Comparative Case Study – Randomized Search Trees (Treaps)

Overview

This project is designed as a comparative case study to analyze and evaluate the performance of three advanced tree-based data structures --- Treaps (Randomized Binary Search Trees), and Binary Search Trees (BSTs) when used for maintaining dynamically prioritized data with dual ordering requirements. Students will implement tree structures from scratch, integrate them into a social media feed management system, and analyze runtime, operation performance, and structural properties (e.g., tree height, number of nodes, rotation behavior).

The study also includes a large-scale performance test with real-world social media datasets of varying sizes and update frequencies.

Learning Objectives

- Implement two tree-based data structures: Treaps, and Binary Search Trees.
- Integrate these structures into a dynamic feed management system and simulate real-world social media conditions.
- Evaluate performance for insert, delete, search, and priority update operations.
- Record and visualize tree properties: height, number of rotations, and balancing patterns.

Case Study Background

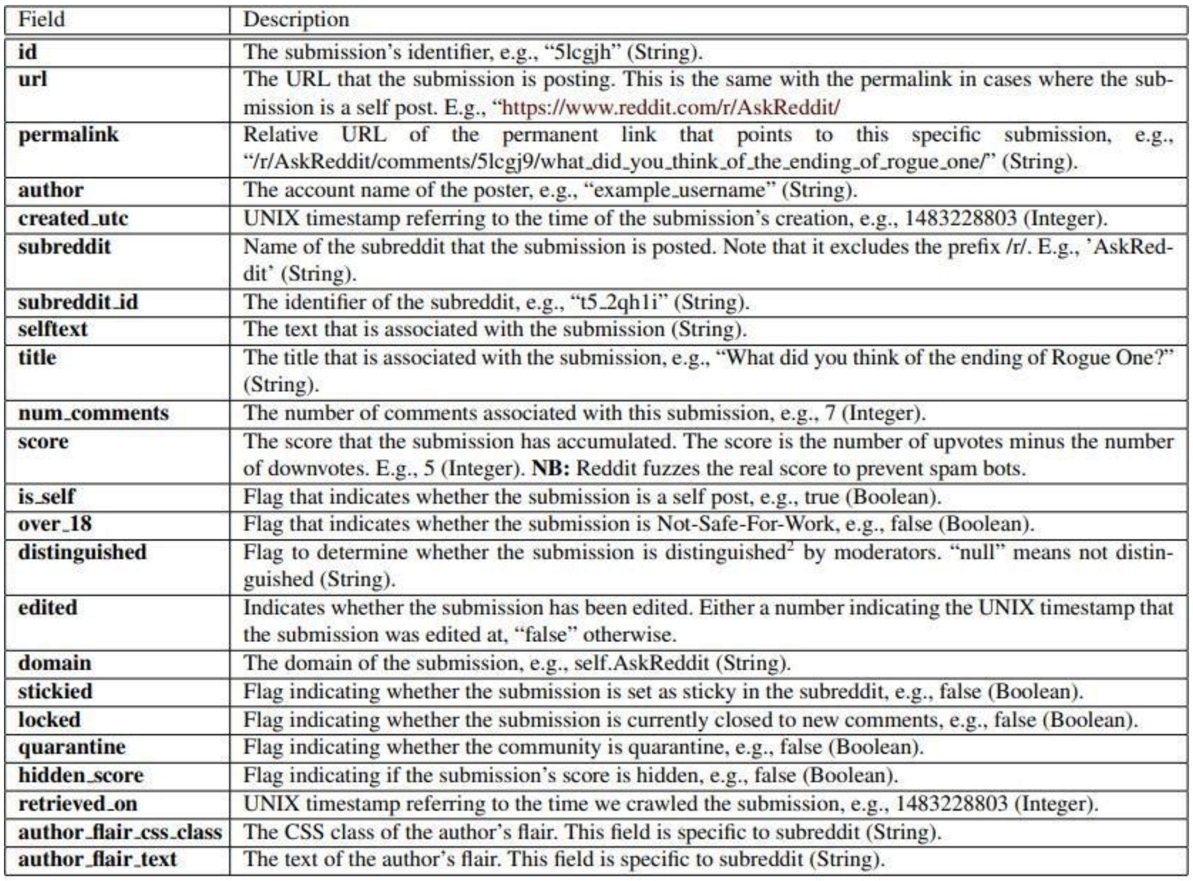
Modern social media platforms such as Twitter and Reddit maintain real-time content feeds that must update continuously as new posts appear and engagement metrics evolve. Post timestamps and popularity (likes or upvotes) both change frequently, requiring the system to keep the feed simultaneously sorted by recency and responsive to popularity spikes.
Since traditional solutions rely on maintaining multiple data structures: a sorted list or BST for chronological ordering and a heap or priority queue for popularity ranking, rapid updates often lead to expensive rebalancing and synchronization overhead. Choosing a unified structure that efficiently supports both ordering dimensions can significantly improve performance at scale.

Students are tasked to:

1. Implement two data structures: a Treap and a standard BST.
2. Use the provided dataset to perform insertion, deletion, and update operations on both structures.
3. Simulate real-time feed activity using the dataset’s stream of timestamps and score updates.
4. Compare the performance of the BST and Treap to determine which structure performs better under different conditions.

Dataset Information

Dataset Link: pushshift-reddit-dataset
The dataset consists of JSON file with following keys, in which you are most interested in `{id, created_utc, score}`



Field	Description
id	The submission's identifier, e.g., "5lcgjh" (String).
url	The URL that the submission is posting. This is the same with the permalink in cases where the submission is a self post. E.g., "https://www.reddit.com/r/AskReddit/
permalink	Relative URL of the permanent link that points to this specific submission, e.g., "/r/AskReddit/comments/5lcgj9/what_did_you_think_of_the_ending_of_rogue_one/" (String).
author	The account name of the poster, e.g., "example_username" (String).
created_utc	UNIX timestamp referring to the time of the submission's creation, e.g., 1483228803 (Integer).
subreddit	Name of the subreddit that the submission is posted. Note that it excludes the prefix /r/. E.g., 'AskReddit' (String).
subreddit_id	The identifier of the subreddit, e.g., "t5_2qh1i" (String).
selftext	The text that is associated with the submission (String).
title	The title that is associated with the submission, e.g., "What did you think of the ending of Rogue One?" (String).
num_comments	The number of comments associated with this submission, e.g., 7 (Integer).
score	The score that the submission has accumulated. The score is the number of upvotes minus the number of downvotes. E.g., 5 (Integer). **NB:** Reddit fuzzes the real score to prevent spam bots.
is_self	Flag that indicates whether the submission is a self post, e.g., true (Boolean).
over_18	Flag that indicates whether the submission is Not-Safe-For-Work, e.g., false (Boolean).
distinguished	Flag to determine whether the submission is distinguished[2] by moderators. "null" means not distinguished (String).
edited	Indicates whether the submission has been edited. Either a number indicating the UNIX timestamp that the submission was edited at, "false" otherwise.
domain	The domain of the submission, e.g., self.AskReddit (String).
stickied	Flag indicating whether the submission is set as sticky in the subreddit, e.g., false (Boolean).
locked	Flag indicating whether the submission is currently closed to new comments, e.g., false (Boolean).
quarantine	Flag indicating whether the community is quarantine, e.g., false (Boolean).
hidden_score	Flag indicating if the submission's score is hidden, e.g., false (Boolean).
retrieved_on	UNIX timestamp referring to the time we crawled the submission, e.g., 1483228803 (Integer).
author_flair_css_class	The CSS class of the author's flair. This field is specific to subreddit (String).
author_flair_text	The text of the author's flair. This field is specific to subreddit (String).

Table 1: `Submissions` data description.

Dataset Challenges and Handling Strategy

The dataset's scale poses major constraints: the compressed file is ~15 GB and the fully decompressed version expands to nearly 180 GB. Such volume cannot be loaded into memory at once, and even storing the uncompressed version may exceed available disk capacity. Students must therefore design their system to process the data in a memory-efficient, streaming-friendly manner while still supporting continuous updates to their feed manager.

To address these limitations, students should clearly explain the strategy they adopt. Two recommended approaches include:

- **Stream Processing in Compressed Form:**
  Instead of decompressing the entire .zst archive, the dataset should be consumed as a

compressed stream. This enables line-by-line processing, keeps memory usage minimal, and avoids generating the massive 180 GB intermediate file.

- **Partitioning via Multiple Treaps (n-Treaps):**
  Rather than constructing one extremely large Treap, students may split the dataset into n segments, build n independent Treaps, and then merge them using Treap union operations. This reduces memory pressure, allows parallel processing, and maintains efficient structural balance.

Note: Students must justify how their chosen method effectively handles the dataset's size and operational constraints.

Implementation Requirements

1. **Implement a Treap data structure that supports the following operations.**

Function	**Description**
`addPost(postid, timestamp, score)`	Insert a new post into the Treap.
`likePost(postid)`	Increase the like count; reheapify to maintain heap property.
`deletePost(postid)`	Remove a post by ID
`getMostPopular()`	Return post with highest popularity (max-heap root).
`getMostRecent(k)`	Return k-most recent posts (reverse in-order traversal).

2. **Implement a BST that supports the following operations**

Function	**Description**
`addPost(postid, timestamp, score)`	Insert a post into BST, ordered by Timestamp.
`likePost(postid)`	Increase the like count; does not affect structure.
`deletePost(postid)`	Remove a post by ID
`getMostPopular()`	Traverse the BST to find the post with the highest score count.
`getMostRecent(k)`	Return k-most recent posts (reverse in-order traversal).

3. **A comparison dashboard where you compare BST with Treap**

To draw comparison of both data structures you have to perform ablation studies, i.e you have to test the two data structures under the same conditions. You will use the same data set, same sample size for the experimentation to test the data structure in terms of time, height, no of rotations and memory usage (optional). Ensure the comparison is clearly illustrated through visual analyses, including graphs and plots created with suitable visualization libraries.

4. **Metrics to record for the performance measurements**

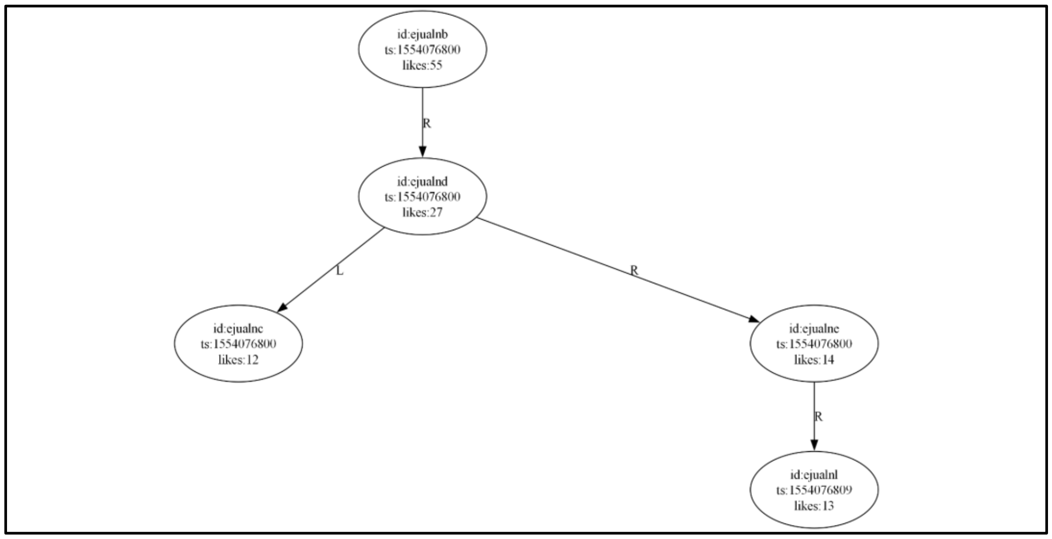
Metrics	**Treaps**	**Binary Search Tree**
Insertion Time (avg)		
Deletion Time (avg)		
Search Time (avg)		
Height of the Tree		
Tree Balancing Factor		

Bonus Implementation

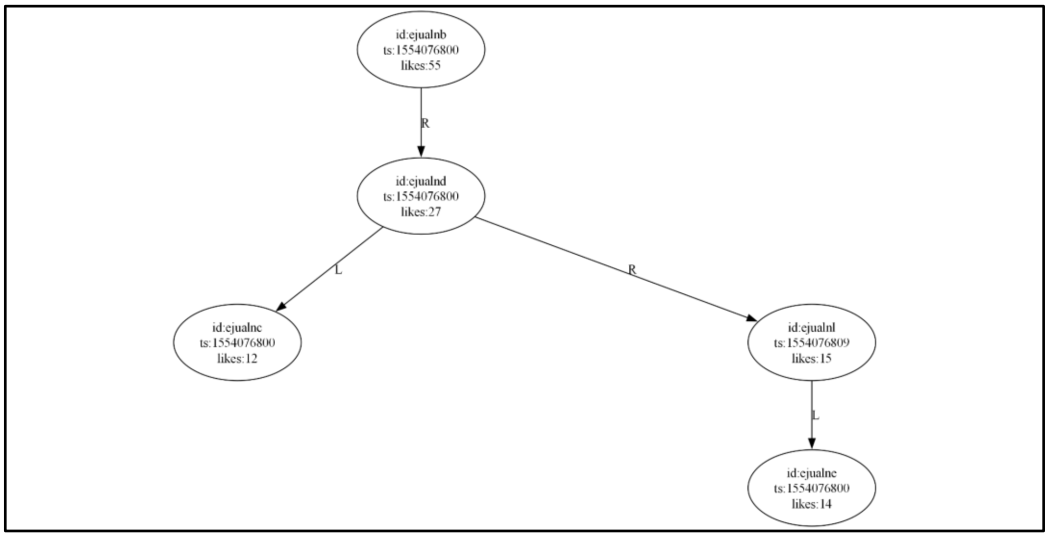
1. Treap operations like Union (merging two treaps to get one Treap), and Intersection (finds common elements between the treaps)
2. Tree visualizations for Treap and BST to inspect height, node distribution, and rotations

Test Case (sample size: 4)

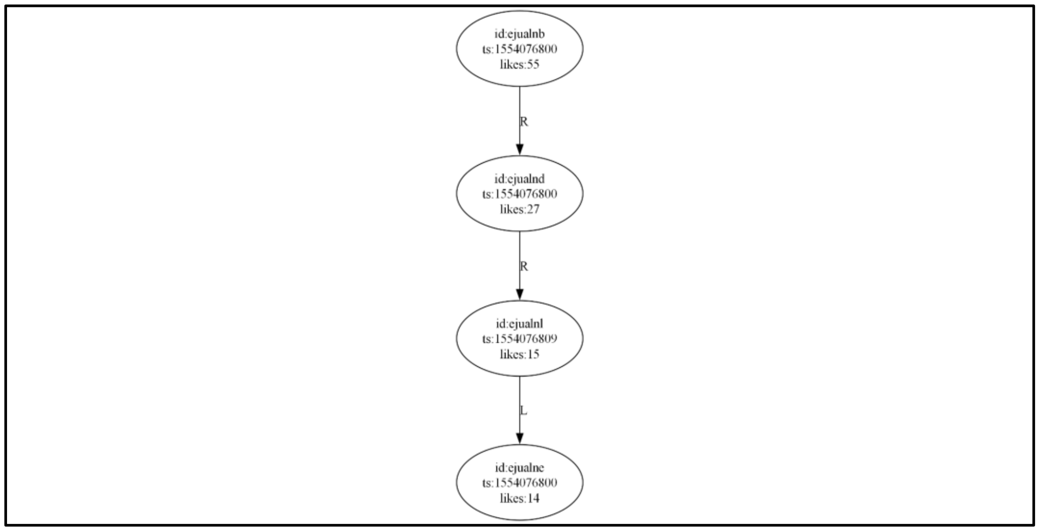
1. Add the following into the Treap data structure, and the tree will look like:
```
addPost("ejualnb", 1554076800, 55)
addPost("ejualnc", 1554076800, 12)
addPost("ejualnd", 1554076800, 27)
addPost("ejualne", 1554076800, 14)
addPost("ejualnl", 1554076809, 13)
```



2. Like the post with ID ‘ejunalnl’ 2 times.
```
likePost(‘ejualnl’)
likePost(‘ejualnl’)
```



3. Delete the post with ID ‘ejualnc’.
   `deletePost(‘ejualnc’)`



4. Get the post with the most likes.
   `getMostPopular() ----- returns -----> (‘ejunalb’, 1554076800, 55)`

Deliverables

1. **Source Code:**
   - Treaps implementation and the performance metrics
   - Binary Search Tree (BSTs) implementation and the performance metrics.
2. **Report (with the following requirements):**
   - Introduction & Theoretical Overview of both structures (Treaps and BSTs)
   - Implementation Details – Programming language used, Dataset handling method, Duplication handling method, Tree operations used etc.)
   - Performance metrics and their visualizations
   - Challenges faced in the implementation
   - Conclusion