ScapeGoatTree

Generated by Doxygen 1.16.0

Chapter 1

Scapegoat Tree Implementation

▯() ▯() ▯()

1.0.1 A Self-Balancing BST with Multi-Platform Support

Academic Project: Data Structures Course
Language: C++26 with Python Bindings
Constraint: Minimal STL Usage (Custom Containers)

Note: This is the most comprehensive publicly available Scapegoat Tree project, including multiple user interfaces, Python bindings, advanced operations, and extensive testing.

1.1 Documentation

→ View Full API Documentation
Complete API reference with:

- Class hierarchies and relationships
- Detailed method descriptions with complexity analysis
- Code examples and usage patterns
- Interactive search functionality

1.2 Features

1.2.1 Core Tree Features

- -Weight-Balanced Scapegoat Tree
- Automatic height-balanced rebalancing
- Supports insert, delete, search
- Sum in range — efficiently compute the sum of all values within a given range

- Values in range — retrieve all elements within a specified range
- Kth smallest element — find the element at a specific order in sorted sequence
- Get successor — find the next higher element in the tree
- Get minimum / maximum — retrieve the smallest or largest element in the tree
- Batch operations for efficiency
- Undo/Redo system
- Tree merging with duplicate handling
- Operator overloading for intuitive syntax

1.2.2 Custom Data Structures

- Vector: Dynamic array, automatic resizing, minimal memory overhead
- Queue: Singly-linked list for level-order traversal
- Stack: Built on Vector, used for undo/redo

1.2.3 User Interfaces

- Terminal UI (TUI) with color-coded menus
- Python Tkinter GUI with animations
- DirectX 11 + ImGui GUI (Windows only)

1.2.4 Advanced Usage

- Cross-language Python bindings via pybind11
- Custom parameter for tree balancing
- Detailed balance checking and traversal outputs

1.3 Quick Start

1.3.1 Running the Python GUI (Cross-Platform)

```
# 1. Build the project
mkdir build && cd build
cmake .. && make

# 2. Run the animated visualizer
python ../py.py
```

1.3.2 Running the C++ Terminal UI

```
# After building, run the executable
./TUI   # Terminal-based User Interface
```

1.3.3 Running via Docker

The provided Dockerfile runs the Terminal UI directly:

```
docker build -t scapegoat .
docker run -it --rm scapegoat
```

1.4 Algorithm Overview

A Scapegoat Tree is a self-balancing BST that maintains balance through periodic rebuilding:

- -weight-balanced: No subtree can exceed × parent's size (= 2/3)
- Height bound: h log . (n), where n = number of nodes
- Lazy rebalancing: Rebuilds only when balance is violated

Advantages:

- Simpler than AVL or Red-Black trees (no color/height metadata)
- Amortized efficiency: rebuilds are rare, fast average-case operations
- Space-efficient: minimal per-node overhead

1.4.1 Complexity Analysis

Operation	Time Complexity	Space
Search	O(log n) worst-case	O(1)
Insert	O(log n) amortized	O(1)
Delete	O(log n) amortized	O(1)
Rebuild	O(n) occasional	O(n) temporary
Traversal	O(n)	O(n) for level-order

1.5 Prerequisites

1.5.1 Required

- C++ Compiler: GCC 9+, Clang 10+, or MSVC 2019+
- CMake 3.15+
- Python 3.7+
- pybind11 (pip install pybind11)

1.5.2 Optional

- DirectX 11 SDK (for Windows GUI)
- Tkinter (usually included with Python)
- Doxygen (for generating documentation locally)

1.6 Installation

1.6.1 Windows

```
pip install pybind11

mkdir build && cd build
cmake ..
cmake --build . --config Release
```

1.6.2 Linux/Mac

```
pip install pybind11

mkdir build && cd build
cmake ..
make
```

1.7 Usage Examples

1.7.1 Python Interface

```python
import scapegoat_tree_py as sgt

# Create tree and insert values
tree = sgt.ScapeGoatTree()
tree.insert_batch([10, 20, 30, 5, 15])

# Use undo/redo
tree.undo()
tree.redo()

# Range queries
sum_result = tree.SuminRange(10, 30)
values = tree.ValuesInRange(5, 20)

# Merge trees
tree2 = sgt.ScapeGoatTree()
tree2.insert_batch([25, 35])
merged = tree + tree2
```

1.7.2 C++ Interface

```cpp
#include "ScapeGoatTree.hpp"

ScapeGoatTree<int> tree;

// Insert values
tree.insert(100);
tree + 200;  // Operator overload

// Delete values
tree - 100;  // Operator overload

// Batch operations
tree.insertBatch({10, 20, 30});

// Undo/Redo
tree.undo();
tree.redo();

// Range queries
int sum = tree.sumInRange(10, 50);
Vector<int> values = tree.valuesInRange(10, 50);
```

1.8 Testing

1.8.1 Run Unit Tests

```
cd build
./unit_tests
```

Comprehensive test suite includes:

- Basic operations and edge cases
- Automatic rebalancing verification

- Operator overloading
- Undo/Redo system
- Batch operations
- Copy and move semantics
- Stress testing with 50,000 operations

1.9 Project Structure

```
ScapeGoatTree/
  .github/
    workflows/
      docs.yml              # Auto-generate documentation
  CPP/
    ScapeGoatTree.hpp/tpp     # Main tree implementation
    Node.hpp                  # Node structure
    vector.hpp               # Custom dynamic array
    queue.hpp/tpp              # Custom queue
    stack.hpp                # Custom stack
    bindings.cpp              # Pybind11 bindings
    iTree.cpp/hpp              # Terminal UI
    TreeDriver.cpp             # DirectX + ImGui GUI
    RunTUI.cpp                 # Entry for terminal interface
    tests.cpp                # Unit test suite
  py.py                      # Python Tkinter GUI
  CMakeLists.txt               # Build configuration
  Doxyfile                   # Documentation config
  Dockerfile                 # Container deployment
  LICENSE.md
  README.md
```

1.10 Learning Resources

For more information about Scapegoat Trees:

- Original Paper (1993) by Galperin & Rivest
- API Documentation

1.11 License

1.12 Author

Omar Ahmed Abdel Hameed

- GitHub: @omarahmedthe25th
- Project Link: https://github.com/omarahmedthe25th/ScapeGoatTree

---

1.13 Acknowledgments

- Course: Data Structures

- Inspiration: Self-balancing tree algorithms
- Special thanks to the open-source community

Star this repo if you found it helpful!

Chapter 2

Directory Hierarchy

2.1 Directories

Chapter 3

Class Index

3.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 4

File Index

4.1 File List

Here is a list of all files with brief descriptions:

Chapter 5

Directory Documentation

5.1 CPP Directory Reference

Files

- file benchmark.cpp
- file bindings.cpp
- file iTree.cpp
- file iTree.hpp
- file Node.hpp
- file queue.hpp
- file RunTUI.cpp
- file ScapeGoatTree.hpp
- file stack.hpp
- file tests.cpp
- file TreeDriver.cpp
- file vector.hpp

Chapter 6

Class Documentation

6.1 Command< T > Struct Template Reference

#include <ScapeGoatTree.hpp>

Public Attributes

- OpType type
- T value

6.1.1 Detailed Description

template<typename T>
struct Command< T >

Encapsulates a command that can be undone or redone.

6.1.2 Member Data Documentation

6.1.2.1 type

template<typename T>
OpType Command< T >::type

6.1.2.2 value

template<typename T>
T Command< T >::value

The documentation for this struct was generated from the following file:

- CPP/ScapeGoatTree.hpp

6.2 ScapeGoatTree< T >::iterator Class Reference

Public Member Functions

- iterator (TreeNode ∗node)
- T & operator∗ ()
- iterator & operator++ ()
- iterator operator++ (int)
- bool operator!= (const iterator &other) const

Private Attributes

- TreeNode ∗ curr

6.2.1 Constructor & Destructor Documentation

6.2.1.1 iterator()

template<typename T>
ScapeGoatTree< T >::iterator::iterator (
    TreeNode ∗ node) [inline]

6.2.2 Member Function Documentation

6.2.2.1 operator"!=()

template<typename T>
bool ScapeGoatTree< T >::iterator::operator!= (
    const iterator & other) const [inline]

6.2.2.2 operator∗()

template<typename T>
T & ScapeGoatTree< T >::iterator::operator∗ () [inline]

6.2.2.3 operator++() [1/2]

template<typename T>
iterator & ScapeGoatTree< T >::iterator::operator++ () [inline]

6.2.2.4 operator++() [2/2]

template<typename T>
iterator ScapeGoatTree< T >::iterator::operator++ (
    int) [inline]

6.2.3 Member Data Documentation

6.2.3.1 curr

template<typename T>
TreeNode∗ ScapeGoatTree< T >::iterator::curr [private]
The documentation for this class was generated from the following file:

- CPP/ScapeGoatTree.hpp

6.3 ITree Class Reference

#include <iTree.hpp>

Static Public Member Functions

- static void TreeUI ()

Static Private Member Functions

- static void handleBatches (ScapeGoatTree< ElemenType > &A, ScapeGoatTree< ElemenType > &B, opcodes op)
- static void handleOperations (ScapeGoatTree< ElemenType > &A, ScapeGoatTree< ElemenType > &B, opcodes op)
- static void handleDisplay (ScapeGoatTree< ElemenType > &A, ScapeGoatTree< ElemenType > &B, opcodes op)
- static void handleBalance (ScapeGoatTree< ElemenType > &A, ScapeGoatTree< ElemenType > &B)
- static void handleCoreOperators (ScapeGoatTree< ElemenType > &A, ScapeGoatTree< ElemenType > &B, opcodes op)

- static void handleOperatorEmpty (ScapeGoatTree< ElemenType > &A, ScapeGoatTree< ElemenType > &B)
- static void handleOperatorMerge (ScapeGoatTree< ElemenType > &A, ScapeGoatTree< ElemenType > &B)
- static void handleOperatorCompare (const ScapeGoatTree< ElemenType > &A, const ScapeGoatTree< ElemenType > &B)
- static ScapeGoatTree< ElemenType > & selectTree (ScapeGoatTree< ElemenType > &A, ScapeGoatTree< ElemenType > &B)
- static void handleClear (ScapeGoatTree< ElemenType > &A, ScapeGoatTree< ElemenType > &B)
- static void handleUndoRedo (ScapeGoatTree< ElemenType > &A, ScapeGoatTree< ElemenType > &B, opcodes op)
- static void handleSuminRange (ScapeGoatTree< ElemenType > &A, ScapeGoatTree< ElemenType > &B)
- static void hanleMinMax (ScapeGoatTree< ElemenType > &A, ScapeGoatTree< ElemenType > &B, opcodes op)
- static void handleValuesinRange (ScapeGoatTree< ElemenType > &A, ScapeGoatTree< ElemenType > &B)
- static void handleKthSmallestElement (ScapeGoatTree< ElemenType > &A, ScapeGoatTree< ElemenType > &B)
- static void handleSucessor (ScapeGoatTree< ElemenType > &A, ScapeGoatTree< ElemenType > &B)

6.3.1 Member Function Documentation

6.3.1.1 handleBalance()

```
void ITree::handleBalance (
                ScapeGoatTree< ElemenType > & A,
                ScapeGoatTree< ElemenType > & B)   [static], [private]
```

Handles checking and reporting the balance status of the trees.

6.3.1.2 handleBatches()

```
void ITree::handleBatches (
                ScapeGoatTree< ElemenType > & A,
                ScapeGoatTree< ElemenType > & B,
                opcodes op)   [static], [private]
```

Handles batch insertion and deletion operations.

6.3.1.3 handleClear()

```
void ITree::handleClear (
                ScapeGoatTree< ElemenType > & A,
                ScapeGoatTree< ElemenType > & B)   [static], [private]
```

Handles clearing the contents of the trees.
Handles clearing the contents of a selected tree.

6.3.1.4 handleCoreOperators()

```
void ITree::handleCoreOperators (
                ScapeGoatTree< ElemenType > & A,
                ScapeGoatTree< ElemenType > & B,
                opcodes op)   [static], [private]
```

Handles core operators like insertion and deletion.
Handles core operators like insertion and deletion using overloaded + and - operators.

6.3.1.5 handleDisplay()

void ITree::handleDisplay (
ScapeGoatTree< ElemenType > & A,
ScapeGoatTree< ElemenType > & B,
opcodes op) [static], [private]

Handles display operations for the trees.

6.3.1.6 handleKthSmallestElement()

void ITree::handleKthSmallestElement (
ScapeGoatTree< ElemenType > & A,
ScapeGoatTree< ElemenType > & B) [static], [private]

6.3.1.7 handleOperations()

void ITree::handleOperations (
ScapeGoatTree< ElemenType > & A,
ScapeGoatTree< ElemenType > & B,
opcodes op) [static], [private]

Handles standard tree operations like search.

6.3.1.8 handleOperatorCompare()

void ITree::handleOperatorCompare (
const ScapeGoatTree< ElemenType > & A,
const ScapeGoatTree< ElemenType > & B) [static], [private]

Handles comparing two trees for equality.
Handles comparing two trees for equality using the == operator.

6.3.1.9 handleOperatorEmpty()

void ITree::handleOperatorEmpty (
ScapeGoatTree< ElemenType > & A,
ScapeGoatTree< ElemenType > & B) [static], [private]

Handles checking if the trees are empty.

6.3.1.10 handleOperatorMerge()

void ITree::handleOperatorMerge (
ScapeGoatTree< ElemenType > & A,
ScapeGoatTree< ElemenType > & B) [static], [private]

Handles merging two trees together.
Handles merging two trees together using the + operator.

6.3.1.11 handleSucessor()

void ITree::handleSucessor (
ScapeGoatTree< ElemenType > & A,
ScapeGoatTree< ElemenType > & B) [static], [private]

6.3.1.12 handleSuminRange()

void ITree::handleSuminRange (
ScapeGoatTree< ElemenType > & A,
ScapeGoatTree< ElemenType > & B) [static], [private]

6.3.1.13 handleUndoRedo()

void ITree::handleUndoRedo (
ScapeGoatTree< ElemenType > & A,

ScapeGoatTree< ElemenType > & B,
opcodes op) [static], [private]

6.3.1.14 handleValuesinRange()

void ITree::handleValuesinRange (
ScapeGoatTree< ElemenType > & A,
ScapeGoatTree< ElemenType > & B) [static], [private]

6.3.1.15 hanleMinMax()

void ITree::hanleMinMax (
ScapeGoatTree< ElemenType > & A,
ScapeGoatTree< ElemenType > & B,
opcodes op) [static], [private]

6.3.1.16 selectTree()

ScapeGoatTree< ElemenType > & ITree::selectTree (
ScapeGoatTree< ElemenType > & A,
ScapeGoatTree< ElemenType > & B) [static], [private]

Prompts the user to select one of the two available trees.
Prompts the user to select one of the two available trees (Tree A or Tree B).

6.3.1.17 TreeUI()

void ITree::TreeUI () [static]

Launches the terminal-based user interface for interacting with the trees.
The documentation for this class was generated from the following files:

- CPP/iTree.hpp
- CPP/iTree.cpp

6.4 MenuItem Struct Reference

Public Attributes

- string name
- opcodes opcode
- MenuHandler func

6.4.1 Member Data Documentation

6.4.1.1 func

MenuHandler MenuItem::func

6.4.1.2 name

string MenuItem::name

6.4.1.3 opcode

opcodes MenuItem::opcode

The documentation for this struct was generated from the following file:

- CPP/iTree.cpp

6.5 Node< T > Class Template Reference

#include <Node.hpp>

Public Member Functions

- Node (const T &v, Node ∗parentPtr=nullptr)

Public Attributes

- T value {}
- Node ∗ left {}
- Node ∗ right {}
- Node ∗ parent {}
- unsigned int size =1

Friends

- template<typename>
  class ScapeGoatTree

6.5.1 Constructor & Destructor Documentation

6.5.1.1 Node()

```
template<typename T>
Node< T >::Node (
                const T & v,
                Node< T > ∗ parentPtr = nullptr)   [inline], [explicit]
```

Initializes a node with a value and an optional parent pointer.

6.5.2 Friends And Related Symbol Documentation

6.5.2.1 ScapeGoatTree

```
template<typename T>
template<typename>
friend class ScapeGoatTree   [friend]
```

6.5.3 Member Data Documentation

6.5.3.1 left

```
template<typename T>
Node∗ Node< T >::left {}
```

6.5.3.2 parent

```
template<typename T>
Node∗ Node< T >::parent {}
```

6.5.3.3 right

```
template<typename T>
Node∗ Node< T >::right {}
```

6.5.3.4 size

```
template<typename T>
unsigned int Node< T >::size =1
```

6.5.3.5 value

template<typename T>
T Node< T >::value {}
The documentation for this class was generated from the following file:

- CPP/Node.hpp

6.6 QNode< T > Class Template Reference

#include <queue.hpp>

Private Member Functions

- QNode (T value)

Private Attributes

- QNode ∗ next {}
- T value {}

Friends

- template<typename>
  class Queue

6.6.1 Constructor & Destructor Documentation

6.6.1.1 QNode()

```
template<typename T>
QNode< T >::QNode (
                    T value)    [inline], [explicit], [private]
```

6.6.2 Friends And Related Symbol Documentation

6.6.2.1 Queue

template<typename T>
template<typename>
friend class Queue [friend]

6.6.3 Member Data Documentation

6.6.3.1 next

template<typename T>
QNode∗ QNode< T >::next {} [private]

6.6.3.2 value

template<typename T>
T QNode< T >::value {} [private]
The documentation for this class was generated from the following file:

- CPP/queue.hpp

6.7 Queue< T > Class Template Reference

#include <queue.hpp>

Public Member Functions

- ~Queue ()
- void push (T value)
- void pop ()
- bool isEmpty () const
- T front ()
- int size () const

Private Attributes

- QNode< T > ∗ head {}
- QNode< T > ∗ tail {}
- int nNodes {}

6.7.1 Constructor & Destructor Documentation

6.7.1.1 ~Queue()

template<typename T>
Queue< T >::~Queue () [inline]
Destroys the queue and releases memory by popping all elements.

6.7.2 Member Function Documentation

6.7.2.1 front()

template<typename T>
T Queue< T >::front ()
Returns the value of the front element without removing it.

6.7.2.2 isEmpty()

template<typename T>
bool Queue< T >::isEmpty () const [nodiscard]
Checks if the queue is empty.

6.7.2.3 pop()

template<typename T>
void Queue< T >::pop ()
Removes the front element from the queue.

6.7.2.4 push()

template<typename T>
void Queue< T >::push (
T value)
Adds a new value to the back of the queue.

6.7.2.5 size()

template<typename T>
int Queue< T >::size () const [nodiscard]
Returns the current number of elements in the queue.

6.7.3 Member Data Documentation

6.7.3.1 head

template<typename T>
QNode<T>∗ Queue< T >::head {} [private]

6.7.3.2 nNodes

template<typename T>
int Queue< T >::nNodes {} [private]

6.7.3.3 tail

template<typename T>
QNode<T>∗ Queue< T >::tail {} [private]
The documentation for this class was generated from the following file:

- CPP/queue.hpp

6.8 ScapeGoatTree< T > Class Template Reference

#include <ScapeGoatTree.hpp>

Classes

- class iterator

Public Member Functions

- ScapeGoatTree ()
- void insert (T value)
- void insertBatch (const Vector< T > &values)
- bool deleteValue (T value)
- void deleteBatch (const Vector< T > &values)
- bool search (const T &key) const
- void clear ()
- void undo ()
- void redo ()
- T sumInRange (T min, T max)
- T getMin ()
- T getMax ()
- Vector< T > valuesInRange (T min, T max)
- T getSuccessor (T value) const
- T kthSmallest (int k) const
- std::string isBalanced () const
- const TreeNode ∗ getRoot ()
- iterator begin ()
- ScapeGoatTree (const ScapeGoatTree &Otree)
- ScapeGoatTree (ScapeGoatTree &&other) noexcept
- ∼ScapeGoatTree ()
- std::string displayPreOrder ()
- std::string displayInOrder ()
- std::string displayPostOrder ()
- std::string displayLevels ()
- bool operator[] (T value) const
- ScapeGoatTree operator+ (const ScapeGoatTree &other) const
- ScapeGoatTree & operator= (const ScapeGoatTree &other)
- ScapeGoatTree & operator= (ScapeGoatTree &&other) noexcept
- ScapeGoatTree & operator= (int value)
- bool operator== (const ScapeGoatTree &tree) const
- bool operator!= (const ScapeGoatTree &tree) const
- bool operator! () const
- void operator+ (const T &value)
- bool operator- (const T &value)
- bool operator-= (const T &value)
- void operator+= (const T &value)

Static Public Member Functions

- static iterator end ()

Private Types

- using TreeNode = Node<T>

Private Member Functions

- TreeNode ∗ rebuildTree (int start, int end, TreeNode ∗parent_node, T ∗array)
- void inorderTraversal (const TreeNode ∗node, int &i, T ∗&array) const
- void preorderTraversal (const TreeNode ∗node)
- void displayPreOrder (const TreeNode ∗node, std::ostream &os)
- void displayInOrder (const TreeNode ∗node, std::ostream &os)
- void displayPostOrder (const TreeNode ∗node, std::ostream &os)
- int getThreshold () const
- void DeletionRebuild ()
- bool areTreesEqual (const TreeNode ∗n1, const TreeNode ∗n2) const
- void restructure_subtree (TreeNode ∗newNode)
- T sumHelper (TreeNode ∗node, T min, T max)
- void rangeHelper (TreeNode ∗node, T min, T max, Vector< T > &range)
- T kthSmallestHelper (TreeNode ∗node, int k) const

Static Private Member Functions

- static int findH (const TreeNode ∗node)
- static unsigned int countN (const TreeNode ∗node)
- static TreeNode ∗ findTraitor (TreeNode ∗node)
- static void postorderTraversal (const TreeNode ∗node)
- static TreeNode ∗ getSuccessor (TreeNode ∗node)

Private Attributes

- TreeNode ∗ root {}
- int nNodes {}
- int rebuildCount = 0
- Stack< Command< T > > undoStack
- Stack< Command< T > > redoStack
- bool isUndoing = false
- int max_nodes = 0

6.8.1 Member Typedef Documentation

6.8.1.1 TreeNode

template<typename T>
using ScapeGoatTree< T >::TreeNode = Node<T> [private]

6.8.2 Constructor & Destructor Documentation

6.8.2.1 ScapeGoatTree() [1/3]

template<typename T>
ScapeGoatTree< T >::ScapeGoatTree ()
Default constructor for an empty Scapegoat Tree.

6.8.2.2 ScapeGoatTree() [2/3]

template<typename T>
ScapeGoatTree< T >::ScapeGoatTree (
    const ScapeGoatTree< T > & Otree)

Copy constructor for deep copying another ScapeGoatTree.

6.8.2.3 ScapeGoatTree() [3/3]

template<typename T>
ScapeGoatTree< T >::ScapeGoatTree (
    ScapeGoatTree< T > && other) [noexcept]

Move constructor for transferring ownership from another ScapeGoatTree.

6.8.2.4 ~ScapeGoatTree()

template<typename T>
ScapeGoatTree< T >::~ScapeGoatTree ()

Destructor that cleans up all nodes in the tree.

6.8.3 Member Function Documentation

6.8.3.1 areTreesEqual()

template<typename T>
bool ScapeGoatTree< T >::areTreesEqual (
    const TreeNode * n1,
    const TreeNode * n2) const [private]

Compares two subtrees for structural and value equality.

6.8.3.2 begin()

template<typename T>
iterator ScapeGoatTree< T >::begin ()

6.8.3.3 clear()

template<typename T>
void ScapeGoatTree< T >::clear ()

Removes all nodes from the tree and resets its state.

6.8.3.4 countN()

template<typename T>
unsigned int ScapeGoatTree< T >::countN (
    const TreeNode * node) [static], [private]

Counts the total number of nodes in the subtree rooted at the given node.

6.8.3.5 deleteBatch()

template<typename T>
void ScapeGoatTree< T >::deleteBatch (
    const Vector< T > & values)

Removes multiple values from a Vector from the tree.

6.8.3.6 deleteValue()

template<typename T>
bool ScapeGoatTree< T >::deleteValue (
    T value)

Removes a value from the tree and maintains balance if needed.

6.8.3.7 DeletionRebuild()

template<typename T>
void ScapeGoatTree< T >::DeletionRebuild () [private]
Checks if a rebuild is needed after a deletion and performs it if necessary.

6.8.3.8 displayInOrder() [1/2]

template<typename T>
std::string ScapeGoatTree< T >::displayInOrder ()
Returns a string representing the tree in in-order traversal.

6.8.3.9 displayInOrder() [2/2]

template<typename T>
void ScapeGoatTree< T >::displayInOrder (
    const TreeNode ∗ node,
    std::ostream & os) [private]
Formats the tree in in-order.

6.8.3.10 displayLevels()

template<typename T>
std::string ScapeGoatTree< T >::displayLevels ()
Returns a string representing the tree in level-order traversal.

6.8.3.11 displayPostOrder() [1/2]

template<typename T>
std::string ScapeGoatTree< T >::displayPostOrder ()
Returns a string representing the tree in post-order traversal.

6.8.3.12 displayPostOrder() [2/2]

template<typename T>
void ScapeGoatTree< T >::displayPostOrder (
    const TreeNode ∗ node,
    std::ostream & os) [private]
Formats the tree in post-order.

6.8.3.13 displayPreOrder() [1/2]

template<typename T>
std::string ScapeGoatTree< T >::displayPreOrder ()
Returns a string representing the tree in pre-order traversal.

6.8.3.14 displayPreOrder() [2/2]

template<typename T>
void ScapeGoatTree< T >::displayPreOrder (
    const TreeNode ∗ node,
    std::ostream & os) [private]
Formats the tree in pre-order.

6.8.3.15 end()

template<typename T>
iterator ScapeGoatTree< T >::end () [static]

6.8.3.16 findH()

template<typename T>
int ScapeGoatTree< T >::findH (
  const TreeNode ∗ node) [static], [private]

Calculates the height of a given node in the tree.

6.8.3.17 findTraitor()

template<typename T>
TreeNode ∗ ScapeGoatTree< T >::findTraitor (
  TreeNode ∗ node) [static], [private]

Finds the highest node that violates the alpha-weight-balance property.

6.8.3.18 getMax()

template<typename T>
T ScapeGoatTree< T >::getMax ()

6.8.3.19 getMin()

template<typename T>
T ScapeGoatTree< T >::getMin ()

6.8.3.20 getRoot()

template<typename T>
const TreeNode ∗ ScapeGoatTree< T >::getRoot ()

Returns a pointer to the root node of the tree.

6.8.3.21 getSuccessor() [1/2]

template<typename T>
T ScapeGoatTree< T >::getSuccessor (
  T value) const

6.8.3.22 getSuccessor() [2/2]

template<typename T>
TreeNode ∗ ScapeGoatTree< T >::getSuccessor (
  TreeNode ∗ node) [static], [private]

6.8.3.23 getThreshold()

template<typename T>
int ScapeGoatTree< T >::getThreshold () const [inline], [nodiscard], [private]

Calculates the maximum allowed height before a rebuild is triggered.

6.8.3.24 inorderTraversal()

template<typename T>
void ScapeGoatTree< T >::inorderTraversal (
  const TreeNode ∗ node,
  int & i,
  T ∗& array) const [private]

Performs an in-order traversal to populate a sorted array with node values.

6.8.3.25 insert()

template<typename T>
void ScapeGoatTree< T >::insert (
  T value)

Inserts a new value into the tree and maintains balance if needed.

6.8.3.26 insertBatch()

template<typename T>
void ScapeGoatTree< T >::insertBatch (
 const Vector< T > & values)

Inserts multiple values from a Vector into the tree.

6.8.3.27 isBalanced()

template<typename T>
std::string ScapeGoatTree< T >::isBalanced () const [nodiscard]

Returns a string report indicating if the tree is currently balanced.

6.8.3.28 kthSmallest()

template<typename T>
T ScapeGoatTree< T >::kthSmallest (
 int k) const

6.8.3.29 kthSmallestHelper()

template<typename T>
T ScapeGoatTree< T >::kthSmallestHelper (
 TreeNode ∗ node,
 int k) const [private]

6.8.3.30 operator"!()

template<typename T>
bool ScapeGoatTree< T >::operator! () const

Checks if the tree is empty.

6.8.3.31 operator"!=()

template<typename T>
bool ScapeGoatTree< T >::operator!= (
 const ScapeGoatTree< T > & tree) const

Checks if two trees are not equal.

6.8.3.32 operator+() [1/2]

template<typename T>
ScapeGoatTree ScapeGoatTree< T >::operator+ (
 const ScapeGoatTree< T > & other) const

Creates a new tree containing elements from both trees.

6.8.3.33 operator+() [2/2]

template<typename T>
void ScapeGoatTree< T >::operator+ (
 const T & value)

Overloaded plus operator for inserting a value.

6.8.3.34 operator+=()

template<typename T>
void ScapeGoatTree< T >::operator+= (
 const T & value)

Overloaded addition assignment operator for inserting a value.

6.8.3.35 operator-()

```
template<typename T>
bool ScapeGoatTree< T >::operator- (
                const T & value)
```

Overloaded minus operator for deleting a value.

6.8.3.36 operator-=()

```
template<typename T>
bool ScapeGoatTree< T >::operator-= (
                const T & value)
```

Overloaded subtraction assignment operator for deleting a value.

6.8.3.37 operator=() [1/3]

```
template<typename T>
ScapeGoatTree & ScapeGoatTree< T >::operator= (
                const ScapeGoatTree< T > & other)
```

Assignment operator for deep copying.

6.8.3.38 operator=() [2/3]

```
template<typename T>
ScapeGoatTree & ScapeGoatTree< T >::operator= (
                int value)
```

Clears the current tree and initializes it with a single value.

6.8.3.39 operator=() [3/3]

```
template<typename T>
ScapeGoatTree & ScapeGoatTree< T >::operator= (
                ScapeGoatTree< T > && other)   [noexcept]
```

Move assignment operator.

6.8.3.40 operator==()

```
template<typename T>
bool ScapeGoatTree< T >::operator== (
                const ScapeGoatTree< T > & tree) const
```

Checks if two trees are equal.

6.8.3.41 operator[]()

```
template<typename T>
bool ScapeGoatTree< T >::operator[] (
                T value) const
```

Overloaded subscript operator to search for a value in the tree.

6.8.3.42 postorderTraversal()

```
template<typename T>
void ScapeGoatTree< T >::postorderTraversal (
                const TreeNode * node)   [static], [private]
```

Recursively deletes all nodes in the subtree using post-order traversal.

6.8.3.43 preorderTraversal()

```
template<typename T>
void ScapeGoatTree< T >::preorderTraversal (
                const TreeNode * node)   [private]
```

Performs a pre-order traversal for internal processing.

6.8.3.44 rangeHelper()

template<typename T>
void ScapeGoatTree< T >::rangeHelper (
    TreeNode ∗ node,
    T min,
    T max,
    Vector< T > & range) [private]

6.8.3.45 rebuildTree()

template<typename T>
TreeNode ∗ ScapeGoatTree< T >::rebuildTree (
    int start,
    int end,
    TreeNode ∗ parent_node,
    T ∗ array) [private]

Recursively rebuilds a balanced BST from a sorted array of values.

6.8.3.46 redo()

template<typename T>
void ScapeGoatTree< T >::redo ()

6.8.3.47 restructure_subtree()

template<typename T>
void ScapeGoatTree< T >::restructure_subtree (
    TreeNode ∗ newNode) [private]

Initiates a subtree rebuild starting from the scapegoat node.

6.8.3.48 search()

template<typename T>
bool ScapeGoatTree< T >::search (
    const T & key) const [nodiscard]

Searches for a specific value in the tree.

6.8.3.49 sumHelper()

template<typename T>
T ScapeGoatTree< T >::sumHelper (
    TreeNode ∗ node,
    T min,
    T max) [private]

6.8.3.50 sumInRange()

template<typename T>
T ScapeGoatTree< T >::sumInRange (
    T min,
    T max)

6.8.3.51 undo()

template<typename T>
void ScapeGoatTree< T >::undo ()

6.8.3.52 valuesInRange()

template<typename T>
Vector< T > ScapeGoatTree< T >::valuesInRange (
T min,
T max)

6.8.4 Member Data Documentation

6.8.4.1 isUndoing

template<typename T>
bool ScapeGoatTree< T >::isUndoing = false [private]
Flag to prevent operations triggered by undo/redo from being recorded. This avoids infinite recursion and keeps the undo history clean.

6.8.4.2 max_nodes

template<typename T>
int ScapeGoatTree< T >::max_nodes = 0 [private]

6.8.4.3 nNodes

template<typename T>
int ScapeGoatTree< T >::nNodes {} [private]

6.8.4.4 rebuildCount

template<typename T>
int ScapeGoatTree< T >::rebuildCount = 0 [private]

6.8.4.5 redoStack

template<typename T>
Stack<Command<T> > ScapeGoatTree< T >::redoStack [private]
Stack to store commands that have been undone and can be redone.

6.8.4.6 root

template<typename T>
TreeNode∗ ScapeGoatTree< T >::root {} [private]

6.8.4.7 undoStack

template<typename T>
Stack<Command<T> > ScapeGoatTree< T >::undoStack [private]
Stack to store commands that can be undone.
The documentation for this class was generated from the following file:

- CPP/ScapeGoatTree.hpp

6.9 Stack< T > Class Template Reference

#include <stack.hpp>

Public Member Functions

- void push (const T &value)
- T pop ()
- T top ()
- unsigned int size () const
- bool isEmpty () const

Private Attributes

- Vector< T > data

6.9.1 Member Function Documentation

6.9.1.1 isEmpty()

template<typename T>
bool Stack< T >::isEmpty () const [inline], [nodiscard]
Checks if the stack is empty.

6.9.1.2 pop()

template<typename T>
T Stack< T >::pop () [inline]
Removes and returns the top element from the stack. Throws std::out_of_range if the stack is empty.

6.9.1.3 push()

template<typename T>
void Stack< T >::push (
const T & value) [inline]
Pushes a new element onto the stack.

6.9.1.4 size()

template<typename T>
unsigned int Stack< T >::size () const [inline], [nodiscard]
Returns the number of elements currently in the stack.

6.9.1.5 top()

template<typename T>
T Stack< T >::top () [inline]

6.9.2 Member Data Documentation

6.9.2.1 data

template<typename T>
Vector<T> Stack< T >::data [private]
The documentation for this class was generated from the following file:

- CPP/stack.hpp

6.10 Vector< T > Class Template Reference

#include <vector.hpp>

Public Member Functions

- ~Vector ()
- Vector ()=default
- unsigned int size () const
- void push_back (const T &value)
- T pop_back ()
- T * begin ()
- T * end ()
- T & operator[ ] (unsigned int index)

- const T & operator[] (unsigned int index) const
- Vector (const Vector &other)
- Vector & operator= (const Vector &)=delete
- Vector (Vector &&)=delete
- Vector & operator= (Vector &&)=delete

Private Attributes

- unsigned int _size = 50
- int nElements = 0
- T ∗ data = new T[_size]{}

Friends

- template<typename>
  class Stack
- template<typename>
  class ScapeGoatTree

6.10.1 Constructor & Destructor Documentation

6.10.1.1 ∼Vector()

template<typename T>
Vector< T >::∼Vector () [inline]
Destroys the vector and releases the dynamically allocated memory.

6.10.1.2 Vector() [1/3]

template<typename T>
Vector< T >::Vector () [default]
Default constructor for the Vector class.

6.10.1.3 Vector() [2/3]

template<typename T>
Vector< T >::Vector (
    const Vector< T > & other) [inline]

6.10.1.4 Vector() [3/3]

template<typename T>
Vector< T >::Vector (
    Vector< T > &&) [delete]

6.10.2 Member Function Documentation

6.10.2.1 begin()

template<typename T>
T ∗ Vector< T >::begin () [inline]

6.10.2.2 end()

template<typename T>
T ∗ Vector< T >::end () [inline]

6.10.2.3 operator=() [1/2]

template<typename T>
Vector & Vector< T >::operator= (
    const Vector< T > &) [delete]

6.10.2.4 operator=() [2/2]

template<typename T>
Vector & Vector< T >::operator= (
    Vector< T > &&) [delete]

6.10.2.5 operator[]() [1/2]

template<typename T>
T & Vector< T >::operator[] (
    unsigned int index) [inline]

Provides access to the element at the specified index.

6.10.2.6 operator[]() [2/2]

template<typename T>
const T & Vector< T >::operator[] (
    unsigned int index) const [inline]

Provides read-only access to the element at the specified index.

6.10.2.7 pop_back()

template<typename T>
T Vector< T >::pop_back () [inline]

Removes and returns the last element. Throws std::out_of_range if the vector is empty. Shrinks internal storage when usage falls to 1/4 of capacity (min capacity 50).

6.10.2.8 push_back()

template<typename T>
void Vector< T >::push_back (
    const T & value) [inline]

Appends a new element to the end of the vector, resizing if necessary.

6.10.2.9 size()

template<typename T>
unsigned int Vector< T >::size () const [inline], [nodiscard]

Returns the number of elements currently stored in the vector.

6.10.3 Friends And Related Symbol Documentation

6.10.3.1 ScapeGoatTree

template<typename T>
template<typename>
friend class ScapeGoatTree [friend]

6.10.3.2 Stack

template<typename T>
template<typename>
friend class Stack [friend]

6.10.4 Member Data Documentation

6.10.4.1 _size

template<typename T>
unsigned int Vector< T >::_size = 50 [private]

6.10.4.2 data

template<typename T>
T* Vector< T >::data = new T[_size]{} [private]

6.10.4.3 nElements

template<typename T>
int Vector< T >::nElements = 0 [private]

The documentation for this class was generated from the following file:

- CPP/vector.hpp

Chapter 7

File Documentation

7.1 CPP/benchmark.cpp File Reference

#include <iostream>
#include <chrono>
#include <set>
#include ”ScapeGoatTree.hpp”

Functions

- void benchmark_sequential_ops ()
- int main ()

7.1.1 Function Documentation

7.1.1.1 benchmark_sequential_ops()

void benchmark_sequential_ops ()

7.1.1.2 main()

int main ()

7.2 CPP/bindings.cpp File Reference

#include <pybind11/pybind11.h>
#include <pybind11/operators.h>
#include <pybind11/stl.h>
#include ”ScapeGoatTree.hpp”

Typedefs

- typedef long long Type

Functions

- PYBIND11_MODULE (scapegoat_tree_py, m)

7.2.1 Typedef Documentation

7.2.1.1 Type

typedef long long Type

7.2.2 Function Documentation

7.2.2.1 PYBIND11_MODULE()

PYBIND11_MODULE (
    scapegoat_tree_py ,
    m)

Pybind11 module for exposing the ScapeGoatTree implementation to Python.

7.3 CPP/iTree.cpp File Reference

#include "iTree.hpp"
#include <limits>
#include <iostream>
#include "print"

Classes

- struct MenuItem

Typedefs

- typedef void(∗ MenuHandler) (ScapeGoatTree< ElemenType > &, ScapeGoatTree< ElemenType > &, opcodes)
- typedef const unsigned long long Long

Functions

- void printError (const string &msg)
- void printSuccess (const string &msg)
- void printInfo (const string &msg)
- void printHeader (const std::string &title)
- bool validateCinLine ()

Variables

- const string RED = "\033[31m"
- const string GREEN = "\033[32m"
- const string CYAN = "\033[36m"
- const string RESET = "\033[0m"
- const string WHITE = "\033[37m"

7.3.1 Typedef Documentation

7.3.1.1 Long

typedef const unsigned long long Long

7.3.1.2 MenuHandler

typedef void(∗ MenuHandler) (ScapeGoatTree< ElemenType > &, ScapeGoatTree< ElemenType > &, opcodes)

7.3.2 Function Documentation

7.3.2.1 printError()

void printError (
    const string & msg)

7.3.2.2 printHeader()

```
void printHeader (
                    const std::string & title)
```

7.3.2.3 printInfo()

```
void printInfo (
                    const string & msg)
```

7.3.2.4 printSuccess()

```
void printSuccess (
                    const string & msg)
```

7.3.2.5 validateCinLine()

```
bool validateCinLine ()
```

7.3.3 Variable Documentation

7.3.3.1 CYAN

```
const string CYAN = "\033[36m"
```

7.3.3.2 GREEN

```
const string GREEN = "\033[32m"
```

7.3.3.3 RED

```
const string RED = "\033[31m"
```

7.3.3.4 RESET

```
const string RESET = "\033[0m"
```

7.3.3.5 WHITE

```
const string WHITE = "\033[37m"
```

7.4 CPP/iTree.hpp File Reference

#include "ScapeGoatTree.hpp"

Classes

- class ITree

Typedefs

- typedef int ElemenType

Enumerations

- enum class opcodes {
  INSERT , DELETEOP , SEARCH , DISPLAY_INORDER ,
  DISPLAY_PREORDER , DISPLAY_POSTORDER , DISPLAY_LEVELS , EXIT ,
  BALANCE , COMPARE , MERGE , EMPTY ,
  BATCH_INSERT , BATCH_DELETE , CLEAR , UNDO ,

REDO , SUMINRANGE , VALUESINRANGE , MIN ,
MAX , KTH , SUCC }

7.4.1 Typedef Documentation

7.4.1.1 ElemenType

typedef int ElemenType

7.4.2 Enumeration Type Documentation

7.4.2.1 opcodes

enum class opcodes [strong]

Enumerator

INSERT	
DELETEOP	
SEARCH	
DISPLAY_INORDER	
DISPLAY_PREORDER	
DISPLAY_POSTORDER	
DISPLAY_LEVELS	
EXIT	
BALANCE	
COMPARE	
MERGE	
EMPTY	
BATCH_INSERT	
BATCH_DELETE	
CLEAR	
UNDO	
REDO	
SUMINRANGE	
VALUESINRANGE	
MIN	
MAX	
KTH	
SUCC	

7.5 iTree.hpp

Go to the documentation of this file.

```
//
// Created by DELL on 24/12/2025.
//

#ifndef TREE_ITREE_HPP
#define TREE_ITREE_HPP
#include "ScapeGoatTree.hpp"
using namespace std;
typedef int ElemenType;
```

```

enum class opcodes {INSERT, DELETEOP, SEARCH, DISPLAY_INORDER, DISPLAY_PREORDER,
    DISPLAY_POSTORDER,
    DISPLAY_LEVELS,EXIT,BALANCE,COMPARE,MERGE,EMPTY,BATCH_INSERT,BATCH_DELETE,CLEAR,UNDO,REDO,SUMINRAN

class ITree {
    static  void handleBatches(ScapeGoatTree<ElemenType> &A, ScapeGoatTree<ElemenType> &B, opcodes op);

    static void handleOperations(ScapeGoatTree<ElemenType> &A, ScapeGoatTree<ElemenType> &B, opcodes op);

    static void handleDisplay(ScapeGoatTree<ElemenType> &A, ScapeGoatTree<ElemenType> &B, opcodes op);

    static void handleBalance(ScapeGoatTree<ElemenType> &A, ScapeGoatTree<ElemenType> &B);

    static void handleCoreOperators(ScapeGoatTree<ElemenType> &A, ScapeGoatTree<ElemenType> &B, opcodes op);

    static void handleOperatorEmpty(ScapeGoatTree<ElemenType> &A, ScapeGoatTree<ElemenType> &B);

    static void handleOperatorMerge(ScapeGoatTree<ElemenType> &A, ScapeGoatTree<ElemenType> &B);

    static void handleOperatorCompare(const ScapeGoatTree<ElemenType> &A, const ScapeGoatTree<ElemenType>
    &B);

    static ScapeGoatTree<ElemenType>& selectTree(ScapeGoatTree<ElemenType> &A, ScapeGoatTree<ElemenType>
    &B);

    static  void handleClear(ScapeGoatTree<ElemenType> &A, ScapeGoatTree<ElemenType> &B);

    static void handleUndoRedo(ScapeGoatTree<ElemenType> &A, ScapeGoatTree<ElemenType> &B, opcodes op);
    static void handleSuminRange(ScapeGoatTree<ElemenType> &A, ScapeGoatTree<ElemenType> &B);
    static void hanleMinMax(ScapeGoatTree<ElemenType> &A, ScapeGoatTree<ElemenType> &B, opcodes op);
    static void handleValuesinRange(ScapeGoatTree<ElemenType> &A, ScapeGoatTree<ElemenType> &B);
    static void handleKthSmallestElement(ScapeGoatTree<ElemenType> &A, ScapeGoatTree<ElemenType> &B);
    static void handleSucessor(ScapeGoatTree<ElemenType> &A, ScapeGoatTree<ElemenType> &B);

public:
    static void TreeUI();

};


#endif //TREE_ITREE_HPP
```

7.6 CPP/Node.hpp File Reference

Classes

- class Node< T >

7.7 Node.hpp

Go to the documentation of this file.

```
//
// Created by DELL on 10/25/2025.
//

#ifndef SCAPEGOATTREE_NODE_HPP
#define SCAPEGOATTREE_NODE_HPP
template<typename T>

class Node {
public:
    T value{};      // stored value
    Node* left{};    // left child pointer
    Node* right{};   // right child pointer
    Node* parent{};  // parent pointer
    unsigned int size=1;       // subtree size


    explicit Node(const T& v, Node* parentPtr = nullptr)
        : value(v), parent(parentPtr){}
    template<typename>
    friend class ScapeGoatTree;
};


#endif //SCAPEGOATTREE_NODE_HPP
```

7.8 CPP/queue.hpp File Reference

#include ”queue.tpp”

Classes

- class QNode< T >
- class Queue< T >

7.9 queue.hpp

Go to the documentation of this file.

```
//
// Created by DELL on 24/12/2025.
//

#ifndef TREE_QUEUE_HPP
#define TREE_QUEUE_HPP

template<typename T>
class QNode {
    QNode* next{};
    T value{};
    explicit QNode(T value): value(value){}
    template<typename>
    friend class Queue;
};
template<typename T>
class Queue {
    QNode<T>* head{};
    QNode<T>* tail{};
    int nNodes{};
public:
    ~Queue() {
        while (!isEmpty()) pop();
    }

    void push(T value);

    void pop();

    [[nodiscard]] bool isEmpty() const;

    T front();

    [[nodiscard]] int size() const;

};

#include "queue.tpp"

#endif //TREE_QUEUE_HPP
```

7.10 CPP/RunTUI.cpp File Reference

#include ”iTree.hpp”

Functions

- int main ()

7.10.1 Function Documentation

7.10.1.1 main()

int main ()

7.11 CPP/ScapeGoatTree.hpp File Reference

#include <string>
#include "Node.hpp"
#include <cmath>
#include "vector.hpp"
#include "stack.hpp"
#include "scapegoatTree.tpp"

Classes

- struct Command< T >
- class ScapeGoatTree< T >
- class ScapeGoatTree< T >::iterator

Enumerations

- enum class OpType { Insert , Delete , BatchStart , BatchEnd }

7.11.1 Enumeration Type Documentation

7.11.1.1 OpType

enum class OpType [strong]
ScapeGoat Tree Implementation
A self-balancing BST that maintains balance through periodic rebuilding.
Key Properties:

- -weight-balanced: No node's subtree is heavier than × parent's subtree
- = 2/3 for this implementation
- Height bound: h log . (n) where n = number of nodes

Time Complexity:

- Insert: O(log n) amortized, O(n) worst case during rebuild
- Delete: O(log n) amortized, O(n) worst case during rebuild
- Search: O(log n) worst case (tree stays balanced)

Space Complexity: O(n) for tree + O(n) temporary array during rebuild Represents the type of operation performed on the tree for undo/redo purposes.

Enumerator

Insert	
Delete	
BatchStart	
BatchEnd	

7.12 ScapeGoatTree.hpp

Go to the documentation of this file.

```
//
// Created by DELL on 10/25/2025.
//
#ifndef SCAPEGOATTREE_SCAPEGOATTREE_HPP
#define SCAPEGOATTREE_SCAPEGOATTREE_HPP

```

```cpp
#include <string>
#include "Node.hpp"
#include <cmath>
#include "vector.hpp"
#include "stack.hpp"
enum class OpType {
    Insert,     // Insertion of a single value
    Delete,     // Deletion of a single value
    BatchStart, // Marker for the beginning of a batch operation
    BatchEnd    // Marker for the end of a batch operation
};

template<typename T>
struct Command {
    OpType type; // Type of the operation
    T value;     // Value associated with the operation
};

template<typename T>
class ScapeGoatTree {

    using TreeNode = Node<T>;
    TreeNode* root{};
    int nNodes{};
    int rebuildCount = 0;
    Stack<Command<T>> undoStack;
    Stack<Command<T>> redoStack;
    bool isUndoing = false;
    int max_nodes = 0;
    // iterator class
    class iterator {
        TreeNode* curr;  // stores current node

    public:
        // constructor
        iterator(TreeNode* node) : curr(node) {}

        // dereference
        T& operator*() { return curr->value; }

        // pre-increment
        iterator& operator++() {
            curr = getSuccessor(curr);
            return *this;
        }

        // post-increment
        iterator operator++(int) {
            iterator temp = *this;
            ++(*this);
            return temp;
        }

        // comparison
        bool operator!=(const iterator& other) const { return curr != other.curr; }
    };

    static int findH(const TreeNode* node);
    static  unsigned int countN(const TreeNode* node);
    static TreeNode* findTraitor(TreeNode* node);
    TreeNode* rebuildTree(int start,int end,TreeNode* parent_node,T* array);
    void inorderTraversal(const TreeNode*node, int &i,T*& array) const;
    static void postorderTraversal(const TreeNode* node);
    void preorderTraversal(const TreeNode* node);
    void displayPreOrder(const TreeNode* node, std::ostream& os);
    void displayInOrder(const TreeNode* node, std::ostream& os);
    void displayPostOrder(const TreeNode* node, std::ostream& os);
    [[nodiscard]] int getThreshold() const {return static_cast<int>(log(nNodes) / log(1.5));}
    void DeletionRebuild();
    bool areTreesEqual(const TreeNode* n1, const TreeNode* n2) const;
    void restructure_subtree(TreeNode *newNode);
    T sumHelper(TreeNode* node,T min,T max);
    void rangeHelper(TreeNode* node,T min,T max,Vector<T>& range);
    T kthSmallestHelper(TreeNode *node, int k) const;
  static TreeNode* getSuccessor(TreeNode* node);




public:

    ScapeGoatTree();

    void insert(T value);

    void insertBatch( const Vector<T> &values);

    bool deleteValue(T value);

    void deleteBatch(const Vector<T> &values);

    [[nodiscard]] bool search(const T & key) const;

    void clear();
    void undo();
    void redo();
    T sumInRange(T min, T max);
    T getMin();
    T getMax();
    Vector<T> valuesInRange(T min,T max);
    T getSuccessor(T value) const;
    T kthSmallest(int k) const;


    [[nodiscard]] std::string isBalanced() const;


    const TreeNode* getRoot();
    iterator begin();

    static iterator end();

    ScapeGoatTree(const ScapeGoatTree &Otree);

    ScapeGoatTree(ScapeGoatTree&& other) noexcept;

    ~ScapeGoatTree();

    std::string displayPreOrder(); // for display

    std::string displayInOrder() ; // for display

    std::string displayPostOrder() ; // for display

    std::string displayLevels(); // for display

    bool operator[](T value) const;

    ScapeGoatTree operator+(const ScapeGoatTree &other) const;

    ScapeGoatTree& operator=(const ScapeGoatTree& other);

    ScapeGoatTree& operator=(ScapeGoatTree&& other) noexcept;

    ScapeGoatTree &operator=(int value);

    bool operator==(const ScapeGoatTree &tree) const;

    bool operator!=(const ScapeGoatTree &tree) const;

    bool operator!() const;

    void operator+(const T& value);

    bool operator-(const T& value);

    bool operator-=(const T& value);

    void operator+=(const T& value);



};
#include "scapegoatTree.tpp"

#endif //SCAPEGOATTREE_SCAPEGOATTREE_HPP
```

7.13 CPP/stack.hpp File Reference

#include <stdexcept>
#include "vector.hpp"

Classes

- class Stack< T >

7.14 stack.hpp

Go to the documentation of this file.

```
//
// Created by DELL on 03/01/2026.
//

#ifndef SCAPEGOATPROJECT_STACK_HPP
#define SCAPEGOATPROJECT_STACK_HPP
#include <stdexcept>
#include "vector.hpp"

template<typename T>
class Stack {
    Vector<T> data;
public:
    void push(const T& value) {
        data.push_back(value);
    }

    T pop() {
        if (data.size() == 0) {
            throw std::out_of_range("pop on empty Stack");
        }
        return data.pop_back();
    }
    T top() {
        return data.data[0];
    }

    [[nodiscard]] unsigned int size() const {
        return data.size();
    }

    [[nodiscard]] bool isEmpty() const {
        return data.size() == 0;
    }
};


#endif //SCAPEGOATPROJECT_STACK_HPP
```

7.15 CPP/tests.cpp File Reference

#include <iostream>
#include <cassert>
#include <vector>
#include <random>
#include <algorithm>
#include <set>
#include "ScapeGoatTree.hpp"

Typedefs

- typedef int Type

Functions

- template<typename T>
  bool containsAll (const ScapeGoatTree< T > &tree, const std::vector< T > &values)
- void testBasicInsertion ()
- void testDeletion ()
- void testRebuilding ()
- void testOperators ()
- void testBatchOperations ()
- void testCopyAndMove ()
- void testUandR ()
- void testNewMethods ()
- void stressTest ()

- void testOrderedInsertion ()
- void testIterator ()
- int main ()

7.15.1 Typedef Documentation

7.15.1.1 Type

typedef int Type

7.15.2 Function Documentation

7.15.2.1 containsAll()

template<typename T>
bool containsAll (
    const ScapeGoatTree< T > & tree,
    const std::vector< T > & values)

7.15.2.2 main()

int main ()

7.15.2.3 stressTest()

void stressTest ()

7.15.2.4 testBasicInsertion()

void testBasicInsertion ()

7.15.2.5 testBatchOperations()

void testBatchOperations ()

7.15.2.6 testCopyAndMove()

void testCopyAndMove ()

7.15.2.7 testDeletion()

void testDeletion ()

7.15.2.8 testIterator()

void testIterator ()

7.15.2.9 testNewMethods()

void testNewMethods ()

7.15.2.10 testOperators()

void testOperators ()

7.15.2.11 testOrderedInsertion()

void testOrderedInsertion ()

7.15.2.12 testRebuilding()

void testRebuilding ()

7.15.2.13 testUandR()

void testUandR ()

7.16 CPP/TreeDriver.cpp File Reference

#include "imgui/imgui.h"
#include "imgui/imgui_impl_win32.h"
#include "imgui/imgui_impl_dx11.h"
#include <d3d11.h>
#include <windows.h>
#include "iTree.hpp"

Functions

- bool CreateDeviceD3D (HWND hWnd)
- void CleanupDeviceD3D ()
- void CreateRenderTarget ()
- void CleanupRenderTarget ()
- LRESULT WINAPI WndProc (HWND hWnd, UINT msg, WPARAM wParam, LPARAM lParam)
- int main (int, char **)
- IMGUI_IMPL_API LRESULT ImGui_ImplWin32_WndProcHandler (HWND hWnd, UINT msg, WPARAM wParam, LPARAM lParam)

Variables

- static ID3D11Device * g_pd3dDevice = nullptr
- static ID3D11DeviceContext * g_pd3dDeviceContext = nullptr
- static IDXGISwapChain * g_pSwapChain = nullptr
- static ID3D11RenderTargetView * g_mainRenderTargetView = nullptr
- static UINT g_ResizeWidth = 0
- static UINT g_ResizeHeight = 0
- static bool g_SwapChainOccluded = false
- static ImVec2 g_WindowPos = ImVec2(200, 200)
- static ImVec2 g_WindowVel = ImVec2(4.0f, 3.5f)
- static ImVec2 g_WindowSize = ImVec2(300, 120)

7.16.1 Function Documentation

7.16.1.1 CleanupDeviceD3D()

void CleanupDeviceD3D ()

7.16.1.2 CleanupRenderTarget()

void CleanupRenderTarget ()

7.16.1.3 CreateDeviceD3D()

bool CreateDeviceD3D (
    HWND hWnd)

7.16.1.4 CreateRenderTarget()

void CreateRenderTarget ()

7.16.1.5 ImGui_ImplWin32_WndProcHandler()

```
IMGUI_IMPL_API LRESULT ImGui_ImplWin32_WndProcHandler (
        HWND hWnd,
        UINT msg,
        WPARAM wParam,
        LPARAM lParam) [extern]
```

7.16.1.6 main()

```
int main (
        int ,
        char ** )
```

Entry point for the ImGui-based visualizer application.

7.16.1.7 WndProc()

```
LRESULT WINAPI WndProc (
        HWND hWnd,
        UINT msg,
        WPARAM wParam,
        LPARAM lParam)
```

7.16.2 Variable Documentation

7.16.2.1 g_mainRenderTargetView

ID3D11RenderTargetView* g_mainRenderTargetView = nullptr [static]

7.16.2.2 g_pd3dDevice

ID3D11Device* g_pd3dDevice = nullptr [static]

7.16.2.3 g_pd3dDeviceContext

ID3D11DeviceContext* g_pd3dDeviceContext = nullptr [static]

7.16.2.4 g_pSwapChain

IDXGISwapChain* g_pSwapChain = nullptr [static]

7.16.2.5 g_ResizeHeight

UINT g_ResizeHeight = 0 [static]

7.16.2.6 g_ResizeWidth

UINT g_ResizeWidth = 0 [static]

7.16.2.7 g_SwapChainOccluded

bool g_SwapChainOccluded = false [static]

7.16.2.8 g_WindowPos

ImVec2 g_WindowPos = ImVec2(200, 200) [static]

7.16.2.9 g_WindowSize

ImVec2 g_WindowSize = ImVec2(300, 120) [static]

7.16.2.10 g_WindowVel

ImVec2 g_WindowVel = ImVec2(4.0f, 3.5f) [static]

7.17 CPP/vector.hpp File Reference

#include <stdexcept>
#include "vector"

Classes

- class Vector< T >

7.18 vector.hpp

Go to the documentation of this file.

```
#ifndef VECTOR_HPP
#define VECTOR_HPP
#include <stdexcept>
#include "vector"
template <typename T>
class Vector {

    unsigned int _size = 50;
    int nElements = 0;
    T* data = new T[_size]{};


public:
    template<typename>
    friend class Stack;
    ~Vector() { delete[] data; }

    Vector()=default;

    [[nodiscard]] unsigned int size() const { return nElements; }

    void push_back(const T& value) {
       if (nElements >= _size) {
           _size *= 2;
           T* newData = new T[_size]{};
           for (unsigned int i = 0; i < nElements; ++i)
               newData[i] = data[i];
           delete[] data;
           data = newData;
       }
       data[nElements++] = value;
    }
    T pop_back() {
        if (nElements == 0) {
            throw std::out_of_range("pop_back on empty Vector");
        }
        --nElements;
        T value = data[nElements];

        // Shrink when underutilized, keep minimum capacity of 50
        if (nElements > 0 && static_cast<unsigned int>(nElements) <= _size / 4 && _size > 50) {
            _size = 50u > _size ? 50u : _size;
            T* newData = new T[_size]{};
            for (unsigned int i = 0; i < nElements; ++i)
                newData[i] = data[i];
            delete[] data;
            data = newData;
        }
        return value;
    }
    T* begin() { return data; }
    T* end()   { return data + _size; }

    T& operator[](unsigned int index) { return data[index]; }

    const T& operator[](unsigned int index) const { return data[index]; }
    template <typename>
    friend class ScapeGoatTree;

    Vector(const Vector& other) {
      for (int i =0; i<other.nElements;i++) {
            push_back(other.data[i]);
        }
    }
    Vector& operator=(const Vector&) = delete;
    Vector(Vector&&) = delete;
    Vector& operator=(Vector&&) = delete;
};
#endif
```

7.19 README.md File Reference