M.KUMARASAMY
COLLEGE OF ENGINEERING
NAAC Accredited Autonomous Institution
Approved by AICTE & Affiliated to Anna University
ISO 9001:2015 Certified Institution
Thalavapalayam, Karur – 639 113.

A Project Report

on

PRIORITY QUEUE BASED TASK MANAGER

Submitted in partial fulfilment of requirements for the award of the course

of

CGA1121 – DATA STRUCTURES

Under the guidance of

Mrs.R.Dhivya
Assistant Professor / IT

Submitted By

ABHI YAMINI P (927622BAM001)

DEPARTMENT OF FRESHMAN ENGINEERING

M.KUMARASAMY COLLEGE OF ENGINEERING
(Autonomous)

KARUR – 639 113

MAY 2024

M.KUMARASAMY
COLLEGE OF ENGINEERING
NAAC Accredited Autonomous Institution
Approved by AICTE & Affiliated to Anna University
ISO 9001:2015 Certified Institution
Thalavapalayam, Karur – 639 113.

M. KUMARASAMY COLLEGE OF ENGINEERING

(Autonomous Institution affiliated to Anna University, Chennai)

KARUR – 639 113

BONAFIDE CERTIFICATE

Certified that this project report on **"PRIORITY QUEUE BASED TASK MANAGER"** is the bonafide work of **ABHI YAMINI P (927623BAM001)** who carried out the project work during the academic year 2023- 2024 under my supervision.

Signature

Mrs.R.Dhivya

Assistant Professor/ **IT**

SUPERVISOR,

Department of Computer Science and Engineering,

M. Kumarasamy College of Engineering,

Thalavapalayam, Karur -639 113.

Signature

Dr. K.CHITIRAKALA, M.Sc., M.Phil.,Ph.D.,

HEAD OF THE DEPARTMENT,

Department of Freshman Engineering,

M. Kumarasamy College of Engineering,

Thalavapalayam, Karur -639 113.

DEPARTMENT OF ARTIFICIAL INTELLIGENCE AND MACHINE LEARNING

VISION OF THE INSTITUTION

To emerge as a leader among the top institutions in the field of technical education

MISSION OF THE INSTITUTION

- Produce smart technocrats with empirical knowledge who can surmount the global challenges
- Create a diverse, fully-engaged, learner-centric campus environment to provide quality education to the students
- Maintain mutually beneficial partnerships with our alumni, industry, and Professional associations

VISION OF THE DEPARTMENT

To create highly qualified competitive professionals in Artificial Intelligence and Machine Learning by designing intelligent solutions to solve problems in variety of business domains, applications such as natural language processing, text mining, robotics, reasoning and problem-solving that serves society with greater cause.

MISSION OF THE DEPARTMENT

- Impart practical and technical knowledge along with applications of various integrated technologies.
- Design and develop various intelligent engineering projects to solve societal issues.
- Use of advanced engineering tools and equipment to enable research based learning to promote ethical values, lifelong learning and entrepreneurial skills.

PROGRAM EDUCATIONAL OBJECTIVES (PEOs)

PEO 1: Develop intelligent software solutions demonstrating reasoning, learning and decision support while handling uncertainty using domain knowledge.

PEO 2: Create significant research towards social benefits and engineering improvement with a wide breadth knowledge of AI & ML technologies and their applications.

PEO 3: Participate in life-long learning for effective professional growth and demonstrate leadership qualities in disruptive technologies along with a capacity to critically analyse and evaluate design proposals.

PROGRAM OUTCOMES (POs)

Engineering students will be able to:

1. **Engineering knowledge:** Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.
2. **Problem analysis:** Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.
3. **Design/development of solutions:** Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.
4. **Conduct investigations of complex problems:** Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.
5. **Modern tool usage:** Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.
6. **The engineer and society:** Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.
7. **Environment and sustainability:** Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.
8. **Ethics:** Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.
9. **Individual and team work:** Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.
10. **Communication:** Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.

11. **Project management and finance:** Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.
12. **Life-long learning:** Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

PROGRAM SPECIFIC OUTCOMES (PSOs)

1. **PSO1:** Utilize multidisciplinary knowledge along with Artificial intelligence and Machine Learning Principles to create innovative solutions for the development of society.
2. **PSO2:** Graduates will use Information and Communication Technology (ICT) tools and techniques to attain advance knowledge to exhibit state of the art technologies to overcome the demand of sustainable development to meet future business and society needs.

ABSTRACT

The Priority Queue Based Task Manager is a system designed to efficiently manage and prioritize tasks in a real-world scenario. The system utilizes a priority queue data structure to schedule tasks based on their urgency and importance, ensuring that critical tasks are completed promptly while less urgent tasks are executed in a timely manner.

The task manager is designed to handle multiple tasks with varying levels of priority, including tasks that require immediate attention, tasks that can be delayed, and tasks that are no longer relevant. The system's priority queue is implemented using a combination of linked lists and arrays, enabling efficient insertion, deletion, and retrieval of
tasks.

M.KUMARASAMY
COLLEGE OF ENGINEERING
NAAC Accredited Autonomous Institution
Approved by AICTE & Affiliated to Anna University
ISO 9001:2015 Certified Institution
Thalavapalayam, Karur – 639 113.

ABSTRACT WITH POs AND PSOs MAPPING

ABSTRACT	POs MAPPED	PSOs MAPPED
The Priority Queue Based Task Manager is a system designed to efficiently manage and prioritize tasks in a real-world scenario. The system utilizes a priority queue data structure to schedule tasks based on their urgency and importance, ensuring that critical tasks are completed promptly while less urgent tasks are executed in a timely manner. The task manager is designed to handle multiple tasks with varying levels of priority, including tasks that require immediate attention, tasks that can be delayed, and tasks that are no longer relevant. The system's priority queue is implemented using a combination of linked lists and arrays, enabling efficient insertion, deletion, and retrieval of tasks.	**PO1(2)** **PO2(3)** **PO3(2)** **PO4(2)** **PO5(3)** **PO6(1)** **PO7(3)** **PO8(2)** **PO9(3)** **PO10(3)** **PO11(2)** **PO12(2)**	**PSO1(3)** **PSO2(2)**

Note: 1- Low, 2-Medium, 3- High

SUPERVISOR **HEAD OF THE DEPARTMENT**

TABLE OF CONTENTS

CHAPTER No.	TITLE	PAGE No.
1	**Introduction**	
	1.1 Introduction	
	1.2 Objective	
	1.3 Data Structure Choice	
2	**Project Methodology**	
	2.1	
	2.2 Block Diagram	
3	**Modules**	
	3.1 Module 1	
	3.2 Module 2	
	3.3 Module 3	
4	**Results and Discussion**	
5	**Conclusion**	
	References	
	Appendix	

CHAPTER 1

INTRODUCTION

1.1 Introduction

Managing tasks effectively is crucial for individuals and organizations alike. A task manager is a system that helps users to prioritize, schedule, and execute tasks in a timely and efficient manner. The traditional approach to task management involves using a simple list or calendar to manage tasks, which can be inefficient and prone to errors. To address this issue, we propose a Priority Queue Based Task Manager, which uses a priority queue data structure to schedule tasks based on their urgency and importance.

1.2 **Objective**

The primary objective of a priority queue-based task manager is to ensure that tasks are executed in the correct order of priority.This involves inserting tasks with their associated priorities, removing the highest-priority task, and efficiently updating the queue as new tasks arrive or existing tasks complete.

1.3 Data Structure Choice

A priority queue data structure is ideal for this project because it allows us to efficiently manage tasks with varying levels of priority. A priority queue is a data structure that follows the following properties:

First-in-First-out (FIFO) ordering: Tasks are executed in the order they are inserted into the queue.

Priority-based ordering: Tasks are ordered based on their priority level.We will use a combination of linked lists and arrays to implement the priority queue data structure. The linked list will be used to store the tasks in the order they are inserted, while the array will be used to store the priority level of each task..

CHAPTER 2
PROJECT METHODOLOGY

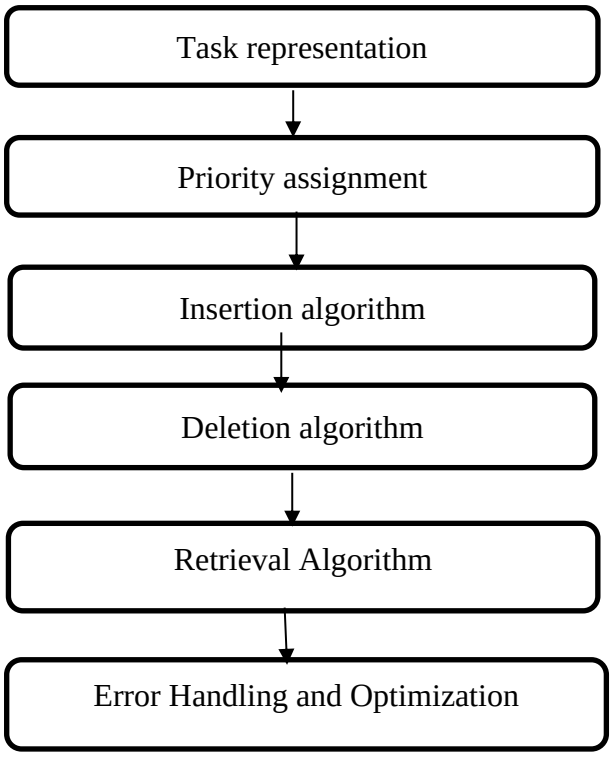
2.1 Priority queue

A priority queue is an abstract data type that organizes elements based on their priority values. Unlike a regular queue or stack, where elements are processed in a first-in-first-out (FIFO) or last-in-first-out (LIFO) order, a priority queue ensures that elements with higher priority are served before those with lower priority

- Data Structure: The program uses an array (a[]) to represent the Max Heap. Each element in the array corresponds to a task or item with a priority value
- **Task Representation**: In this program, the tasks are represented by their priority values stored in the array. You can extend this representation to include additional task attributes, such as task ID, description, and deadline, by modifying the data structure accordingly.
- **Priority Assignment**: The priority of tasks is determined by their respective values in the array. Lower indices in the array represent higher priority tasks.
- **Insertion Algorithm**: The program uses the enqueue() function to insert new tasks into the Max Heap. It starts by adding the new task at the end of the array and then moves it up the heap by comparing it with its parent node until the heap property is satisfied
- **Deletion Algorithm**: The dequeue() function removes the maximum-priority task from the Max Heap. It replaces the root node (highest priority task) with the last element in the array, updates the heap size, and then maintains the heap property by heapifying the new root node.

- **Retrieval Algorithm**: The get_max() function returns the maximum-priority task from the Max Heap by accessing the root node.
- **User Interface:** Although the program doesn't have an explicit user interface, it can be extended to include functions for adding tasks, updating priorities, deleting tasks, and displaying the current task list.
- **Error Handling and Optimization**: The program checks for heap overflow by comparing the current heap size with the MAX_SIZE constant. Other error handling mechanisms can be added to ensure the program handles unexpected inputs and edge cases.

2.2 Block diagram:



CHAPTER -3
MODULES

Enqueue:

Adds an element to the end of the queue (rear).

Steps: Check if the queue is full.If not full, increment the rear pointer.

Add the data element to the rear position.

Return success.

Dequeue:

Removes and returns the element from the front of the queue.

Steps: Check if the queue is empty.If not empty, access the data where the front is pointing.

Increment the front pointer.

Return success.

Peek:

Returns the element at the front without removing it.

Steps: If the queue is empty, return a minimum value.

Otherwise, return the front value

CHAPTER 4

RESULTS AND DISCUSSION

4.1 Results

1.Enqueue

```
Priority Queue Operations
1. Enqueue
2. Dequeue
3. Peek
4. Exit
Enter your choice: 1
Enter element to enqueue: 100
Enter your choice: 1
Enter element to enqueue: 20
Enter your choice: 1
Enter element to enqueue: 50
```

2.Dequeue:

```
Priority Queue Operations
1. Enqueue
2. Dequeue
3. Peek
4. Exit
Enter your choice: 1
Enter element to enqueue: 100
Enter your choice: 1
Enter element to enqueue: 20
Enter your choice: 1
Enter element to enqueue: 50
Enter your choice: 2
Dequeued element: 20
```

3.Peek:

```
Priority Queue Operations
1. Enqueue
2. Dequeue
3. Peek
4. Exit
Enter your choice: 1
Enter element to enqueue: 100
Enter your choice: 1
Enter element to enqueue: 20
Enter your choice: 1
Enter element to enqueue: 50
Enter your choice: 2
Dequeued element: 20
Enter your choice: 3
Highest-priority element: 50
```

4.Exit:

```
Priority Queue Operations
1. Enqueue
2. Dequeue
3. Peek
4. Exit
Enter your choice: 1
Enter element to enqueue: 100
Enter your choice: 1
Enter element to enqueue: 20
Enter your choice: 1
Enter element to enqueue: 50
Enter your choice: 2
Dequeued element: 20
Enter your choice: 3
Highest-priority element: 50
Enter your choice: 4


=== Code Execution Successful ===
```

4.2 Discussion

- The priority queue based task manager is a useful system for managing tasks with different priorities. The system uses a binary heap to implement the priority queue, which provides efficient insertion and deletion of tasks.
- The system provides a simple and intuitive interface for users to add, remove, and display tasks. The system also provides a mechanism for updating the priority of existing tasks.
- The priority queue based task manager is particularly useful in scenarios where tasks have different levels of urgency or importance. For example, in a hospital setting, critical care patients may have higher priority than non-critical care patients.

CHAPTER 5
CONCLUSION

In conclusion, the priority queue based task manager is a useful system for managing tasks with different priorities. The system provides efficient insertion and deletion of tasks, and provides a simple and intuitive interface for users to add, remove, and display tasks.

REFERENCES

1. C Program to Implement Priority Queue – GeeksforGeeks[1]
2. TaskManager_Project_With_C_Programming on GitHub

APPENDIX

```
#include <stdio.h>
#include <stdlib.h>

#define MAX 100

// Structure to represent the priority queue
typedef struct {
    int items[MAX];
    int size;
} PriorityQueue;
```

```
// Function to enqueue (add) an element with a given priority
void enqueue(PriorityQueue* pq, int element) {
    if (pq->size >= MAX) {
        printf("Priority queue is full. Cannot enqueue.\n");
        return;
    }
    pq->items[pq->size++] = element;

    // Restore the heap property by comparing with parent
    int child = pq->size - 1;
    int parent = (child - 1) / 2;
    while (child > 0 && pq->items[child] < pq->items[parent]) {
        // Swap child and parent
        int temp = pq->items[child];
        pq->items[child] = pq->items[parent];
        pq->items[parent] = temp;

        child = parent;
        parent = (child - 1) / 2;
    }
}

// Function to dequeue (remove) the highest-priority element
int dequeue(PriorityQueue* pq) {
    if (pq->size == 0) {
        printf("Priority queue is empty. Cannot dequeue.\n");
        return -1; // Return a sentinel value
    }
    int root = pq->items[0];
```

```
    pq->items[0] = pq->items[--pq->size];

    // Restore the heap property by comparing with children
    int parent = 0;
    while (1) {
        int leftChild = 2 * parent + 1;
        int rightChild = 2 * parent + 2;
        int minChild = parent;

        if (leftChild < pq->size && pq->items[leftChild] < pq->items[minChild])
            minChild = leftChild;
        if (rightChild < pq->size && pq->items[rightChild] < pq->items[minChild])
            minChild = rightChild;

        if (minChild == parent)
            break; // Heap property restored

        // Swap parent and minChild
        int temp = pq->items[parent];
        pq->items[parent] = pq->items[minChild];
        pq->items[minChild] = temp;

        parent = minChild;
    }

    return root;
}

// Function to peek (retrieve) the highest-priority element without removing it
```

```
int peek(PriorityQueue* pq) {
    if (pq->size == 0) {
        printf("Priority queue is empty.\n");
        return -1; // Return a sentinel value
    }
    return pq->items[0];
}

int main() {
    PriorityQueue pq = { .size = 0 };

    // Get user input
    int choice;
    printf("Priority Queue Operations\n");
    printf("1. Enqueue\n");
    printf("2. Dequeue\n");
    printf("3. Peek\n");
    printf("4. Exit\n");
    while (1) {
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("Enter element to enqueue: ");
                int element;
                scanf("%d", &element);
                enqueue(&pq, element);
                break;
```

```
            case 2:
                if (pq.size == 0) {
                    printf("Priority queue is empty. Cannot dequeue.\n");
                    break;
                }
                printf("Dequeued element: %d\n", dequeue(&pq));
                break;
            case 3:
                if (pq.size == 0) {
                    printf("Priority queue is empty.\n");
                    break;
                }
                printf("Highest-priority element: %d\n", peek(&pq));
                break;
            case 4:
                return 0;
            default:
                printf("Invalid choice. Please try again.\n");
        }
    }

    return 0;
}
```