Ganesh -Gundekarw
11700551.

Algo's Assignment-04

(1) Huffman Code is

string : ACCGGTCGAGTGCGCGGAAGCCGGCGAA

frequencies are :-

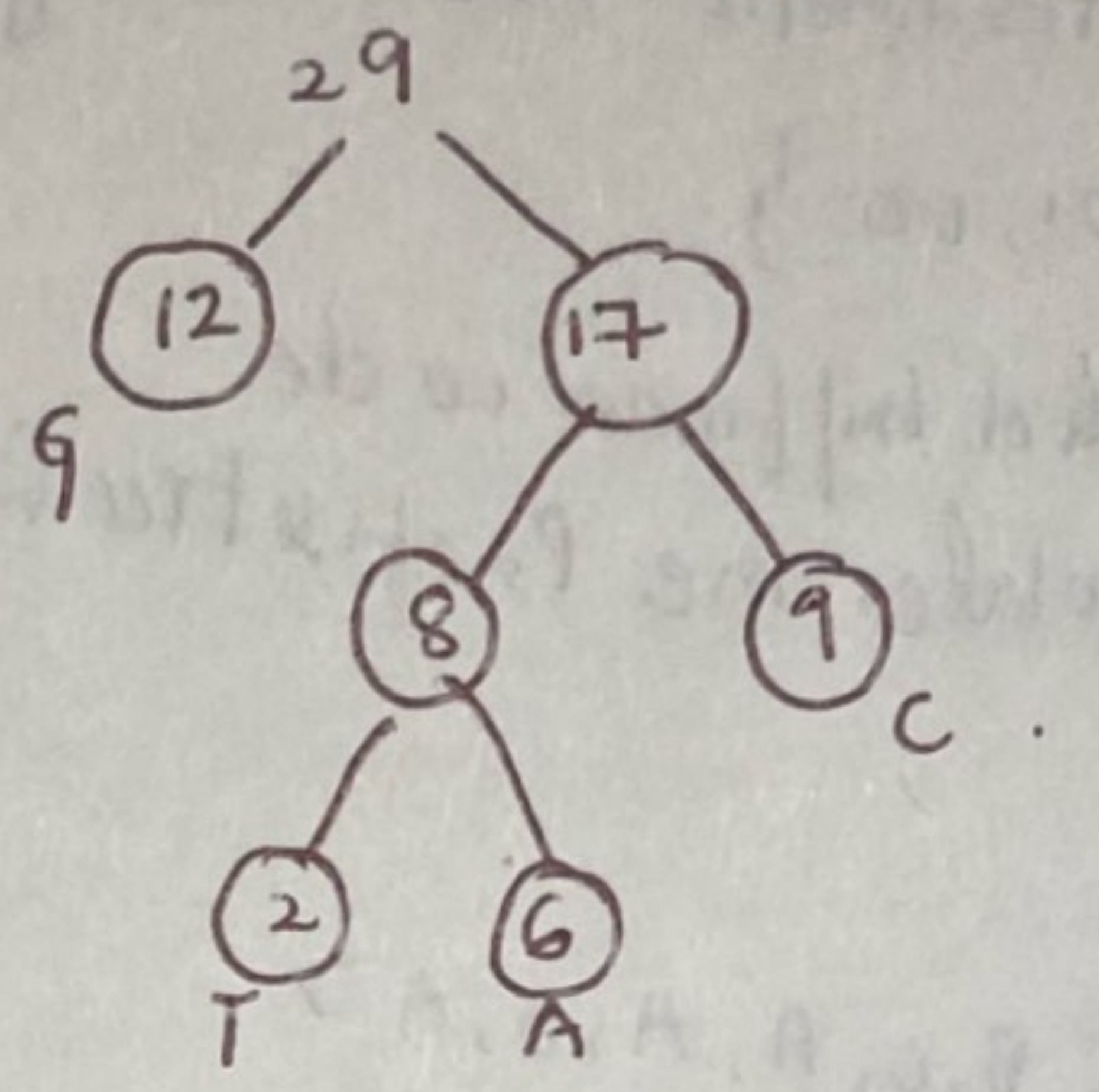
A:-6

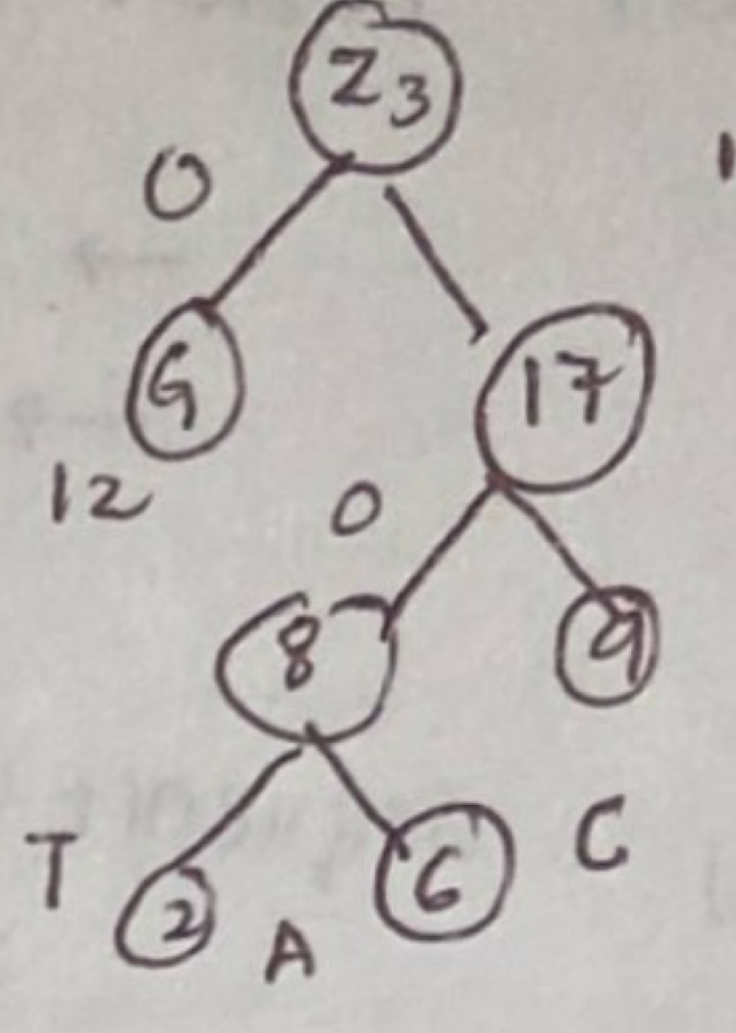
C:-9

G:-12

T: 2

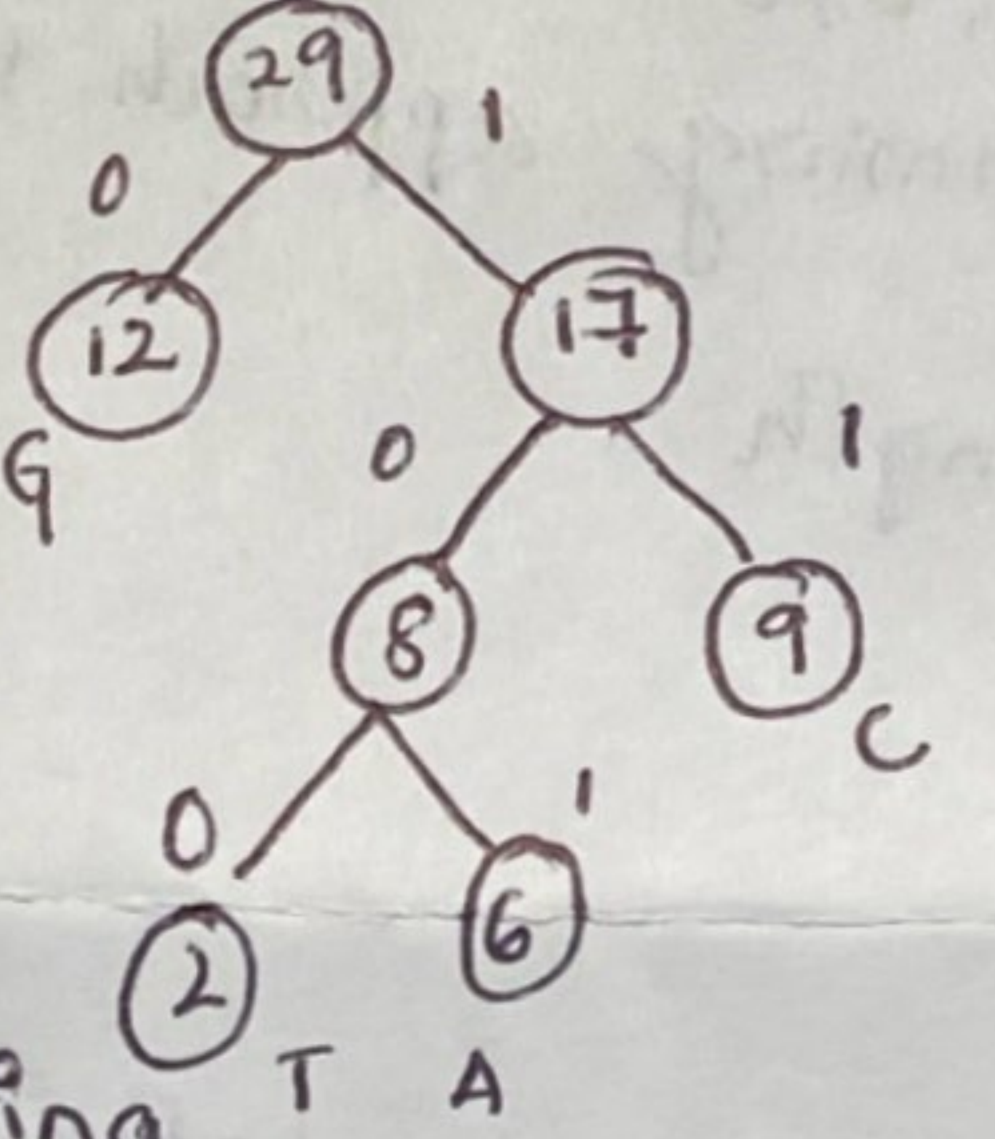
Tree —





Code :- leftchild:- 0

Right child :- 0



| Char | freq | codeword | bits |
|---|---|---|---|
| G | 12 | 0(2pts) | 1 |
| C | 9 | 11(2pts\| | 2 |
| A | 6 | 101(2pts) | 3 |
| T | 2 | 100(2pts) | 3 |

A:- 101 → length :- 3

C: 11 = 2

G: 0 = 1

T: 100 = 3

Number of bits required to encode string

$= 3\times6 + 2\times9 + 1\times12 + 2\times3$

$= 18+18+12+6$

$= 54$ bits //

2) a) code : { 0, 10, 11 }

b) code: { 0, 1, 00 }

c) code : { 10, 01, 00 }

Properties of Huffman Algorithm :-

1. The code must be prefix free, meaning the No code is the prefix of another codeword.

2. Here, the code should be minimizing the weighted Average Code length, where the weight is the frequency of each symbol.

a> code : {0,10,11}

→ valid huffman code

→ freq statisty  $f_a > f_b + f_c$

· This is a prefix (-free) code, and it is possible for enample, the freq $f_c = 4$, $f_b = 2$ and $f_c = 3$ not unique but they should have $f_a \geq f_b \geq f_c$ and $f_a \leq f_b + f_c$ gives code (0,10,

b> code : {0,1,00}

→ Invalid huffman code

→ violates Prefix free Property

// 2. The code '0' fore the letter "a" is a prefix of code "00" for letter c.

(c) code : {10,01,00}

→ Invalid huffman code

→ Violates the Prefix Free Property

// The code is not optimal since code {1,01,00} gives a shorter encoding. doesn't correspond to a full binary tree.

(3) sequences are

$X = \langle B, C, A, A, B, A \rangle$

$Y = \langle A, B, A, C, B \rangle$

here, Dynamic Programming approch to fing Longest Common Subsequence.

Pseudocode for LCS length

```
LCS-length (x, y)
    m = length(x)
    n = length(y)
    let c[0...m, 0...n] be new table
    let b[0...m, 0...n] be new table
    for i = 0 to u
        c[i,0] = 0
    for j = 0 to n
        c[0,j] = 0
    for i = 1 to m
        for j = 1 to n
            if x[i] = y[j]
                c[i,j] = c[i-1,j-1] + 1
                b[i,j] = "↖" // upper left
            else if c[i-1, j] >= c[i,j-1]
                c[i,j] = c[i-1, j]
                b[i,j] = "↑" // up arrow
```

```
else
    c[i,j] = c[i,j-1]
    b[i,j] = "←" lef
return c and b
```

let's apply pseudocode for sequence

$X = \langle B, C, A, A, B, A \rangle$ and $Y = \langle A, B, A, C, B \rangle$

C-table

| | A | B | A | C | B |
|---|---|---|---|---|---|
| B | 0 | 0 | 0 | 0 | 0 |
| C | 0 | 0 | 0 | 0 | 0 |
| A | 0 | 1 | 1 | 1 | 1 |
| A | 0 | 1 | 2 | 2 | 2 |
| B | 0 | 1 | 2 | 2 | 2 |
| A | 0 | 1 | 2 | 3 | 3 |

b-table →

| | A | B | A | C | B |
|---|---|---|---|---|---|
| B | O | O | O | O | O |
| C | O | O | O | O | O |
| A | O | ↑ | ← | ← | ← |
| A | O | ↖ | ↑ | ← | ← |
| B | O | ↑ | ↖ | ← | ← |
| A | O | ↑ | ↑ | ↑ | ↖ |

following Arrows in btable to the top-left
LCS : $\langle A, A, B \rangle$ for X, Y.

(4) $\{(v_i, w_i)\}$ for $i = 1, 2, 3, 4, 5$

value $v = [2, 3, 3, 4, 4]$

weight $w = [2, 3, 1, 2, 3]$

Total allowable weight $w = 5$

using 0/1 Knapsack Algo.

c table:- using Formula

$$c[i][w] = \max\left[\, c[i-1][w],\ V_i + c[i-1][w-w_i] \,\right]$$

| Item | 0 | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 2 | 2 | 2 | 2 |
| 2 | 0 | 0 | 2 | 3 | 3 | 5 |
| 3 | 0 | 3 | 3 | 5 | 6 | 6 |
| 4 | 0 | 3 | 4 | 5 | 7 | 8 |
| 5 | 0 | 3 | 4 | 5 | 7 | 8 |

Tracing Back:-

starting at c [5][5] = 8

c [5][4] = 7 → Not Equal (value 4 weight 3) included this in the solut

c [4] [2] = 3 → equal, not included in sol$^n$

c [3] [2] = 3 → equal, not included in sol$^n$

c[2][1] = 0 → not Equal, (value 3, weight 3) item 2 not included in sol$^n$

c[1] [0] = 0 → Equal, item - 1 (value 2, weight - 2) is included in sol$^n$.

optimal sol$^n$:- 1, 3, 5.

57 a) change for M amount using denominations $d_1, d_2, \ldots d_n$ to form a greedy choice property

Selecting largest Denomination that is less than the Amount M (or) remain Amount.

(b)

```
def min-change (denom, target):
    den= [float ('inf')]*(target +1)
    den [0] = 0
    for coin in denom:
        for i in range (coin, target +1):
            den[i]= min( den[i], den [i- coin]+1)
    return den[target]
denominations = [1,5,10,25]
amount = 17
```

```
result = min_change (denominations,amount)
print (f"min no of coins to make {amount} cents"
          {results"}
```

e) Time Complexity

= O(n* target)

n → No of coin denominat^n's

target → target Amount