****

**Project Report**

***Electricity Company Management system -ECMS-***

**Defining the Problem Treated**

The Electricity Company Management System (ECMS) is a focused project addressing the challenges faced by a national electricity company in managing power consumption, injection and billing. The objective is to develop an efficient system.

The ECMS tackles the complexities of managing electricity consumption, billing, and the integration of photovoltaic electricity within a vast customer network. Utilizing advanced data structures such as hash tables, binary search trees, AVL trees, and heaps, the system aims to streamline data access, optimize operations, and enhance the overall customer experience.

**Project Objectives**

1. Efficient Data Organization:

The ECMS employs hierarchical structures like hash tables and BSTs for quick and efficient data retrieval, crucial for managing extensive datasets.

2. Optimized Operations:

Leveraging AVL trees and heaps, the project focuses on optimizing billing, departmental performance analysis, and customer summary retrieval, ensuring system responsiveness.

3. Dynamic Budget Management:

The system incorporates dynamic budget adjustments based on departmental performance, showcasing adaptability to the company's financial dynamics.

1. User-Friendly Results Display:

The ECMS facilitates user-friendly functionalities, such as generating individual customer bills, summarizing bills for specific regions or cities, and displaying departmental rankings.

**Explanation of the Data chosen:**

In the context of our project, we generated datasets to simulate the electricity management system for the national network of a fictional Algerian electricity company. The datasets were created to represent the geographic structure of Algeria (regions, cities, districts), along with the details of individual customers. Two tables are created for the manipulation of the data format chosen in CSV files containing tables for previously mentioned instances.

Here's how we formulated and utilized these datasets:

1. **Regions, Cities, and Districts Dataset:**

We obtained data for regions, cities, and districts from a publicly available GitHub file \*₁. Then we have modified and adapted this data to suit the needs of our project. Each entry in the dataset includes:

- Region ID - Region Name - City ID - City Name - District ID - District Name

For example ***Figure 1***

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Region\_ID | | Region\_Name | City\_ID | City\_Name | District\_ID | District\_Name |
| 01 | | Adrar | 01 | Adrar | 001 | Adrar |
| 01 | | Adrar | 01 | Adrar | 013 | Ouled Ahmed Timmi |
| 01 | | Adrar | 02 | Aoulef | 022 | Timekten |
| 02 | | Chlef | 01 | Abou El Hassane | 060 | Talassa |
| 02 | | Chlef | 12 | Tenes | 062 | Tenes |
| 05 | | Batna | 19 | Theniet El Abed | 156 | Oued Taga |
| **05** | | **Batna** | **12** | **Merouana** | **152** | **Merouana** |
| 06 | Béjaïa | | 10 | El Kseur | 229 | Toudja |

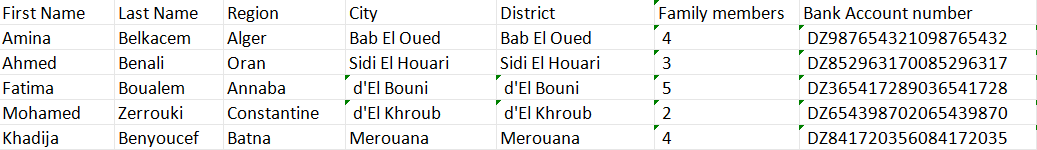
***Figure 1***: Dataset sample of geographic structure of Algeria

**2. Customers Dataset:**

To represent the households within the districts, we generated a customer dataset using data sourced from «https://www.bard.google.com». Each customer is initialized with the following parameters:

- First Name - Last Name - Region - City - District - Number of Family Members - Bank Account

For example ***Figure 2***



***Figure 2:*** Dataset Sample of cutomers information

Additionally, a unique customer ID was generated based on the region, city, district, and electricity account number, ensuring a distinctive identifier for each customer by concatenating the followings:

Customer\_ID = region\_id + city\_id + district\_id + electricity account number

The customer ID generation process, for instance, might look like this:

|  |  |  |
| --- | --- | --- |
| Khadija | Benyoucef | **512152**841720356084172035 |

1. **weather Dataset:**

To represent the weather data, we generated a dataset using data sourced from «https://www.bard.google.com» respecting the climate change from one month to another.

Each day weather is initialized with the following parameters:

Date,MaxTemperature,MinTemperature,SunnyHours,Condition

***Figure 3:*** Dataset Sample of weather.

**Solution Explanation**

* **ADTs & design choices**

\*\*\*\*\*Where and how this adt is used; why?->Time complexity\*\*\*\*\*

1. Hash Tables: Region City District Lina
2. Binary Search Trees: Wissam
3. Hierarchical Nested Classes and Vectors in Weather Data Management:

• **Weather Data Hierarchy:** A sophisticated hierarchy is used for weather data management using classes and vectors. This hierarchy includes data from daily weather records to monthly and yearly aggregations, gathered in a weather\_Table. This dynamically allocated vector is adept at managing multi-year weather data. The system facilitates data manipulation and retrieval through well-defined constructors, setters, and getters, handling comprehensive weather details like date, temperature, sunny hours, and weather conditions.

**• File Reading Implementation:** enhancing security and data management efficiency by implementing file reading capabilities for weather information and customer data.

**• Modularity and Clarity:** The architecture is designed for modularity and maintainability. It achieves this through a clear structure of distinct classes, each with specific responsibilities, ensuring a streamlined and understandable codebase.

III. Class Menu - User Interface Management

• **Interface Functionality:** The menu class serves as the primary user interface, facilitating interaction with the system. It provides role-based options, allowing for either Administrator or Customer interactions. *Administrators* can manage customer data, view and update bills, access detailed weather information, and evaluate performance of departments and customer contributions. *Customers*, on the other hand, can access their electricity bills for specified months, years, or defined periods.

**• Guided Interaction:** The system employs a clear and structured console output, guiding users through various functionalities. This approach ensures a user-friendly experience even for complex operations.

**• Error Handling and System Stability**: error handling mechanisms are in place to manage exceptions during runtime and ensuring the stability and reliability of the system.

**• Data Integrity and Input Validation:** To maintain data integrity and prevent input-related errors, the system includes comprehensive input validation processes.

**• Decision-Making Support for Administrators:** Administrators have access to specific date-based weather information, a feature that offers valuable insights for decision-making processes, particularly in the context of electricity management.

**• Secure File Reading for Sensitive Information:** The system is equipped with specialized file reading capabilities, especially for handling sensitive information like administrator passwords, enhancing overall data security.

1. Heaps: Departments Implementation

In ECMS, the Department implementation is fundamental to managing the financial aspects of marketing departments associated with each city. The solution involves the creation of a **Department** class and a **DepartmentHeap** class, utilizing heaps to efficiently identify and update the budgets of the best and worst performing departments.

**Department Class:**

The `Department` class encapsulates information related to a city's marketing department. It includes attributes such as `city\_id`, `budget`, and `totalAmountPaid`. The `Department` class allows setting and retrieving budget and total amount paid, crucial for financial management.

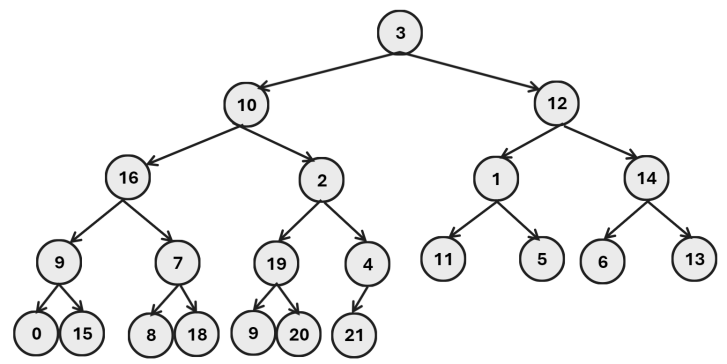
**DepartmentHeap Class:**

The `DepartmentHeap` class efficiently maintains the top-performing and bottom-performing departments using max and min heaps, respectively. The vector of departments holds all department instances, and the separate vectors for max and min heaps provide quick access to the best and worst departments based on their profit.

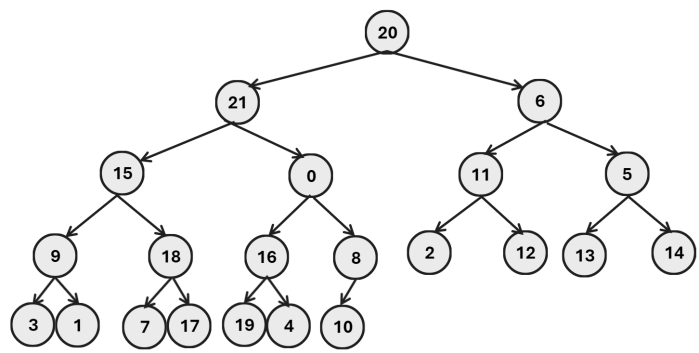
This implementation facilitates efficient identification of the top and bottom departments each year, enabling dynamic budget adjustments. ***-Figure 3-***

- `departments`: This vector stores instances of the `Department` class. Each element in this vector represents a department, and the vector serves as the main container for all departments.

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1. D1   P(12) | 1. D2   P(45) | 2.D3  P(78) | 3.D4  P(100) | 4.D5  P(50) | 5.D6  P(30) | 6.D7  P(25) | 7.D8  P(65) | 8.D9  P(19) | 9.D10  P(36) | 10.D11  P(87) |
| 11.D12  P(28) | 12.D13  P(90) | 13.D14  P(44) | 14.D15  P(55) | 15.D16  P(21) | 16.D17  P(36) | 17.D18  P(66) | 18.D19  P(48) | 19.D20  P(73) | 20.D21  P(05) | 21.D22  P(10) |



- **minHeap:** This vector is used as a min heap, where the indices of departments in the `departments` vector are stored based on their totalAmountPaid. The department with the lowest totalAmountPaid is at the top of this heap.



- **maxHeap:** This vector is used as a max heap, where the indices of departments in the `departments` vector are stored based on their totalAmountPaid. The department with the highest totalAmountPaid is

at the top of this heap.

**Heap Maintenance Functions:**

\*Insertion `insertDepartment`:

- When a new department is inserted using the `insertDepartment` function, its index is added to both the max and min heaps.

\*Max Heapify `maxHeapifyUp` and `maxHeapifyDown`:

- These functions are used to maintain the max heap property. `maxHeapifyUp` is called when a new element is added, and `maxHeapifyDown` is called when elements are removed or their values are updated.

\*Min Heapify `minHeapifyUp` and `minHeapifyDown`:

- Similar to max heapify, these functions maintain the min heap property.

**Budget Update Algorithm:**

- Each each year, budget updates are performed on departments such that:

1. The budget of the best-performing department is increased by 15%.

2. The budgets of the next 9 best departments are increased by 10%.

3. The budget of the worst-performing department is decreased by 15%.

4. The budgets of the next 9 worst departments are decreased by 10%.

- After budget updates, `make\_heap` is used to rearrange the elements in the heaps based on the new totalAmountPaid values of the incoming year, ensuring the heap property is maintained.***-Figure 4-***

**Printing Operations:**

- `printHeap`, `printBestDepartments`, `printWorstDepartments`, `getBest10`, and `getWorst10` functions are provided to display information about the departments in the heaps.

**Time Complexity:**

- The time complexity of heap operations (insertion, updating, and heapify operations) is O(log n), where 'n' is the number of elements in the heap.

- `make\_heap` operation is considered O(n), but its impact is limited to the top 10 elements, so it's practically O(1).

**Why Heaps?**

It is right that extra space will be used for this implementation but here we care about time complexity more than space complexity.

- Efficient Rankings: Heaps efficiently maintain the ordering of departments based on profit margins, enabling quick identification of top and bottom performers.

- Dynamic Updates: Heap operations are crucial for updating budgets dynamically, responding to yearly performance changes.

- Optimized Time Complexity: The logarithmic time complexity of heap operations ensures efficient processing, especially for a large number of departments.

//AVL: Nouha

* **Results**

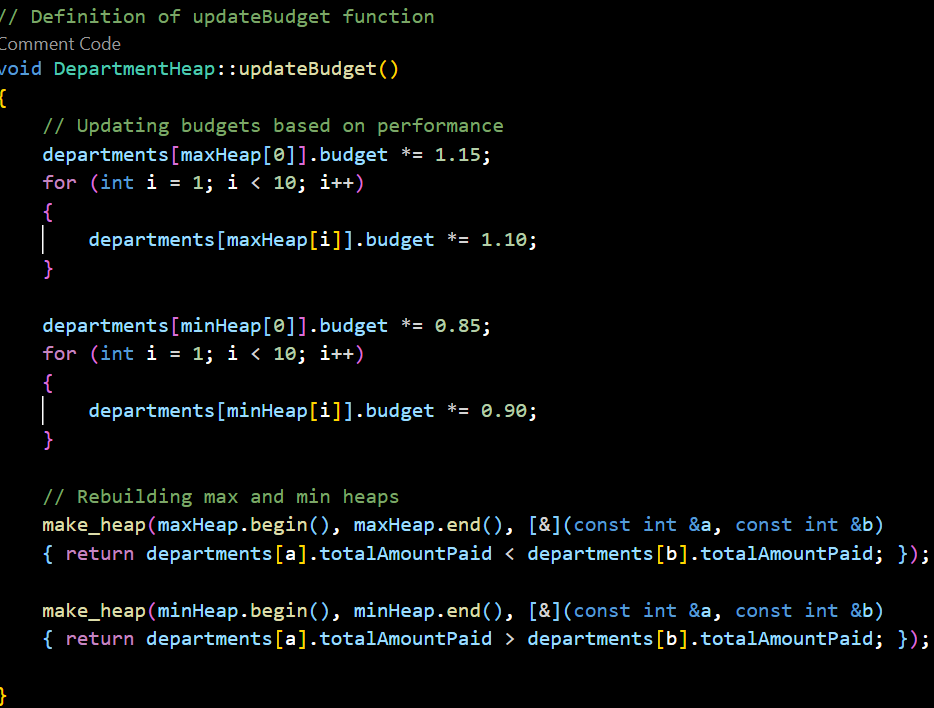
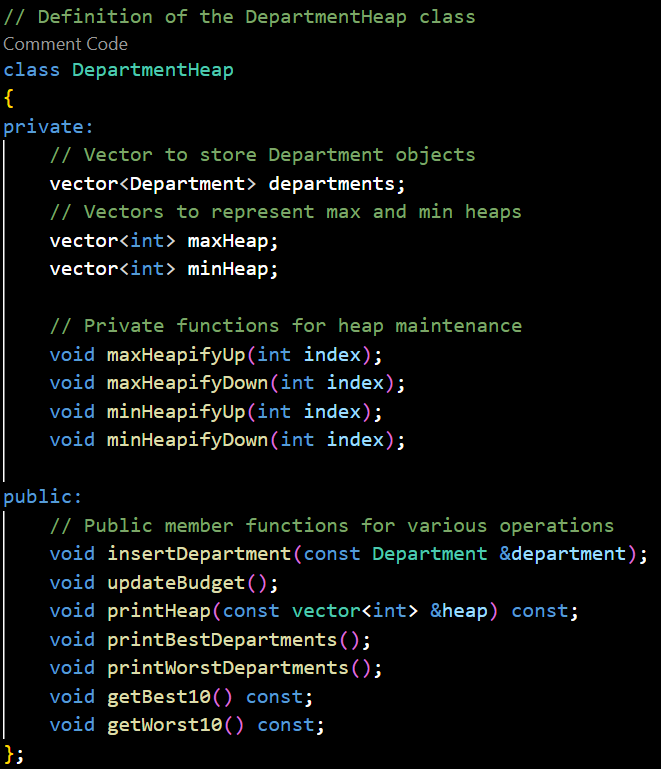
- Answer the question 7; using comparative tables (time; space; efficiency)

- Give results summary of how the program runs including guidelines (examples of running)

**Team members tasks**

* **Mami Nourhane**
* **Slama Lina Nour**
* **Lakhdari Wissam**
* **Demmouche Lina**

**Appendix**



***- Figure 3:*** DepartmentHeap class ***-Figure 4:*** UpdateBudget Function

**References**

DO NOT FORGET THE SCANNED PAPER!