L-Università ta' Malta
Faculty of Information & Communication Technology

Assignment Report

Manwel Bugeja

June 2, 2020

Contents

1 Introduction

1.1 Note on the code

This assignment is implemented in c++. Readability was prioritized over efficiency since the code needs to be easily understood by others.

2 Part 1

2.1 How the problem was tackled

2.1.1 Structures

First off, the needed structures where created: 'literal_e' in an enumerated type containing the possible variables along with their negation. An 'inv' literal was also created whose use is for error catching purposes.

Then a clause was defined as a vector of literals. Similarly, a formula was defined as a vector of clauses.

Apart from those, 'expression_t' was also defined as an array of alphabet. Alphabet being an enumeration containing the possible alphabet characters received as input.

These structures are defined in 'formula.h'.

2.1.2 Parsing

In the parsing, the string is first converted to an 'expression_t'. Then it is translated to a formula. Since all variables are only a character long, the commas can be ignored completly when the expression is inputted. For example "(wx), (!w)" will still be parsed successfully. This will not reslut in an error as the input can still be successfully parse. Still, inserting "(," will still cause the program to exit since after an open parethesis, the parser expects a literal (variable or negation followed by a variable).

2.2 The DPLL algorithm

The DPLL algorithm was implemented according to the pseudo code listed in the course notes. In the 'choose_literal()' part of the algorithm, the first literal from the left is chosen.

2.3 Testing

For the testing of the algorithm, some expressions where tried, with the output compared. Included in this document are examples from the notes.

Listing 1: 1-literal rule test

```
$ ./part_1_binary
Hello, World!
Enter expression
(w), (!w, x)
RECIVED INPUT: (w), (!w, x)
Checking if input is alphabetically correct
Checking if syntax is valid
Printing formula
w
!w x
Running algorithm
Starting DPLL
w
!w x
Removing trivially sat
w
!w x
Applying one lit rule
Applying pure lit rule
SAT
```

Listing 2: pure literal rule test

```
$ ./part_1_binary
Hello, World!
Enter expression
(w, x), (w, y), (!x, !y)
RECIVED INPUT: (w, x), (w, y), (!x, !y)
Checking if input is alphabetically correct
Checking if syntax is valid
Printing formula
w x
w y
!x !y
Running algorithm
Starting DPLL
w x
w y
!x !y
Removing trivially sat
w x
w y
!x !y
```

```
Applying one lit rule
w x
w y
!x !y
Applying pure lit rule
!x !y
Starting DPLL
!x !y
!x
Removing trivially sat
!x !y
!x
Applying one lit rule
!x !y
Applying pure lit rule
SAT
```

Listing 3: DPLL recursion test

```
$ ./part_1_binary
Hello, World!
Enter expression
(x, y, z), (x, !y), (y, !z), (z, !x), (!x, !y, !z)
RECIVED INPUT: (x, y, z), (x, !y), (y, !z), (z, !x), (!x, !y,
    !z)
Checking if input is alphabetically correct
Checking if syntax is valid
Printing formula
x y z
x !y
y !z
z !x
!x !y !z
Running algorithm
Starting DPLL
x y z
x !y
y !z
z !x
!x !y !z
Removing trivially sat
x y z
x !y
y !z
z !x
!x !y !z
Applying one lit rule
x y z
x !y
```

```
y !z
z !x
!x !y !z
Applying pure lit rule
x y z
x !y
y !z
z !x
!x !y !z
Starting DPLL
x y z
x !y
y !z
z !x
!x !y !z
x
Removing trivially sat
x y z
x !y
y !z
z !x
!x !y !z
x
Applying one lit rule
x y z

Applying pure lit rule

Starting DPLL
x y z
x !y
y !z
z !x
!x !y !z
!x
Removing trivially sat
x y z
x !y
y !z
z !x
!x !y !z
!x
Applying one lit rule

z !x
!x !y
Applying pure lit rule

UNSAT
```

3 Part 2

3.1 How the problem was tackled

First off, the input is checked for any invalid characters, printing an error if any are found. In the huffman algorithm, the input is iterated and the values are stored in a map. The map contains the character and the corresponding weight.

Following that, the map is translated to a vector of nodes (which are all leaves at this point in time).

3.1.1 Huffman coding

The algorithm is implemented as follows:

- (The following is repeated while the vector is longer then 1 element)
- The vector is sorted greatest first
- The last two elements are merged into a new node with '_' as value and the sum of their weights
- These last two elements are removed from the vector
- The new node is added to the vector

3.2 Testing

For testing, some outputs where tested manually by hand. Online generators where also used to compare the output. Included are a sample output of the program and the tree obtained by an online generator. The output match but the values are flipped.

Listing 4: Program output

```
$ ./part_2_binary input
Hello, World!
Building tree:
Compressing: bbbbbbbbbbbccceeeeeeeeeeeeeeeeeeeeeee
eeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeiiiiiiiiiiiiiiii
iiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiooooooooooooooooo
ooooooooooooooopppppppppppppppppppppp
COLLECTED DATA:
b 12 c 3 e 57 i 51 o 33 p 20
Printing tree:
11 o
```

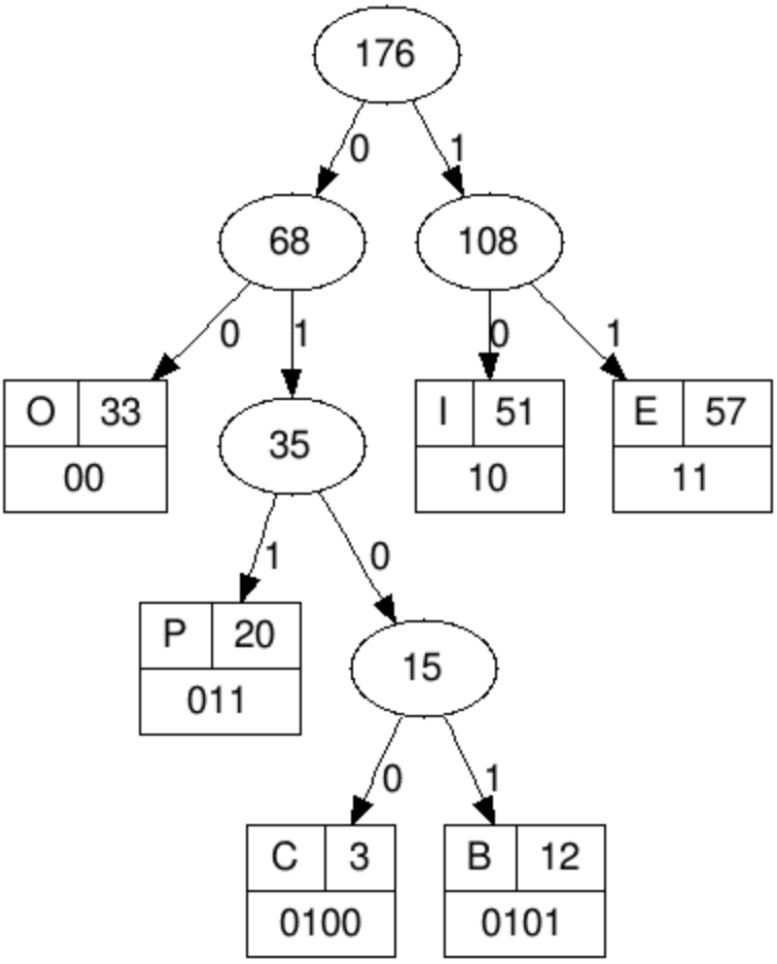

Figure 1: Website used: http://huffman.ooz.ie/

```
1011 c
1010 b
100 p
01 i
00 e
```