알고리즘(해밍코드) 설명

문자 한개 8비트에 오류검출과 복구를 위한 패리티 비트 4비트를 더해 총 12비트로 구성된 한 단위로 한 문자를 저장하는 해밍코드 방식을 채택하였습니다. 4개의 패리티 비트는 각 2^0, 2^1, 2^2, 2^3자리에 위치하여 각 패리티 비트가 담당하는 비트가 1이되는 비트 수에 짝수를 맞춰주는 역할을 합니다. 이를 짝수 패리티 기법이라고 합니다. 만약, 디코딩 시 짝수 패리티를 만족한다면 신드롬 단어로 0 0 0 0 이 나오고, 만족하지 않는다면 신드롬 단어가 다른 단어가 출력이 되고 그 해당하는 10진수가 오류가 발생한 비트임을 알 수 있고 해당하는 비트 값을 반전 시키면 원본 값으로 복원이 가능합니다.

그런데 이 해밍코드 알고리즘에는 단점이 있습니다. 2비트 이상의 오류는 검출할 수 없습니다. 변조 프로그램을 사용하면, 1개의 변조는 1바이트를 수정합니다. 즉, 패리티비트를 가진 12비트의 정보 중 8비트가 수정될 수 있고, 이렇게 되면, 오류를 수정할 수 없습니다. 그래서 우리는 이 12비트를 저장하는 방식에 변형을 주었습니다.

12비트의 이진수 정보를 문자열 형태로 저장하고, 이 문자열을 16개 저장하는 char data[16][12]의 배열을 만들어 저장합니다. 그리고 문자열의 방향과 수직되는 방향으로 새로운 문자열로 저장을 합니다. 예시로 KMU\_CS23/HONG CH을 저장해 보겠습니다. 안의 이진수는 임의로 넣겠습니다.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **1** | 0 | 0 | 0 | 1 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 1 | **K** |
| **2** | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 1 | 1 | 0 | 1 | 0 | **M** |
| **3** | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 0 | 1 | 1 | 1 | 0 | **U** |
| **4** | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 1 | 0 | 1 | 1 | **\_** |
| **5** | 0 | 1 | 0 | 0 | 1 | 0 | 1 | 1 | 1 | 0 | 0 | 1 | **C** |
| **6** | 0 | 0 | 1 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 1 | **S** |
| **7** | 0 | 1 | 1 | 1 | 0 | 1 | 0 | 1 | 0 | 0 | 1 | 1 | **2** |
| **8** | 0 | 1 | 0 | 1 | 0 | 1 | 1 | 1 | 0 | 0 | 1 | 1 | **3** |
| **9** | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 1 | 0 | 1 | 0 | **/** |
| **10** | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 1 | **H** |
| **11** | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | **O** |
| **12** | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | **N** |
| **13** | 0 | 0 | 1 | 0 | 1 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | **G** |
| **14** | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 1 | 0 | 1 | 0 |  |
| **15** | 0 | 1 | 0 | 1 | 1 | 1 | 1 | 0 | 1 | 0 | 0 | 1 | **C** |
| **16** | 0 | 1 | 1 | 0 | 1 | 1 | 1 | 0 | 0 | 1 | 0 | 1 | **H** |
|  | **0** | **131** | **435** | **311** | **46** | **657** | **3242** | **312** | **534** | **312** | **3242** | **23423** |  |

이 상태에서 첫 번째 문자열에 이진수들이 의미하는 것은 K입니다. 두 번째 문자열에 이진수들이 의미하는 것은 M입니다. 기존에는 K, M을 그대로 저장하는 것이었지만, 여기서 정보를 가로방향이 아닌, 세로 방향으로 읽습니다. 그렇게 되면 16개의 이진수 정보를 가진 데이터 12개가 됩니다. 이 정보들은 2바이트를 저장하는 short[12]에 담고, 바이너리 파일에 저장하게 됩니다.

여기서 이렇게 저장하는 것에 이유는 만약 데이터 변조 프로그램으로 변조를 한다면, 1바이트 단위로 바뀔 것 입니다. 만약 1바이트가 바뀐다면, 위의 표에 파란색으로 칠해진 부분처럼 바뀔 것 입니다. 이 상태에서 디코딩을 할 때, short[12]에서 data[16][12]로 복원하게 되면, 8개의 정보가 각각 1비트씩 바뀐 것이 될 것입니다. 즉, 1비트의 오류를 수정할 수 있는 해밍코드로 데이터를 복구할 수 있습니다. 이렇게 되면, 저희 알고리즘으로 오류 검출과 복구 모두 할 수 있습니다.

하지만 여전히 단점이 있습니다. 표에 파란색과 주황색으로 칠해진 부분이 변조된다면, 해밍코드로 데이터 복구를 할 수 없습니다. 이 확률을 계산해 보면, 인코딩 파일이 480바이트 이면, 240개의 short데이터가 담기고, 같은 12bit에 2개이상의 오류가 검출 되는 경우여야 함으로, 240/12=20, 즉, 1x1/20x1/2(20개 중 1개 x 20개 중 앞의 것과 같을 확률 x 2바이트 중 같은 위치의 바이트일 확률)=1/40입니다. 약 2퍼센트의 확률로 실패합니다. 하지만, 바이트 수와 복구성공률, 구현 난이도, 시간 등을 고려했을 때, 적합하다고 생각되어 이 알고리즘을 채택하였습니다.