Analysis of Algorithms

BLG 335E

Project 3

Res. Assist. Aycan Sahin

sahinay21@itu.edu.tr

Res. Assist. Büşra Çalışkan

caliskanb21@itu.edu.tr

Instructors:
Assoc. Prof. Dr. İlkay Öksüz
Asst. Prof. Dr. Mehmet Baysan
Asst. Prof. Dr. Tuğçe Bilen

Faculty of Computer and Informatics Engineering
Department of Computer Engineering
Date of submission: 20.12.2024

1.Project Brief

Imagine you are part of the Data Science Department at a leading video game publishing company, GameSoft Inc., which has been tracking the sales of its video games across different regions. GameSoft has been growing rapidly and needs a robust system for managing its sales data, which spans from 1980 to 2020. The company operates in three key marketplaces: North America (NA), Europe (EU), and Other Regions, with numerous publishers releasing games each year on various platforms such as PlayStation, Xbox, and PC.

To streamline sales tracking, GameSoft has tasked you with building a data structure capable of managing and querying the sales data of various video game publishers efficiently. You are given a dataset that includes sales numbers for video games released between 1980 and 2020. Your job is to compare two different data structures—Binary Search Tree (BST) and Red-Black Tree (RBT)—and determine which one is better suited for handling the company's sales data management needs.

You have been provided with a dataset containing the following features for each video game released by a publisher:

- **Name**: Name of the video game.
- **Platform**: The platform on which the game was released (e.g., *PlayStation*, *Xbox*, *PC*).
- **Year of Release**: The year the game was released (1980-2020).
- **Publisher**: The name of the publisher.
- **NA Sales**: Sales in North America (in million units).
- **EU Sales**: Sales in Europe (in million units).
- **Other Sales**: Sales in the Rest of the World (in million units).

As a part of the data analysis, you are tasked with building two trees using the **publisher's name** as the key. The tree should store the **cumulative sales** data for each publisher across the three regions. If a publisher appears multiple times (for different games, years), the sales numbers must be updated by adding the new data to the existing cumulative sales.

2.Implementation Details

A.Data Insertion

Construct a Binary Search Tree and a Red-Black Tree to store the dataset. As you insert data, update the cumulative sales for each publisher. Ensure that when a publisher already exists in the tree, its sales data is updated appropriately. (You should set the color attribute of the struct as 0 for “black” nodes and 1 for “red” nodes.)

Record the time taken to insert all the data for each tree. (µs : microseconds)

B.Search Efficiency

Perform 50 random searches (e.g., for specific publishers) in both trees.

Measure and compare the average search time. (ns : microseconds)

C.Best-Selling Publishers at the End of Each Decade:

At the end of each decade (1990, 2000, 2010, 2020), generate a list of the best-selling publishers in each marketplace for both trees. Use pre-order traversal to print the tree’s nodes in a structured format, displaying the depth and color (red or black) of each node for RBT.

D.Final Tree Structure:

How balanced is the Binary Search Tree compared to the Red-Black Tree?

E. Write Your Recommendation

Based on your analysis, recommend which tree structure is better suited for managing the Dataset Valley and justify your answer.

F. Ordered Input Comparison

Order the data by NA_Sales, then EU_Sales, and finally Other_Sales in case of ties. After sorting, construct both a Binary Search Tree (BST) and a Red-Black Tree (RBT) based on the ordered data. Then, perform 50 random search operations on the constructed trees. Measure the time taken for each search operation and calculate the average search time for both trees. Record the time in nanoseconds and compare the results to evaluate the search performance of the two tree structures with ordered data

3. Explanations of Algorithms

A. Classes:

- ✓ **Publisher Class**: This class represents a publisher, storing the publisher's name and sales data in three regions (North America, Europe, and others).
- ✓ **Node Class**: The Node class is used to represent a node.
- ✓ **RB_tree Class**: This is the core class for implementing the Red-Black Tree. It manages operations such as inserting nodes, fixing up the tree to maintain Red-Black properties, performing tree rotations, and searching for the best-selling publishers in different regions.
- ✓ **BST_tree Class**:The BST_tree class is the core class for implementing the Binary Search Tree (BST). It manages operations such as inserting nodes and searching for the best-selling publishers in different regions.

B. Red Black Tree Functions:

RB_tree() function implements the core functionality of the Red-Black Tree, including insertion, rotations, and maintaining the Red-Black properties.

B.1.Insertion:

- RB_insert(): This function inserts a new node into the tree, based on its publisher's cumulative sales.
- RB_insert_fixup(): After inserting a node, this function ensures that the Red-Black properties are maintained.
- **B.1.1.Rotations**:
  - RB_left_rotate() and RB_right_rotate(): These functions perform left and right rotations, which are used to maintain the balance of the tree during insertions and deletions.

B.2.Finding Best Sellers:

- find_best_seller(): This function traverses the tree and finds the best-selling publisher in each region (North America, Europe, and others) by comparing the sales of publishers.

B.3.Traversal:

- preorder(): This function performs a preorder traversal of the tree and print the publisher's name and color.

C. Binary Search Tree Functions:

BST_tree() function implements the core functionality of the Binary Search Tree (BST), which allows basic operations like insertion.

C.1. Insertion:

The BST_insert() function inserts a new node into the tree based on the publisher's cumulative sales.

4. Evaluation of Criteria

1) Red-Black Tree implementation (35 points)
2) Binary Search Tree implementation (15 points)
3) All tests providing correct results (25 points)
4) Report: Comparison of Red-Black Tree (RBT) and Binary Search Tree (BST) [Task : A, B, C, D, E, F] (25 points)

WARNINGS:

- Do not add any new libraries. Complete the functions within the two provided C++ files. But You may employ additional functions to be used within the existing functions.
- Do not alter the names of the two C++ files.
- Make sure to submit the task within the given time frame. Tasks submitted after the deadline will not be accepted.
- Please utilize Visual Studio Code with Docker to achieve the expected results.

Sample Input File

```
Name,Platform,Year_of_Release,Publisher,NA_Sales,EU_Sales,Other_Sales
Asteroids,2600,1980,Atari,4,0.26,0.05
Missile Command,2600,1980,Atari,2.56,0.17,0.03
Kaboom!,2600,1980,Activision,1.07,0.07,0.01
Defender,2600,1980,Atari,0.99,0.05,0.01
Boxing,2600,1980,Activision,0.72,0.04,0.01
Ice Hockey,2600,1980,Activision,0.46,0.03,0.01
Freeway,2600,1980,Activision,0.32,0.02,0
Bridge,2600,1980,Activision,0.25,0.02,0
Checkers,2600,1980,Atari,0.22,0.01,0
Pitfall!,2600,1981,Activision,4.21,0.24,0.05
Frogger,2600,1981,Parker Bros.,2.06,0.12,0.02
Demon Attack,2600,1981,Imagic,1.99,0.12,0.02
E.T.: The Extra Terrestrial,2600,1981,Atari,1.84,0.11,0.02
Ms. Pac-Man,2600,1981,Atari,1.54,0.1,0.02
River Raid,2600,1981,Activision,1.49,0.09,0.02
Donkey Kong,2600,1981,Coleco,1.36,0.08,0.02
Centipede,2600,1981,Atari,1.26,0.08,0.01
Atlantis,2600,1981,Imagic,1.18,0.08,0.01
Megamania,2600,1981,Activision,1.03,0.06,0.01
Cosmic Ark,2600,1981,Imagic,0.99,0.05,0.01
Donkey Kong Junior,2600,1981,Atari,0.9,0.05,0.01
Spider-Man,2600,1981,Parker Bros.,0.87,0.05,0.01
Custer's Revenge,2600,1981,Mystique,0.76,0.05,0.01
Alien,2600,1981,20th Century Fox Video Games,0.74,0.04,0.01
Air Raid,2600,1981,Men-A-Vision,0.72,0.04,0.01
Berzerk,2600,1981,Atari,0.68,0.04,0.01
King Kong,2600,1981,Tigervision,0.65,0.04,0.01
Adventures of Tron,2600,1981,Mattel Interactive,0.63,0.03,0.01
BurgerTime,2600,1981,Mattel Interactive,0.55,0.03,0.01
```

SAMPLE OUTPUTS

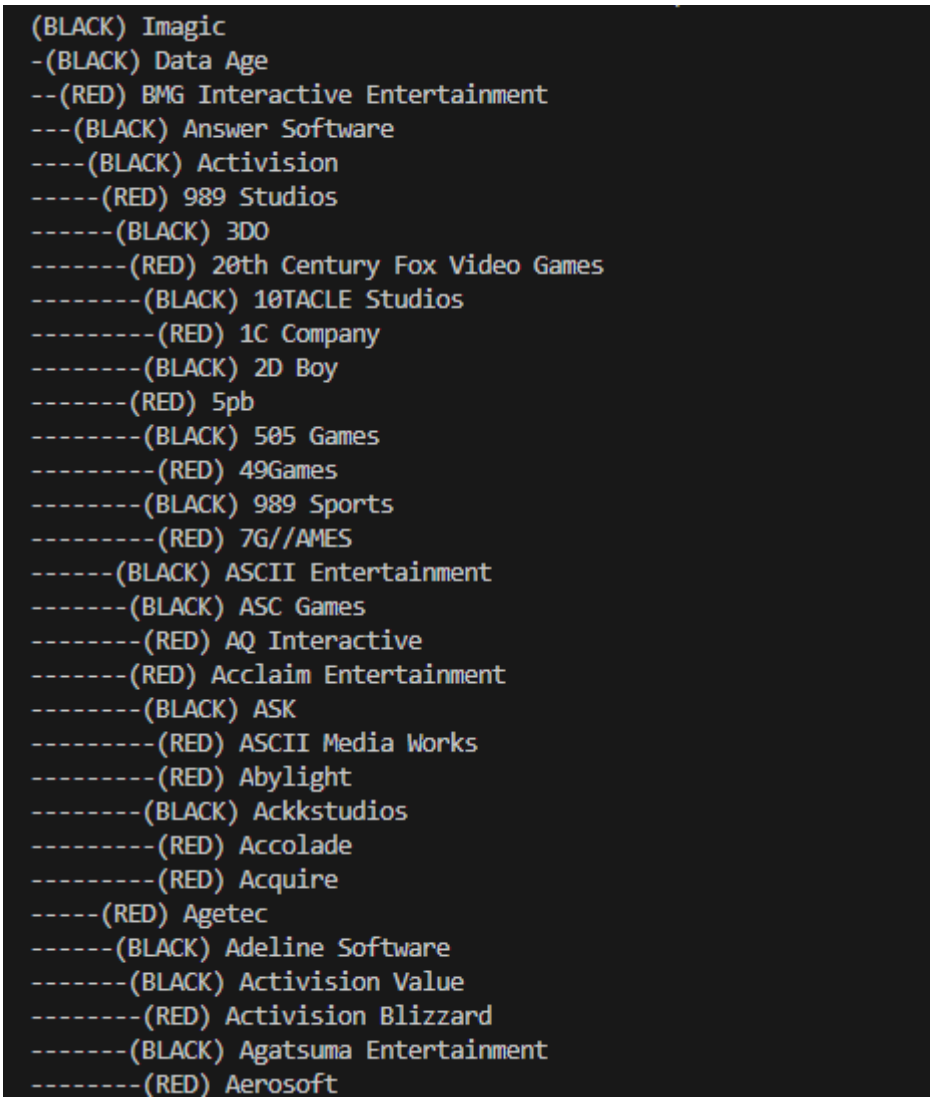
BST:

```
test@blg335e:~/hostvolume/blg335e_hw3$ g++ solution_BST_tree.cpp -o solution_BST
test@blg335e:~/hostvolume/blg335e_hw3$ ./solution_BST VideoGames.csv
End of the 1990 Year
Best seller in North America: Nintendo - 160.02 million
Best seller in Europe: Nintendo - 30.03 million
Best seller rest of the World: Nintendo - 5.65 million
End of the 2000 Year
Best seller in North America: Nintendo - 334.75 million
Best seller in Europe: Nintendo - 101.97 million
Best seller rest of the World: Nintendo - 15.76 million
End of the 2010 Year
Best seller in North America: Nintendo - 722.26 million
Best seller in Europe: Nintendo - 350.91 million
Best seller rest of the World: Electronic Arts - 89.2 million
End of the 2020 Year
Best seller in North America: Nintendo - 814.43 million
Best seller in Europe: Nintendo - 418.36 million
Best seller rest of the World: Electronic Arts - 126.82 million
Time taken to insert all data into BST: 25024 µs
Average time for 50 random searches : 1167.9 ns
```

RBT:

```
test@blg335e:~/hostvolume/blg335e_hw3$ g++ solution_RBT_tree.cpp -o solution_RBT
test@blg335e:~/hostvolume/blg335e_hw3$ ./solution_RBT VideoGames.csv
End of the 1990 Year
Best seller in North America: Nintendo - 160.02 million
Best seller in Europe: Nintendo - 30.03 million
Best seller rest of the World: Nintendo - 5.65 million
End of the 2000 Year
Best seller in North America: Nintendo - 334.75 million
Best seller in Europe: Nintendo - 101.97 million
Best seller rest of the World: Nintendo - 15.76 million
End of the 2010 Year
Best seller in North America: Nintendo - 722.26 million
Best seller in Europe: Nintendo - 350.91 million
Best seller rest of the World: Electronic Arts - 89.2 million
End of the 2020 Year
Best seller in North America: Nintendo - 814.43 million
Best seller in Europe: Nintendo - 418.36 million
Best seller rest of the World: Electronic Arts - 126.82 million
Time taken to insert all data into RBT: 26820 µs
Average time for 50 random searches : 649.76 ns
```

PREORDER PRINT OF RBT:



: