단일 dx\_root block이 500 개 이상의 htree block을 인덱싱 할 수 있고 해당 block에 수백 개의 파일 이름 항목이 포함될 수 있다는 점을 감안하면 htree가 단일 레벨 이상을 필요로 하는 경우는 거의 없습니다. 실제로, offset 30의 "tree 깊이"byte(offset 30)는 항상 0x00이며 flat tree를 나타냅니다. 이는 중첩 tree를 허용하지만 실제 응용 프로그램에서는 본 적이 없습니다.

byte 34-35는 따라야 할 실제 dx\_entry 레코드 수이며 36-39 byte의 "zero hash"레코드를 dx\_entry 레코드 중 하나로 계산합니다. 각 dx\_entry 레코드는 4 byte hash 값이다. 각각의 hash값은 디렉토리 파일 시작에서 4byte relative block offset으로 표시됩니다. 명확성을 위해이 예제의 처음 몇 개의 dx\_entry 레코드를 표 형식으로 살펴보겠습니다.



dx\_entry 레코드는 hash 값으로 정렬된 조회 테이블입니다. 초기 "zero hash"항목은 이름이 0x0F3FFEA2보다 작은 값으로 hash되는 모든 파일을 디렉토리 파일의 block 번호 1에서 찾을 수 있음을 의미합니다. hash 값이 0x0F3FFEA2 이상이지만 0x1D8171F2보다 작은 파일 이름은 파일의 16 번 block에서 찾을 수 있습니다. dx\_entry 레코드는 hash 값을 기준으로 정렬되므로 EXT 파일 시스템 코드가 이진 검색을 수행하여 적절한 block offset을 더 빨리 찾을 수 있습니다.

마지막 dx\_entry 레코드 이후 나머지 block은 여유 공간에 있습니다. 일반적으로이 여유 공간에는 디렉토리가 단일 block에 들어가기에 충분히 작은 디렉토리 항목이 포함됩니다. 예를 들어, 마지막 dx\_entry 레코드가 종료된 직후, "alsa-utils"라는 서브 디렉토리 (파일 유형 0x02)에 대한 대부분의 원래 항목을 볼 수 있습니다. 그런 다음 inode 0x000C14D7 (791767)에 "anacron"이라는 다른 하위 디렉토리에 대한 항목이 있습니다. 일반적으로 hash 된 htree block에서이 디렉토리에 대한 라이브 항목을 찾을 수 있습니다. 그러나 이건 그 3개의 디렉토리가 삭제될 수 있고, 이 디렉토리 여유에 관한 항목들이 그 자료들이 존재했던걸 나타내는 유일한 기록이다.

디렉토리 파일의 나머지 부분은 htree의 "리프 block"입니다. 이 block은 일반 디렉토리 항목으로 가득 차 있으며 단순히 순차적으로 읽습니다. htree 형식은 이전 버전과의 호환성을 위해 특별히 설계되었으므로 이전 코드는 여전히 디렉토리 항목을 통해 정상적인 순차적 검색을 수행할 수 있습니다.

삭제된 디렉토리 파일을 carving 할 생각이라면 한 block보다 큰 디렉토리 파일은 일반적으로 조각된다는 점에 유의하십시오. EXT block 할당 알고리즘은 파일을 상위 디렉토리와 동일한 block 그룹에 배치하는 것을 선호합니다. 디렉토리가 단일 block보다 커질 때까지 근처 block이 모두 소비되었습니다.