1.Linear queue using arrays

#include <stdio.h>

#define MAX\_SIZE 100

int queue[MAX\_SIZE];

int front = -1;

int rear = -1;

void enqueue(int value) {

if (rear == MAX\_SIZE - 1) {

printf("Queue is full. Cannot enqueue.\n");

return;

}

if (front == -1) {

front = 0;

}

rear++;

queue[rear] = value;

}

void dequeue() {

if (front == -1 || front > rear) {

printf("Queue is empty. Cannot dequeue.\n");

return;

}

front++;

}

int front\_element() {

if (front == -1 || front > rear) {

printf("Queue is empty. No front element.\n");

return -1;

}

return queue[front];

}

int find\_element(int value) {

if (front == -1 || front > rear) {

printf("Queue is empty. Element not found.\n");

return -1;

}

for (int i = front; i <= rear; i++) {

if (queue[i] == value) {

return i;

}

}

printf("Element not found in the queue.\n");

return -1;

}

int main() {

// Example usage

enqueue(10);

enqueue(20);

enqueue(30);

printf("Front element: %d\n", front\_element());

dequeue();

printf("Front element after dequeue: %d\n", front\_element());

int index = find\_element(20);

if (index != -1) {

printf("Element 20 found at index: %d\n", index);

}

return 0;

}

2.Circular queue using arrays

#include <stdio.h>

#define MAX\_SIZE 100

int queue[MAX\_SIZE];

int front = -1;

int rear = -1;

void enqueue(int value) {

if ((front == 0 && rear == MAX\_SIZE - 1) || (rear == (front - 1) % (MAX\_SIZE - 1))) {

printf("Queue is full. Cannot enqueue.\n");

return;

} else if (front == -1) {

front = rear = 0;

} else if (rear == MAX\_SIZE - 1 && front != 0) {

rear = 0;

} else {

rear++;

}

queue[rear] = value;

}

void dequeue() {

if (front == -1) {

printf("Queue is empty. Cannot dequeue.\n");

return;

}

printf("Dequeued element: %d\n", queue[front]);

if (front == rear) {

front = rear = -1;

} else if (front == MAX\_SIZE - 1) {

front = 0;

} else {

front++;

}

}

int find\_element(int value) {

if (front == -1) {

printf("Queue is empty. Element not found.\n");

return -1;

}

int i = front;

do {

if (queue[i] == value) {

return i;

}

i = (i + 1) % MAX\_SIZE;

} while (i != (rear + 1) % MAX\_SIZE);

printf("Element not found in the queue.\n");

return -1;

}

int main() {

// Example usage

enqueue(10);

enqueue(20);

enqueue(30);

dequeue();

enqueue(40);

int index = find\_element(20);

if (index != -1) {

printf("Element 20 found at index: %d\n", index);

}

return 0;

}

Assignment 2

3.#include <stdio.h>

#include <stdbool.h>

#define N 4

void printSolution(int board[N][N]) {

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

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

printf("%d ", board[i][j]);

}

printf("\n");

}

}

bool isSafe(int board[N][N], int row, int col) {

int i, j;

// Check this row on left side

for (i = 0; i < col; i++)

if (board[row][i])

return false;

// Check upper diagonal on left side

for (i = row, j = col; i >= 0 && j >= 0; i--, j--)

if (board[i][j])

return false;

// Check lower diagonal on left side

for (i = row, j = col; j >= 0 && i < N; i++, j--)

if (board[i][j])

return false;

return true;

}

bool solveNQUtil(int board[N][N], int col) {

if (col >= N)

return true;

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

if (isSafe(board, i, col)) {

board[i][col] = 1;

if (solveNQUtil(board, col + 1))

return true;

board[i][col] = 0; // BACKTRACK

}

}

return false;

}

bool solveNQ() {

int board[N][N] = { {0, 0, 0, 0},

{0, 0, 0, 0},

{0, 0, 0, 0},

{0, 0, 0, 0} };

if (solveNQUtil(board, 0) == false) {

printf("Solution does not exist");

return false;

}

printSolution(board);

return true;

}

int main() {

solveNQ();

return 0;

}