Data Structures and Algorithms
HashMap linked AVL
Graph using Adjacency List

•••

- Moni, JP, Kelvin, Saurav

Agenda

- Context
- Problem Statement
- Solutions Overview
- Data Structure I - Hashmap Linked AVL
- Data Structure 2 - Graph
- Algorithm 1 - Data Science Queries
- Algorithm 2 - Centrality
- Complexity Analysis
- Conclusion
- Challenges

Context

- Four of us are developing an **e-commerce startup** where we need to capture user data to increase **efficiency of our marketing efforts**.
- The best way to analyze our data is using two main pictures, **the macro environment and the micro environment**.
- We define the **macro environment** as the overall best place, best location, best age group to target, to give us the best returns on the marketing spend.
- We define the **micro environment** as how to particularly target the niche we identify from the macro-analysis.
- Our aim in this project is to efficiently answer two problem statements as mentioned on the next slide.

Problem Statement

1. **What is an efficient way to store all user data captured on our website and query it as we want? (Macro Problem)**
    - This problem statement helps us identify the macro we want to target to effectively increase our user activity.
2. **What is the best way to strategize marketing efforts based on the macro we identify? (Micro Problem)**
    - This statement helps us identify who in the macro would be the right people to target to build out a network effect, where our aim will be to capture a percentage of the network.

Solution Overview

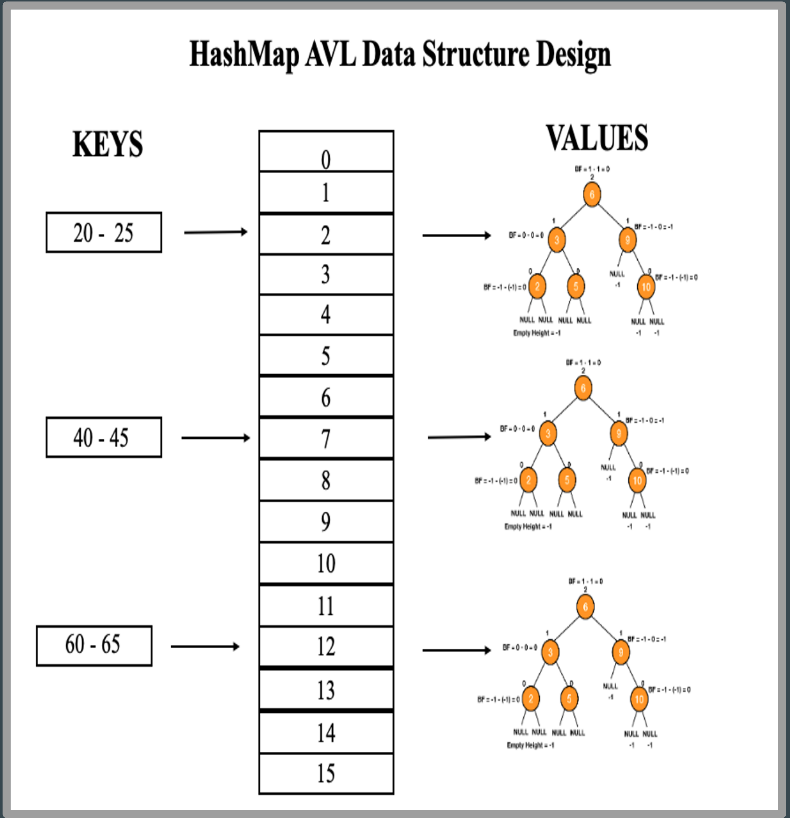
After a lot of brainstorming, we came down to the below data structures to solve our problems:

- **HashMap Linked AVL** - This particular data structure gives us the ability to differentiate data and set it in a different **AVL based on an attribute**. For the purposes of this project, we used age group as a factor to create multiple keys. This would **effectively help us quickly search, filter data** on age attributes, and further location and name attributes as well.

- **Graph Data Structure** - We decided to use this data structure as it would help us effectively **build networks and show connections between users.** This would help us find the **centrality of the network**.. The centrality would give us a good output on who would be the best bet to target and get our money's worth!

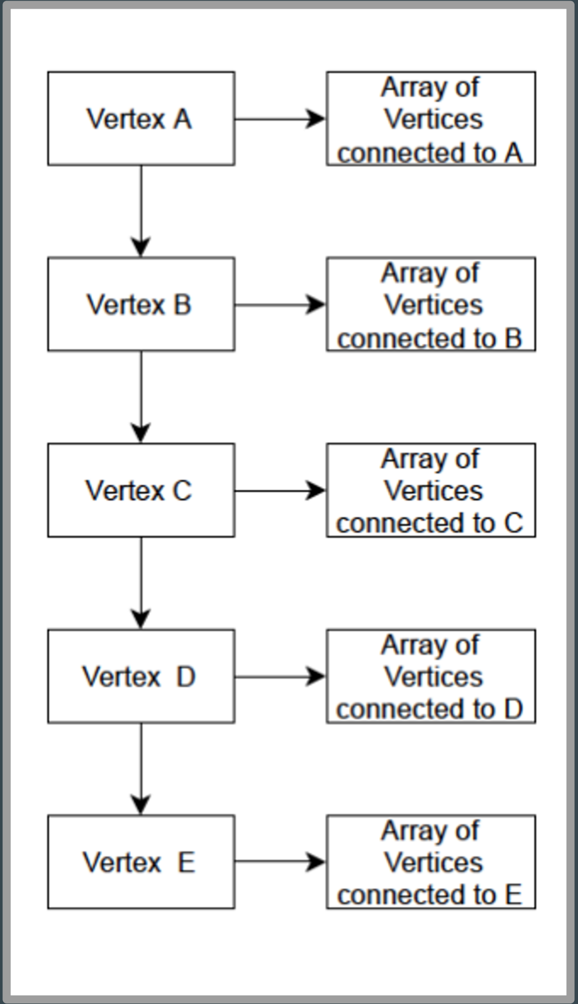
HashMap Linked AVL

- HashMap:
  - Key feature: **hash function**: make or break
  - A good hash function:
    - Evenly distribute keys in the data structure
    - **Prevent collisions**
- AVL Tree:
  - A binary search tree with self - balancing property
  - Key feature: **Rotations** play a critical role
    - Left - left
    - Left right
    - Right - right
    - Right - left
  - Maintaining the **balance** at every level of the tree



Graph

- We used an **adjacency list** representation compared to the adjacency matrix representation because of the **high possibility of sparse networks**.
- Our graph data structure represents an **unweighted bidirectional graph**.
- Each of the Vertices are part of the **singly linked list** inside a graph data structure, and each of the **vertices** have an **array inside their structure** that represents the vertices it is connected to.



Data Science Queries - Answer to the Macro

- With the Hashmap Linked AVL, we have implemented a **search function** which can give us really fast results when we know additional parameters for the user.
- For instance, if we wanted to search for a user who was of a particular age, we could first access the AVL with those ages, and then quickly search through AVL in log n time.
- Because we have 4 age groups, essentially with the age, we are able to search through for a user in log n/4 time, which is much faster.
- In the future, we can further develop this to include more querying, where we could perform fast map, filter, reduce operations.

Centrality - Answer to the Micro

- **Degree Centrality** - This algorithm looks at the number of nodes each vertices are connected, the one with the **most connections** are deemed as the **center** of the graph.
- We used this algorithm to **estimate our center**, as it was ideal for us to target the users with the most amount of connections. This would give us the **best shot at capturing most users** in that particular micro environment.

Space and Time Complexities

Hashmap Linked AVL

- **Space Complexity:** O(n) (for each)

- **Time Complexity:**
  - Insertion of key - value: O(1)
  - Search of key - value: O(1)
  - Deletion of key - value: O(1)
  - Insertion of node in tree: O(log n)
  - Search of node in tree: O(log n)
  - Deletion of node in tree: O(log n)

Graph

- **Space Complexity:** O(n + |E|)

- **Time Complexity:**
  - Insertion of Vertex: O(1)
  - Deletion of Vertex: O(n + |E|)
  - Add Edge: O(1)
  - Deletion of Edge: O(|E|)
  - Search: O(n)

Conclusion

- We wanted to **integrate the knowledge** that we have gained this semester in the domain of data structures and algorithms and apply it to a **real world problem**.
- We **successfully managed** to come up with intriguing data structures to tackle both the problems and implement them to the best of our abilities.
- We do believe that what we have learnt from this project is the **start of our exploration about using data structures and algorithms** in the field of computer science.

Challenges

- Implementation was challenging especially the **self balancing aspect** of the AVL tree and managing all the **different rotations**.
- Implementation of the **graph data structures** had some roadblocks as well, we had difficulty figuring out what kind of **list to use for the graph struct**.
- Dealing with **C was a challenge**