디스크(Disk)

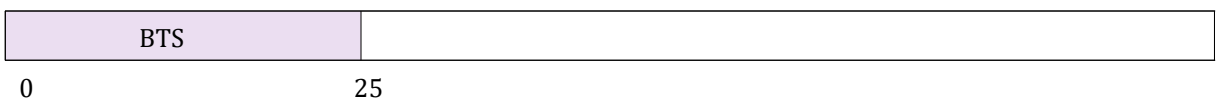
[문제] 용량이 V인 일반적인 디스크에 파일을 저장, 관리한다. 즉 디스크는 메모리가 순차적으로 연결된 공간이다. 이 디스크에 적용할 수 있는 작업(operation)은 다음과 같다.

작업	명령어 양식	의미
파일 쓰기	**write fa sa**	파일 fa (크기는 sa)를 적절한 곳에 아래 설명한 알고리즘으로 저장(write)한다.
파일 제거	**delete fa**	파일 fa를 disk에서 삭제하고 그 공간을 free로 만들어 둔다.
파일 출력	**show fa**	파일 fa가 디스크에 저장된 정보 출력. fa가 disk에 분산되어 기록된 정보(시작,끝)를 순서대로 출력
압축정리	**compact**	파일을 옮겨 disk의 중간 빈(free) 공간을 정리한다. 단 원래 순서는 그대로 유지된다. 빈 공간만 삭제된다.

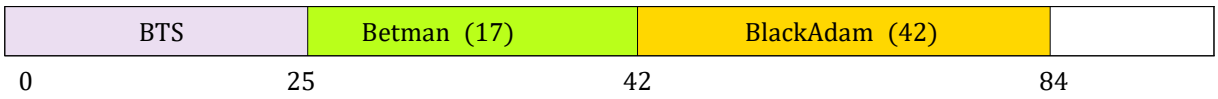
파일 쓰기 작업의 규칙은 다음과 같다. 만일 disk의 남아있는 빈(free) 공간의 합이 sa보다 작으면 write는 거부되고 시스템(OS)는 "**diskfull**"을 출력한다. 만일 이 경우가 아니라면 fa를 하나의 덩어리로 저장할 수 있는 가장 앞쪽(주소가 빠른) 공간에 저장한다. 즉 가능하면 파일을 쪼개지 않은 상태를 유지한다. 만일 남아있는 disk 짜투리 공간의 합은 sa보다 크지만 fa를 한 덩어리로 넣을 수 있는 "통" 공간이 없을 경우에는 잘라 넣을 수 밖에 없다. 이 경우에는 앞에서부터 남아있는 공간에 파일을 잘라서 순서대로 채워 넣는다. 즉 이 경우에는 파일이 분할 기록된다.

파일 제거(delete file)는 해당 파일을 disk에서 삭제하고 그 공간은 다른 파일의 쓰기가 가능한 free 공간으로 되돌리는 작업이다. 그리고 파일 출력(show file) 명령어는 해당 파일 file의 시작 위치 주소를 출력한다. 만일 fa가 하나 이상의 공간에 조각으로 나누어져 있는 상태라면 각 조각의 모든 시작 위치를 순서대로 출력해야 한다. **응축(compact)**은 현재 disk의 중간 중간에 비어있는 공간이 없도록 파일 조각을 앞쪽으로 옮겨서 전체를 하나의 연속된 파일로 만드는 작업이다. 즉 disk 내의 모든 파일(조각난 파일까지 포함) 사이에 빈 공간은 없어진다. V=100인 예제로 이 과정을 살펴보자.

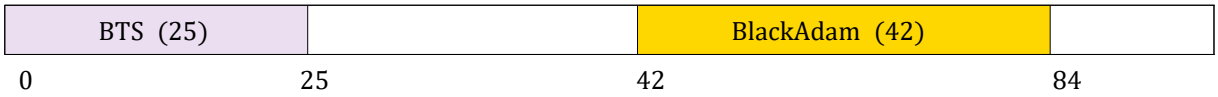
초기 상태에서 **"write BTS 25"** 명령은 파일 이름이 "BTS"이며 크기가 25인 파일을 디스크 공간은 쓰는 것이다. 초기 disk는 모두 비어있기 때문에 이 파일은 제일 앞, 0번지에 배치된다.



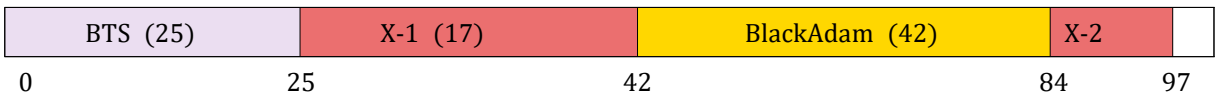
다시 이 상황에서 **"write Betman 17", "write BlackAdam 42"**를 수행한 후의 disk 모습을 보자. 즉 Betman은 다시 [25,42]에 기록되고, BlackAdam은 다시 Betman 뒤에 순차적으로 기록되어 그 최종 dsik map의 모습은 다음과 같아진다.



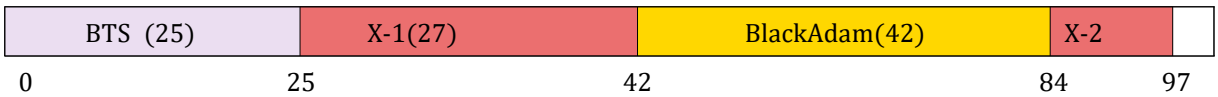
이 상황에서 **"show Betman"**을 수행하면 그 파일의 첫 위치 주소인 "25"를 출력해야 한다. 이 상황에서 **"delete Betman"**으로 삭제하면 disk map은 다음과 같아진다.

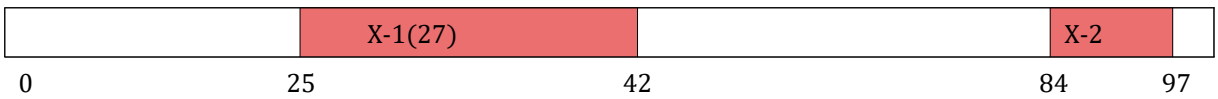


이 상황에서 **"write Night 48"**을 수행하면 현재 남아있는 공간이 100-(25+42)=33이므로 크기가 44인 Night 파일의 저장은 불가능하다. 따라서 여러분은 "**diskfull**"을 출력한다. 그리고 이후 **"write X 30"**을 수행하면 이 파일을 다음과 같이 2개 {X-1, X-2}로 쪼개져서 저장된다.

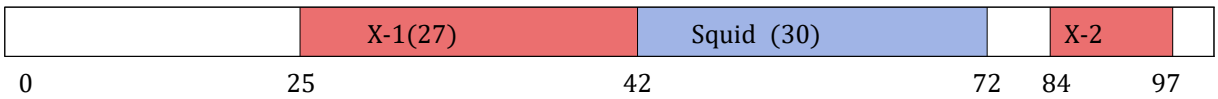


여기에서 **"show X"**를 수행하면 이 쪼개진 파일의 첫 주소인 **25, 84**를 출력해야 한다. 여기에서 BTS와 BlackAdam을 순차적으로 삭제하면 다음과 같이 변한다.

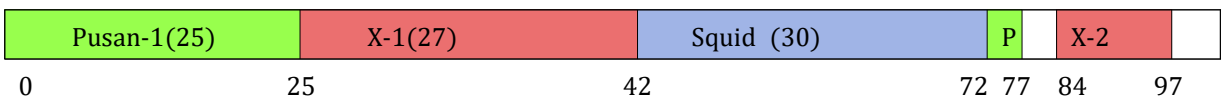




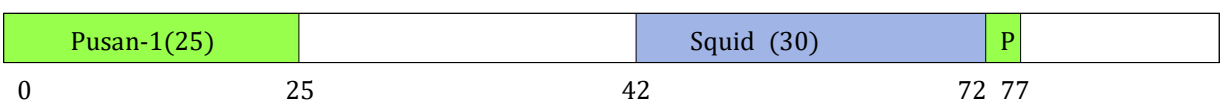
이 상황에서 **"write Squid 30"**을 수행하면 이 파일 Squid는 아래와 같이 통으로 들어갈 수 있는 공간 [42,84]가 존재하므로 아래와 같이 저장된다.



이 상황에서 "**"write Pusan 30"**를 수행해보자. 이제 크기 30인 파일을 쓸 수 있는 "통"공간은 없으므로 이제 잘라서 넣을 수 밖에 없다. 이 파일을 구간 [0,25]와 [72,77]에 저장된다.



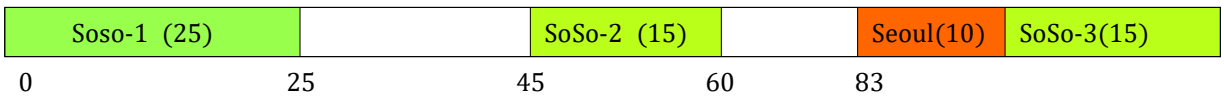
"delete X"을 수행하면 다음과 같이 변한다.



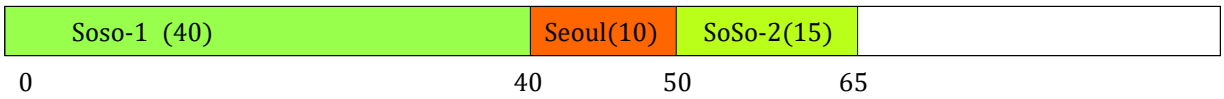
이 상황에서 **"compact"**를 수행하면 중간 빈 공간 [25,42]이 정리, 축약되어 다음과 같이 된다.

여기에서 다시 **"write Jazz 23"**를 수행하면 다음과 같이 변화한다.

만일 compact후 이전에 쪼개진 파일이 연속된 상태로 붙게되면 이것은 하나의 덩어리로 취급됩니다. 예를 들어 아래 상황에서 compact를 수행하면



이렇게 정리되고 이 상황에서 "show Soso"를 수행하면 그 답은 '0 50'이 된다.



만일 이 상황에서 Soso-1과 Soso-2가 합쳐지면 '0'만 출력한다.

[입출력] 입력 파일 **stdin** 명의 첫 줄에 disk의 용량 V가 정수로 제시된다. 단 $10 \leq V \leq 100{,}000$ 이다. 디스크에 적용되는 명령어는 이 파일의 각 줄에 하나씩 제시된다. 여러분은 제시된 명령어에 대한 출력 결과를 순차적으로 **stdout**에 출력해야 한다. 단 없는 파일을 지우거나, 동일한 이름의 파일을 다시 쓰는(write) 경우, 또 없는 파일을 show할 경우에는 "**error**"를 출력해야 한다. 입력 명령어의 끝은 문자열 **"end"**로 표시된다. 입력 **stdin**에 표시된 명령어의 수는 최소 5개, 최대 100개이다.

[예제]

stdin	**stdout**
100 // V=100 write BTS 25 write Betman 17 write BlackAdam 42 show Betman delete Betman write Night 48 delete Box //error write X 30 show X delete BTS delete BlackAdam write Squid 30 show Squid write Pusan 30 delete X compact show Pusan write Jazz 23 show X // error show Jazz **end** // end of input	25 //show Betman diskfull //Night 48 **error** // delete Box 25 84 //show X 42 //show Squid 0 55 // show Pusan **error** // show X 60 //show Jazz

[제한조건] 프로그램의 이름은 **`disk.{c,cpp,java,py}`**이다. 수행 속도는 각 데이터별로 최대 1초이다. 제출 횟수는 25회이며 최대 사용 token은 800이다. V 크기의 array나 vector를 사용하면 time-out이 될 수 있으므로 양방향 연결 리스트인 **`list< FILE_TYPE >`**를 구성해서 사용하기를 권한다. 즉 각 파일의 연속된 덩어리(chunk)와 빈 공간(free space)를 list node로 관리하면 간결하게 해결할 수 있다. 단 동시에 FILE의 위치를 위한 FILE table도 가지고 있어야 할 것이다. FILE의 개수는 최대 100개이다. 만점자 중에서 **가장 우아한 코드 작성자 5인에게 +5%의 가점이 있다.**