100)

spread inodes?    False
spread data?      False
contig alloc:     1

      0000000000 0000000000 1111111111 2222222222
      0123456789 0123456789 0123456789 0123456789

group inodes     data
    0 /a-------- /aaaaaaaaa aaaaaaaaaa aaaaaaaaaa
    1 ---------- aaaaaaaaaa a---------- ----------
    2 ---------- ---------- ---------- ----------
    3 ---------- ---------- ---------- ----------
    4 ---------- ---------- ---------- ----------
    5 ---------- ---------- ---------- ----------
    6 ---------- ---------- ---------- ----------
    7 ---------- ---------- ---------- ----------
    8 ---------- ---------- ---------- ----------
    9 ---------- ---------- ---------- ----------
```

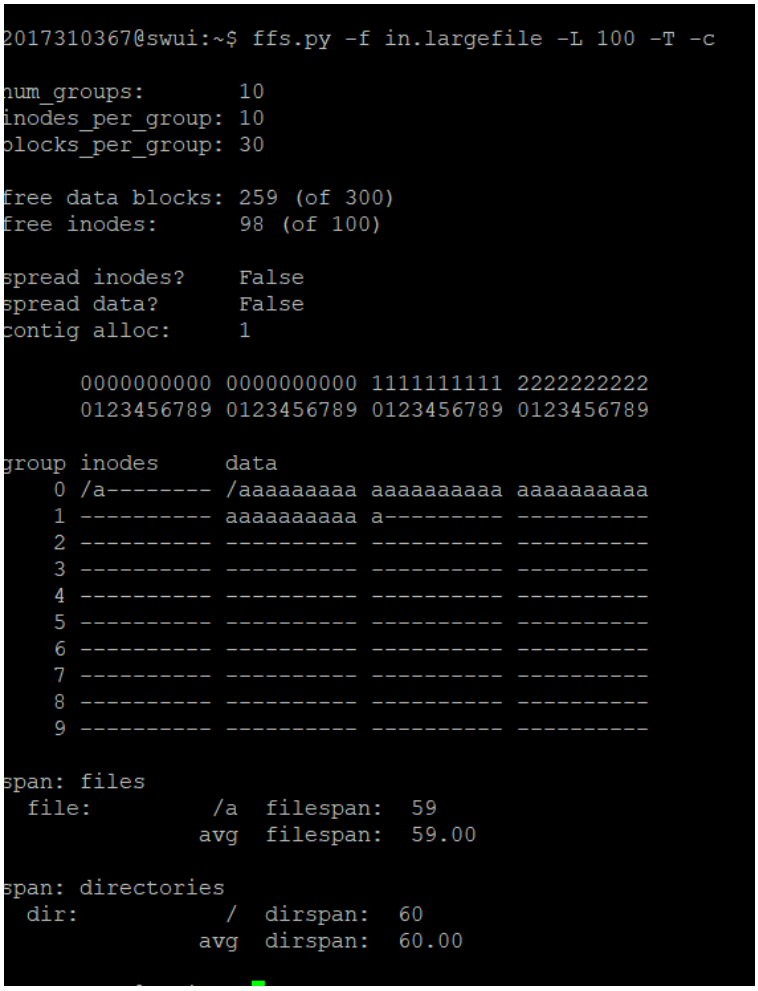
실제 어느 블록이 a를 담았는지 확인해보자. Inode_per_group은 10이고 block_per_group은 30이라고 나온다.

3. Now we will compute some statistics about the file. The first is something we call filespan, which is the max distance between any two data blocks of the file or between the inode and any data block. Calculate the filespan of /a. Run ffs.py -f in.largefile -L 4 -T -c to see what it is. Do the same with -L 100. What difference do you expect in filespan as the large-file exception parameter changes from low values to high values?

```
2017310367@swui:~$ ffs.py -f in.largefile -L 4 -T -c

num_groups:       10
inodes_per_group: 10
blocks_per_group: 30

free data blocks: 259 (of 300)
free inodes:      98 (of 100)

spread inodes?    False
spread data?      False
contig alloc:     1

      0000000000 0000000000 1111111111 2222222222
      0123456789 0123456789 0123456789 0123456789

group inodes     data
    0 /a-------- /aaaa----- ---------- ----------
    1 ---------- aaaa------ ---------- ----------
    2 ---------- aaaa------ ---------- ----------
    3 ---------- aaaa------ ---------- ----------
    4 ---------- aaaa------ ---------- ----------
    5 ---------- aaaa------ ---------- ----------
    6 ---------- aaaa------ ---------- ----------
    7 ---------- aaaa------ ---------- ----------
    8 ---------- aaaa------ ---------- ----------
    9 ---------- aaaa------ ---------- ----------

span: files
  file:         /a  filespan: 372
               avg  filespan: 372.00

span: directories
  dir:           /  dirspan: 373
               avg  dirspan: 373.00
```

다른 두 데이터블록, 혹은 아이노드와 다른 데이터 블록 간의 최대거리를 알려주는 filespan을 구해볼 수 있다. -L 4 일 때의 파일 a의 filespan은 372인 것을 확인할 수 있다.

그러나 만약 -L 100인 경우라면 filespan이 더 줄어들게 된다. -L 4은 의도적으로 파일을 작은 크기로 나누어서 저장하였기 때문에 블록들 간의 거리가 멀다. 그러나 -L 100인 경우 파일들을 하나의 블록 그룹에 모아서 저장하였기 때문에 블록들간의 거리가 가까운 것이다.



실제로 -L 100 일 때의 파일 a의 filespan은 59로 372보다 확연히 낮은 것을 확인할 수 있다.

4. Now let's look at a new input file, in.manyfiles. How do you think the FFS policy will lay these files out across groups? (you can run with -v to see what files and directories are created, or just cat in.manyfiles). Run the simulator with -c to see if you were right.

일단 cat으로 in.manyfiles 에 얼마나 많은 파일이 있는지 확인해보자.

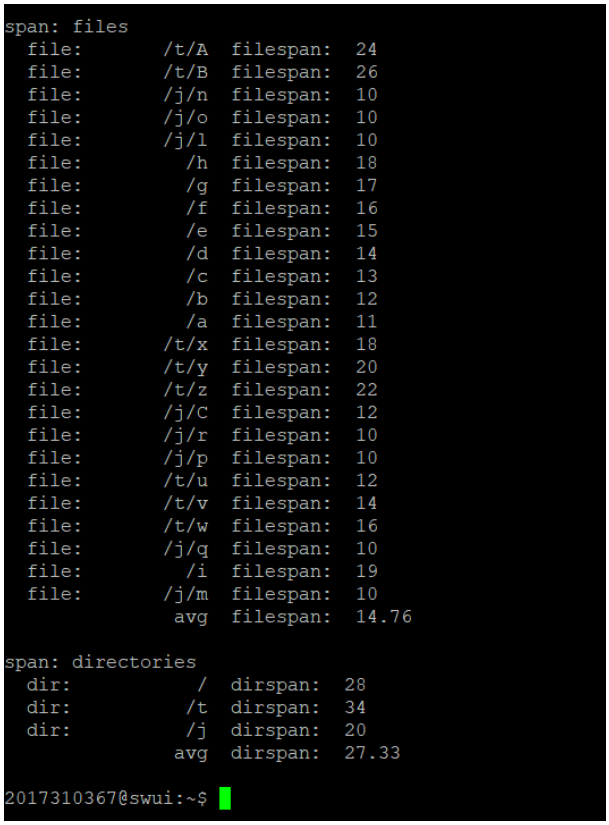
```
2017310367@swui:~$ cat in.many
file /a 2
file /b 2
file /c 2
file /d 2
file /e 2
file /f 2
file /g 2
file /h 2
file /i 2

dir /j
dir /t

file /t/u 3
file /j/l 1
file /t/v 3
file /j/m 1
file /t/w 3
file /j/n 1
file /t/x 3
file /j/o 1
file /t/y 3
file /j/p 1
file /t/z 3
file /j/q 1
file /t/A 3
file /j/r 1
file /t/B 3
file /j/C 3
```

이 파일들을 ffs가 어떻게 저장하는지 확인해보자.

```
2017310367@swui:~$ ./ffs.py -f in.manyfiles -c

num_groups:       10
inodes_per_group: 10
blocks_per_group: 30

free data blocks: 245 (of 300)
free inodes:      72 (of 100)

spread inodes?    False
spread data?      False
contig alloc:     1

      0000000000 0000000000 1111111111 2222222222
      0123456789 0123456789 0123456789 0123456789

group inodes     data
    0 /abcdefghi /aabbccdde effgghhii- ----------
    1 jlmnopqrC- jlmnopqrCC C--------- ----------
    2 tuvwxyzAB- tuuuvvvwww xxxyyyzzzA AABBB-----
    3 ---------- ---------- ---------- ----------
    4 ---------- ---------- ---------- ----------
    5 ---------- ---------- ---------- ----------
    6 ---------- ---------- ---------- ----------
    7 ---------- ---------- ---------- ----------
    8 ---------- ---------- ---------- ----------
    9 ---------- ---------- ---------- ----------
```

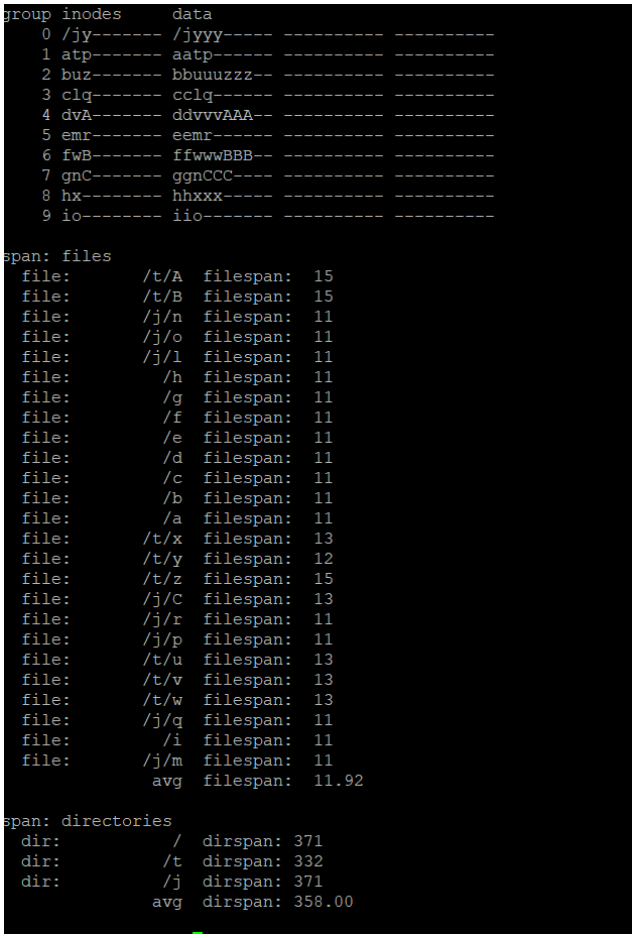
그룹별로 정렬해주는 것을 확인할 수 있다. 같은 디렉토리에 있는 abcdefghi파일을 하나의 그룹에 묶어 저장해주었다. 또한 j와 j의 하위 디렉토리인 lmnopqrC 또한 같은 그룹에 저장해주었으며 t와 t의 하위 디렉토리인 uvwxyzAB도 같은 그룹에 저장되어 있다.

5. A metric to evaluate FFS is called dirspan. This metric calculates the spread of files within a particular directory, specifically the max distance between the inodes and data blocks of all files in the directory and the inode and data block of the directory itself. Run with in.manyfiles and the -T flag, and calculate the dirspan of the three directories. Run with -c to check. How good of a job does FFS do in minimizing dirspan?



디렉토리의 아이노드와 모든 데이터블록의 최대 거리를 계산하여 평가기준이 되어주는 dirspan를 사용해 ffs의 효과를 보자. 상위 디렉토리의 dirspan은 28, t 디렉토리의 dirspan는 34, j 디렉토ㄹ의 dirspan는 20으로 모두 상당히 작은 숫자를 보이고 있다. 이를 통해 ffs가 얼마나 효과적으로 아이노드와 데이터블록들에 파일을 저장하였는지 확인할 수 있다.

6. Now change the size of the inode table per group to 5 (-I 5). How do you think this will change the layout of the files? Run with -c to see if you were right. How does it affect the dirspan?



```
group inodes     data
    0 /jy------- /jyyy----- ---------- ----------
    1 atp------- aatp------ ---------- ----------
    2 buz------- bbuuuzzz-- ---------- ----------
    3 clq------- cclq------ ---------- ----------
    4 dvA------- ddvvvAAA-- ---------- ----------
    5 emr------- eemr------ ---------- ----------
    6 fwB------- ffwwwBBB-- ---------- ----------
    7 gnC------- ggnCCC---- ---------- ----------
    8 hx-------- hhxxx----- ---------- ----------
    9 io-------- iio------- ---------- ----------

span: files
  file:       /t/A  filespan:  15
  file:       /t/B  filespan:  15
  file:       /j/n  filespan:  11
  file:       /j/o  filespan:  11
  file:       /j/l  filespan:  11
  file:         /h  filespan:  11
  file:         /g  filespan:  11
  file:         /f  filespan:  11
  file:         /e  filespan:  11
  file:         /d  filespan:  11
  file:         /c  filespan:  11
  file:         /b  filespan:  11
  file:         /a  filespan:  11
  file:       /t/x  filespan:  13
  file:       /t/y  filespan:  12
  file:       /t/z  filespan:  15
  file:       /j/C  filespan:  13
  file:       /j/r  filespan:  11
  file:       /j/p  filespan:  11
  file:       /t/u  filespan:  13
  file:       /t/v  filespan:  13
  file:       /t/w  filespan:  13
  file:       /j/q  filespan:  11
  file:         /i  filespan:  11
  file:       /j/m  filespan:  11
              avg  filespan:  11.92

span: directories
  dir:          /  dirspan: 371
  dir:         /t  dirspan: 332
  dir:         /j  dirspan: 371
              avg  dirspan: 358.00
```

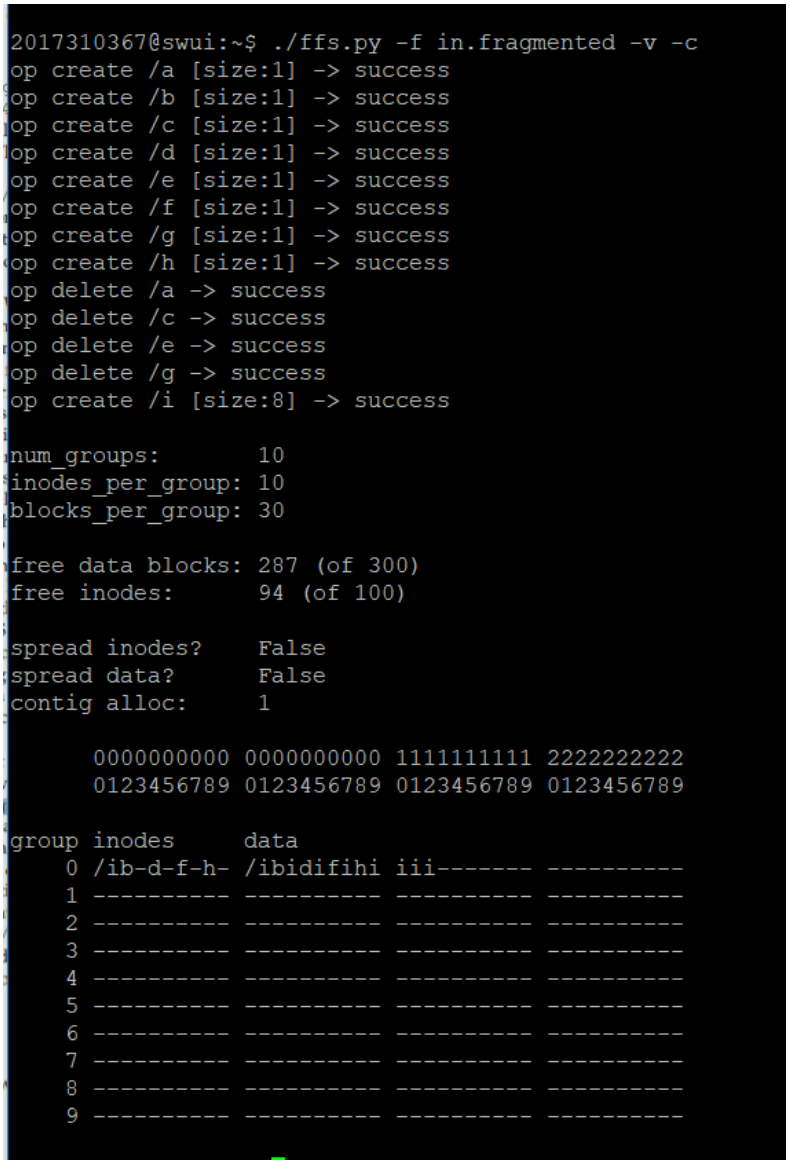
그룹의 아이노드 테이블의 개수를 5개로 제한하였다. 그 결과 파일들이 상위 디렉토리와 같은 그룹에 저장되는 것이 아니라 흩어져 저장되었으며 dirspan 또한 각각 371, 332, 371로 굉장히 좋지 않은 성능을 보이고 있음을 확인하였다.

7. Which group should FFS place inode of a new directory in? The default (simulator) policy looks for the group with the most free inodes. A different policy looks for a set of groups with the most free inodes. For example, if you run with -A 2, when allocating a new directory, the simulator will look at groups in pairs and pick the best pair for the allocation. Run ./ffs.py -f in.manyfiles -I 5 -A 2 -c to see how allocation changes with this strategy. How does it affect dirspan? Why might this policy be good?

```
group inodes     data
    0 /ejmyr---- /eejmyyyr- ---------- ----------
    1 ---------- ---------- ---------- ----------
    2 aftwpB---- aafftwwwpB BB-------- ----------
    3 ---------- ---------- ---------- ----------
    4 bgunzC---- bbgguuunzz zCCC------ ----------
    5 ---------- ---------- ---------- ----------
    6 chlxq----- cchhlxxxq- ---------- ----------
    7 ---------- ---------- ---------- ----------
    8 divoA----- ddiivvvoAA A--------- ----------
    9 ---------- ---------- ---------- ----------

span: files
  file:       /t/A  filespan:  16
  file:       /t/B  filespan:  16
  file:       /j/n  filespan:  14
  file:       /j/o  filespan:  14
  file:       /j/l  filespan:  12
  file:         /h  filespan:  12
  file:         /g  filespan:  12
  file:         /f  filespan:  12
  file:         /e  filespan:  11
  file:         /d  filespan:  11
  file:         /c  filespan:  11
  file:         /b  filespan:  11
  file:         /a  filespan:  11
  file:       /t/x  filespan:  14
  file:       /t/y  filespan:  13
  file:       /t/z  filespan:  16
  file:       /j/C  filespan:  18
  file:       /j/r  filespan:  13
  file:       /j/p  filespan:  14
  file:       /t/u  filespan:  14
  file:       /t/v  filespan:  14
  file:       /t/w  filespan:  14
  file:       /j/q  filespan:  14
  file:         /i  filespan:  12
  file:       /j/m  filespan:  11
              avg  filespan:  13.20

span: directories
  dir:          /  dirspan: 333
  dir:         /t  dirspan: 336
  dir:         /j  dirspan: 335
              avg  dirspan: 334.67
```

-A 2 플래그를 주면 시뮬레이터는 디렉토리를 할당할 때 두 개의 그룹 씩 짝짓는다. 그 후 가장 연관있는 페어 그룹에 디렉토리를 할당한다. 연관되어 있는 디렉토리를 해당 그룹에 넣기 수월해지기 때문에 더 효율적이다. 실제로 dirspan을 본 결과 333, 3356, 335로 그냥 ffs보다 조금 더 효율적으로 파일을 저장한 것을 확인할 수 있다.

8. One last policy change we will explore relates to file fragmentation. Run ./ffs.py -f in.fragmented -v and see if you can predict how the files that remain are allocated. Run with -c to confirm your answer. What is interesting about the data layout of file /i? Why is it problematic?



```
2017310367@swui:~$ ./ffs.py -f in.fragmented -v -c
op create /a [size:1] -> success
op create /b [size:1] -> success
op create /c [size:1] -> success
op create /d [size:1] -> success
op create /e [size:1] -> success
op create /f [size:1] -> success
op create /g [size:1] -> success
op create /h [size:1] -> success
op delete /a -> success
op delete /c -> success
op delete /e -> success
op delete /g -> success
op create /i [size:8] -> success

num_groups:       10
inodes_per_group: 10
blocks_per_group: 30

free data blocks: 287 (of 300)
free inodes:      94 (of 100)

spread inodes?    False
spread data?      False
contig alloc:     1

      0000000000 0000000000 1111111111 2222222222
      0123456789 0123456789 0123456789 0123456789

group inodes     data
    0 /ib-d-f-h- /ibidifihi iii------- ----------
    1 ---------- ---------- ---------- ----------
    2 ---------- ---------- ---------- ----------
    3 ---------- ---------- ---------- ----------
    4 ---------- ---------- ---------- ----------
    5 ---------- ---------- ---------- ----------
    6 ---------- ---------- ---------- ----------
    7 ---------- ---------- ---------- ----------
    8 ---------- ---------- ---------- ----------
    9 ---------- ---------- ---------- ----------
```

Fragment란 block보다 작은 단위인 fragment를 도입하여 fragment단위로 할당 받을 수 있게 하는 것이다. I를 보면 다른 bdfh에 비해 큰 파일이기 때문에 나누어져 할당받았다. 그러면 연관된 디렉토리인 ibdfh를 모두 같은 그룹에 저장할 수 있다.

9. A new policy, which we call contiguous allocation (-C), tries to ensure that each file is allocated contiguously. Specifically, with -C n, the file system tries to ensure that n contiguous blocks are free within a group before allocating a block. Run ./ffs.py -f in.fragmented -v -C 2 -c to see the difference. How does layout change as the parameter passed to -C increases? Finally, how does -C affect filespan and dirspan?