2254. Design Video Sharing Platform

Problem Description

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

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 dislikes for each 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`.

Example

1. Upload a video: Let's assume a user uploads a video with content "12345". The platform assigns `videoId` 0 to this video.
2. View a video: A viewer watches the video with `videoId` 0 from minute 1 to minute 3. The platform should return the content "234".
3. Like and dislike: The viewer likes the video with `videoId` 0. Now the video has 1 like. Another viewer dislikes the video. Now it has 1 dislike.
4. Get statistics: The platform should return the statistics for the video with `videoId` 0 → views: 1, likes: 1, dislikes: 1.
5. Delete a video: A user deletes the video with `videoId` 0.
6. 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

We can use the following data structures for this problem:

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

1. 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.
2. 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.
3. 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".
4. For liking and disliking a video, first check if the video with the given `videoId` exists in the `videoIdToVideo` map. If yes, increment the likes or dislikes count in the respective maps.
5. For getting statistics, first check if the video with the given `videoId` exists in the `videoIdToVideo` map. If yes, return the views, likes, and dislikes from the respective maps. If not, return -1.

Solution

```
python
from queue import PriorityQueue
from collections import defaultdict

class VideoSharingPlatform:

    def __init__(self):
        self.currVideoId = 0
        self.usedIds = PriorityQueue()
        self.videoIdToVideo = {}
        self.videoIdToViews = defaultdict(int)
        self.videoIdToLikes = defaultdict(int)
        self.videoIdToDislikes = defaultdict(int)

    def getVideoId(self):
        if self.usedIds.empty():
            videoId = self.currVideoId
            self.currVideoId += 1
        else:
            videoId = self.usedIds.get()
        return videoId

    def upload(self, video: str) -> int:
        videoId = self.getVideoId()
        self.videoIdToVideo[videoId] = video
        return videoId

    def remove(self, videoId: int) -> None:
        if videoId in self.videoIdToVideo:
            self.usedIds.put(videoId)
            del self.videoIdToVideo[videoId]
            del self.videoIdToViews[videoId]
            del self.videoIdToLikes[videoId]
            del self.videoIdToDislikes[videoId]

    def watch(self, videoId: int, startMinute: int, endMinute: int) -> str:
        if videoId not in self.videoIdToVideo:
            return "-1"
        self.videoIdToViews[videoId] += 1
        video = self.videoIdToVideo[videoId]
        duration = min(endMinute, len(video) - 1) - startMinute + 1
        return video[startMinute:startMinute + duration]

    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 else

    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

```
javascript
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;
    }

    remove(videoId) {
        if (this.videoIdToVideo.has(videoId)) {
            this.usedIds.push(videoId);
            this.videoIdToVideo.delete(videoId);
            this.videoIdToViews.delete(videoId);
            this.videoIdToLikes.delete(videoId);
            this.videoIdToDislikes.delete(videoId);
        }
    }

    watch(videoId, startMinute, endMinute) {
        if (!this.videoIdToVideo.has(videoId)) {
            return "-1";
        }
        this.videoIdToViews.set(videoId, (this.videoIdToViews.get(videoId) || 0) + 1);
        const video = this.videoIdToVideo.get(videoId);
        const duration = Math.min(endMinute, video.length - 1) - startMinute + 1;
        return video.slice(startMinute, startMinute + duration);
    }

    like(videoId) {
        if (this.videoIdToVideo.has(videoId)) {
            this.videoIdToLikes.set(videoId, (this.videoIdToLikes.get(videoId) || 0) + 1);
        }
    }

    unlike(videoId) {
        if (this.videoIdToVideo.has(videoId)) {
            this.videoIdToDislikes.set(videoId, (this.videoIdToDislikes.get(videoId) || 0) + 1);
        }
    }

    getLikesAndDislikes(videoId) {
        if (this.videoIdToVideo.has(videoId)) {
            return [this.videoIdToLikes.get(videoId) || 0, this.videoIdToDislikes.get(videoId) || 0];
        } else {
            return [-1];
        }
    }

    getViews(videoId) {
        return this.videoIdToViews.get(videoId) || -1;
    }
}
```

Java Solution

```
java
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);
    }
}
```

Time Complexity

Solution Implementation

Python

Java

C++

TypeScript

Recommended Readings