Name: Ganath Avinash G.R

CSE – B

CH.SC.U4CSE24118

Week –8 (22/2/2026)

1. Huffman Coding – Qn “dataanalyticsandintelligencelaboratory”

Code:

```
//CH.SC.U4CSE24118

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define MAX_TREE_HT 100
#define MAX_CHAR 256
struct MinHeapNode {
    char data;
    unsigned freq;
    struct MinHeapNode *left, *right;
};
struct MinHeap {
    unsigned size;
    unsigned capacity;
    struct MinHeapNode** array;
};
struct MinHeapNode* newNode(char data, unsigned freq) {
    struct MinHeapNode* temp = (struct MinHeapNode*)malloc(sizeof(struct MinHeapNode));
    temp->left = temp->right = NULL;
    temp->data = data;
    temp->freq = freq;
    return temp;
}
struct MinHeap* createMinHeap(unsigned capacity) {
    struct MinHeap* minHeap = (struct MinHeap*)malloc(sizeof(struct MinHeap));
    minHeap->size = 0;
    minHeap->capacity = capacity;
    minHeap->array = (struct MinHeapNode**)malloc(minHeap->capacity * sizeof(struct
MinHeapNode*));
    return minHeap;
}
void swapMinHeapNode(struct MinHeapNode** a, struct MinHeapNode** b) {
    struct MinHeapNode* t = *a;
    *a = *b;
    *b = t;
}
```

```
void minHeapify(struct MinHeap* minHeap, int idx) {
    int smallest = idx;
    int left = 2 * idx + 1;
    int right = 2 * idx + 2;

    if (left < minHeap->size && minHeap->array[left]->freq < minHeap->array[smallest]->freq)
        smallest = left;

    if (right < minHeap->size && minHeap->array[right]->freq < minHeap->array[smallest]->freq)
        smallest = right;

    if (smallest != idx) {
        swapMinHeapNode(&minHeap->array[smallest], &minHeap->array[idx]);
        minHeapify(minHeap, smallest);
    }
}
struct MinHeapNode* extractMin(struct MinHeap* minHeap) {
    struct MinHeapNode* temp = minHeap->array[0];
    minHeap->array[0] = minHeap->array[minHeap->size - 1];
    --minHeap->size;
    minHeapify(minHeap, 0);
    return temp;
}
void insertMinHeap(struct MinHeap* minHeap, struct MinHeapNode* minHeapNode) {
    ++minHeap->size;
    int i = minHeap->size - 1;
    while (i && minHeapNode->freq < minHeap->array[(i - 1) / 2]->freq) {
        minHeap->array[i] = minHeap->array[(i - 1) / 2];
        i = (i - 1) / 2;
    }
    minHeap->array[i] = minHeapNode;
}
void buildMinHeap(struct MinHeap* minHeap) {
    int n = minHeap->size - 1;
    for (int i = (n - 1) / 2; i >= 0; --i)
        minHeapify(minHeap, i);
}
int isLeaf(struct MinHeapNode* root) {
    return !(root->left) && !(root->right);
}
void printCodes(struct MinHeapNode* root, int arr[], int top) {
    if (root->left) {
        arr[top] = 0;
        printCodes(root->left, arr, top + 1);
    }
```

```
    if (root->right) {
        arr[top] = 1;
        printCodes(root->right, arr, top + 1);
    }
    if (isLeaf(root)) {
        printf("  %c   |   %3d   | ", root->data, root->freq);
        for (int i = 0; i < top; ++i)
            printf("%d", arr[i]);
        printf("\n");
    }
}
void calculateSpaceSaving(struct MinHeapNode* root, int arr[], int top, int* totalOriginalBits, int*
totalCompressedBits) {
    if (root->left) {
        arr[top] = 0;
        calculateSpaceSaving(root->left, arr, top + 1, totalOriginalBits, totalCompressedBits);
    }
    if (root->right) {
        arr[top] = 1;
        calculateSpaceSaving(root->right, arr, top + 1, totalOriginalBits, totalCompressedBits);
    }
    if (isLeaf(root)) {
        *totalOriginalBits += root->freq * 8; // Assuming 8 bits per character
        *totalCompressedBits += root->freq * top;
    }
}
void HuffmanCodes(char data[], int freq[], int size) {
    struct MinHeapNode *left, *right, *top;
    struct MinHeap* minHeap = createMinHeap(size);
    for (int i = 0; i < size; ++i)
        minHeap->array[i] = newNode(data[i], freq[i]);
    minHeap->size = size;
    buildMinHeap(minHeap);
    while (minHeap->size != 1) {
        left = extractMin(minHeap);
        right = extractMin(minHeap);
        top = newNode('$', left->freq + right->freq);
        top->left = left;
        top->right = right;
        insertMinHeap(minHeap, top);
    }
    int arr[MAX_TREE_HT], topIdx = 0;
    printf("\nChar | Freq | Huffman Code\n");
    printf("-------------------------\n");
    struct MinHeapNode* root = extractMin(minHeap);
    printCodes(root, arr, topIdx);
```

```
    int totalOriginalBits = 0, totalCompressedBits = 0;
    calculateSpaceSaving(root, arr, 0, &totalOriginalBits, &totalCompressedBits);
    printf("\nSpace Saving Analysis:\n");
    printf("----------------------\n");
    printf("Original size (8 bits per character): %d bits\n", totalOriginalBits);
    printf("Compressed size (Huffman coding): %d bits\n", totalCompressedBits);
    printf("Space saved: %d bits (%.2f%%)\n",
           totalOriginalBits - totalCompressedBits,
           ((float)(totalOriginalBits - totalCompressedBits) / totalOriginalBits) * 100);
}
int main() {
    char str[1000];
    int freq[MAX_CHAR] = {0};
    printf("Enter the string: ");
    scanf("%s", str);
    for (int i = 0; str[i] != '\0'; i++) {
        freq[(unsigned char)str[i]]++;
    }
    char unique_chars[MAX_CHAR];
    int unique_freqs[MAX_CHAR];
    int n = 0;
    for (int i = 0; i < MAX_CHAR; i++) {
        if (freq[i] > 0) {
            unique_chars[n] = (char)i;
            unique_freqs[n] = freq[i];
            n++;
        }
    }
    HuffmanCodes(unique_chars, unique_freqs, n);
    return 0;
}
```

OUTPUT:

```
PS C:\Users\Ganath Avinash\OneDrive\ドキュメント\Back-end\DAA> cd
Huffman_C.c -o Huffman_C } ; if ($?) { .\Huffman_C }
  y   |     2   | 1010
  s   |     1   | 10110
  d   |     2   | 10111
  e   |     3   | 1100
  i   |     3   | 1101
  a   |     7   | 111

Space Saving Analysis:
----------------------
Original size (8 bits per character): 304 bits
Compressed size (Huffman coding): 138 bits
Space saved: 166 bits (54.61%)
PS C:\Users\Ganath Avinash\OneDrive\ドキュメント\Back-end\DAA>
```

Huffman:

Data Analytics and intelligence laboratory

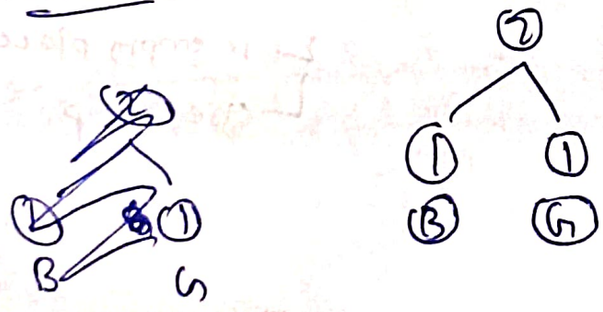
Letter	D	A	T	N	L	Y	I	C	S	E	G	B	R	O
Frequency	2	7	4	4	4	2	3	2	1	3	1	1	2	2

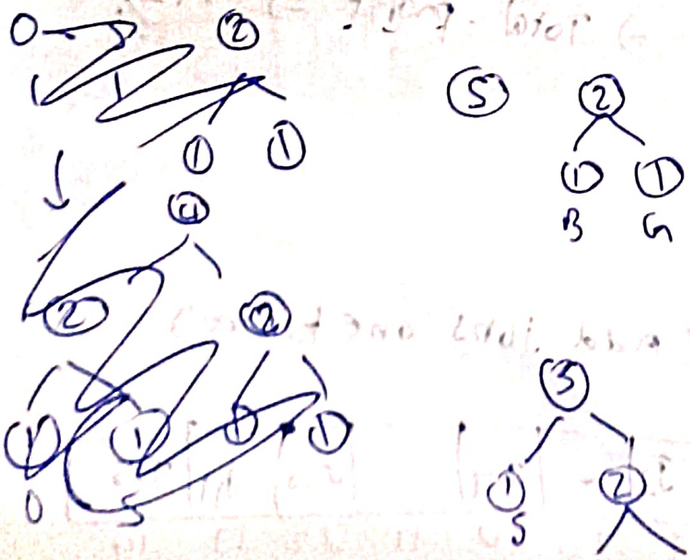
① Sort:

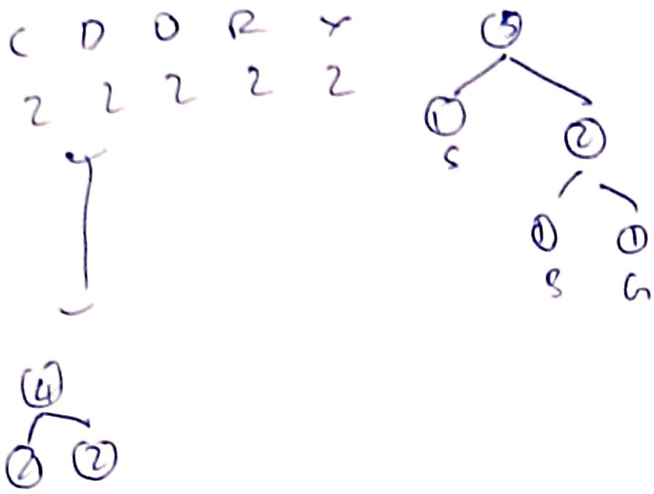
S G B O D Y C R N I E T L A
1 1 1 1 2 2 2 2 3 3 3 4 4 5 8

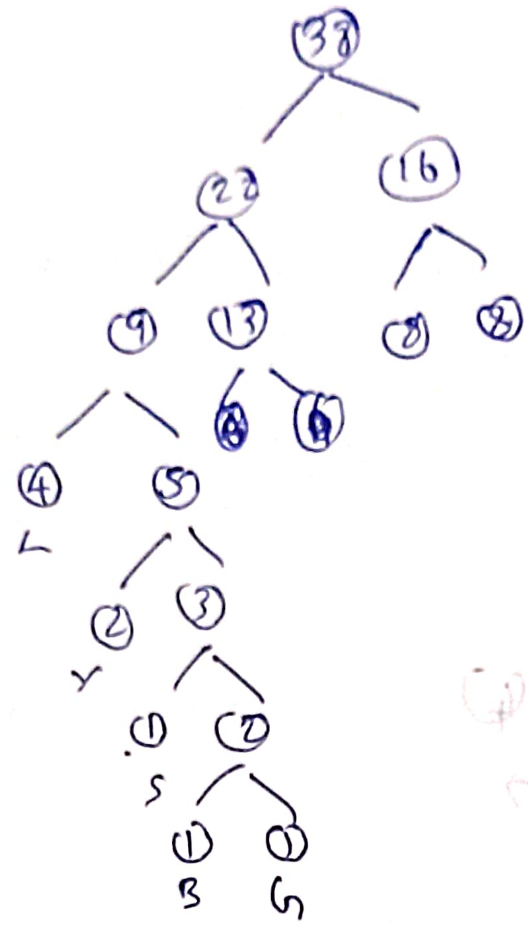
B	G	S	C	DOR	Y	E	I	N	T	L	A
1	1	1	2	222	2	3	3	4	4	4	8

② Build tree:






C D O R X
2 2 2 2 2
5
1 2
S
0 1
S G
4
1 2
O R X 3 4 4
5
2 3
X
1 2
S
1 1
B G
4
2 1
O R
4
2 2
C D
E I
3 3
6
4 4 N T L 5 6 A
8
N T L 5 6 A 8
L 5 6 A 8 8
9 6 A 8 8 9
3 8 4 13 => 9 13 16 => 21 16
=> 37



=)

=) C D O R

0000 0001 0010 011

=) Y T

010 011

=) L Y S B G

100 1010 10110 10110 10111

⇒ E I
1100 110 1

⇒ A
111

⇒ Total cost

⇒ Σ (Code L x freq) ⇒

= (4 x 2) + (2 x 4) + (4 x 1) + (3 x 2) + (3 x 5) + (4 x 2)
+ (5 x 1) + 0 (5 x 1)
+ (5 x 1)
+ (4 x 3) + (4 x 3)
+ (7 x 3)

⇒ 8 + 8 + 4 + 6 + 15 + 8 + 5 + 6 + 6 + 12 + 12 + 21

⇒ 111 //