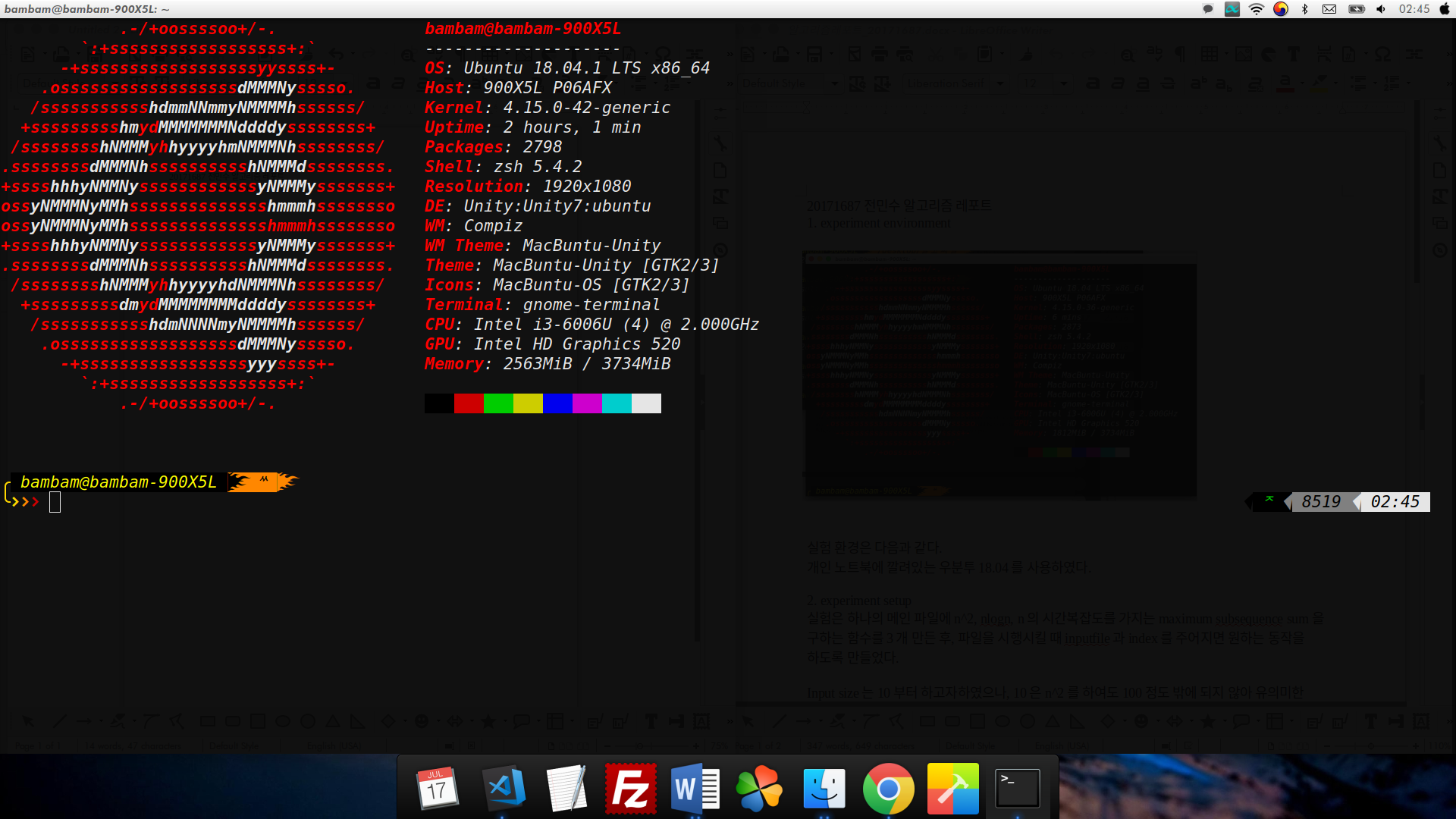
20171687 전민수 알고리즘 레포트

1. experiment envrionment



실험 환경은 다음과같다.

개인 노트북에깔려 있는우분투 18.04를 사용하였다.

2. experiment setup

최대한 다양한파일을 압축해보고자 노력하였다.

압축한 파일은 다음과 같다.

1) .txt file

- 최악의 케이스

char 값 기준 -128~127까지 바이트에서나올수 있는모든경우의 수를 가지고 있는 케이스

- 일반적인 케이스

제 생각에 이번 실험을 돌린 source코드가 꽤나 좋은 테스트 케이스가 될 것이라 생각했습니다.

- 숫자만 있는 file

sorting 과제를 할때 사용하였던 파일을 사용하였습니다. 숫자와 공백문자만이 포함되어있기 때문에, 압축이잘 되는 케이스일거라 판단했습니다.

2) .mp3 file

컴퓨터에 있는 파일 아무거나하나 돌려봤습니다.

파일의 이름은 Madagascar 2 - I Like To Move It.mp3으로, youtube에서 오디오파일을 다운로드받 았습니다.

3)mp3\_20171687 : 바이너리 파일

4) .exe 파일

3) experiment

1. .txt file

|  |  |  |  |
| --- | --- | --- | --- |
|  | 크기 | 압축 크기 | 압축률 |
| 최악의 케이스 | 1000000 byte(1MB) | 1000321 byte | 1.000321 |
| 일반적인 케이스 | 6346 byte | 4238 byte | 0.6678 |
| 숫자로 되어 있는 케이스 | 10.5 MB (10,481,596 bytes) | 4.7 MB (4,710,235 bytes) | 0.44938 |

2. .mp3 file

|  |  |  |  |
| --- | --- | --- | --- |
|  | 크기 | 압축 크기 | 압축률 |
| 마다가스카 | 5.3 MB (5,322,848 bytes) | 5.3 MB (5,295,794 bytes) | 0.994917 |

3. 바이너리 파일

|  |  |  |  |
| --- | --- | --- | --- |
|  | 크기 | 압축 크기 | 압축률 |
| mp3\_20171687 | 63.3 kB (63,320 bytes) | 43.6 kB (43,552 bytes) | 0.68780796 |

4. .exe 파일

|  |  |  |  |
| --- | --- | --- | --- |
|  | 크기 | 압축 크기 | 압축률 |
| a (2).exe | 38.9 MB (38,854,527 bytes) | 38.8 MB (38,827,610 bytes) | 0.99307236 |

5. .PNG 파일

|  |  |  |  |
| --- | --- | --- | --- |
|  | 크기 | 압축 크기 | 압축률 |
| Asd.png | 327.9 kB (327,932 bytes) | 328.1 kB (328,066 bytes) | 1.000408621 |

4) data structure & algorithm

data structure는 공부하는 ppt에 적혀있는 node structure를 그대로 따왔다.

typedef struct \_node {

char symbol;

int freg;

struct \_node\* left;

struct \_node\* right;

}node;

이진트리를 구성하는 왼쪽, 오른쪽, symbol이 나온 횟수, 심볼이 저장되어있다.

ppt와 다른점은 queue 헤더파일에 있는 priority queue를 min heap으로 사용하기 위해 다음과 같은 operaotor를 따로 정의해 줬다는 점이다.

bool operator()(node\* t, node\* u) {

return t->freg > u->freg;

}

};

이를 통해 priority queue는 다음과 같이 사용하였다.

priority\_queue<node\*,vector<node\*>,cmp> pq;

pq를 사용한 이유는 허프만 코드를 사용하기 위해서 필요하기 떄문이다.

압축을 하기 위해 사용한 내 알고리즘은 기본적으로 트리 구성을 허프만 알고리즘을 따른다.

즉, pq에 저장되어 있는 노드들은 frequency를 기준으로 작은 순서대로 저장되어 있는데, 가장 작은 두개를 뽑아 같은 부모를 가지는 노드로 만들어주는 과정을 노드가 한개가 될때까지 만들어주었다.

이를 통해, 허프만 트리를 얻을 수 있다.

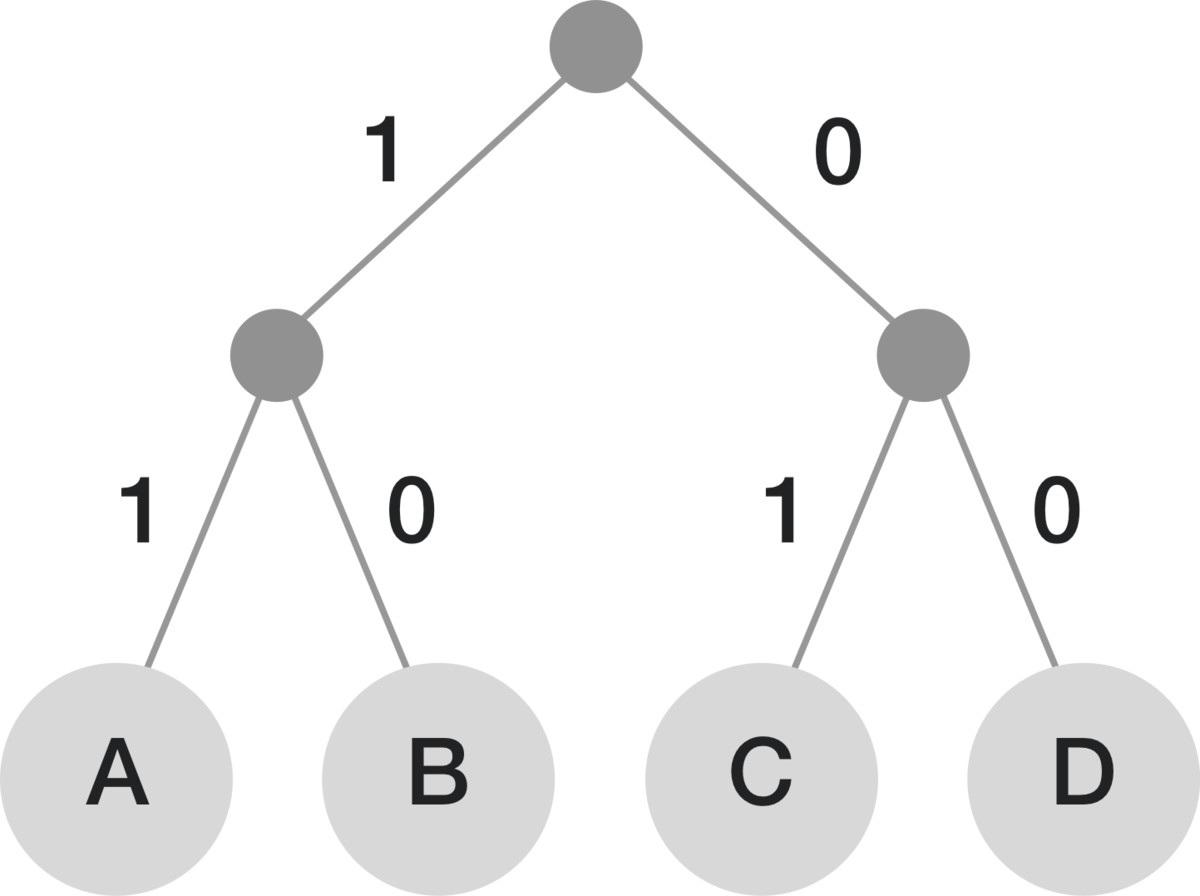
내 생각에 압축률에서 차이를 보일 수 있는 부분은

1)트리를 압축 파일에 어떻게 가공을 해서 넣을것인지

2)그 후 각각의 문자가 나타내는 비트들을 어떻게 가공해서 집어 넣을것인지

라고 생각하였다.

1)트리를 압축 파일에 가공해서 넣은 방법은 다음과 같다.



다음과 같은 허프만 트리가 있다고 하자. 나는 이 트리를 dfs를 돌며

“현재 트리의 root에서 leaf가 아니면 0, leaf면 1이라 놓는다. 만약 이번에 놓는 값이 1이라면, 그 뒤에 현재 노드의 symbol을 적어둔다.”

라는 규칙을 가지고 넣는다. 위의 트리는 다음과 같이 만들어질 것이다.

001A1B01C1D

다음과 같이 적힌 것은 트리는 만들 때 무조건 이진트리로 만들어졌기 떄문에, 복구를 하기 위해서는

입력값을 읽으며 다음과 같은 역할을 하는 재귀함수를 돌리면 된다.

만약 이번 입력값이 0이라면 새로운 노드를 만들고 재귀함수를 두번 돌린 후, 첫번째 재귀함수의 return 값을 그 노드의 left node로, 두번째 것을 right node로 추가한다. 그 후 그 노드를 return 한다.

만약 1이라면 새로운 노드를 만들고 추가적으로 8byte를 읽어들여 읽은 값을 sybol로 하고, left 와 right가 모두 NULL으로 만든 후 그 노드를 RETURN한다.

이를 통해 트리를 완벽히 복구할 수 있다.

실제로 돌려보니, 최악의 케이스에서 트리의 바이트가 320BYTE밖에 되지 않는 것을 보고 큰 영향을 주지 않음을 알 수 있었다.

2) 각각의 문자는 다음과 같이 가공하였다.

문자들을 가공한 process는 비슷하기 때문에, 하나의 예시를 들어보겠다.

int tt = 128;

for (int i = 0; i<8; i++) {

int k = t / tt;

bits\_to\_byte += k \* mul;

mul /= 2;

t %= tt;

tt /= 2;

if (mul == 0) {

fprintf(out, "%c", bits\_to\_byte);

mul = 128;

bits\_to\_byte = 0;

}

}

이것은 위의 트리를 만들 때, symbol들을 만드는 방식이다.

t는 현재 받은 symbol 값이다.(0~255)

이것을 관리 할 때,

mul : 현재 몇번째 비트를 가공하고 있는 것인가

(128이 0번 비트, 1이 7번비트이다.)

bits\_to\_byte : 비트들이 어떻게 가공되었는가

이것을 전역변수로 관리하며 현재 비트를 가공시켜준다.

그 후,

mul이 0이 된 순간 == 8번째의 비트를 가공해야하는 순간

지금까지 가공한 비트를 파일에 넣고,mul을 다시 128로 만들고, 가공한 비트를 다시 0으로 만드는 과정을 반복한다.

Decoding 할 때에는 하나의 과정이 추가되는데,

값을 새로 읽어 들어온다.

정확도에서 차이를 보일 수 있는 부분은 마지막 바이트이다.

마지막 바이트는 다 채우지 않고 끝날 수도 있다.

애매하게 끝났을 때, 그것이 몇번째 바이트에서 끝났는지를 적어두지 않으면, 알 방법이 없다.

그렇기 때문에, 파일이 끝이났을 때, compress단계에서 어쩔 수 없이 1 byte를 사용하여 몇번째 비트에서 끝났는지를 적어주었다.

그 후, decoding 하는 과정에서 마지막 바이트를 미리 읽고 마지막에서 두번째 바이트를 어디까지 읽을지를 결정할 수 있다.

내 생각에 압축 프로그램은 압축하고자 하는 프로그램의 용량에 상관 없이 압축을 진행할 수 있어야 한다고 생각하였다. 따라서, 압축하고자 하는 파일을 미리 읽어 배열에 저장하고 그것을 처리하는 방식이 아닌, 필요한 바이트가 생길때마다 처리하는 방식을 사용하였다.

그 결과, 1gb의 파일도 문제 없이 돌아감을 확인할 수 있었다.

5) 실험 결과에 대한 고찰

1. 실험 결과는 파일의 확장자에 따라 다르지 않고, 파일 각각의 바이트가 얼마나 다양한지에 따라 영향을 받는다.

실험 결과 모든 캐릭터들이 들어 있는 .txt파일이 가장 압축률이 낮고, 숫자와 공백문자만이 들어있는 파일이 압축률이 가장 높음을 보면 확인할 수 있다.

2. 그 실험결과는 어찌보면 매우 자명하다. 알고리즘이 몇개의 문자가 나왔는지, 어떤 빈도수로 나왔는지를 체크하고 문자 수가 적으면 각 비트에 대응시킬 비트의 길이 또한 줄어들기 때문이다.

3. 과제 description에서 소개되어 있던 2개의 문자를 만드는 방법은 최악의 케이스에서는 255c2갯수의 리프 노드를 가지는 트리가 구성될것이 자명하게 보이기 때문에, 평균 압축률은 비슷할 것이라 생각한다.

4. 내가 최악의 케이스에서 트리의 크기가 320바이트임을 알 수 있었던 방법은 encoding 해주는 함수 중 트리를 출력하는 부분을 빼고 압축을 시도해 보았다. 압축 파일의 크기가 그냥 파일보다 321바이트가 큰데, 트리의 크기를 빼면 1바이트가 큰 것이다. 그것에 대해 생각을 해 보았더니, 일단 1 바이트는 마지막 바이트가 몇비트까지 사용되었는가를 확인하는 용도이니, 그것을 빼면, 트리와 1바이트를 빼면 파일의 크기가 같은 것이다. Encoding 하기 전, 각각의 캐릭터는 1바이트씩 저장되어 있다.

최악의 케이스이므로, 256개의 서로 다른 캐릭터가 모두 나온 케이스인데, 그러면 완전 이진트리로 만들어졌다고 가정하였을 때, 각각의 비트스트링은 8개 즉 1바이트의 크기를 가진다.

1바이트에 대응되는 비트스트링이 1바이트이기 때문에 파일의 크기가 같게 나왔다는 것이라는 결론을 낼 수 있다. 또한, 허프만 코드를 사용하였을 때 어차피 나올 수 있는 바이트는 255가지이므로, 압축률은 트리 크기를 고려하여도 원본 파일과 매우 비슷한 크기를 가지거나 더 작다는 결론을 낼 수 있다.