COSC2469 - Algorithms and Analysis

Group Project

Group G12
Nguyen Cong Thinh - s3926387
Nguyen Minh Nhat - s3924871
Nguyen Dang Ha - s3924594
Ha Thuy Chi - s3930417

Lecturer’s name: Hoang Tuan Anh
09/05/2024

1. Overview and High-Level Design
   a. Overview

Our group project is an application that is similar to Google Maps, which was developed using Java and without the use of any classes or interfaces from the Java Collection Framework or external libraries. The application has a map represented by the x and y axis, with a size of up to 10,000,000 x 10,000,000 and has 100,000,000 available places. A place is defined as a point in the system with unique x-y coordinates and some services. The application will support four major operations: Add, Edit, Remove, and Search. The "Add" feature allows users to add a new place with coordinates and services. Users can choose from 10 different types of services. The "Edit" feature allows users to edit the service array with any x and y values. The "Remove" feature removes a place. Finally, the "Search" feature is used to search for a list of places based on an input service and a bounding rectangle. This rectangle is defined by a center point and side lengths. The search will return the closest points with the matched service within this rectangle to the users.

   b. High-Level Design

In our application, all places are stored in an array, and in this array, there are many other arrays which are used to store the places with the same x coordinate. The small arrays are sorted ascendingly based on the x values, while in these arrays, the elements are sorted based on y values.

- Classes:
  - Service: This class will define the y coordinate and the services of each place.
  - X: This class will define the x coordinate with a list of y coordinates and services of those places sharing the same x coordinate.
  - Map2D: This class has some methods to process the required operations, such as sorting the array by quick sort, adding a new place, removing a place, editing a place, searching, etc. This is also used to execute those features.
- Methods:
  - Setter, Getter: Those methods are in class X and Service and are used to set and get the attributes of a place such as x-y coordinates and services.
  - Sorting methods: Using the quick sort algorithm, those methods are used to sort the big array by x coordinates and then sort the small arrays inside the big one based on the y coordinates for further processes.
  - Generating methods: Methods are used to randomly generate xy coordinates and the services and then remove the duplicated values to ensure uniqueness.
  - Operation methods: Methods perform adding, editing, removing and searching actions by taking the inputs from users to process their demands.
    - “Search”: There are some methods to process the logic behind it. It will use the x-y coordinates of the current point, width, and height of the rectangle and the type of service the user wants to search for to return 50 places closest to the current point.
    - “Add”: The program will ask for the xy coordinates and the services of the new place, then process to add it to the correct index of the big array.

        - “Edit”: The application will ask for the xy coordinates and use them to find the place and modify its services.
        - “Remove”: It will use the xy coordinates from the users to detect the place and then remove it from the big array.
    - Main method: It executes the methods to adapt the required features.
2. Data Structures and Algorithms
    a. Data Structures

    In the realm of data structuring, the array of X objects, coupled with arrays of services, presents a compact yet potent means of representing spatial information and services. Each X object encapsulates an integer value representing a coordinate along the x-axis, serving as an unique identifier. Paired with this identifier is an array of Service objects, each comprising an integer value for the y-coordinate and an array of strings of services associated with the place located on that x and y axis.

    This dynamic structure facilitates efficient access to spatial data and service information, enabling seamless addition, removal, and modification of objects and services. Applications span various domains, from urban planning to tourism, where precise spatial information and service availability are crucial.

    Challenges such as memory management and data consistency exist, requiring careful consideration. However, the array of X objects with associated services stands as a versatile tool, offering insights into geographical landscapes and enhancing decision-making processes.

    b. Algorithms

    In the Map2D class, various algorithms drive the functionality of operations such as sorting, adding, removing, and editing places on a 2D map. Let's delve into the analysis of these algorithms, examining their design, efficiency, and practical implications.

    Sorting Algorithms:
    - Quick Sort by X Coordinate: This algorithm is employed to sort the array of X objects based on their x-coordinate values. Quick sort is a comparison-based sorting algorithm known for its efficiency.
    - Quick Sort by Y Coordinate: Similar to sorting by X coordinate, this algorithm sorts Service objects based on their y-coordinate values.

    Searching Algorithms:
    - Binary Search: Utilized to efficiently locate the appropriate X object within a specified range based on its x-coordinate. It has a divide-and-conquer approach, making it efficient for sorted arrays.

    Manipulation Algorithms:
    - Adding a Place: The code implements methods to add new places to the 2D map, including handling scenarios where the X coordinate already exists and where it doesn't.
    - Removing a Place: Algorithms are implemented to remove places from the map, considering scenarios where only one place exists for an X coordinate or multiple places.

- Editing a Place: Enables modification of existing places on the map, involving locating the place based on its coordinates and updating its service information.

Miscellaneous Algorithms:

- Duplicate Removal: Methods are implemented to remove duplicate X objects and duplicate Service objects from the arrays, enhancing data integrity and efficiency. These operations involve iterating through the arrays and removing duplicates.

Distance Calculation:

- Euclidean Distance Calculation: The code includes a method to calculate the Euclidean distance between two coordinates which involves simple arithmetic calculations.

3. Complexity Analysis

In the Map2D class, a suite of algorithms orchestrates the manipulation and management of a 2D map structure. Each algorithm's efficiency is paramount for ensuring the smooth functioning of map-related operations. Here, we delve into a detailed complexity analysis of these algorithms, shedding light on their computational characteristics and performance implications.

Sorting Operations:

- Quick Sort by X Coordinate: The average-case time complexity of quick sort is O(n log n), where 'n' is the number of X objects in the array. This sorting operation efficiently arranges the X objects based on their x-coordinate values.
- Quick Sort by Y Coordinate: Similar to sorting by X coordinate, sorting by Y coordinate also has an average-case time complexity of O(n log n), where 'n' is the number of Service objects in the array.

Searching Operations:

- Binary Search: The binary search algorithm has a time complexity of O(log n), where 'n' is the number of X objects in the array. This algorithm efficiently locates the appropriate X object within a specified range based on its x-coordinate.

Manipulation Operations:

- Adding a Place: Adding a place involves various steps such as searching for the appropriate position and inserting the new place. In the worst-case scenario, if the array needs to be resized and elements need to be shifted, the time complexity could be O(n). If not, it involves finding the appropriate X object in the places array, determining the insertion index using binary search (searchHigher), and then inserting the new Service object at the appropriate position in the Service array of that X object. The time complexity of finding the X object is O(log n) due to binary search.
- Removing a Place: Removing a place typically involves locating the place to be removed and deleting it from the array. If the X object only contains one Service object, the removal operation also includes removing the entire X object. The time complexity of finding the X object is O(log n) due to binary search. The time complexity of finding the Service object within the X object is also O(log k), where k is the number of services in that X object. The removal operation within the Service array of the X object involves shifting elements, which takes linear time, but is typically small, O(k). Therefore, the overall time complexity of removePlace is O(log n) if removing only a Service object and O(n) if removing the entire X object.

- Editing a Place: This method involves finding the appropriate X object in the places array, finding the appropriate Service object in the Service array of that X object, and then modifying its services. Both finding the X object and finding the Service object involve binary search, with time complexities of O(log n) and O(log k), respectively. Therefore, the overall time complexity of editPlace is O(log n).

Miscellaneous Operations:

- Duplicate Removal: Removing duplicate X objects and Service objects from the arrays involves iterating through the arrays and removing duplicates. The time complexity for this operation is typically O(n).

Distance Calculation:

- Euclidean Distance Calculation: Calculating the Euclidean distance between two coordinates has a constant time complexity of O(1) as it involves simple arithmetic calculations.

4. Evaluation

The system we test on has InitialHeapSize is approximately 254 MB, and both MaxHeapSize and SoftMaxHeapSize are approximately 4GB. CPU: 3.30 GHz.

We have conducted test cases for all functions located in the "Test" folder. Based on these test cases, we can confirm that the correctness of our methods is 100%. We also tested the performance of our program by increasing the map size to 10,000,000, the searching bounding to 100,000, and the number of places up to 100,000,000. The system took around 40-60 seconds to complete the execution of the maximum map size (10,000,000) searching bounding (100,000), and 100,000,000 random generated places.

In Map2D.java file,
- binarySearch: Searches for a target value within a sorted array and returns its index.
- searchHigher: Finds the index of the next higher or same value compared to the target.
- searchLower: Finds the index of the next lower or same value compared to the target.

These methods perform binary search operations on an array of objects X. These methods are designed for use with the 'X' object in Map2D.java. The implementation of the logic and algorithms are the same with the 3 methods in the file X.java, where the design is for an array of object Service. Refer to XTest.java for further testing details on binarySearch, searchHigher and searchLower methods.

5. Conclusions

a. Advantages

After those above analysis, there are several points that can be concluded. According to our insight, applying would make our application most consistent. Thus, it is obvious that we had to implement some side functions that utilize our main functions and support our algorithm to run more efficiently. Overall, the greatest part in our application is the consistent runtime among the same amount of data inputs, without the exaggerated amount of space usage. In detail, for most of the requirements in this project, we used transfer and conquer techniques as our approach. As mentioned above, several side smaller functions were written to aid the performance of our primary operations. By doing that, most of our final operations have an average runtime complexity of O(nlogn), even for the worst-case run, the

runtime complexity is at most of O(n^2). Beside the complexity of runtime, space complexity for most functions is also under control, with the average among them is O(n).

b. Limitation

Despite our consideration of the performance and optimization of our product, limitations and drawbacks are just something that is inevitable. Back to our analysis on this project, from our point of view, transfer and conquer was still the most suitable approach for our project. Most of the main operations were more optimized and functionable with this approach. For example, by applying Quick Sort algorithm to develop a ‘sort’ function, functionality of some requirements such as searching, add, remove was optimized in terms of runtime, as iterating or accessing a sorted array is much better. However, drawbacks arise along this optimization. In general, more functions being developed means more arrays, more structures were initiated during the application runtime, which also means the memory used is multiplied by a constant. Despite our effort to minimize the application space usage, it is likely that our final product is only suitable for devices with a considerable amount of memory, such as computers and laptops, devices like phones or other portable ones might not be able to run our application due to lack of memory allocation. Furthermore, the data structure we used for this project was not the ideal one either. From the complexity analysis, even though no functions have complexity that is over-exaggerated, the performance was just fine, at the average rate. Besides, there are also functions that could be improved even more to achieve better performance.