**Assignment-1**

**CS1105 (Design and Analysis of Algorithm)**



**FACULTY GUIDE:**

Mr Devendra Bhavsar

Dr. Renu Jain

**SUBMITTED BY:** Siddhi Nyati

**Roll No:** 2022Btech101

**Define “what is priority queue”.**

A priority queue is a type of abstract data structure that operates similarly to a regular queue or stack, but with a key difference: Each item in the queue has a priority attached to it. Those elements with a higher priority than others are served ahead. The priority queue has a property that the highest priority element is always first and will leave after extraction. Priority queues are widely used in many computer science applications, including graph algorithms operating systems and simulation systems.

*Implementation of Priority Queue:*

Ways to implement a priority queue is by using a heap, Array, Binary search tree, and linked list. However, an efficient way to implement a priority queue is by using a heap data structure.

There are two types of priority queues based on the priority of elements:

* Min Priority Queue: The element with the smallest value has the highest priority.
* Max Priority Queue: The element with the highest value has the highest priority.

**How is it different from the normal queue?**

A priority queue differs from a normal queue in the following ways:

|  |  |  |
| --- | --- | --- |
| Ways in which they differ | Normal Queue | Priority Queue |
| Priority Assignment | No priority is assigned to elements and the FIFO principle has been adopted. The elements are served in the same order they were added. | The elements are prioritized. The higher-priority items are served first irrespective of the sequence in which they have been included. |
| Order of Removal | Items are removed in the order they are enqueued and this follows the FIFO principle. | An element with the highest priority is taken off first regardless of when it was added to the queue. |
| Data Structure | While a normal queue can be implemented using basic data structures like arrays or linked lists | Can be implemented using a variety of data structures, including arrays, linked lists, heap data structures, and binary search trees. |
| Comparable Elements | Does not support comparable elements. | Supports only comparable elements. This allows for efficient retrieval of the highest or lowest priority element. |

**An interactive program for Priority Queue**

I have used the array method to implement the Priority Queue

*Code:*

#include <iostream>

#include <vector>

using namespace std;

struct Process {

int id;

int priority;

};

class PriorityQueue {

private:

vector<Process> processes;

public:

void insertProcess() {

Process newProcess;

cout << "Enter process ID: ";

cin >> newProcess.id;

cout << "Enter priority: ";

cin >> newProcess.priority;

processes.push\_back(newProcess);

cout << "Process with ID " << newProcess.id << " inserted with priority " << newProcess.priority << endl;

}

void deleteProcess() {

if (processes.empty()) {

cout << "Queue is empty. Nothing to delete." << endl;

return;

}

int maxPriorityIndex = 0;

for (int i = 1; i < processes.size(); ++i) {

if (processes[i].priority > processes[maxPriorityIndex].priority) {

maxPriorityIndex = i;

}

}

Process deletedProcess = processes[maxPriorityIndex];

processes.erase(processes.begin() + maxPriorityIndex);

cout << "Process with ID " << deletedProcess.id << " and priority " << deletedProcess.priority << " deleted." << endl;

}

void displayProcesses() {

if (processes.empty()) {

cout << "Queue is empty." << endl;

} else {

cout << "Processes in the queue:" << endl;

for (const Process &p : processes) {

cout << "ID: " << p.id << ", Priority: " << p.priority << endl;

}

}

}

};

int main() {

PriorityQueue priorityQueue;

while (true) {

cout << "\nChoose an operation:" << endl;

cout << "1. Insert\n2. Delete\n3. Display\n4. Exit\n";

int choice;

cin >> choice;

switch (choice) {

case 1:

priorityQueue.insertProcess();

break;

case 2:

priorityQueue.deleteProcess();

break;

case 3:

priorityQueue.displayProcesses();

break;

case 4:

cout << "Exiting the program." << endl;

return 0;

default:

cout << "Invalid choice. Please choose again." << endl;

}

}

return 0;

}

*Output:*

Choose an operation:

1. Insert

2. Delete

3. Display

4. Exit

1

Enter process ID: 101

Enter priority: 3

Process with ID 101 inserted with priority 3

Choose an operation:

1. Insert

2. Delete

3. Display

4. Exit

1

Enter process ID: 201

Enter priority: 1

Process with ID 201 inserted with priority 1

Choose an operation:

1. Insert

2. Delete

3. Display

4. Exit

3

Processes in the queue:

ID: 101, Priority: 3

ID: 201, Priority: 1

Choose an operation:

1. Insert

2. Delete

3. Display

4. Exit

2

Process with ID 101 and priority 3 deleted.

Choose an operation:

1. Insert

2. Delete

3. Display

4. Exit

3

Processes in the queue:

ID: 201, Priority: 1

Choose an operation:

1. Insert

2. Delete

3. Display

4. Exit

4

Exiting the program.