**Implementation of Data Structures and Algorithm in Weather Forecast API**



**BTech/II Year CSE/IV Semester**

**19CSE213/Data Structures and Algorithms**

**Case Study Report**

|  |  |
| --- | --- |
| **Roll no.** | **Name** |
| **CB.EN.U4CSE21257** | **Shreenithi S** |
| **CB.EN.U4CSE21270** | **Yash Puthalath** |

**Department of Computer Science and Engineering**

**Amrita School of Computing, Coimbatore**

**Amrita Vishwa Vidyapeetham**

**2022 - 2023 Even Semester**

**Introduction:**

Hybrid data structures combine the benefits of individual data structures and forms a multiple data structure, hybrid data structures can provide

1. Optimize complex problems
2. Handling large datasets
3. Efficient storage
4. Improved performance
5. Reduced usage of time and resources

**Overview of the Project:**

We built a simple GUI weather forecasting application using OpenWeatherMap API, it provides the locations and retrieve their current weather information and display in a label, including the temperature, humidity, and weather description. The application also displays an image related to each location, if available.

The weather information provided by the application is used to determine whether it is safe to go out, and if any extra precautions should be taken. For example

* If the temperature is very high, the application might suggest staying indoors and drinking plenty of water to avoid heatstroke.
* If it is rainy or cloudy, the application might suggest bringing an umbrella.
* If it is snowy, the application might suggest wearing snow boots and appropriate clothing to protect against the cold.

By providing this information, the application can help users make informed decisions about how to dress and prepare for the weather and avoid any discomfort.

**Overview of the Hybrid Data Structure:**

We used ‘Treap data structure’ to hold the locations and their randomly generated priorities in addition to displaying the weather information.

Treap = Binary Search Tree + Heap

A treap is a mixture of binary search tree and heap, which means that each node in the tree has a **key and a priority**, and the nodes are arranged such that

* The keys satisfy the BST property
* The priorities satisfy the heap property
* Maintain balance in the face of random insertions and deletions

This makes it an efficient data structure for solving problems that require both sorted order and priority-based access. The treap is balanced by rotating nodes left or right when a node violates the heap property.

In the context of the weather forecasting application, it is used to **store the locations** entered by the user, along with their randomly generated priorities. It can **handle dynamic updates efficiently**, meaning new locations can be added and old locations can be removed dynamically, and it is handled by performing rotations.

We are also using Hash Maps, when the hash function evenly distributes the keys across the array:

* Collisions are minimized
* Resulting in efficient retrieval
* The performance increases

Collisions occur when multiple keys map to the same array index, leading to slower retrieval of collision resolution techniques. The performance of hash tables is based on the principle of hashing, which allows for quick data access by using a hash function to map keys to array indices.

**Implementation Details:**

Firstly, we started off by getting our API key from OpenWeatherMap and did basic code for

“Input location: Output information”

Secondly, we decided on treap data structure and collected information and code and tweaked it to suit our project, wherein it integrates the binary search tree and heap properties to efficiently store and access locations and their priorities.

1. Location entered ---> New node created & inserted ---> BST property.
2. Node ---> Randomly generated priority value ---> Heap property.
3. If the node violates the heap property, rotations restore the balance between the BST and heap properties.
4. Handles dynamic updates ---> new locations can be added and old locations can be removed dynamically.
5. Location is removed ---> node is deleted from the BST ---> Heap property is restored by performing rotations.

Thirdly, we added a simple GUI using tkinter which displays the temperature, humidity, and weather description with an image related to each location and a short description on what to wear for the weather in the place we look for and if travelling is safe.

Implementation of Hash Maps in Flight Status Checker

* Here, we Acquire Flight Details using the Aviation Stack API and the program prompts the user to enter a flight number and all the details of that flight is displayed in a Tkinter UI.
* Here, Python Dictionary Data Structures are implemented as Hash Maps
* The flight\_details hashmap is used to store the flight details for all the flights provided by the user. This makes it easy to access the flight details for a particular flight by its IATA number.
* For example, if the user wants to know the departure airport for flight number AA123, they can simply look up the flight details hashmap using the key AA123. The value associated with the key AA123 will be the departure airport for that flight.
* The flight\_details hashmap is used to print the flight details in the loop from line 47 to 56. The IATA number of each flight is used as the key to look up the flight details in the hashmap.

**Design choices and trade-offs made during the implementation phase:**

Our design choice of using treap is because the overall structure remains balanced even as nodes are inserted and deleted. As a result, the treap is well-suited for applications where dynamic updates are frequent, such as real-time systems and weather forecast is one such application which uses treap data structure to its maximum potential.

Trade-offs may be needed between memory usage and performance. Increasing the number of nodes in the Treap can improve the efficiency of priority-based access, but can also:

* Increase memory usage.
* Increasing computational cost of updates.

**GitHub repository link:** https://github.com/Shree-nithi/Weather-Forcast-Treap-Data-Structure

**Practical Applications:**

Treap data structure is a versatile and efficient data structure that can be used in a variety of practical applications where both sorted order and priority-based access are required along with dynamic updates. A few such live examples are as follows:

1. Real-time stock trading - Treap data structure can be used to store and access stock prices based on both their price and priority, such as the urgency of the order or the trading volume.
2. Network routing - Treap data structure can be used to efficiently route messages through computer networks based on both their destination and priority.
3. Web caching - Treap data structure can be used to prioritize web pages for caching based on their popularity and access frequency.
4. Task scheduling - Treap data structure can be used to schedule tasks based on their priority and deadline.

Hashmaps

1.Hashmaps are Standard in Java Packages, it maintains an array of buckets. Each bucket is a linkedlist of key value pairs encapsulated as Entry objects, without Hashmaps, many Java Functionalities Would not Work.

2. Hash maps are used to link the file name with the path of the file. To store the correspondence between the file name and path and the physical location of that file on the disk, the system uses a map.

**Performance Analysis:**

**1. Treap**

The Time Complexity for the Binary Tree+ Heap Data Structure incorporates the Time Complexity of each of them and results in O(Log n) for searching, inserting and deleting.

We can use a Similar Approach to calculate the Space Complexity which stands as O(n).

1. Treap provides efficient average-case performance for insertion, deletion, and search operations.
2. It maintains a balanced binary search tree structure based on both key values and priorities, resulting in good overall performance.
3. By combining multiple data structures, it is possible to achieve improved efficiency and additional functionality.

2. Hash Maps (Stand-alone)

The Time Complexity of this stand-alone data Structure stands as O(1) for Insertion and Deletion.

The Space Complexity Stands as O(n) it is proportional to the number of elements, where n is the number of key-value pairs in the Hash Table.

In the worst-case scenario, where all keys collide, the time complexity can deteriorate to O(n), where n is the number of elements in the hash table.

**Discussion:**

* Applications of Treap:  
  Randomized algorithms: Treaps can be used to implement randomized algorithms, such as quickselect and randomized quicksort.
* Streaming algorithms: Treaps can be used to implement streaming algorithms, which are algorithms that process data as it arrives.

Applications of Hash maps

* Data compression: Hash maps can be used in data compression algorithms, such as the Huffman coding algorithm, to encode data efficiently.
* Search algorithms: Hash maps can be used to implement search algorithms, such as the Rabin-Karp algorithm, for fast lookups and queries.
* Cryptography: Hash maps are used in cryptography, for example for digital signatures in public key encryption, for MACs in secret key encryption (e.g., HMAC), for hash-based pseudo-random number generators, for user authentication (e.g. salted password hashing) and so on.

Limitations, challenges, and potential future improvements:

Limitations:

* Randomness of priorities: If the priorities are not truly random, the tree may become unbalanced, leading to decreased efficiency.
* Memory usage: It can require more memory as each node contains both a key and a priority value.
* Performance: While it is efficient for priority-based access, it may not be efficient for other types of operations, such as searching or deletion.

Hashmaps:

* Space overhead. Hashmaps store both the key and the value of each element, so they require more space than some other data structures, such as linked lists.
* Collisions. When two different keys hash to the same bucket, this is called a collision. Collisions can slow down lookups, as the hashmap must then iterate through the linked list in the bucket to find the correct element.

Challenges:

* Implementation complexity: The implementation can be complex, especially when it comes to balancing the binary search tree and heap properties.
* Trade-offs: The Treap data structure requires a trade-off between priority-based access and memory usage, as increasing the number of nodes can improve access efficiency but can also increase memory usage.

Potential Future Improvements:

* Improved balancing algorithms: The development of new algorithms to better balance the binary search tree and heap properties, particularly in cases where the priorities are not truly random.
* Hybrid data structures: Combine the properties of the Treap data structure with other data structures to improve performance for other types of operations, such as searching or deletion.

**Conclusion:**

Overall, the project was successful in implementing a weather forecasting application using the Treap data structure as a hybrid data structure. The project demonstrated the practicality and effectiveness treap for efficiently storing and accessing data based on both sorted order and priority-based access.

The implementation and evaluation of the Treap data structure provided valuable insights into its strengths and limitations, which can help with future improvements and applications.

**References:**

[**https://chat.openai.com/**](https://chat.openai.com/)

[**https://openweathermap.org/api**](https://openweathermap.org/api)

[**https://aviationstack.com/documentation**](https://aviationstack.com/documentation)