|  |
| --- |
| **SAVEETHA INSTITUTE OF MEDICAL AND TECHNICAL SCIENCES** |
| **COMPUTER SCIENCE AND ENGINEERING PROGRAMME** |

**SUB CODE: CSA0392 SUB NAME: Data Structures for Hashing Techniques**

**LIST OF PROGRAMS**

**DAY 6 : 30.07.2024**

**Lab Questions to be practiced with test cases**

1. Write a program to implement Circular Queue with insertion and deletion operations.

Answer:

#include <stdio.h>

#include <stdlib.h>

#define MAX 100 // Define the maximum size of the queue

// Define the structure for the circular queue

typedef struct {

int arr[MAX];

int front;

int rear;

int size;

} CircularQueue;

// Function to initialize the queue

void initializeQueue(CircularQueue\* queue) {

queue->front = 0;

queue->rear = -1;

queue->size = 0;

}

// Function to check if the queue is empty

int isEmpty(CircularQueue\* queue) {

return queue->size == 0;

}

// Function to check if the queue is full

int isFull(CircularQueue\* queue) {

return queue->size == MAX;

}

// Function to add an element to the end of the queue

void enqueue(CircularQueue\* queue, int value) {

if (isFull(queue)) {

printf("Queue is full. Cannot enqueue %d.\n", value);

return;

}

queue->rear = (queue->rear + 1) % MAX; // Circular increment

queue->arr[queue->rear] = value;

queue->size++;

printf("Enqueued %d.\n", value);

}

// Function to remove an element from the front of the queue

int dequeue(CircularQueue\* queue) {

if (isEmpty(queue)) {

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

return -1; // Return -1 to indicate an error

}

int value = queue->arr[queue->front];

queue->front = (queue->front + 1) % MAX; // Circular increment

queue->size--;

return value;

}

// Function to get the element at the front of the queue without removing it

int peek(CircularQueue\* queue) {

if (isEmpty(queue)) {

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

return -1; // Return -1 to indicate an error

}

return queue->arr[queue->front];

}

// Function to display the elements of the queue

void displayQueue(CircularQueue\* queue) {

if (isEmpty(queue)) {

printf("Queue is empty.\n");

return;

}

printf("Queue elements are:\n");

int i;

for (i = 0; i < queue->size; i++) {

printf("%d ", queue->arr[(queue->front + i) % MAX]);

}

printf("\n");

}

int main() {

CircularQueue queue;

initializeQueue(&queue);

int choice, value;

while (1) {

printf("\nCircular Queue Menu:\n");

printf("1. Enqueue\n");

printf("2. Dequeue\n");

printf("3. Peek\n");

printf("4. Display\n");

printf("5. EXIT\n");

printf("Enter your choice: ");

scanf("%d", &choice);

switch (choice) {

case 1:

printf("Enter the value to enqueue: ");

scanf("%d", &value);

enqueue(&queue, value);

break;

case 2:

value = dequeue(&queue);

if (value != -1) {

printf("Dequeued %d.\n", value);

}

break;

case 3:

value = peek(&queue);

if (value != -1) {

printf("Front element is %d.\n", value);

}

break;

case 4:

displayQueue(&queue);

break;

case 5:

printf("Exiting...\n");

exit(0);

default:

printf("Invalid choice. Please try again.\n");

}

}

return 0;

}

1. Write a program to implement Double Ended Queue with insertion and deletion operations.

Answer:

#include <stdio.h>

#include <stdlib.h>

// Define the structure for a node in the doubly linked list

typedef struct Node {

int data;

struct Node\* prev;

struct Node\* next;

} Node;

// Define the structure for the deque

typedef struct {

Node\* front;

Node\* rear;

int size;

} Deque;

// Function to initialize the deque

void initializeDeque(Deque\* deque) {

deque->front = NULL;

deque->rear = NULL;

deque->size = 0;

}

// Function to check if the deque is empty

int isEmpty(Deque\* deque) {

return deque->size == 0;

}

// Function to insert an element at the front of the deque

void insertFront(Deque\* deque, int value) {

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

newNode->data = value;

newNode->prev = NULL;

newNode->next = deque->front;

if (isEmpty(deque)) {

deque->rear = newNode;

} else {

deque->front->prev = newNode;

}

deque->front = newNode;

deque->size++;

printf("Inserted %d at the front.\n", value);

}

// Function to insert an element at the rear of the deque

void insertRear(Deque\* deque, int value) {

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

newNode->data = value;

newNode->next = NULL;

newNode->prev = deque->rear;

if (isEmpty(deque)) {

deque->front = newNode;

} else {

deque->rear->next = newNode;

}

deque->rear = newNode;

deque->size++;

printf("Inserted %d at the rear.\n", value);

}

// Function to delete an element from the front of the deque

int deleteFront(Deque\* deque) {

if (isEmpty(deque)) {

printf("Deque is empty. Cannot delete from the front.\n");

return -1; // Return -1 to indicate an error

}

Node\* temp = deque->front;

int value = temp->data;

deque->front = deque->front->next;

if (deque->front == NULL) {

deque->rear = NULL;

} else {

deque->front->prev = NULL;

}

free(temp);

deque->size--;

return value;

}

// Function to delete an element from the rear of the deque

int deleteRear(Deque\* deque) {

if (isEmpty(deque)) {

printf("Deque is empty. Cannot delete from the rear.\n");

return -1; // Return -1 to indicate an error

}

Node\* temp = deque->rear;

int value = temp->data;

deque->rear = deque->rear->prev;

if (deque->rear == NULL) {

deque->front = NULL;

} else {

deque->rear->next = NULL;

}

free(temp);

deque->size--;

return value;

}

// Function to get the element at the front of the deque without removing it

int getFront(Deque\* deque) {

if (isEmpty(deque)) {

printf("Deque is empty. Cannot get the front element.\n");

return -1; // Return -1 to indicate an error

}

return deque->front->data;

}

// Function to get the element at the rear of the deque without removing it

int getRear(Deque\* deque) {

if (isEmpty(deque)) {

printf("Deque is empty. Cannot get the rear element.\n");

return -1; // Return -1 to indicate an error

}

return deque->rear->data;

}

// Function to display the elements of the deque

void displayDeque(Deque\* deque) {

if (isEmpty(deque)) {

printf("Deque is empty.\n");

return;

}

Node\* current = deque->front;

printf("Deque elements are:\n");

while (current != NULL) {

printf("%d ", current->data);

current = current->next;

}

printf("\n");

}

int main() {

Deque deque;

initializeDeque(&deque);

int choice, value;

while (1) {

printf("\nDeque Menu:\n");

printf("1. Insert Front\n");

printf("2. Insert Rear\n");

printf("3. Delete Front\n");

printf("4. Delete Rear\n");

printf("5. Get Front\n");

printf("6. Get Rear\n");

printf("7. Display\n");

printf("8. EXIT\n");

printf("Enter your choice: ");

scanf("%d", &choice);

switch (choice) {

case 1:

printf("Enter the value to insert at the front: ");

scanf("%d", &value);

insertFront(&deque, value);

break;

case 2:

printf("Enter the value to insert at the rear: ");

scanf("%d", &value);

insertRear(&deque, value);

break;

case 3:

value = deleteFront(&deque);

if (value != -1) {

printf("Deleted %d from the front.\n", value);

}

break;

case 4:

value = deleteRear(&deque);

if (value != -1) {

printf("Deleted %d from the rear.\n", value);

}

break;

case 5:

value = getFront(&deque);

if (value != -1) {

printf("Front element is %d.\n", value);

}

break;

case 6:

value = getRear(&deque);

if (value != -1) {

printf("Rear element is %d.\n", value);

}

break;

case 7:

displayDeque(&deque);

break;

case 8:

printf("Exiting...\n");

exit(0);

default:

printf("Invalid choice. Please try again.\n");

}

}

return 0;

}

1. Write a program to implement Priority Queue with insertion and deletion operations.

Answer:

#include <stdio.h>

#include <stdlib.h>

#define MAX 100 // Define the maximum number of elements in the heap

// Define the structure for the priority queue (max-heap)

typedef struct {

int arr[MAX];

int size;

} PriorityQueue;

// Function to initialize the priority queue

void initializeQueue(PriorityQueue\* pq) {

pq->size = 0;

}

// Function to get the parent index of a node

int parent(int i) {

return (i - 1) / 2;

}

// Function to get the left child index of a node

int leftChild(int i) {

return 2 \* i + 1;

}

// Function to get the right child index of a node

int rightChild(int i) {

return 2 \* i + 2;

}

// Function to swap two elements in the heap

void swap(int\* a, int\* b) {

int temp = \*a;

\*a = \*b;

\*b = temp;

}

// Function to insert a new element into the priority queue

void insert(PriorityQueue\* pq, int value) {

if (pq->size == MAX) {

printf("Priority Queue is full. Cannot insert %d.\n", value);

return;

}

// Insert the new value at the end of the heap

pq->arr[pq->size] = value;

pq->size++;

// Fix the max-heap property if it is violated

int i = pq->size - 1;

while (i != 0 && pq->arr[parent(i)] < pq->arr[i]) {

swap(&pq->arr[i], &pq->arr[parent(i)]);

i = parent(i);

}

printf("Inserted %d into the priority queue.\n", value);

}

// Function to delete the maximum element from the priority queue

int deleteMax(PriorityQueue\* pq) {

if (pq->size <= 0) {

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

return -1; // Return -1 to indicate an error

}

if (pq->size == 1) {

pq->size--;

return pq->arr[0];

}

// Store the maximum value and remove it from the heap

int root = pq->arr[0];

pq->arr[0] = pq->arr[pq->size - 1];

pq->size--;

// Heapify the root element to maintain max-heap property

int i = 0;

while (leftChild(i) < pq->size) {

int largest = i;

int left = leftChild(i);

int right = rightChild(i);

if (left < pq->size && pq->arr[left] > pq->arr[largest]) {

largest = left;

}

if (right < pq->size && pq->arr[right] > pq->arr[largest]) {

largest = right;

}

if (largest != i) {

swap(&pq->arr[i], &pq->arr[largest]);

i = largest;

} else {

break;

}

}

return root;

}

// Function to get the maximum element from the priority queue without deleting it

int getMax(PriorityQueue\* pq) {

if (pq->size <= 0) {

printf("Priority Queue is empty. Cannot get the maximum element.\n");

return -1; // Return -1 to indicate an error

}

return pq->arr[0];

}

// Function to display the elements of the priority queue

void displayQueue(PriorityQueue\* pq) {

if (pq->size <= 0) {

printf("Priority Queue is empty.\n");

return;

}

printf("Priority Queue elements are:\n");

for (int i = 0; i < pq->size; i++) {

printf("%d ", pq->arr[i]);

}

printf("\n");

}

int main() {

PriorityQueue pq;

initializeQueue(&pq);

int choice, value;

while (1) {

printf("\nPriority Queue Menu:\n");

printf("1. Insert\n");

printf("2. Delete Max\n");

printf("3. Get Max\n");

printf("4. Display\n");

printf("5. EXIT\n");

printf("Enter your choice: ");

scanf("%d", &choice);

switch (choice) {

case 1:

printf("Enter the value to insert: ");

scanf("%d", &value);

insert(&pq, value);

break;

case 2:

value = deleteMax(&pq);

if (value != -1) {

printf("Deleted max value %d.\n", value);

}

break;

case 3:

value = getMax(&pq);

if (value != -1) {

printf("Maximum value is %d.\n", value);

}

break;

case 4:

displayQueue(&pq);

break;

case 5:

printf("Exiting...\n");

exit(0);

default:

printf("Invalid choice. Please try again.\n");

}

}

return 0;

}