Chloe Antonozzi - 1670980

Problem 1

Tree 1: ('A', 4, (('B',2,'C'),3,('D',1,'E')))

a)

```
; Inverse Huffman for ('A', 4, (('B',2,'C'),3,('D',1,'E')))
(declare-const A Int)
(declare-const B Int)
(declare-const C Int)
(declare-const D Int)
(declare-const E Int)

(assert
(and
(= 100 (+ A B C D E)) ; Must equal to 1,0 which is the same as 100 (since we are using integers)
(> A 0) ; Each frequency has to be higher than 0
(> B 0)
(> C 0)
(> D 0)
(> E 0)

; The first cycle, making sure D and E have the smallest frequencies
(< D A)
(< D B)
(< D C)
(< E A)
(< E B)
(< E C)

; The second cycle, making sure B and C have smaller frequencies than A and D + E (combined)
(< B A)
(< B (+ D E))
(< C A)
(< C (+ D E))
))

(check-sat)
(get-model)
```

b)

```
; Inverse Huffman for ('A', 4, (('B',2,'C'),3,('D',1,'E')))
(declare-const A Int)
(declare-const B Int)
(declare-const C Int)
(declare-const D Int)
(declare-const E Int)
(define-const A_len Int 1) ; length of encoding for message ’A’
(define-const B_len Int 3) ; length of encoding for message ’B’
(define-const C_len Int 3) ; length of encoding for message ’C’
(define-const D_len Int 3) ; length of encoding for message ’D’
(define-const E_len Int 3)  ; length of encoding for message ’E’
(declare-const avg_code_len Real)

(assert
(and
(= 100 (+ A B C D E)) ; Must equal to 1,0 which is the same as 100 (since we are using integers)
(> A 0) ; Each frequency has to be higher than 0
```

```
(> B 0)
(> C 0)
(> D 0)
(> E 0)

; The first cycle, making sure D and E have the smallest frequencies
(< D A)
(< D B)
(< D C)
(< E A)
(< E B)
(< E C)

; The second cycle, making sure B and C have smaller frequencies than A and D + E (combined)
(< B A)
(< B (+ D E))
(< C A)
(< C (+ D E))

(= avg_code_len (/ (+ (* A_len A) (* B_len B) (* C_len C) (* D_len D) (* E_len E)) 100))

))

(check-sat)
(get-model)
```

c)

Minimize: Input

```
; Inverse Huffman for ('A', 4, (('B',2,'C'),3,('D',1,'E')))
(declare-const A Int)
(declare-const B Int)
(declare-const C Int)
(declare-const D Int)
(declare-const E Int)
(define-const A_len Int 1) ; length of encoding for message ’A’
(define-const B_len Int 3) ; length of encoding for message ’B’
(define-const C_len Int 3) ; length of encoding for message ’C’
(define-const D_len Int 3) ; length of encoding for message ’D’
(define-const E_len Int 3) ; length of encoding for message ’E’
(declare-const avg_code_len Real)

(assert
(and
(= 100 (+ A B C D E)) ; Must equal to 1,0 which is the same as 100 (since we are using integers)
(> A 0) ; Each frequency has to be higher than 0
(> B 0)
(> C 0)
(> D 0)
(> E 0)

; The first cycle, making sure D and E have the smallest frequencies
(< D A)
(< D B)
(< D C)
(< E A)
(< E B)
(< E C)

; The second cycle, making sure B and C have smaller frequencies than A and D + E (combined)
```

```
(< B A)
(< B (+ D E))
(< C A)
(< C (+ D E))

(= avg_code_len (/ (+ (* A_len A) (* B_len B) (* C_len C) (* D_len D) (* E_len E)) 100))

))
(minimize avg_code_len)

(check-sat)
(get-model)
```

Minimize: Output

```
sat
(
  (define-fun C () Int
    3)
  (define-fun E_len () Int
    3)
  (define-fun D () Int
    2)
  (define-fun B_len () Int
    3)
  (define-fun B () Int
    3)
  (define-fun C_len () Int
    3)
  (define-fun E () Int
    2)
  (define-fun D_len () Int
    3)
  (define-fun A_len () Int
    1)
  (define-fun avg_code_len () Real
    (/ 6.0 5.0))
  (define-fun A () Int
    90)

The minimum average for the code length is 1,2 bit/symbol
```

Maximize: Output

```
sat
(
  (define-fun C () Int
    21)
  (define-fun E_len () Int
    3)
  (define-fun D () Int
    16)
  (define-fun B_len () Int
    3)
  (define-fun B () Int
```

```
    21)
  (define-fun C_len () Int
    3)
  (define-fun E () Int
    20)
  (define-fun D_len () Int
    3)
  (define-fun A_len () Int
    1)
  (define-fun avg_code_len () Real
    (/ 64.0 25.0))
  (define-fun A () Int
    22)
)
```

The maximum average for the code length is 2,56 bit/symbol

Tree 2: ((('U', 1, 'V'), 2, x),5, ('W', 4, ('Y', 3, 'Z')))

a)

```
; Inverse Huffman for ((('U', 1, 'V'), 2, x),5, ('W', 4, ('Y', 3, 'Z')))
(declare-const U Int)
(declare-const V Int)
(declare-const W Int)
(declare-const X Int)
(declare-const Y Int)
(declare-const Z Int)

(assert
(and

(= 100 (+ U V W X Y Z)) ; Must equal to 1,0 which is the same as 100 (since we are using integers)
(> U 0) ; Each frequency has to be higher than 0
(> V 0)
(> W 0)
(> X 0)
(> Y 0)
(> Z 0)

; The first cycle, making sure U and V have the smallest two frequencies
(<= U W)
(<= U X)
(<= U Y)
(<= U Z)
(<= V W)
(<= V X)
(<= V Y)
(<= V Z)

; The second cycle, making sure U + V (combined) have smaller frequency than W, Y and z
(<= (+ U V) W)
(<= (+ U V) Y)
(<= (+ U V) Z)

; The thrid cycle, making sure X has a smaller frequency than W,Y and Z
(<= X Y)
```

```
(<= X Z)
(<= X W)

; The fourth cycle, making sure U + V + X (combined) have a greater frequency than W,Y and Z
(>= (+ U V X) W)
(>= (+ U V X) Y)
(>= (+ U V X) Z)

; The fifth cycle, making sure Y and Z have smaller frequencies than W and U + V + X (combined)
(<= Y W)
(<= Y (+ U V X))
(<= Z W)
(<= Z (+ U V X))

; The last cycle, making sure W has a smaller frequency than Y + Z (combined) and U + V + X
(combined)
(<= (+ Y Z) (+ U V X))
(<= W (+ Y Z))
(<= W (+ U V X))

))
(check-sat)
(get-model)
```

b)

```
; Inverse Huffman for ((('U', 1, 'V'), 2, x),5, ('W', 4, ('Y', 3, 'Z')))
(declare-const U Int)
(declare-const V Int)
(declare-const W Int)
(declare-const X Int)
(declare-const Y Int)
(declare-const Z Int)
(define-const U_len Int 3) ; length of encoding for message ’U’
(define-const V_len Int 3) ; length of encoding for message ’V’
(define-const W_len Int 2) ; length of encoding for message ’W’
(define-const X_len Int 2) ; length of encoding for message ’X’
(define-const Y_len Int 3) ; length of encoding for message ’Y’
(define-const Z_len Int 3) ; length of encoding for message ’Z’
(declare-const avg_code_len Real)

(assert
(and

(= 100 (+ U V W X Y Z)) ; Must equal to 1,0 which is the same as 100 (since we are using integers)
(> U 0) ; Each frequency has to be higher than 0
(> V 0)
(> W 0)
(> X 0)
(> Y 0)
(> Z 0)

; The first cycle, making sure U and V have the smallest two frequencies
(<= U W)
(<= U X)
(<= U Y)
(<= U Z)
(<= V W)
(<= V X)
(<= V Y)
(<= V Z)
```

```
; The second cycle, making sure U + V (combined) have smaller frequency than W, Y and z
(<= (+ U V) W)
(<= (+ U V) Y)
(<= (+ U V) Z)

; The thrid cycle, making sure X has a smaller frequency than W,Y and Z
(<= X Y)
(<= X Z)
(<= X W)

; The fourth cycle, making sure U + V + X (combined) have a greater frequency than W,Y and Z
(>= (+ U V X) W)
(>= (+ U V X) Y)
(>= (+ U V X) Z)

; The fifth cycle, making sure Y and Z have smaller frequencies than W and U + V + X (combined)
(<= Y W)
(<= Y (+ U V X))
(<= Z W)
(<= Z (+ U V X))

; The last cycle, making sure W has a smaller frequency than Y + Z (combined) and U + V + X
(combined)
(<= (+ Y Z) (+ U V X))
(<= W (+ Y Z))
(<= W (+ U V X))

(= avg_code_len (/ (+ (* U_len U) (* V_len V) (* W_len W) (* X_len X) (* Y_len Y) (* Z_len Z)) 100))
))
(check-sat)
(get-model)
```

c)

Minimize: Input

```
; Inverse Huffman for ((('U', 1, 'V'), 2, x),5, ('W', 4, ('Y', 3, 'Z')))
(declare-const U Int)
(declare-const V Int)
(declare-const W Int)
(declare-const X Int)
(declare-const Y Int)
(declare-const Z Int)
(define-const U_len Int 3) ; length of encoding for message ’U’
(define-const V_len Int 3) ; length of encoding for message ’V’
(define-const W_len Int 2) ; length of encoding for message ’W’
(define-const X_len Int 2) ; length of encoding for message ’X’
(define-const Y_len Int 3) ; length of encoding for message ’Y’
(define-const Z_len Int 3) ; length of encoding for message ’Z’
(declare-const avg_code_len Real)

(assert
(and

(= 100 (+ U V W X Y Z)) ; Must equal to 1,0 which is the same as 100 (since we are using integers)
(> U 0) ; Each frequency has to be higher than 0
(> V 0)
(> W 0)
(> X 0)
(> Y 0)
(> Z 0)
```

```
; The first cycle, making sure U and V have the smallest two frequencies
(<= U W)
(<= U X)
(<= U Y)
(<= U Z)
(<= V W)
(<= V X)
(<= V Y)
(<= V Z)

; The second cycle, making sure U + V (combined) have smaller frequency than W, Y and z
(<= (+ U V) W)
(<= (+ U V) Y)
(<= (+ U V) Z)

; The thrid cycle, making sure X has a smaller frequency than W,Y and Z
(<= X Y)
(<= X Z)
(<= X W)

; The fourth cycle, making sure U + V + X (combined) have a greater frequency than W,Y and Z
(>= (+ U V X) W)
(>= (+ U V X) Y)
(>= (+ U V X) Z)

; The fifth cycle, making sure Y and Z have smaller frequencies than W and U + V + X (combined)
(<= Y W)
(<= Y (+ U V X))
(<= Z W)
(<= Z (+ U V X))

; The last cycle, making sure W has a smaller frequency than Y + Z (combined) and U + V + X
(combined)
(<= (+ Y Z) (+ U V X))
(<= W (+ Y Z))
(<= W (+ U V X))

(= avg_code_len (/ (+ (* U_len U) (* V_len V) (* W_len W) (* X_len X) (* Y_len Y) (* Z_len Z)) 100))
))
(minimize avg_code_len)
(check-sat)
(get-model)
```

Minimize: Output

```
sat
(
  (define-fun W () Int
    32)
  (define-fun V_len () Int
    3)
  (define-fun V () Int
    1)
  (define-fun Y () Int
    17)
  (define-fun Z () Int
    17)
  (define-fun X () Int
```

```
    17)
  (define-fun Z_len () Int
    3)
  (define-fun X_len () Int
    2)
  (define-fun Y_len () Int
    3)
  (define-fun U_len () Int
    3)
  (define-fun W_len () Int
    2)
  (define-fun avg_code_len () Real
    (/ 251.0 100.0))
  (define-fun U () Int
    16)
)
```

The minimum average for the code length is 2,51 bit/symbol

Maximize: Output

```
sat
(
  (define-fun W () Int
    20)
  (define-fun V_len () Int
    3)
  (define-fun V () Int
    1)
  (define-fun Y () Int
    20)
  (define-fun Z () Int
    20)
  (define-fun X () Int
    20)
  (define-fun Z_len () Int
    3)
  (define-fun X_len () Int
    2)
  (define-fun Y_len () Int
    3)
  (define-fun U_len () Int
    3)
  (define-fun W_len () Int
    2)
  (define-fun avg_code_len () Real
    (/ 13.0 5.0))
  (define-fun U () Int
    19)
)
```

The maximum average for the code length is 2,6 bit/symbol

Problem 2

1)

9 91 1 1 1 1 1
(9 x 3) + (9 x 1) + (1 x 3) + (1 x 1) + (1 x 3) + (1 x 1) + (1 x 3) + (1 x 1) = multiple of 10
27 + 9 + 3 + 1 + 3 + 1+ 3 + 1 = 48 🚫 **not valid**

2 9 9 2 2 2 2 2
(2 x 3) + (9 x 1) + (9 x 3) + (2 x 1) + (2 x 3) + (2 x 1) + (2 x 3) + (2 x 1) = multiple of 10
6 + 9 + 27 + 2 + 6 + 2 + 6 + 2 = 60 ✅ **valid**

3 3 9 9 3 3 3 3
(3 x 3) + (3 x 1) + (9 x 3) + (9 x 1) + (3 x 3) + (3 x 1) + (3 x 3) + (3 x 1) = multiple of 10
9 + 3 + 27 + 9 + 9 + 3 + 9 + 3 = 66 🚫 **not valid**

4 4 4 9 9 4 4 4
(4 x 3) + (4 x 1) + (4 x 3) + (9 x 1) + (9 x 3) + (4 x 1) + (4 x 3) + (4 x 1) = multiple of 10
12 + 4 + 12 + 9 + 27 + 4 + 12 + 4 = 84 🚫 **not valid**

5 5 5 5 9 9 5 5
(5 x 3) + (5 x 1) + (5 x 3) + (5 x 1) + (9 x 3) + (9 x 1) + (5 x 3) + (5 x 1) = multiple of 10
15 + 5 + 15 + 5 + 27 + 9 + 15 + 5 = 96 🚫 **not valid**

6 6 6 6 6 9 9 6
(6 x 3) + (6 x 1) + (6 x 3) + (6 x 1) + (6 x 3) + (9 x 1) + (9 x 3) + (6 x 1) = multiple of 10
18 + 6 + 18 + 6 + 18 + 9 + 27 +6 = 108 🚫 **not valid**

7 7 7 7 7 7 9 9
(7 x 3) + (7 x 1) + (7 x 3) + (7x 1) + (7 x 3) + (7 x 1) + (9 x 3) + (9 x 1) = multiple of 10
21 + 7 + 21 + 7 + 21 + 7 + 27 + 9 = 120 ✅ **valid**

9 8 8 8 8 8 8 9
(9 x 3) + (8 x 1) + (8 x 3) + (8 x 1) + (8 x 3) + (8 x 1) + (8 x 3) + (9 x 1) = multiple of 10
27 + 8 + 24 + 8 + 24 + 8 +24 + 9 = 132 🚫 **not valid**

9 9 9 9 9 9 9 9
(9 x 3) + (9 x 1) + (9 x 3) + (9 x 1) + (9 x 3) + (9 x 1) + (9 x 3) + (9 x 1) = multiple of 10
27 + 9 + 27 + 9 + 27 + 9 + 27 + 9 = 144 🚫 **not valid**

2)

1 2 **?** 4 5 6 7 8
(1 x 3) + (2 x 1) + (a x 3) + (4 x 1) + (5 x 3) + (6 x 1) + (7 x 3) + (8 x 1) = multiple of 10
3 + 2 + 3a + 4 + 15 + 6 + 21 + 8 = multiple of 10
59 + 3a = 80
3a = 21
a = 7

1 2 **7** 5 6 7 8

3)

1 6 7 0 9 8 0
(1 x 3) + (6 x 1) + (7 x 3) + (0 x 1) + (9 x 3) + (8 x 1) + (0 x 3) + (a x 1) = multiple of 10
3 + 6 + 21 + 0 + 27 + 8 + 0 + a = multiple of 10
65 + a = 70
a = 5

1 6 7 0 9 8 0 **5**

Problem 3

a)

```
wlan-187187:downloads chloeantonozzi$ gpg --verify signature-2122-1.asc.txt
LBUI-2122-D.txt

gpg: Signature made Dim  5 déc 21:19:17 2021 CET
gpg:                using EDDSA key AA631A2865697585C1FEB368A32C53DEFA43931A
gpg: Good signature from "Tom Verhoeff (For use in the TU/e bachelor course
2ITX0, Applied Logic, 2021-2022) (For educational use)
<T.Verhoeff@tue.nl>" [unknown]
gpg: WARNING: This key is not certified with a trusted signature!
gpg:          There is no indication that the signature belongs to the owner.
Primary key fingerprint: AA63 1A28 6569 7585 C1FE  B368 A32C 53DE FA43 931A

Therefore Signature-2122-1.asc belongs to LBUI-2122-D.txt


wlan-187187:downloads chloeantonozzi$ gpg --verify signature-2122-2.asc.txt
LBUI-2122-B.txt
gpg: Signature made Dim  5 déc 21:20:23 2021 CET
gpg:                using EDDSA key AA631A2865697585C1FEB368A32C53DEFA43931A
gpg: Good signature from "Tom Verhoeff (For use in the TU/e bachelor course
2ITX0, Applied Logic, 2021-2022) (For educational use)
<T.Verhoeff@tue.nl>" [unknown]
gpg: WARNING: This key is not certified with a trusted signature!
gpg:          There is no indication that the signature belongs to the owner.
Primary key fingerprint: AA63 1A28 6569 7585 C1FE  B368 A32C 53DE FA43 931A

Therefore Signature-2122-2.asc belongs to LBUI-2122-B.txt


wlan-187187:downloads chloeantonozzi$ gpg --verify signature-2122-3.asc.txt
LBUI-2122-C.txt
gpg: Signature made Dim  5 déc 21:20:48 2021 CET
gpg:                using EDDSA key AA631A2865697585C1FEB368A32C53DEFA43931A
gpg: Good signature from "Tom Verhoeff (For use in the TU/e bachelor course
2ITX0, Applied Logic, 2021-2022) (For educational use)
<T.Verhoeff@tue.nl>" [unknown]
gpg: WARNING: This key is not certified with a trusted signature!
gpg:          There is no indication that the signature belongs to the owner.
Primary key fingerprint: AA63 1A28 6569 7585 C1FE  B368 A32C 53DE FA43 931A

Therefore Signature-2122-3.asc belongs to LBUI-2122-C.txt
```