جامـــــعـة المـــــسـتـقـبـل
AL MUSTAQBAL UNIVERSITY

كلية العلوم

قـســــــم الانـــظـــمـــة الـــطـبـيـة الـــذكـــيـــة

Intelligent Medical Systems Department

Lecture: (4)

Priority Queue

Subject: Data structures

Class: second

Lecturer: Prof.Dr. Mehdi Ebady Manaa

Programmer:- Fatima Hussein Jawad

Study Year: 2024-2025

What is a Priority Queue?

A Priority Queue is a data structure similar to a regular queue, but it relies on the priority of each element rather than just its position. In traditional queues, elements are added to the back and removed from the front according to the FIFO (First In, First Out) principle. However, in a Priority Queue, each element has a priority value, and elements are organized in the queue based on this priority.

Applications of Priority Queue :

Priority Queues are used in various applications, including:

1. Task Scheduling, where priority is given to more important tasks.
2. Algorithms such as Dijkstra's algorithm for finding the shortest path.
3. Operating Systems to determine the priority of processes.

A Priority Queue supports the following operations:

1. **Enqueue/Insert:** Add a new element based on a specified priority.
2. **Dequeue/Delete:** Remove the element with the highest priority.
3. **Peek:** Access the value of the highest-priority element without removing it.

We'll start by creating a Priority Queue as a list and implementing fundamental operations on it without using any external libraries or classes.

1. Creating the Priority Queue and Inserting Elements:

```
priority_queue = []          # Create an empty list to represent the priority queue

priority_queue.append(5)         # Insert elements into the priority queue

priority_queue.append(1)

priority_queue.append(3)

priority_queue.append(10)

priority_queue.append(2)

# Sort the list so that the smallest element (highest priority) is at the front

priority_queue.sort()

print("Priority queue after inserting elements:", priority_queue)
```

output:

```
Priority queue after inserting elements: [1, 2, 3, 5, 10]
```

2. Accessing the Highest Priority Element (Peek)

To view the element with the highest priority (the smallest element in this case), simply check the first element in the list.

```
# Peek at the highest priority element without removing it
```

```
if priority_queue:
    print("Highest priority element (Peek):", priority_queue[0])
else:
    print("The priority queue is empty.")
```

output:

```
Highest priority element (Peek): 1
```

3. Removing the Highest Priority Element (Dequeue)

To remove the element with the highest priority, we remove the first element in the list.

```
# Remove and print elements in priority order
print("Removing elements from the priority queue:")
while priority_queue:
    highest_priority = priority_queue.pop(0)  # Remove the highest priority element
    print("Removed element:", highest_priority)
    print("Current state of the priority queue:", priority_queue)
```

output:

```
Removed element: 1
Current state of the priority queue: [2, 3, 5, 10]
 Removed element: 2
Current state of the priority queue: [3, 5, 10]
```

```
Removed element: 3

Current state of the priority queue: [5, 10]

Removed element: 5

Current state of the priority queue: [10]

Removed element: 10

Current state of the priority queue: []
```

Task 2: Arithmetic Operations on the Priority Queue Elements :

Now, let's perform basic arithmetic operations on the elements of the Priority Queue.

1. Sum of All Elements:

We calculate the sum of all elements in the queue.

```
# Sum of all elements in the priority queue

priority_queue = []

priority_queue.append(5)

priority_queue.append(1)

priority_queue.append(3)

priority_queue.append(10)

priority_queue.append(2)

priority_queue.sort()

print("Priority queue after inserting elements:", priority_queue)
```

```
if priority_queue:
    print("Highest priority element (Peek):", priority_queue[0])
else:
    print("The priority queue is empty.")
sum_elements = 0
for item in priority_queue:
    sum_elements += item
print("Sum of elements in the priority queue:", sum_elements)
```

output:

```
Priority queue after inserting elements: [1, 2, 3, 5, 10]
Highest priority element (Peek): 1
Sum of elements in the priority queue: 21
```

2. Subtraction

Subtract each element from the highest priority element (the first element).

```
# Subtract elements from the highest priority element
if priority_queue:
    difference = priority_queue[0]
    for item in priority_queue[1:]:
        difference -= item
    print("Result of subtracting all elements from the highest priority element:", difference)
```

output:

```
Result of subtracting all elements from the highest priority element: -19
```

3. Division:

Divide the highest priority element by each other element in the queue, if possible.

```
# Divide the highest priority element by other elements
if priority_queue[0] != 0:
    division_results = [priority_queue[0] / item if item != 0 else 'Error: Division by zero' for item in priority_queue[1:]]
    print("Results of dividing the highest priority element by other elements:", division_results)
else:
    print("Error: The highest priority element is zero, division not possible.")
```

output:

```
Results of dividing the highest priority element by other elements: [0.5, 0.3333333333333333, 0.2, 0.1]
```

4. Multiplication:

Calculate the product of all elements in the queue.

```
# Product of all elements in the priority queue
product = 1
for item in priority_queue:
   product *= item
print("Product of elements in the priority queue:", product)
```

output:

```
Product of elements in the priority queue: 300
```

5. Modulus (Remainder)

Compute the remainder when the highest priority element is divided by each of the other elements.

```
# Modulus of the highest priority element with other elements
modulus_results = [priority_queue[0] % item for item in priority_queue[1:] if item != 0]
print("Results of modulus operation (highest priority element % other elements):", modulus_results)
```

output:

```
Results of modulus operation (highest priority element % other elements): [1, 1, 1, 1]
```

6. Checking for Even and Odd Numbers:

Identify even and odd numbers in the priority queue.

```python
# Check for even and odd numbers
even_numbers = [item for item in priority_queue if item % 2 == 0]
odd_numbers = [item for item in priority_queue if item % 2 != 0]
print("Even numbers in the priority queue:", even_numbers)
print("Odd numbers in the priority queue:", odd_numbers)
```

output:

```
Even numbers in the priority queue: [2, 10]
Odd numbers in the priority queue: [1, 3, 5]
```

➢ **Summary of Operations**

- **Sum:** Summing all elements in the priority queue.
- **Subtraction**: Subtracting elements starting from the highest priority element.
- **Division:** Dividing the highest priority element by each other element, handling division by zero.
- **Multiplication:** Calculating the product of all elements.
- **Modulus:** Calculating the remainder of the division of the highest priority element by each other element.
- **Even/Odd Check**: Identifying which elements are even or odd.