**확률과 통계**

**Assignment #3**

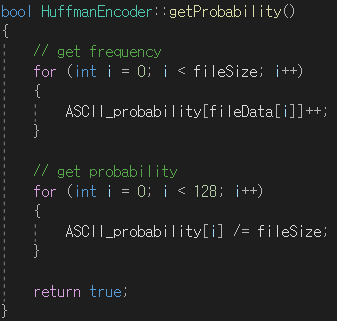
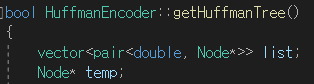
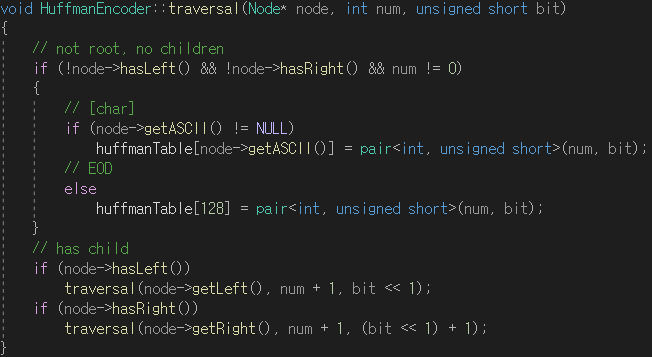
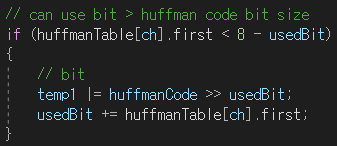
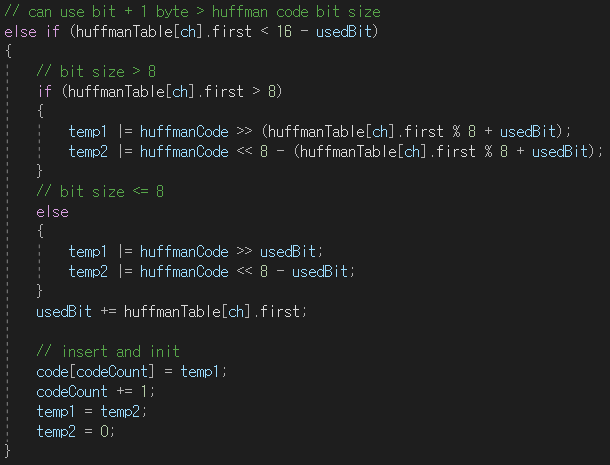
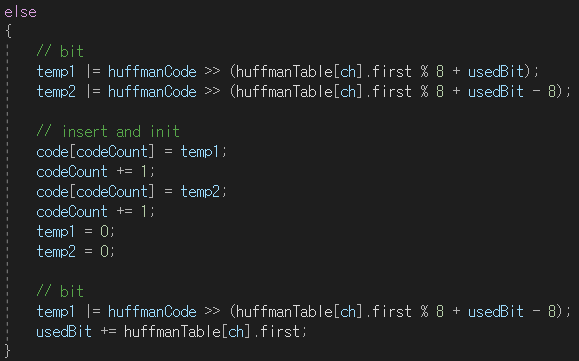
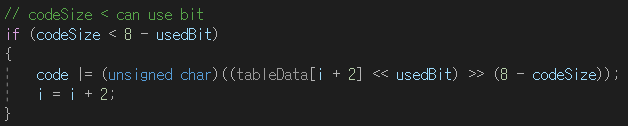
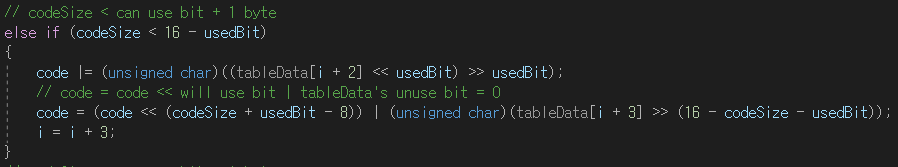
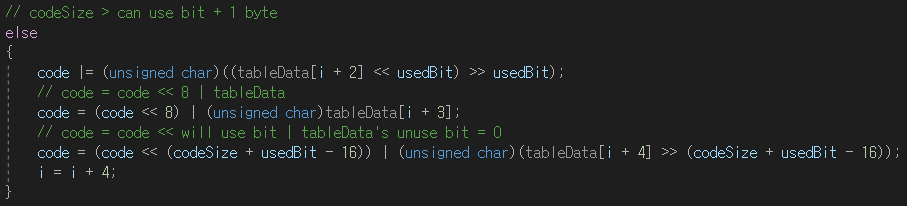
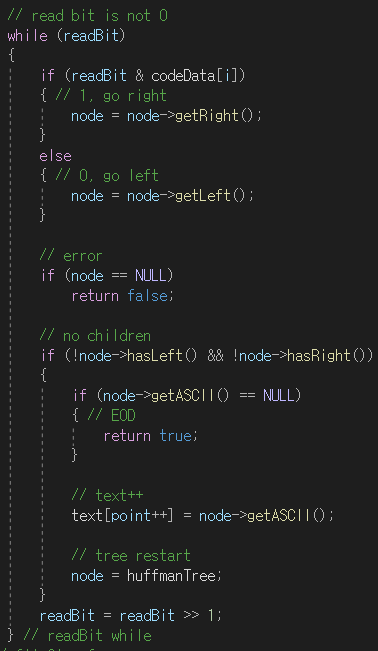
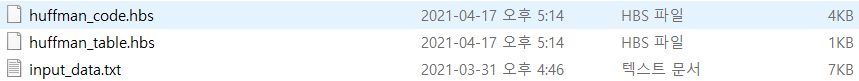
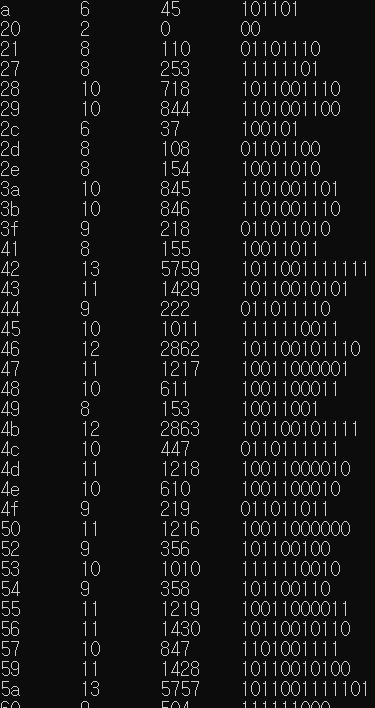
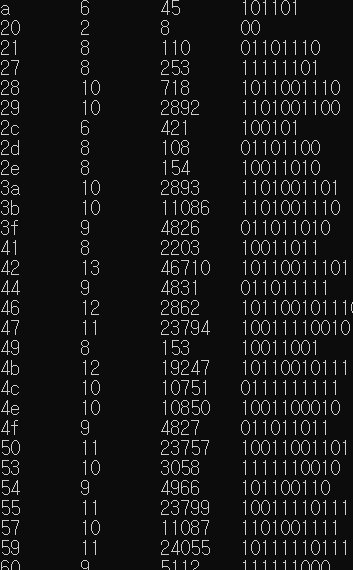
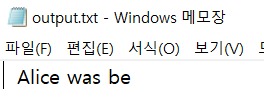
**담당교수 : 심동규**

**제출일 : 2021. 04. 25.**

**학부 : 컴퓨터정보공학부**

**학번 : 2017202088**

**이름 : 신해담**

1. **과제 설명**
   * 입력 text 파일을 Huffman code와 table 파일로 출력하는 encoder와, huffman code와 table으로 출력 text파일을 생성하는 decoder를 구현한다.
2. **접근 방법** 
   * Encoder의 경우 다음과 같이 접근했다.
     + 우선 Text 파일을 읽어서 각 ascii 문자의 확률을 구한다.  
       
     + 위에서 구한 확률을 정렬하고, 낮은 확률끼리 묶어가면서 tree를 구현한다.  
         
       
     + Huffman table을 구현한다. Tree를 순회하면서 각 종단점에 도달할 때마다 table에 추가한다.  
       
     + Huffman table을 이용하여 text 파일의 내용을 Huffman code로 변환한다. 이 때, 1byte 단위로 변환하기 위해 3가지 케이스로 나누었다.  
       1byte 중 사용 가능한 bit의 수가 code의 길이보다 큰 경우에는 temp1에 code를 써넣고 계속 진행했다.  
         
       code 길이가 사용 가능한 bit수보다 크고, 사용 가능 bit 수 + 1byte 보다 작으면 temp1을 가득 채우고 temp2에 남은 bit를 작성한 다음 temp1을 결과에 추가했다.  
         
       code 길이가 사용 가능 bit 수 + 1byte 보다 크면 temp1과 temp2를 채우고 두 값을 모두 결과에 쓴 다음, 남은 bits를 temp1에 쓰고 계속 진행시켰다.  
       
   * Decoder의 경우 다음과 같이 시도했다.
     + Huffman table 파일을 읽어서 Huffman table을 얻는다. Encoder에서와 비슷하게 3가지 케이스로 나누었다.  
       먼저 사용할 수 있는 bit가 code size보다 크면 code부분만 읽어와서 table을 만들었다.  
         
       code size가 사용 가능 bit보다 크고 사용 가능 bit + 1byte보다 작으면, i+2의 남은 전부와 i+3의 필요한만큼 bit를 읽었다.  
         
       code size가 사용 가능 bit + 1byte보다 크면, I + 2의 남은 전부와 i+3 전부, i+4의 필요한 부분만큼 bit를 읽었다.  
       
     + 그렇게 해서 얻은 table로 tree를 구현한다. Code의 MSB에서 LSB까지 읽으면서 tree를 따라가고, 통과할 tree부분이 비어있으면 생성한다.  
       
     + Tree를 이용하여 huffman code에 맞는 ascii code를 찾아서 원본 입력 파일을 복호화한다.  
       
3. **실험 및 검증 결과**
   * Encoding 결과 huffman\_code.hbs 파일과 Huffman\_table.hbs 파일이 만들어졌다. Input\_data.txt는 7KB이므로 생성한 code는 약 1.75배 줄어들었다.  
     
   * Table이 제대로 만들어졌는지 확인하기 위해 printHuffmanTable 함수를 구현했다. 왼쪽이 encoder의 table이고, 오른쪽이 decoder가 재구성한 table이다.  
        
     두 곳의 차이가 어느 부분 때문에 발생했는지 huffman\_code.hps를 잘 찾아보니 0x42와 0x43 사이의 bit pattern이 예상과 달랐다. 따라서 Encoder의 storeHuffmanTable 함수 부분에서 문제가 있는 것으로 추정된다. storeHuffmanTable을 기반으로 구현한 storeHuffmanCode 또한 올바른 출력이 아닐 것이다.
   * Decoding한 output.txt. 디버깅 과정에서 tree 형성이 비정상적임을 확인했다.  
     
4. **고찰**
   * Bit 단위의 데이터들을 출력하는 데에 상당한 어려움을 겪었다. Shift를 사용해 쉽게 될 줄 알았는데, bit누락이나 증발현상이 발생하여 구간별로 나누어 처리하려 했다. 이번 케이스에서는 2byte를 넘기지 않아 3구간으로 나누었지만 Huffman code의 bit 수가 2byte를 넘기면 오류가 발생할 것으로 예상된다.
   * Decoder가 tree를 참조하여 decoding하는 과정에서, tree를 탐색하던 node가 NULL이 되는 error가 발생했다. 디버깅해보니 원래라면 leaf가 되어야 할 node가 child를 가지고 있었다. Huffman code를 MSB부분에서 따라 내려가면서 구현한 tree라 어느 곳을 잘못 구현했는지 찾지 못했다.