Heap and Priority Que
in Python

Difficulty Level : Medium • Last Updated : 01 Oct, 2020

Heaps are widely used tree-like data structures in which the parent nodes satisfies any one of the criteria given below.

- The value of the parent node in each level is less than or equal to its children's values – **min-heap.**
- The value of the parent node in each level higher than or equal to its children's values – **max-heap.**

The heaps are complete binary trees and are used in the implementation of the priority queues. The min-heaps play a vital role in scheduling jobs, scheduling emails or in assigning the resources to tasks based on the priority.

Priority queues

These are abstract data types and are a special form of queues. The elements in the queue have priorities assigned to them. Based on the priorities, the first element in the priority queue will be the one with the highest priority. The basic operations associated with these priority queues are listed below:

- **is_empty:** To check whether the queue is empty.
- **insert :** To insert an element along with its priority. The element will be placed in the order of its priority only.
- **pop :** To pop the element with the highest priority. The first element will be the

The priority queues can be used for all scheduling kind of processes. The programmer can decide whether the largest number is
number will be considered as the highest
then they appear in the order in which the

heapq mo

Heapq module is an implementation of he
in which the property of min-heap is pres
rearranges it such that they satisfy the following criteria of min-heap:

- The parent node in index **'i'** is less than or equal to its children.
- The left child of a node in index **'i'** is in index **'(2*i) + 1'**.
- The right child of a node in index **'i'** is in index **'(2*i) + 2'**.

Priority queues using heapq module

The priority queue is implemented in Python as a list of tuples where the tuple contains the priority as the first element and the value as the next element.

Example : [(1, 2), (2, 3), (4, 5), (6,7)]

consider (1,2) :

- ***Priority : 1***

Example:

Consider a simple priority queue impleme
students based on their roll number. Here
to present. Since it is a min-heap, roll nun

Python3

```
# import modules
import heapq as hq

# list of students
list_stu = [(5,'Rina'),(1,'Anish'),(3,'Moana'),(2,'cathy'),(4,'Lucy')]

# Arrange based on the roll number
hq.heapify(list stu)
```


Output

```
The order of presentation is :
1 : Anish
2 : cathy
3 : Moana
5 : Rina
4 : Lucy
```

Example 2:

Now let us implement a simple scheduler that assigns the jobs to the processor. The priority queue is used by the scheduler to decide which task has to be performed. Apart from the tasks, there will be interrupts approaching the scheduler. So the scheduler has to decide whether to execute the interrupt or the existing task. If the interrupt has a higher priority, it is executed first otherwise, once all the jobs are completed, the interrupt will be serviced. To implement this the heapq module is used. The approach is

- The tasks to be executed are assigned with priorities. The element that has '1' as priority is considered to be the most im
- All the tasks are in a priority queue and
- The tasks are serviced and while in pro
execution log stating which task is in p
- The interrupts along with their priorit
- The interrupts are pushed into the pric
- The task/interrupt with the highest pri
first element in the queue.
- Once a task.interrupt is serviced, it is popped out from heap queue.

Python3

```
import time
import heapq as hq

# jobs to be executed
jobs = [(2, 'task_1'), (5, 'task_2'), (1, 'task_4'),
        (4, 'task_5'), (3, 'task_3'), (1, 'task_8')]

# interrupts
interrupts = [(1, 'intr_1'), (2, 'intr_2'), (13, 'intr_3')]

i, j = 0, 0

# Arranging jobs in heap
hq.heapify(jobs)

print(jobs, "\n\n")

# scheduling the tasks
while len(jobs) != 0:

    # printing execution log
    print("The ", jobs[0][1], " with priority ",
          jobs[0][0], " in progress", end="")

    # servicing the tasks
    for _ in range(0, 5):

        print(".", end="")
        time.sleep(0.5)

    # pop the job that completed
```

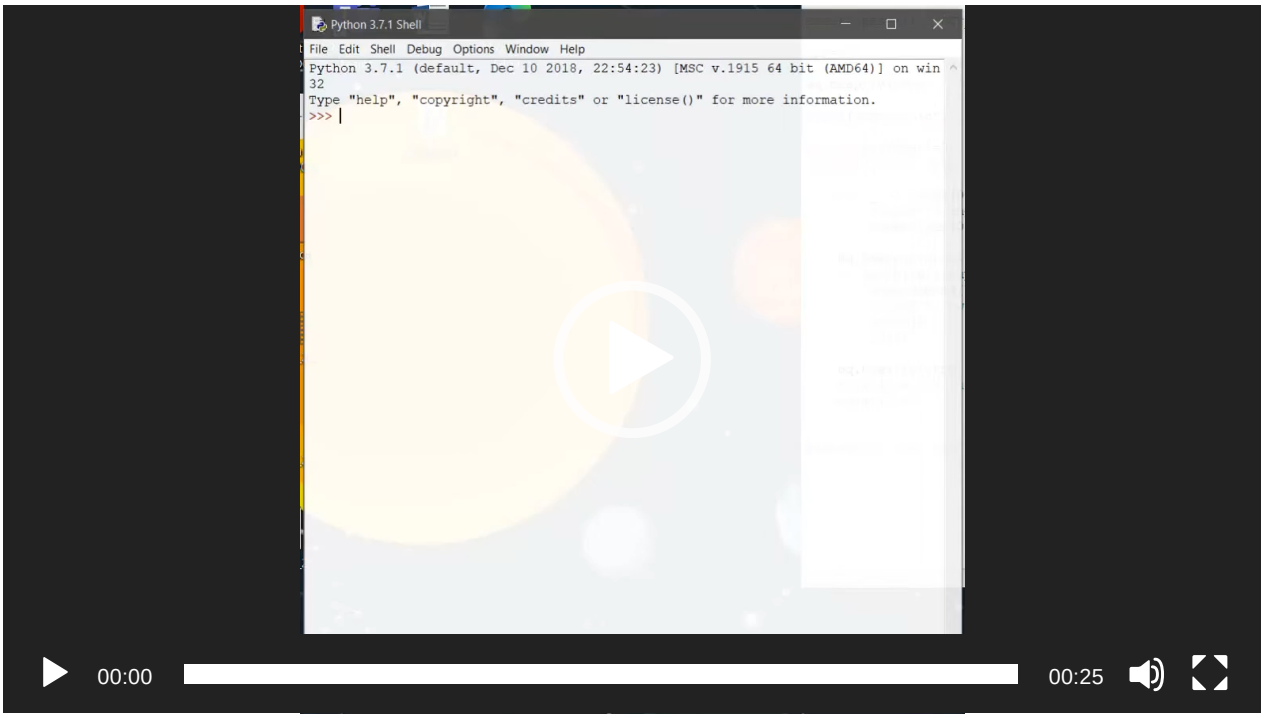
```
    # adding interrupts
    if j < len(interrupts):

        hq.heappush(jobs, interrupt
        print("\n\nNew interrupt ar
        print()
        j = j+1

    # job queue after arrival of in
    print("\n Job queue currently :
    print("\n")


print("\nAll interrupts and jobs completed!")
```


Output



Like 0

Previous

Next

RECOMMENDED ARTICLES

Page : 1 2 3

01 **Heap queue (or heapq) in Python**
27, Sep 16

02 **Priority Queue using Queue and Heapdict module in Python**
13, Mar 20

03 **Difference between Binary Heap, Binomial Heap and Fibonacci Heap**
02, Nov 20

05 **Priority Queue using Binary Heap**
16, Sep 20

06 **Why is Binary Heap Preferred over BST for Priority Queue?**
07, Sep 15

07 **Difference between Circular Queue and Priority Queue**
27, Aug 20

04 priority queue or heap?
25, May 17

queue Module

Article Contributed By :

erakshaya485
@erakshaya485

Vote for difficulty

Current difficulty : Medium

Easy | Normal | Medium | Hard | Expert

Article Tags : Data Structures-Heap, priority-queue, Python

Practice Tags : priority-queue

Improve Article | Report Issue

Writing code in comment? Please use ide.geeksforgeeks.org, generate link and share the link here.

Load Comments

Company

About Us

Careers

Privacy Policy

Contact Us

Copyright Policy

Data Structures

Languages

CS Subjects

Video Tutorials

Web Development

HTML

CSS

JavaScript

Bootstrap

Contribute

Write an Article

Write Interview Experience

Internships

Videos