2254. Design Video Sharing Platform

We can use the following data structures for this problem:

increment the likes or dislikes count in the respective maps.

1. A priority queue usedIds to keep track of the available (deleted) videoIds in increasing order.

2. An integer currVideoId initialized as 0 to keep track of the new videoId to be assigned for the next upload.

You are required to design and implement a simple video sharing platform. The platform allows users to perform the following actions:

Problem Description

dislikes for each video.

1. Upload videos: Each video is represented as a string of digits, where the ith digit of the string represents the content of the video at minute i. For example, the first digit represents the content at minute 0 in the video, the second digit represents the content at minute 1 in the video, and so on. When a video is uploaded, it is assigned the smallest available integer videoId starting from 0.

2. Delete videos: Users can delete videos by specifying the videoId. Once a video is deleted, the videoId associated with that video can be reused for another video. 3. Watch videos: Viewers can watch a part of the video by specifying the videoId, startMinute, and endMinute. The platform should return the content of the video for the specified duration.

4. Like and dislike videos: Viewers can like and dislike videos by specifying the videoId. The platform should keep track of the number of likes and

Like and dislike: The viewer likes the video with videoId 0. Now the video has 1 like. Another viewer dislikes the video. Now it

Example Upload a video: Let's assume a user uploads a video with content "12345". The platform assigns videoId 0 to this video.

5. Get statistics: The platform should be able to provide the number of views, likes, and dislikes for a given video by its videoId.

View a video: A viewer watches the video with videoId 0 from minute 1 to minute 3. The platform should return the content "234".

has 1 dislike. Get statistics: The platform should return the statistics for the video with videoId 0 ⇒ views: 1, likes: 1, dislikes: 1.

Delete a video: A user deletes the video with videoId 0. Upload another video: A user uploads a new video with content "678910". Since 0 was freed up after deleting the previous

video, the platform assigns videoId 0 to this new video.

Approach

for each video by its videoId. The main idea is to use these data structures to efficiently perform the required actions like upload, delete, watch, like, dislike,

and get statistics.

Algorithm

For video upload, first check if the usedIds priority queue is empty. If empty, assign the current currVideoId to the video and increment it by 1. If not empty, get the minimum videoId from the priority queue, remove it, and assign to the video. In both cases, add the video content to the videoIdToVideo map. For video delete, first check if the video with the given videoId exists in the videoIdToVideo map. If yes, add the videoId to the usedIds priority queue and remove the videoId from all the maps. For watching a video, first check if the video with the given videoId exists in the videoIdToVideo map. If yes, increment the

views count in the videoIdToViews map, and return the video content for the specified duration. If not, return "-1".

3. A set of maps videoIdToVideo, videoIdToViews, videoIdToLikes, and videoIdToDislikes to store the video content, views, likes, and dislikes

For getting statistics, first check if the video with the given videoId exists in the videoIdToVideo map. If yes, return the

python

from queue import PriorityQueue

def getVideoId(self):

return videoId

return videoId

return "-1"

remove(videoId) {

like(videoId) {

unlike(videoId) {

} else {

getViews(videoId) {

getLikesAndDislikes(videoId) {

return [-1];

if (this.videoIdToVideo.has(videoId)) {

this.videoIdToVideo.delete(videoId);

this.videoIdToViews.delete(videoId);

this.videoIdToLikes.delete(videoId);

this.videoIdToDislikes.delete(videoId);

const video = this.videoIdToVideo.get(videoId);;

return video.slice(startMinute, startMinute + duration);

this.videoIdToViews.set(videoId, (this.videoIdToViews.get(videoId) || 0) + 1);

this.videoIdToLikes.set(videoId, (this.videoIdToLikes.get(videoId) || 0) + 1);

this.videoIdToDislikes.set(videoId, (this.videoIdToDislikes.get(videoId) || 0) + 1);

return [this.videoIdToLikes.get(videoId) || 0, this.videoIdToDislikes.get(videoId) || 0];

const duration = Math.min(endMinute, video.length - 1) - startMinute + 1;

this.usedIds.push(videoId);

watch(videoId, startMinute, endMinute) {

return "-1";

if (!this.videoIdToVideo.has(videoId)) {

if (this.videoIdToVideo.has(videoId)) {

if (this.videoIdToVideo.has(videoId)) {

if (this.videoIdToVideo.has(videoId)) {

return this.videoIdToViews.get(videoId) || -1;

else:

self.currVideoId = 0

self.videoIdToVideo = {}

if self.usedIds.empty():

self.usedIds = PriorityQueue()

self.videoIdToViews = defaultdict(int)

self.videoIdToLikes = defaultdict(int)

videoId = self.currVideoId

videoId = self.usedIds.get()

self.videoIdToVideo[videoId] = video

self.currVideoId += 1

def upload(self, video: str) -> int:

def remove(self, videoId: int) -> None:

if videoId in self.videoIdToVideo:

del self.videoIdToVideo[videoId]

del self.videoIdToViews[videoId]

del self.videoIdToLikes[videoId]

if videoId not in self.videoIdToVideo:

self.videoIdToViews[videoId] += 1

video = self.videoIdToVideo[videoId]

del self.videoIdToDislikes[videoId]

return video[startMinute:startMinute + duration]

def watch(self, videoId: int, startMinute: int, endMinute: int) -> str:

duration = min(endMinute, len(video) - 1) - startMinute + 1

self.usedIds.put(videoId)

videoId = self.getVideoId()

self.videoIdToDislikes = defaultdict(int)

views, likes, and dislikes from the respective maps. If not, return -1. Solution

For liking and disliking a video, first check if the video with the given videoId exists in the videoIdToVideo map. If yes,

from collections import defaultdict class VideoSharingPlatform: def init (self):

```
def like(self, videoId: int) -> None:
        if videoId in self.videoIdToVideo:
            self.videoIdToLikes[videoId] += 1
    def dislike(self. videoId: int) -> None:
        if videoId in self.videoIdToVideo:
            self.videoIdToDislikes[videoId] += 1
    def getLikesAndDislikes(self, videoId: int) -> list[int]:
         return [self.videoIdToLikes[videoId], self.videoIdToDislikes[videoId]] if videoId in self.videoIdToVideo els
    def getViews(self. videoId: int) -> int:
        return self.videoIdToViews[videoId] if videoId in self.videoIdToVideo else -1
Time Complexity
  The time complexity for each operation in the video sharing platform is O(1) or O(logN) depending on the underlying
  implementation of the priority queue and maps. Since we are using default data structures available in Python, the overall time
  complexity is quite optimal for each operation.## JavaScript Solution
iavascript
class VideoSharingPlatform {
    constructor() {
        this.currVideoId = 0;
        this.usedIds = new PriorityQueue();
        this.videoIdToVideo = new Map();
        this.videoIdToViews = new Map();
        this.videoIdToLikes = new Map();
        this.videoIdToDislikes = new Map();
    getVideoId() {
        if (this.usedIds.isEmpty()) {
            const videoId = this.currVideoId;
            this.currVideoId += 1;
            return videoId;
        } else {
            return this.usedIds.pop();
    upload(video) {
        const videoId = this.getVideoId();
        this.videoIdToVideo.set(videoId, video);
        return videoId;
```

Java Solution

```
iava
import java.util.*;
public class VideoSharingPlatform {
    private int currVideoId:
    private PriorityQueue<Integer> usedIds;
    private Map<Integer, String> videoIdToVideo;
    private Map<Integer, Integer> videoIdToViews:
    private Map<Integer, Integer> videoIdToLikes:
    private Map<Integer, Integer> videoIdToDislikes;
    public VideoSharingPlatform() {
        currVideoId = 0;
        usedIds = new PriorityQueue<>();
        videoIdToVideo = new HashMap<>();
        videoIdToViews = new HashMap<>();
        videoIdToLikes = new HashMap<>();
        videoIdToDislikes = new HashMap<>();
    public int getVideoId() {
        if (usedIds.isEmpty()) {
            return currVideoId++;
        } else {
            return usedIds.poll();
    public int upload(String video) {
        int videoId = getVideoId();
        videoIdToVideo.put(videoId, video);
        return videoId;
    public void remove(int videoId) {
        if (videoIdToVideo.containsKey(videoId)) {
            usedIds.add(videoId);
            videoIdToVideo.remove(videoId);
            videoIdToViews.remove(videoId);
            videoIdToLikes.remove(videoId);
            videoIdToDislikes.remove(videoId);
    public String watch(int videoId, int startMinute, int endMinute) {
        if (!videoIdToVideo.containsKey(videoId)) {
            return "-1";
        videoIdToViews.put(videoId, videoIdToViews.getOrDefault(videoId, 0) + 1);
        String video = videoIdToVideo.get(videoId);
        int duration = Math.min(endMinute, video.length() - 1) - startMinute + 1;
        return video.substring(startMinute, startMinute + duration);
    public void like(int videoId) {
        if (videoIdToVideo.containsKey(videoId)) {
            videoIdToLikes.put(videoId, videoIdToLikes.getOrDefault(videoId, 0) + 1);
    public void unlike(int videoId) {
        if (videoIdToVideo.containsKey(videoId)) {
            videoIdToDislikes.put(videoId, videoIdToDislikes.getOrDefault(videoId, 0) + 1);
    public int[] getLikesAndDislikes(int videoId) {
        if (videoIdToVideo.containsKey(videoId)) {
            int[] result = new int[2];
            result[0] = videoIdToLikes.getOrDefault(videoId, 0);
            result[1] = videoIdToDislikes.getOrDefault(videoId, 0);
            return result;
        } else {
            return new int[]{-1};
    public int getViews(int videoId) {
        return videoIdToViews.getOrDefault(videoId, -1);
```