Lesson:

Priority Queue One Shot

PW SKILLS

Pre-Requisites

- Queues
- Heaps

List of Concepts Involved

- Introduction
- Insertion and deletion in a priority queue
- Types of priority queue
- Time complexity analysis
- Implementation of priority queue using heap

Topic 1: Introduction

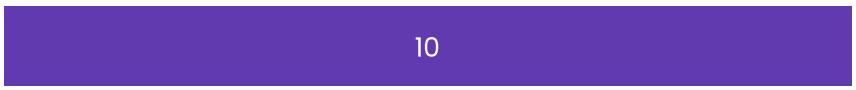
Priority queues are special types of queues where each element is assigned a priority value, which is generally related to the value stored in the element. The arrangement of the elements are based on their priority values. The more priority an element has the closer it will be to the front. For instance, in a school assembly line the person with a smaller height is placed towards the front and a person with a larger height is placed towards the back.

There are several ways to implement a priority queue, including using an array, linked list, heap, or binary search tree. Each method has its own advantages and disadvantages, and the best choice will depend on the specific needs of your application.

Topic 2: Insertion and deletion in a priority queue Insertion

The insertion in a priority queue is different from the insertion in a normal queue. In a normal queue, the elements were added from the rear but in the case of priority queue, the elements are placed according to their priority value. For instance, let's say that we have a priority queue that arranges its elements in the decreasing order of their values. Initially the queue is empty.

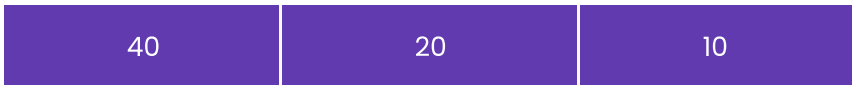
push(10)



push(40)

Since 40 has a value greater than 10, it has a higher priority. Consequently it will be placed before it.

push(2)



20 has a priority value that is greater than 10 and lower than 40, so it will be placed between them.

push(5)

5 has the least priority value among the elements of the priority queue, so it will be placed at the end.

Deletion

Like a normal queue, elements in a priority queue are removed from the front. For instance, if we continue with the above mentioned queue.

pop()

40 is removed from the priority queue.

pop()

20 is removed from the priority queue.

Topic 3: Types of priority queue

Based on the arrangement of the elements in the priority queue, it is classified into 2 categories-

1. Min priority queueIf the element with the smallest value has the highest priority, then that priority queue is called the min priority queue.
2. Max priority queueIf the element with a higher value has the highest priority, then that priority queue is known as the max priority queue

Topic 4: Time complexity analysis

There are various ways by which we can implement a priority queue. Each one has its own advantages and disadvantages. Given below is the time complexity analysis of the various ways by which we can implement a priority queue-

Implementation	push	pop	peek
Array	O(1)	O(n)	O(n)
Linked list	O(1)	O(n)	O(n)
Binary heap	O(logn)	O(logn)	O(1)
Binary search tree	O(logn)	O(logn)	O(1)

Among these data structures, heap data structure provides an efficient implementation of priority queues. Hence, it is generally preferred.

Topic 5: Priority queue in C++ STL

In C++ STL, we have a predefined class for priority queue. By default it displays the elements in decreasing order of their priority. But we can change the order of the elements based on our preference.

Declaration

To use lists in a program, we need to start by including the queue header.

```
#include <queue>
```

There are 3 ways by which we can declare a priority queue-

1. **Max priority queue(default)**

```
priority_queue<data_type> name_of_priority_queue;
```

Example:

```
   a. priority_queue<int> pq1;
```

It will create a priority queue that will arrange the integers stored in it in descending order of their value.

```
   b. priority_queue<char> pq2;
```

It will create a priority queue that will arrange the characters stored in it in descending order of their ASCII value.

2. **Min priority queue**

```
priority_queue<data_type, vector<data_type>, greater<data_type>> name_of_priority_queue;
```

Example:

```
  c. priority_queue<int, vector<int>, greater<int>> pq1;
```

It will create a priority queue that will arrange the integers stored in it in ascending order of their value.

```
  d. priority_queue<char, vector<char>, greater<char>> pq2;
```

It will create a priority queue that will arrange the characters stored in it in ascending order of their ASCII value.

Member function of priority queue in STL

Function	Use
push(obj)	Inserts the object into the priority queue.
pop()	Removes the object from the top of the priority queue.
top()	Returns the first element of the priority queue.
size()	Returns the size of the priority queue.
empty()	Returns 1 if the priority queue is empty, else returns 0.

Example

1. **Max priority queue**

https://pastebin.com/v3Ap8rYu

2. **Min priority queue**

https://pastebin.com/6F6v5TKR

Q1 Given an integer array nums and an integer k, return the kth largest element in the array.
Note that it is the kth largest element in the sorted order, not the kth distinct element.

Input:
nums: [3, 7, 2, 9, 5]
k: 3

Output:
Ans = 5

Approach - 1
Code: https://pastebin.com/JGefQW6e

Steps

- Create a priority queue to store the elements in decreasing order.
- Traverse through the array and push the elements into the priority queue.
- Remove the elements from the priority queue till the kth element comes at the front.
- Return the front element of the queue.

Approach - 2
Code: https://pastebin.com/AHXYdnHq

Steps

- Create a priority queue to store the k largest elements in increasing order.
- Traverse through the array and push the elements into the priority queue.
- At each iteration if the size of the queue becomes greater than k, remove the front element of the queue. This ensures that the queue always contains the k largest elements.
- Return the front element of the queue.

Q2) You are given a set of points in a XY plane. You need to find the k closest points to the origin using manhattan distance.
It is guaranteed that no 2 points will have the same distance from the origin.

Input:
n = 5
k = 3
Points:
5 3
1 2
3 4
5 6
7 8
9 10

Output:
1 2
3 4
5 3

Code: https://pastebin.com/bYgL2QD5

Steps

- Create a priority queue to store the pair of the distance between each point from the origin and the point itself.
- For each point, calculate its distance from the origin and store the distance and the point in the priority queue.
- If the size of the priority queue becomes greater than k, remove the front element of the queue.
- Convert the queue into a vector and return.

Q3 Given an array of n integers. You are supposed to perform k operations on it. At each operation, the smallest 2 elements of the array are removed from the array, multiplied by each other, and the product is added back to the array.
Return the largest element of the array after the k operations.
It is guaranteed that at least one element will remain after the k operations.
Input:
n = 5, k = 3
Array = {2 4 3 1 5}

Output:
20

Code: https://pastebin.com/v2RJbF5b

Steps

- Create a priority queue to store the elements of the array in increasing order.
- At each operation, take out the top 2 elements of the priority queue and insert their product back into the queue.
- Remove all the elements from the queue except one.
- Return the last element of the queue.

Q4. Given a characters array tasks, representing the tasks a CPU needs to do, where each letter represents a different task. Tasks could be done in any order. Each task is done in one unit of time. For each unit of time, the CPU could complete either one task or just be idle.
However, there is a non-negative integer n that represents the cooldown period between two same tasks (the same letter in the array), that is that there must be at least n units of time between any two same tasks.
Return the least number of units of times that the CPU will take to finish all the given tasks.
Input: tasks = ["A","A","A","B","B","B"], n = 2
Output: 8
Explanation:
A -> B -> idle -> A -> B -> idle -> A -> B
There is at least 2 units of time between any two same tasks.
Code link: https://pastebin.com/2ctmUq98

Explanation:

- After counting the frequencies of each task, the frequencies are pushed into the priority queue using pq.push().
- The totalTime variable is initialized to keep track of the total time needed to complete all the tasks.

- The main logic is implemented in a while loop that continues until the priority queue is empty.
- Inside the while loop, a temporary vector called temp is created to store the tasks that are currently being executed. This vector will be used to add the remaining tasks back to the priority queue later.
- A nested for loop iterates from 0 to n, which represents the cooldown period. Within this loop, tasks are extracted from the priority queue using pq.top(), and if the frequency is greater than 1, the remaining tasks (freq - 1) are added to the temp vector.
- The totalTime is incremented for each iteration of the nested for loop.
- After the nested for loop finishes, the remaining tasks in the temp vector are added back to the priority queue using pq.push().
- The while loop continues until both the priority queue and the temporary vector temp are empty, indicating that all tasks have been processed.
- Finally, the totalTime is returned as the least number of units of time required to finish all the tasks.
- In the main function, a sample input is provided, and the leastInterval function is called with the tasks and cooldown period. The result is then printed.

Output:

```
Least number of units of time: 8


...Program finished with exit code 0
Press ENTER to exit console.
```

Q5. Given a stream of integers, find the median of the stream using two priority queues.

Solution Code: https://pastebin.com/1cxvzySx

Output:

```
1.5
2
3.5
4
```

Explanation:

- To find the median of a stream of integers, we can maintain two priority queues: a max-heap to store the lower half of the stream and a min-heap to store the upper half of the stream.
- When a new number is added to the stream, we compare it with the current median and add it to the appropriate heap.
- We balance the heaps by moving an element from the larger heap to the smaller heap if necessary.
- The median will either be the maximum value in the max-heap or the average of the maximum value in the max-heap and the minimum value in the min-heap, depending on whether the stream has an odd or even number of elements.

Time complexity: O(log n) for adding a number and O(1) for finding the median. So overall is O(log n) where n is the number of elements in the stream.

Space complexity: O(n) where n is the number of elements in the stream.

Upcoming lecture

- Graphs