Hacettepe University - Computer Engineering
BBM203 Software Practicum I - Fall 2023

RESIT ASSIGNMENT

SAVING DR. ELARA: MISSION RESCUE

Submission Deadline: 04/02/2024 at 23:59:59
Over submit.cs.hacettepe.edu.tr

Topics: Red-Black Trees, Recursion, Dynamic Memory Allocation, File I/O

Course Instructors: Assoc. Prof. Dr. Adnan ÖZSOY, Asst. Prof. Dr. Engin DEMİR, Assoc. Prof. Dr. Hacer YALIM KELEŞ
TAs: Alperen ÇAKIN, Ardan YILMAZ, Dr. Selma DİLEK
Programming Language: C++11 - **You MUST USE this starter code**
Due Date: **Sunday, 04/02/2024 (23:59:59)**

Saving Dr. Elara: Mission Rescue

Following the groundbreaking decryption of the Celestial Tapestry in *Programming Assignment 1*, a glimmer of hope emerged in the search for the enigmatic Dr. Elara. The tapestry, once thought to be a mere cosmic anomaly, revealed itself as a beacon, guiding us to the whereabouts of Dr. Elara, who vanished while exploring the uncharted depths of space.

The tapestry's hidden messages, a cryptic blend of celestial coordinates and obscure references to distant planets, have led to the formulation of a daring rescue operation: *Mission Rescue*. This high-stakes venture, spearheaded by the same brilliant team of HUBBM and HUAIN students who unraveled the tapestry's secrets, aims to traverse the perilous expanses of space to retrieve Dr. Elara from where she is believed to be stranded.

The path to Dr. Elara is fraught with challenges that demand not only courage but unparalleled scientific and computational expertise. To navigate these unknown realms, a critical task has been identified, forming the backbone of the mission's strategy: *space sector mapping*.

Mission Rescue is not just a simple retrieval mission; it is a journey into the unknown, a testament to human ingenuity and the relentless pursuit of knowledge. As the team embarks on this perilous voyage, they carry not only the hope of finding Dr. Elara but also the aspirations of unraveling the mysteries that lie beyond our known universe.

1 Task 1: Space Sector Mapping

The vastness of uncharted space requires a comprehensive mapping system. In this assignment, you are tasked with implementing the Red-Black Tree (RBT) data structure and algorithms to map space sectors along exploration routes. Your goal is to identify the path to the final destination sector, where Dr. Elara is believed to be stranded on one of the habitable planets. As the journey unfolds, new sectors may be discovered that need to be integrated into the space map. This dynamic process will enable the explorers to adjust the mission's trajectory in real time, ensuring a strategic and responsive approach to the ever-evolving landscape of space exploration.

1.1 Input File and Sector Map Initialization

Space sectors are represented by their position relative to the Earth's position on a 3D coordinates (X, Y, Z) plane. An input file containing the coordinates of the already discovered space **sectors**, structured as a `DAT` file, will be given as the *first command line argument*. Your task is to parse this file within your program to set up the initial `RBT` that will represent the sector map within the `SpaceSectorRBT` class. The member pointer variable `Sector *root` must be set to point to the root node of this tree. The content of a sample input file given as `sectors.dat` is illustrated on the right.

```
X,Y,Z
0,0,0
10,20,30
-5,-15,10
25,5,0
-10,-20,-30
5,15,-10
-5,15,10
-25,-5,0
30,40,50
15,-25,35
0,30,-40
-10,-20,30
15,-20,35
```

Sector nodes should be inserted into the RBT based on their (X, Y, Z) coordinates as follows:

- **Primary, Secondary, and Tertiary Sorting Criteria**: The primary sorting criterion is the X-coordinate. If two sectors have the same X-coordinate, the Y-coordinate becomes the secondary sorting criterion. If both X and Y coordinates are the same, the Z-coordinate is used as the tertiary sorting criterion.
- **Comparison during the insertion Process**: Start by comparing the X-coordinates of sectors. For sectors with the same X-coordinate, compare their Y-coordinates. For sectors with identical X and Y coordinates, compare their Z-coordinates.
- **Tree Structure**: A node's left child has a smaller X-coordinate or, in case of a tie, a smaller Y-coordinate, or, if both are tied, a smaller Z-coordinate. A node's right child has a greater or equal X-coordinate (and then Y, Z are considered similarly in case of ties).

Sector Node Insertion: You are expected to implement the node insertion into the Red-Black Tree recursively. The recursive approach traverses the tree from the root to the appropriate position where the new node should be inserted based on the sorting criteria described above (insertion based on the sector coordinates), while ensuring the tree maintains the RBT properties. The recursive function for inserting a new sector node will follow these steps:

- *Parameter Handling:* Take the current node and its coordinates as parameters.

- *Initiation:* Begin the insertion process with the root node of the RBT. If the RBT is empty (root is `nullptr`), the new node becomes the root and is colored `BLACK`. Otherwise, compare the coordinates as explained above. If the comparison criteria require inserting the sector to the left, proceed to the left child; otherwise, proceed to the right child.
- *Recursive Insertion:* Make a recursive call to the insert function with the left or right child based on the comparison criteria. If the child is `nullptr`, create a new red node with the given coordinates and attach it at this position.
- *Fixing Violations:* After inserting the node, fix any RBT property violations in the tree. This process must adjust the tree to ensure it maintains the Red-Black Tree properties, including the necessary rotations and color changes.
- *Structure Maintenance:* Return the current node after each recursive call to maintain the tree structure. This is crucial as the root of the subtree (or the overall tree) may change due to rotations during the fixing process.
- *Root Assignment:* The node returned from the initial call of the function (starting at the root) is assigned as the new root of the RBT.

Please note that the insertion process in a Red-Black Tree is different from a standard Binary Search Tree due to the additional steps required to maintain the tree's color properties and balance. This includes ensuring that every red node has black children (no two consecutive red nodes are allowed) and that every path from a node to its descendant leaves has the same number of black nodes.

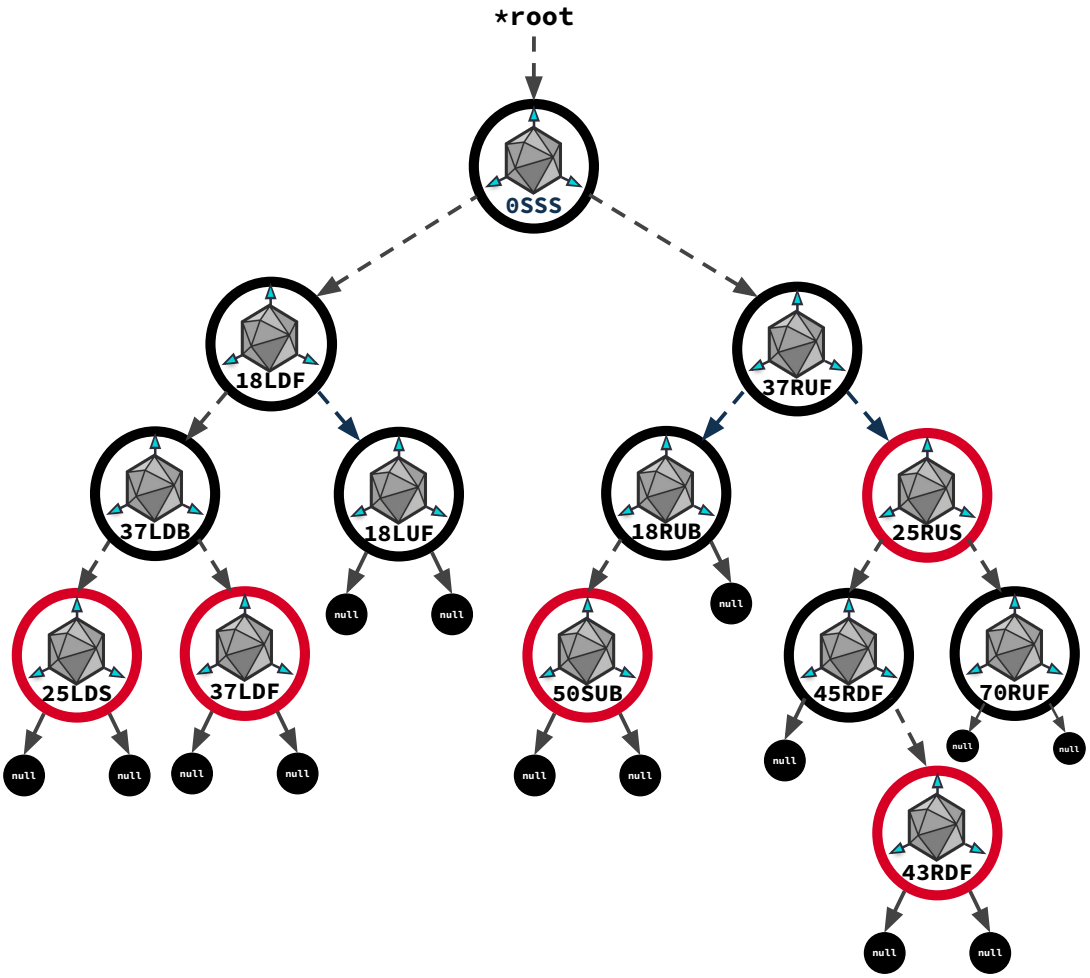
Each sector can be uniquely identified within the RBT map by its **sector code** that is calculated based on its (X, Y, Z) coordinates and its distance from the Earth. To calculate the sector distance in Astro Units (AUs) from the Earth based on their coordinates, you should use the Euclidean distance formula in three dimensions: $d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2 + (z_2 - z_1)^2}$. The Earth is considered to have the origin point of $(0, 0, 0)$.

Sector Code Generation Logic:

- **Distance Component**: The first part of the sector code is derived from the sector's distance from Earth, **truncated to integer (you can think of it as the floor operation)**.
- **Coordinate Components**: The next parts of the code are determined by the sector's X, Y, and Z coordinates.
  - For each coordinate, if the coordinate value is 0 (same as the Earth's), append the letter '**S**' to the code, signifying 'Same' in that dimension.
  - If the coordinate value is positive, append the respective directional letter: '**R**' for Right (X), '**U**' for Up (Y), and '**F**' for Forward (Z).
  - If the coordinate value is negative, append the respective directional letter: '**L**' for Left (X), '**D**' for Down (Y), and '**B**' for Backward (Z).

An example: For a sector located at coordinates $(10, -5, 20)$ and 22.91 AUs away from Earth, the distance component is **22**. The coordinate components are `R` for X (positive), `D` for Y (negative), and `F` for Z (positive). So, the resulting sector code is **22RDF**.

After reading the sample input provided in the given `sectors.dat` file, and inserting the sector nodes into the RBT in order they appear in the input file, while adhering to the sorting criteria and calculating the sector codes correctly, you should obtain the sector map shown on the right.



1.2 Finding a Path to Dr. Elara

In order to be able to search through the sector tree map, you need to implement three tree traversal algorithms: **inorder**, **preorder**, and **postorder**. In an RBT, inorder traversal retrieves elements in sorted order, preorder traversal is useful for creating prefix expressions and operations involving processing the root before the children, while postorder traversal is commonly employed for deleting nodes and processing children before the root. Below, the expected traversal outputs are given for the initial sample tree:

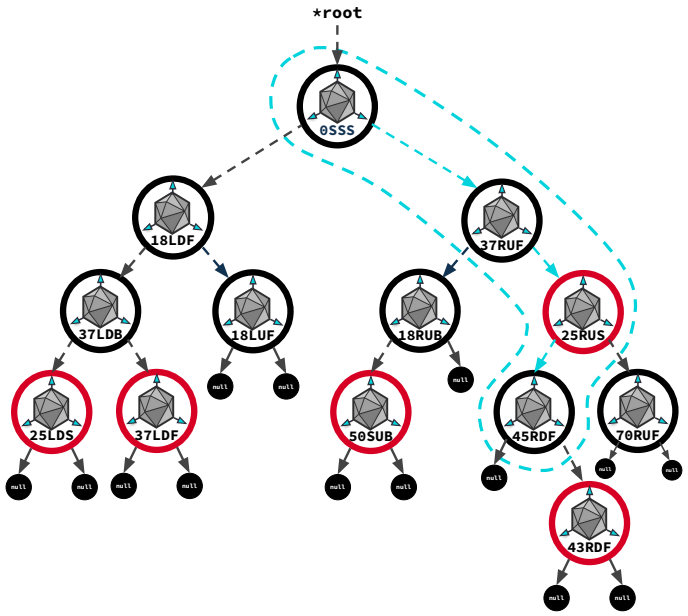
```
Space sectors inorder traversal:
RED sector: 25LDS
BLACK sector: 37LDB
RED sector: 37LDF
BLACK sector: 18LDF
BLACK sector: 18LUF
BLACK sector: 0SSS
RED sector: 50SUB
BLACK sector: 18RUB
BLACK sector: 37RUF
BLACK sector: 45RDF
RED sector: 43RDF
RED sector: 25RUS
BLACK sector: 70RUF
```

```
Space sectors preorder traversal:
BLACK sector: 0SSS
BLACK sector: 18LDF
BLACK sector: 37LDB
RED sector: 25LDS
RED sector: 37LDF
BLACK sector: 18LUF
BLACK sector: 37RUF
BLACK sector: 18RUB
RED sector: 50SUB
RED sector: 25RUS
BLACK sector: 45RDF
RED sector: 43RDF
BLACK sector: 70RUF
```

```
Space sectors postorder traversal:
RED sector: 25LDS
RED sector: 37LDF
BLACK sector: 37LDB
BLACK sector: 18LUF
BLACK sector: 18LDF
RED sector: 50SUB
BLACK sector: 18RUB
RED sector: 43RDF
BLACK sector: 45RDF
BLACK sector: 70RUF
RED sector: 25RUS
BLACK sector: 37RUF
BLACK sector: 0SSS
```

In Programming Assignment 1, you successfully decoded a hidden message within the Celestial Tapestry. This message revealed a sector code, believed to be a secret communication from Dr. Elara, notifying us about her location in the cosmos. It's now imperative that we organize a rescue mission to ensure Dr. Elara's safe return to Earth.

Your mission in this assignment is to navigate the space sector tree map to determine the exact path needed to reach the sector where Dr. Elara is stranded. Envision space as interconnected by wormholes, linking sectors directly in a manner reflected by the child links in our sector tree map. Starting from Earth's sector, the rescue team will use these inter-sector connections to travel, or perform 'space jumps', from one sector to another, ultimately reaching Dr. Elara's location. You are tasked with implementing a function that returns a vector of pointers to Sector nodes, such that, following these pointers in sequence will guide the rescue team from Earth to Dr. Elara's sector. For instance, if based on the decoded message, we were informed that Dr. Elara is in sector **45RDF**. Using the initial sample tree structure, the expected path (illustrated on the right) should be printed like this:



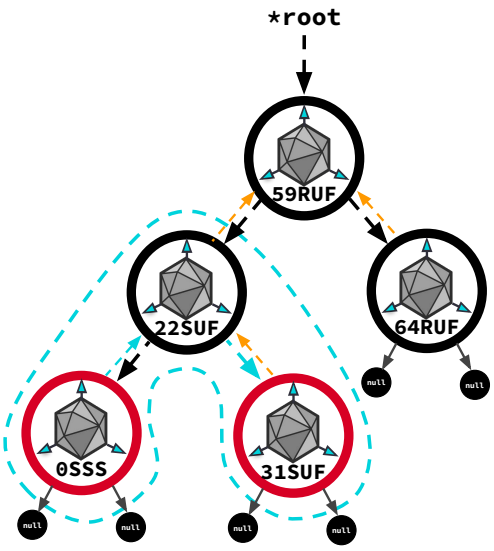

```
The stellar path to Dr. Elara: 0SSS->37RUF->25RUS->45RDF
```

In this example, the expected path to the intended sector necessitates three space jumps to reach the destination from the Earth.

The second sample input file `sectors_sorted.dat` (shown on the right), illustrates an example case in which the sector where the Earth is located may not be the root sector.

```
X,Y,Z
10,40,50
10,30,50
0,10,30
0,10,20
0,0,0
```

In this second example illustrated on the right, we assume that that Dr. Elara is in sector **31SUF**, based on the decoded message. Using the obtained sample tree structure, the expected path should be printed like this:




```
The stellar path to Dr. Elara: 0SSS->22SUF->31SUF
```

Via this example, we observe that the path may or may not visit the root of our tree sector map. Another important note: **make sure that there are no duplicate Sector nodes in the path!**

If a path cannot be found, the expected output is:

```
A path to Dr. Elara could not be found.
```

2 Assignment Implementation Tasks and Requirements

In this section, we outline the classes and functions you are required to implement. As aspiring engineers, it's crucial to base your code on the provided starter code that offers a predetermined class structure. This ensures code clarity, proper encapsulation, and facilitates unit testing. It's imperative that you do not alter the names or signatures of functions provided in the template files. Likewise, refrain from renaming or changing the types of the specified member variables. Other than that, you're free to introduce any additional functions or variables as needed.

`Sector` Class

- **Constructor**:
  `Sector(int x, int y, int z)`

  - Initializes a sector based on the given coordinates. Calculates the sector's distance from the Earth and generates its unique sector code.
- **Operators** $=$, $==$, $!=$: Overloading the assignment and comparison operators.
- **Destructor**:
  `~Sector()`

  - Frees any dynamically allocated memory if necessary.

`SpaceSectorRBT` Class

- **Constructor**:
  `SpaceSectorRBT()`

  - Initializes a space sector RBT instance with a `nullptr root` (already given).
- **Functions**:
  `void readSectorsFromFile(const std::string& filename)`

  - Reads sectors from the given input file and inserts them into the space sector RBT map according to the coordinates-based comparison criteria.

  `void insertSectorByCoordinates(int x, int y, int z)`

  - Instantiates and inserts a new sector into the space sector RBT map according to the coordinates-based comparison criteria.

  `void displaySectorsInOrder()`

  - Traverses the space sector RBT map in-order and prints the sectors to `STDOUT` in the given format.

  `void displaySectorsPreOrder()`

  - Traverses the space sector RBT map in pre-order traversal and prints the sectors to `STDOUT` in the given format.

  `void displaySectorsPostOrder()`

  - Traverses the space sector RBT map post-order traversal and prints the sectors to `STDOUT` in the given format.

`std::vector<Sector*> getStellarPath(const std::string& sector_code)`

- Finds the path from the Earth to the destination sector given by the `sector_code` and returns this path as a vector or `Sector` pointers in the path.

`void printStellarPath(const std::vector<Sector*>& path)`

- Prints the stellar path obtained from the `getStellarPath()` function to `STDOUT` in the given format.

- **Destructor**:
  `~SpaceSectorRBT()`

  - Frees any dynamically allocated memory if necessary.

Must-Use Starter Codes

You MUST use **this starter (template) code**. All headers and classes should be placed directly inside your **zip** archive.

Grading Policy

- No memory leaks and errors: 10%
  - No memory leaks: 5%
  - No memory errors: 5%
- Implementation of the Space Sector Mapping tasks: 80%
  - Reading input and properly implementing the corresponding RBT structure with node insertions: 35%
  - Proper implementation of RBT traversals: 15%
  - Proper implementation of pathfinding: 30%
- Output tests: 10%

Important Notes

- Do not miss the deadline: **Sunday, 04.02.2024 (23:59:59)**.
- Save all your work until the assignment is graded.
- The assignment solution you submit must be your original, individual work. Duplicate or similar assignments are both going to be considered as cheating.
- You can ask your questions via Piazza (https://piazza.com/hacettepe.edu.tr/fall2023/bbm203), and you are supposed to be aware of everything discussed on Piazza.

- You must test your code via **Tur^{3}Bo Grader** https://test-grader.cs.hacettepe.edu.tr/ (**does not count as submission!**).
- You must submit your work via https://submit.cs.hacettepe.edu.tr/ with the file hierarchy given below:
  - **b<studentID>.zip**
    - Sector.cpp <FILE>
    - Sector.h <FILE>
    - SpaceSectorRBT.cpp <FILE>
    - SpaceSectorRBT.h <FILE>
- **You MUST use this starter code**. All classes should be placed directly in your **zip** archive.
- This file hierarchy must be zipped before submitted (not .rar, only .zip files are supported).

Build and Run Configuration

Here is an example of how your code will be compiled (note that instead of `main.cpp` we will use our test files):

```
$ g++ -std=c++11 -g Sector.cpp SpaceSectorRBT.cpp main.cpp -o mission_rescue
```

Or, you can use the provided `Makefile` within the sample input to compile your code:

```
$ make
```

After compilation, you can run the program as follows:

```
$ make
$ ./mission_rescue sectors.dat
```

Testing and Debugging

You **must** use the C++-11 standard in a Linux environment (e.g., Ubuntu under WSL or dev server). Ensure your code is free of memory errors using Valgrind and debug exceptions with gdb before submitting or posting on Piazza.

Academic Integrity Policy

All work on assignments **must be done individually**. You are encouraged to discuss the given assignments with your classmates, but these discussions should be carried out in an abstract way. That is, discussions related to a particular solution to a specific problem (either in actual code or in pseudocode) **will not be tolerated**. In short, turning in someone else's work (including work available on the internet), in whole or in part, as your own will be considered as **a violation of academic integrity**. Please note that the former condition also holds for the material found on the web as everything on the web has been written by someone else.

The submissions will be subjected to a similarity check. Any submissions that fail the similarity check will not be graded and will be reported to the ethics committee as a case of academic integrity violation, which may result in the suspension of the involved students.