CHANAKYA UNIVERSITY

DATA STRUCTURESAND ALOGORITHMS-EVE203

Assignment-02

* MID TERM QUESTION AND ANSWER

Question 1:

#include <stdio.h>

#include <string.h>

#define DAYS 30

struct Member {

char name[50];

int steps[DAYS];

};

void analyzeSteps(struct Member members[], int n) {

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

int count = 0;

int maxSteps = members[i].steps[0];

for (int j = 0; j < DAYS; j++) {

if (members[i].steps[j] > 10000)

count++;

if (members[i].steps[j] > maxSteps)

maxSteps = members[i].steps[j];

}

printf("\nMember: %s\n", members[i].name);

printf("Days exceeded 10,000 steps: %d\n", count);

printf("Maximum steps in month: %d\n", maxSteps);

}

}

int main() {

int n;

printf("Enter number of members: ");

scanf("%d", &n);

struct Member members[n];

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

printf("\nEnter name of member %d: ", i + 1);

scanf("%s", members[i].name);

printf("Enter steps for 30 days: ");

for (int j = 0; j < DAYS; j++) {

scanf("%d", &members[i].steps[j]);

}

}

analyzeSteps(members, n);

return 0;

}

OUTPUT

Member: Kumar

Days exceeded 10,000 steps: 15

Maximum steps in month: 15000

Member: Charan

Days exceeded 10,000 steps: 0

Maximum steps in month: 9000

QUESTION 2:

#include <stdio.h>

#include <stdlib.h>

struct Node {

int id;

struct Node\* next;

};

struct Node\* createNode(int id) {

struct Node\* newNode = (struct Node\*)malloc(sizeof(struct Node));

newNode->id = id;

newNode->next = NULL;

return newNode;

}

void printList(struct Node\* head) {

while (head) {

printf("%d", head->id);

if (head->next) printf(" -> ");

head = head->next;

}

printf("\n");

}

struct Node\* reversePlaylistSegment(struct Node\* head, int m, int n) {

if (!head || m == n) return head;

struct Node dummy;

dummy.next = head;

struct Node\* prev = &dummy;

// Move prev to node before m-th

for (int i = 1; i < m; i++)

prev = prev->next;

struct Node\* start = prev->next;

struct Node\* then = start->next;

// Reverse segment

for (int i = 0; i < n - m; i++) {

start->next = then->next;

then->next = prev->next;

prev->next = then;

then = start->next;

}

return dummy.next;

}

int main() {

int ids[] = {101, 102, 103, 104, 105, 106, 107};

int n = sizeof(ids)/sizeof(ids[0]);

struct Node\* head = createNode(ids[0]);

struct Node\* temp = head;

for (int i = 1; i < n; i++) {

temp->next = createNode(ids[i]);

temp = temp->next;

}

printf("Original playlist:\n");

printList(head);

int m = 2, end = 5;

head = reversePlaylistSegment(head, m, end);

printf("\nModified playlist:\n");

printList(head);

return 0;

}

OUTPUT:

Original playlist:

101 -> 102 -> 103 -> 104 -> 105 -> 106 -> 107

Modified playlist:

101 -> 105 -> 104 -> 103 -> 102 -> 106 -> 107

QUESTION 3:

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

// Structure for tree node

struct Node {

char ch;

int freq;

struct Node \*left, \*right;

};

// Queue structure for level-order insertion

struct Queue {

struct Node\* data[100];

int front, rear;

};

// Initialize Queue

void initQueue(struct Queue\* q) {

q->front = q->rear = -1;

}

int isEmpty(struct Queue\* q) {

return q->front == -1;

}

void enqueue(struct Queue\* q, struct Node\* node) {

if (q->rear == 99) return;

if (q->front == -1) q->front = 0;

q->data[++q->rear] = node;

}

struct Node\* dequeue(struct Queue\* q) {

if (isEmpty(q)) return NULL;

struct Node\* temp = q->data[q->front];

if (q->front == q->rear)

q->front = q->rear = -1;

else

q->front++;

return temp;

}

// Create new node

struct Node\* createNode(char ch) {

struct Node\* newNode = (struct Node\*)malloc(sizeof(struct Node));

newNode->ch = ch;

newNode->freq = 1;

newNode->left = newNode->right = NULL;

return newNode;

}

// Check if character exists in the tree

int incrementIfExists(struct Node\* root, char ch) {

if (root == NULL) return 0;

struct Queue q;

initQueue(&q);

enqueue(&q, root);

while (!isEmpty(&q)) {

struct Node\* curr = dequeue(&q);

if (curr->ch == ch) {

curr->freq++;

return 1;

}

if (curr->left) enqueue(&q, curr->left);

if (curr->right) enqueue(&q, curr->right);

}

return 0;

}

// Insert new character in level order

void insertLevelWise(struct Node\*\* root, char ch) {

if (\*root == NULL) {

\*root = createNode(ch);

return;

}

// If exists, just increment frequency

if (incrementIfExists(\*root, ch))

return;

// Otherwise, insert at first empty position

struct Queue q;

initQueue(&q);

enqueue(&q, \*root);

while (!isEmpty(&q)) {

struct Node\* temp = dequeue(&q);

if (temp->left == NULL) {

temp->left = createNode(ch);

return;

} else enqueue(&q, temp->left);

if (temp->right == NULL) {

temp->right = createNode(ch);

return;

} else enqueue(&q, temp->right);

}

}

// Print tree in level order (character, frequency)

void printLevelOrder(struct Node\* root) {

if (root == NULL) return;

struct Queue q;

initQueue(&q);

enqueue(&q, root);

printf("\nLevel Order Traversal (char, freq):\n");

while (!isEmpty(&q)) {

struct Node\* curr = dequeue(&q);

printf("(%c,%d) ", curr->ch, curr->freq);

if (curr->left) enqueue(&q, curr->left);

if (curr->right) enqueue(&q, curr->right);

}

printf("\n");

}

// Main

int main() {

char str[100];

printf("Enter the string: ");

scanf("%s", str);

struct Node\* root = NULL;

for (int i = 0; i < strlen(str); i++)

insertLevelWise(&root, str[i]);

printLevelOrder(root);

return 0;

}

OUTPUT:

Enter the string: programming

Level Order Traversal (char, freq):

(p,1) (r,2) (o,1) (g,2) (a,1) (m,2) (i,1) (n,1)

NAME: KUSHAL A

REG:25UG00515