Real-Time Stock Price Alert System Using AVL Trees

A CAPSTONE PROJECT

Submitted By

PAVAN CHALLA (192210469)

PUTHANA RAVI KUMAR REDDY (192210367)

In Partial Fulfillment for the completion of the course

CSA0311

DATA STRUCTURES FOR VISUALIZATION

NOV 2024

SAVEETHA INSTITUTE OF MEDICAL AND TECHNICAL SCIENCES

CHENNAI – 602105

TAMIL NADU, INDIA

CONTENTS

Abstract:

The **Real-Time Stock Price Alert System Using AVL Trees** is a computationally efficient solution designed to monitor and manage stock prices dynamically. Stock market operations require systems capable of handling frequent updates and fast queries. AVL trees, being self-balancing binary search trees, ensure that operations like insertion, deletion, and search are performed in$O(\log n)$ time, making them suitable for real-time applications.

This system maintains a database of stock prices where each stock is represented as a node in an AVL tree. The nodes store the stock identifier, its current price, and pointers to child nodes. The AVL tree ensures that the data remains balanced after every update, providing efficient retrieval of stock prices for analysis.

The system includes real-time alert generation, triggered when a stock's price crosses pre-defined thresholds. Users can configure upper and lower limits for each stock, and the system instantly notifies the user if the stock price violates these limits. This ensures timely decision-making for stock trading or investment.

The proposed system combines the reliability of AVL tree operations with the responsiveness required for real-time stock monitoring. It offers features such as dynamic updates of stock prices, quick retrieval for alerts, and in-order traversal for displaying stocks in sorted order. This solution is scalable, accurate, and ideal for modern financial applications where performance is critical.

In addition to its core functionalities, the system offers a streamlined interface for users to view and manage stock data. Through in-order traversal, stocks are displayed in sorted order, making it easier for users to review prices and trends. The AVL tree's balance mechanism ensures consistent performance even with frequent updates, making it scalable for a growing number of stocks. By combining efficiency, accuracy, and scalability, this system addresses the critical requirements of modern stock price monitoring and alert systems, empowering users with a reliable and responsive tool for managing their investments.

The Real-Time Stock Price Alert System Using AVL Trees also emphasizes scalability and adaptability, making it suitable for a wide range of financial applications. As stock markets experience rapid fluctuations and large-scale data inflows, the AVL tree structure ensures that the system can handle increasing volumes of stocks without degrading performance. Its ability to adapt to continuous updates while maintaining balance enables consistent and predictable response times, crucial for real-time systems. Moreover, the flexibility to incorporate additional features, such as historical price tracking or integration with external APIs for live market data, positions this system as a robust foundation for advanced financial tools. This adaptability ensures its relevance in both personal investment management and institutional stock trading environments.

Introduction:

The stock market is a dynamic and fast-paced environment where prices of stocks fluctuate continuously based on market trends, news, and various economic factors. Investors and traders rely on accurate and real-time data to make informed decisions about buying, holding, or selling stocks. Monitoring these price changes manually is not only tedious but also prone to delays and errors. In such scenarios, automated systems that can efficiently track, update, and analyze stock prices are critical. The **Real-Time Stock Price Alert System Using AVL Trees** provides a systematic approach to addressing these challenges by leveraging the power of data structures to ensure quick and efficient operations.

AVL trees, named after their inventors Adelson-Velsky and Landis, are a type of self-balancing binary search tree. They maintain a balance factor for every node, ensuring that the difference in height between the left and right subtrees does not exceed one. This property enables AVL trees to perform all fundamental operations—such as insertion, deletion, and search—inO(logn) time. In the context of stock price monitoring, this efficiency is essential, as it allows the system to process a high volume of updates and queries rapidly. The balance maintained by AVL trees guarantees that performance remains consistent, even as the dataset grows.

The proposed system uses AVL trees to represent stocks as nodes, where each node contains a unique stock identifier and its current price. This representation allows for easy insertion of new stocks, updates to existing stock prices, and quick retrieval of data. Additionally, the system enables users to set upper and lower thresholds for individual stock prices. Whenever a stock's price moves beyond these predefined limits, the system generates alerts in real-time, ensuring users are notified promptly. This feature is particularly valuable in scenarios where rapid decisions are required to capitalize on market opportunities or mitigate losses.

One of the key benefits of the system is its ability to handle large datasets efficiently. As the number of tracked stocks increases, maintaining balance in the AVL tree ensures that operations remain fast and predictable. The system also supports in-order traversal of the tree, which displays stocks in a sorted manner based on their identifiers. This functionality simplifies the process of reviewing stock data, enabling users to analyze and manage their portfolios more effectively. The combination of real-time updates, alerts, and sorted data visualization makes the system a powerful tool for stock market monitoring.

The system is also designed with scalability in mind, making it suitable for both individual investors and institutional traders. Its modular design allows for the integration of additional features, such as fetching live market data from external APIs or storing historical price data for trend analysis. This flexibility ensures that the system can adapt to the evolving needs of users while maintaining its core functionality of efficient stock price monitoring and alerting. By leveraging the robust properties of AVL trees, the system achieves a balance between performance, accuracy, and adaptability.

Objectives and Goals:

1. **Efficient Stock Price Management**

   The primary objective of the **Real-Time Stock Price Alert System Using AVL Trees** is to provide a fast and reliable solution for managing stock price data. By using AVL trees, the system ensures that operations like insertion, deletion, and retrieval of stock prices are performed efficiently, even with large datasets. This structure allows for real-time monitoring of stock price changes, making it ideal for dynamic and high-frequency trading environments.

2. **Dynamic Threshold-Based Alerts**

   Another key objective is to implement a dynamic alert mechanism that notifies users when stock prices cross predefined thresholds. Users can configure upper and lower limits for each stock, ensuring they are alerted promptly when significant price movements occur. This feature empowers traders and investors to act quickly, minimizing risks and maximizing potential gains.

3. **Real-Time Data Updates**

   The system is designed to handle continuous updates in stock prices in real time, reflecting market fluctuations instantly. This ensures that the information displayed to the users is always accurate and up-to-date. The self-balancing nature of AVL trees ensures that the system maintains optimal performance, even with frequent updates, making it suitable for fast-paced financial applications.

4. **Scalability and Adaptability**

   Scalability is a core goal of this system, allowing it to accommodate a growing number of stocks without a decline in performance. As the market expands and the volume of data increases, the AVL tree structure adapts seamlessly to handle the load. Additionally, the system can be extended to include more advanced features, such as integration with live stock market APIs and support for multi-user environments.

5. **Simplified Data Visualization**

   The system provides a straightforward and intuitive way to view stock price data. Through in-order traversal of the AVL tree, stock prices are displayed in a sorted order, making it easy for users to review trends and identify significant changes. This simplifies decision-making for users by presenting information in a clear and organized format.

Project Scope:

The **Real-Time Stock Price Alert System Using AVL Trees** is designed to efficiently manage and monitor stock price data while providing real-time alerts based on user-defined thresholds. The system aims to address the need for rapid price updates, efficient data storage, and dynamic alert mechanisms in financial markets. By leveraging AVL trees, the project ensures that all stock-related operations—such as insertion, deletion, and search—are performed in optimal time, making the system capable of handling large datasets with minimal delay. This system is targeted at traders, investors, and financial institutions that require reliable tools for tracking and responding to stock price changes.

The scope includes the implementation of an AVL tree-based backend that stores and organizes stock data. Each stock is represented as a node containing its identifier, price, and relevant metadata. The AVL tree ensures balance after every operation, preventing inefficiencies caused by skewed structures. The system is capable of handling dynamic updates to stock prices, ensuring that users always have access to the latest data. The use of AVL trees guarantees consistent performance regardless of the volume of data, making it scalable for large-scale applications.

Real-time alert generation is a core feature of this project. Users can set upper and lower price thresholds for individual stocks, and the system will notify them instantly when a stock price crosses these thresholds. This functionality is crucial for enabling quick decision-making in volatile markets. The alerts can be tailored to suit individual preferences, ensuring that users receive timely and actionable information. The system's ability to operate in real-time without delays highlights its suitability for fast-paced financial environments.

Additionally, the system offers functionalities for displaying and managing stock data. Through in-order traversal of the AVL tree, users can view a sorted list of stocks, facilitating quick comparisons and trend analysis. The design also allows for easy integration of additional features, such as historical price tracking and connectivity with live market data feeds. This modularity ensures that the system can evolve and adapt to changing user needs and market conditions.

The project also emphasizes user-friendliness and reliability. With an intuitive interface and robust backend architecture, the system ensures seamless interaction for both novice and experienced users. The AVL tree's self-balancing properties provide consistent performance even during frequent updates, ensuring that the system remains responsive at all times. Furthermore, the system is built to handle high data volumes, making it a valuable tool for both individual investors and large financial institutions.

In conclusion, the **Real-Time Stock Price Alert System Using AVL Trees** is a comprehensive solution that combines efficiency, scalability, and adaptability. By addressing the critical requirements of real-time stock monitoring and alert generation, the project delivers a reliable platform for tracking and managing stock prices. With its robust architecture and versatile features, the system can cater to the diverse needs of users, from personal portfolio management to institutional-level financial operations. Its scope encompasses not only the current features but also the potential for future enhancements, making it a long-term solution for dynamic financial markets.

Technologies & Tools:

The **Real-Time Stock Price Alert System Using AVL Trees** leverages a combination of data structures, programming languages, and software development tools to deliver a scalable, efficient, and reliable solution. The primary technology driving the system is the **AVL Tree**, a self-balancing binary search tree, which ensures logarithmic time complexity for operations like insertion, deletion, and search. This data structure is pivotal for maintaining real-time performance, even with frequent updates to stock prices or thresholds.

The system is implemented using **C/C++ or Python**, chosen for their efficiency and extensive library support. In C/C++, low-level memory management offers granular control over tree operations, while Python's dynamic typing and built-in data structure libraries simplify implementation. These programming languages enable precise and optimized handling of the AVL tree's nodes and balance checks, making them ideal for the system's computational requirements.

For the alerting mechanism, the system integrates **event-driven programming paradigms**, allowing for asynchronous handling of stock price changes. Real-time alerts are implemented using techniques such as callback functions or event listeners, ensuring that users are notified instantly when thresholds are breached. If deployed on a server, the system can utilize **multithreading** or **asynchronous libraries** to handle multiple stock updates and alert notifications concurrently, enhancing scalability.

To visualize stock prices and alert data, the system can integrate with **graphical user interface (GUI) frameworks** like **Tkinter** for Python or **Qt** for C++. These frameworks enable the creation of user-friendly dashboards, where users can monitor stock prices, configure thresholds, and receive alerts in a visually appealing format. Such interfaces make the system accessible to a wide audience, including both novice and experienced investors.

For testing and validation, the system employs **unit testing frameworks** like **Google Test (gTest)** for C++ or **unittest/pytest** for Python. These tools ensure that individual components, such as node insertion, deletion, and alert generation, function correctly under various scenarios. Automated testing guarantees system reliability and correctness, even as the complexity of operations increases with the number of stocks monitored.

Finally, the system is designed for future enhancements and integrations with external data sources, such as **RESTful APIs** for live stock market data. Technologies like **cURL** in C/C++ or **requests** in Python enable seamless communication with APIs, allowing the system to fetch real-time stock prices and update its AVL tree dynamically. This extensibility ensures that the system remains relevant and capable of handling advanced use cases in modern financial applications.

Project Deliverables:

The **Real-Time Stock Price Alert System Using AVL Trees** is designed to deliver a comprehensive and efficient solution for monitoring and managing stock prices. The first deliverable is the development of the AVL tree-based data structure, which acts as the core of the system. This structure ensures that the system maintains balanced and efficient operations for insertion, deletion, and search. The AVL tree implementation will include functions for managing stock data dynamically, enabling seamless updates as prices change in real-time.

The second deliverable is the integration of a user-friendly interface to facilitate stock monitoring. This interface will allow users to input stock details, set price thresholds, and receive alerts. The design will focus on providing a simple yet robust mechanism for configuring stock-specific upper and lower limits, ensuring that users can manage multiple stocks simultaneously with ease.

The third deliverable involves the implementation of the real-time alert system. This feature will continuously monitor stock prices and compare them with user-defined thresholds. Upon detecting a violation of these thresholds, the system will instantly generate notifications or alerts, ensuring that users can act quickly to capitalize on opportunities or mitigate risks in the market.

The fourth deliverable focuses on the visualization and reporting of stock data. The system will include features for displaying stocks in sorted order using an in-order traversal of the AVL tree. This will allow users to view all monitored stocks along with their current prices and thresholds, providing a clear overview of their portfolio. Additionally, the system will support generating reports on stock price trends for further analysis.

The fifth deliverable ensures the system's scalability and performance optimization. The AVL tree's self-balancing nature guarantees efficient handling of large datasets, enabling the system to scale for an increasing number of stocks. Performance testing and optimization will be conducted to ensure that the system remains responsive even during high-frequency updates or when managing large portfolios.

The final deliverable includes comprehensive documentation and future enhancements. The documentation will provide detailed explanations of the system's architecture, user guides, and codebase to facilitate maintenance and potential upgrades. Additionally, the project will outline potential enhancements, such as integration with live stock market APIs, support for multiple users, and advanced analytics features, making the system adaptable for evolving financial needs.

Functionality

1. **Dynamic Stock Price Insertion and Deletion:**

   The system enables dynamic insertion and deletion of stock data. Each stock is represented as a node in an AVL tree, containing its unique identifier (e.g., ticker symbol) and current price. When a new stock is added, the AVL tree ensures that the structure remains balanced by performing necessary rotations. Similarly, when a stock is removed, the tree adjusts to maintain its balance. This ensures that operations like adding or removing stocks are completed efficiently, even as the dataset grows in size.

2. **Threshold-Based Alert System:**

   One of the key functionalities of the system is its ability to monitor stock prices and trigger alerts when prices cross predefined thresholds. Users can configure upper and lower limits for each stock, which the system continuously evaluates against the current price. For example, if a stock price exceeds its upper limit or drops below its lower limit, an alert is generated instantly. This feature ensures users are notified of critical price changes, enabling quick decision-making in real-time trading environments.

3. **Efficient Search and Retrieval:**

   The AVL tree's inherent balancing properties ensure efficient searching and retrieval of stock data. With a time complexity of O(logn) for search operations, users can quickly query any stock to view its current price or compare it with predefined thresholds. This functionality is particularly useful in fast-paced financial scenarios, where users may need to access specific stock information quickly to respond to market trends.

4. **Sorted Data Representation:**

   The system provides an in-order traversal of the AVL tree, displaying all stocks in sorted order based on their identifiers. This feature helps users view the entire stock portfolio in an organized manner, making it easier to analyze data. For example, investors can monitor stocks alphabetically or by unique IDs, facilitating systematic review and management of their portfolio.

5. **Real-Time Updates and Balance Maintenance:**

   The AVL tree's balancing mechanism ensures that the system remains efficient even with frequent updates. Every time a stock price is updated, the tree dynamically adjusts itself to maintain balance, preserving the O(logn) performance for subsequent operations.

GANTT CHART

Task	Day 1	Day 2	Day 3	Day 4	Day 5	Day 6
Research and selection of algorithms, literature review	✓					
Initial design and implementation of hybrid models		✓				
Development of optimization techniques			✓			
Development of user interface and integration of functionalities				✓		
Testing and evaluation on various datasets					✓	
Documentation, final report preparation, and project presentation						✓

Project Timeline & Milestones :

Day 1: Project Initiation

- Activities: Define project objectives, scope, and deliverables. Conduct initial research on AVL trees and real-time alert systems. Create a detailed project plan.
- Milestone: Completion of requirement gathering and finalization of the project plan.

Day 2: System Design

- Activities: Design the architecture of the system, including the structure of the AVL tree, alert generation logic, and system workflow. Document the design specifications and finalize the tools and technologies to be used.
- Milestone: Completion of system design documentation and approval.

Day 3: Core Development - AVL Tree Module

- Activities: Implement the AVL tree with insertion, deletion, and balancing operations. Ensure that the data structure can handle dynamic updates. Begin testing AVL operations for correctness.
- Milestone: Functional AVL tree implementation with basic operations tested.

Day 4: Core Development - Alert System

- Activities: Develop the alert generation mechanism based on predefined thresholds. Integrate the alert system with the AVL tree to trigger notifications when price limits are breached.
- Milestone: Completion of alert system integration and functionality verification.

Day 5: Integration & Testing

- Activities: Integrate the AVL tree and alert system modules. Conduct system-level testing for functionality, performance, and reliability under simulated real-time conditions. Resolve any identified bugs or issues.
- Milestone: Fully integrated system with successful completion of testing.

Day 6: Deployment & User Training

- Activities: Deploy the system in a test environment. Create user documentation and conduct training sessions to guide users in configuring and using the system.
- Milestone: Successful deployment in a test environment and user training completed.

Day 7: Final Review & Enhancements

- Activities: Collect user feedback to identify potential improvements. Implement minor enhancements and optimizations. Prepare and deliver the final project presentation.

CODING:

```
#include <stdio.h>
#include <stdlib.h>
typedef struct AVLNode {
   int stockID;         //
   float stockPrice;
   float lowerLimit;
   float upperLimit;
   struct AVLNode* left;
   struct AVLNode* right;
   int height;
} AVLNode;
AVLNode* createNode(int stockID, float stockPrice, float lowerLimit, float upperLimit) {
   AVLNode* node = (AVLNode*)malloc(sizeof(AVLNode));
   node->stockID = stockID;
   node->stockPrice = stockPrice;
   node->lowerLimit = lowerLimit;
   node->upperLimit = upperLimit;
   node->left = node->right = NULL;
   node->height = 1; // New node is initially at height 1
   return node;
}

int getHeight(AVLNode* node) {
   return node ? node->height : 0;
}

int getBalance(AVLNode* node) {
   return node ? getHeight(node->left) - getHeight(node->right) : 0;
}

AVLNode* rightRotate(AVLNode* y) {
   AVLNode* x = y->left;
   AVLNode* T2 = x->right;

   x->right = y;
   y->left = T2;

   y->height = 1 + (getHeight(y->left) > getHeight(y->right) ? getHeight(y->left) :
getHeight(y->right));
   x->height = 1 + (getHeight(x->left) > getHeight(x->right) ? getHeight(x->left) :
getHeight(x->right));

   return x;
}

AVLNode* leftRotate(AVLNode* x) {
   AVLNode* y = x->right;
```

```
    AVLNode* T2 = y->left;

    y->left = x;
    x->right = T2;

    x->height = 1 + (getHeight(x->left) > getHeight(x->right) ? getHeight(x->left) :
getHeight(x->right));
    y->height = 1 + (getHeight(y->left) > getHeight(y->right) ? getHeight(y->left) :
getHeight(y->right));

    return y;
}

AVLNode* insert(AVLNode* root, int stockID, float stockPrice, float lowerLimit, float
upperLimit) {
    if (!root) return createNode(stockID, stockPrice, lowerLimit, upperLimit);
    if (stockID < root->stockID)
        root->left = insert(root->left, stockID, stockPrice, lowerLimit, upperLimit);
    else if (stockID > root->stockID)
        root->right = insert(root->right, stockID, stockPrice, lowerLimit, upperLimit);
    else {
        root->stockPrice = stockPrice;
        root->lowerLimit = lowerLimit;
        root->upperLimit = upperLimit;
        return root;
    }
    root->height = 1 + (getHeight(root->left) > getHeight(root->right) ? getHeight(root-
>left) : getHeight(root->right));

    int balance = getBalance(root);

    if (balance > 1 && stockID < root->left->stockID)
        return rightRotate(root);

    if (balance < -1 && stockID > root->right->stockID)
        return leftRotate(root);

    if (balance > 1 && stockID > root->left->stockID) {
        root->left = leftRotate(root->left);
        return rightRotate(root);
    }
    if (balance < -1 && stockID < root->right->stockID) {
        root->right = rightRotate(root->right);
        return leftRotate(root);
    }

    return root;
}
```

```
void checkAlerts(AVLNode* node) {
  if (node) {
    checkAlerts(node->left);

    if (node->stockPrice < node->lowerLimit) {
      printf("ALERT: Stock ID %d has fallen below the lower limit! Current Price: %.2f,
Lower Limit: %.2f\n",
          node->stockID, node->stockPrice, node->lowerLimit);
    } else if (node->stockPrice > node->upperLimit) {
      printf("ALERT: Stock ID %d has exceeded the upper limit! Current Price: %.2f,
Upper Limit: %.2f\n",
          node->stockID, node->stockPrice, node->upperLimit);
    }
    checkAlerts(node->right);
  }
}
void inorder(AVLNode* root) {
  if (root) {
    inorder(root->left);
    printf("Stock ID: %d, Price: %.2f, Lower Limit: %.2f, Upper Limit: %.2f\n",
        root->stockID, root->stockPrice, root->lowerLimit, root->upperLimit);
    inorder(root->right);
  }
}
int main() {
  AVLNode* root = NULL;
  printf("Adding stocks to the system...\n");
  root = insert(root, 1, 100.0, 90.0, 110.0);
  root = insert(root, 2, 150.0, 140.0, 160.0);
  root = insert(root, 3, 200.0, 190.0, 210.0);

  printf("\nCurrent Stock Data (In-Order Traversal):\n");
  inorder(root);

  printf("\nChecking for alerts...\n");
  checkAlerts(root);

  printf("\nUpdating stock prices...\n");
  root = insert(root, 1, 85.0, 90.0, 110.0);
  root = insert(root, 2, 165.0, 140.0, 160.0);

  printf("\nCurrent Stock Data After Updates (In-Order Traversal):\n");
  inorder(root);

  printf("\nChecking for alerts after updates...\n");
  checkAlerts(root);

  return 0;
}
```

Conclusion

The **Real-Time Stock Price Alert System Using AVL Trees** demonstrates the effectiveness of combining efficient data structures with real-time monitoring capabilities. By utilizing AVL trees, the system ensures that operations like insertion, deletion, and search are performed in O(logn) time, making it a reliable choice for applications that require high performance and scalability. This solution provides a robust framework for managing dynamic stock price data while maintaining system balance and responsiveness.

One of the most significant advantages of this system is its ability to generate real-time alerts. By allowing users to define specific price thresholds for each stock, the system ensures that they are immediately notified of critical changes in the stock prices. This feature empowers users to make informed and timely decisions, reducing risks and maximizing opportunities in the volatile stock market. The instantaneous response to threshold violations highlights the system's reliability and suitability for real-world applications.

Scalability is another key aspect of this system. As the number of monitored stocks increases, the AVL tree structure ensures that the system continues to perform efficiently. This makes the system capable of handling both individual portfolios and large-scale institutional stock data. Its consistent performance under heavy data loads underscores its potential to be deployed in various financial contexts, from personal investment management tools to enterprise-level trading platforms.

Furthermore, the system offers users an organized way to view and manage stock data through in-order traversal, which displays stocks in sorted order. This functionality helps users quickly assess market trends and track their investments. Coupled with the system's ability to dynamically update stock prices, it ensures that users always have access to accurate and up-to-date information, enhancing their decision-making capabilities.

The adaptability of this system also makes it an excellent foundation for future enhancements. Features like integration with live market data feeds, visualization tools for price trends, and additional alert conditions could be incorporated without compromising performance. These possibilities make the system a versatile solution for financial analytics and stock monitoring, adaptable to evolving user needs and technological advancements.

In conclusion, the **Real-Time Stock Price Alert System Using AVL Trees** provides an efficient, scalable, and reliable approach to managing and monitoring stock prices in real-time. It combines the computational advantages of AVL trees with practical features like alerts and data organization to offer a complete solution for both individual and institutional users. By ensuring real-time responsiveness and scalability, this system meets the critical demands of modern financial markets, laying the groundwork for future innovations in stock management technology.