HOSTEL COUNSELLING AND MANAGEMENT

Rushabh Kela[#1]

[#]*B.Tech – Computer Science with specialization in Data Science (SCOPE)*
Vellore Institute of Technology, Vellore-632014, India.
[1]rushabhbhagwandas.kela2019@vitstudent.ac.in

ABSTRACT – Every year VIT HOSTEL COUNSELLING is held in the second/third week of March during the Winter Semester for students' room reservation for the ensuing academic year. Room reservation criteria and rules will be purely based on the academic performance well as the clean chit record of students' discipline as per Code of Conduct, as practiced every year. Simulation of this project will involve taking the details of the students as inputs, then the will be stored in respective branches (such as CSE, ECE, Mechanical etc.). The desired output shall be a combined list of students from all branches along with their NCGPA's followed by a user – friendly counselling procedure and room allotments.

KEYWORDS – Hash Tables, Key, Value, Collision, Chaining, Queues, FIFO (First In First Out), Linked List, Address, Node, Open Addressing, Data Structures.

I. INTRODUCTION

For many applications of computer science, performance is important. Sometimes, even small performance gains can save a lot of time and memory. Improving performance can be done at different levels. For example, an algorithm can be improved by decreasing the number of calculations, by using extra data structures to reduce the number of redundant calculations, or by improving worst case time complexities.

In this project, we choose and implement a NCGPA list for every student branch in a **HASH TABLE** and then generate a rank list for all the students in the university and then they will come in a queue according to their rank and select the room which they desire accordingly. The branches are represented by a hash table each and the specific branch hash table consists of the classes of the given branches. Then the students are related to the given class by chaining through a given hash function. The registration number gives out the specific class of the student. The allotment is based on the queue concept. Here we make use of the various **data structures** like **queue, hash tables, linked lists etc.**

II. OVERVIEW

Efficient data structures and algorithms can be the key to designing practical problems. In computer science, a **data structure** is a data organization, management, and storage format that enables efficient access and modification. More precisely, a data structure is a collection of data values, the relationships among them, and the functions or operations that can be applied to the data.

A. OBJECTIVE OF THIS PROJECT :

- To study in detail, applications of hash tables, queues and their implementation using linked list.
- To provide a hassle – free and a compact solution for the hostel counselling procedure in VIT.

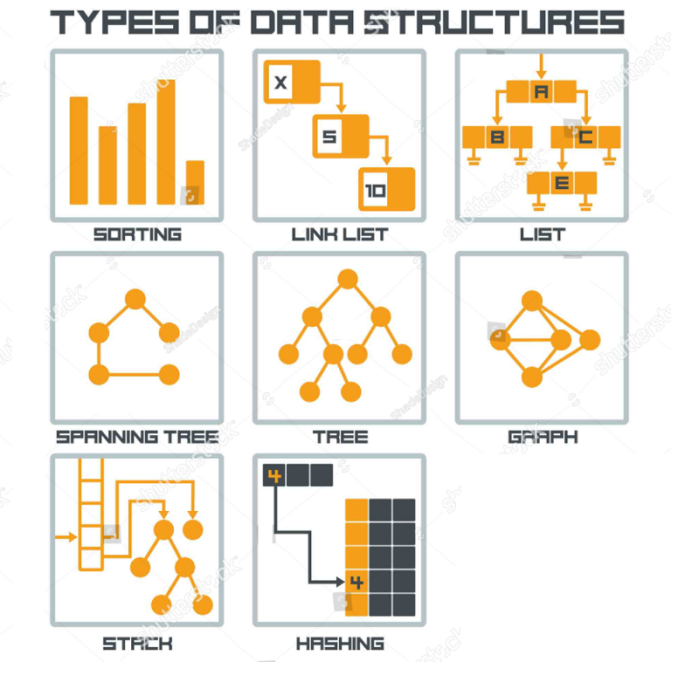

Fig 1: Commonly used Data Structures

III. Data Structures Implemented

In this project, data structures such as linked lists, hash tables, queue etc. are used. This research mainly deals with the aforementioned data structures in detail.

A. LINKED LISTS

A **linked list** is a set of **dynamically allocated** nodes, arranged in such a way that each node contains one value and one pointer. The pointer always points to the next member of the **list**. If the pointer is NULL, then it is the last node in the **list**. A linked list is formed when many such nodes are linked together to form a chain. Each node points to the next node present in the order. The first node is always used as a reference to traverse the list and is called **HEAD**. The last node points to **NULL**.

Structure of a Singly Linked List :

```
struct node
{
      int data;
      struct node *next;
} *head=NULL;
```

Each node in a linked list consists of two fields, one for the information and another is a pointer which contains the address (link) to the next node in the list.

Linked Lists have an edge over arrays mainly because they are dynamic in nature and allocate memory during run-time **(Dynamic Memory Allocation)**, while in the latter, memory is allocated at compile time **(Static Memory Allocation)**. Ease of insertion/deletion in linked lists is also an advantage whereas in arrays, insertion and deletion of elements is expensive and involves shifting of data.

Drawbacks of Linked lists are unlike arrays, random access of elements is not possible and elements have to accessed sequentially. Some amount of memory is wasted since pointers require extra memory for storage.

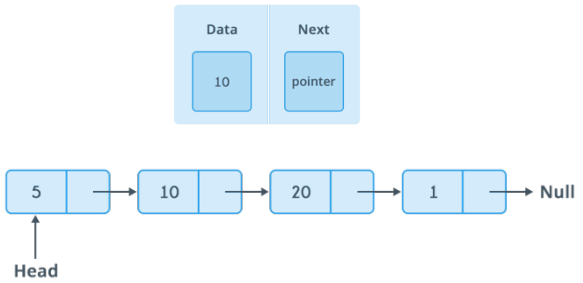

Fig 2: Representation of Linked Lists

B. HASH-TABLE

Hashing is a technique that is used to uniquely identify a specific object from a group of similar objects. An example of how hashing is used is, in universities, each student is assigned a unique roll number that can be used to retrieve information about them.

In hashing, large keys are converted into small keys by using **hash functions**. The values are then stored in a data structure called **hash table**. The idea of hashing is to distribute entries (key/value pairs) uniformly across an array. Each element is assigned a key (converted key). By using that key you can access the element in **O(1)** time. Using the key, the algorithm (hash function) computes an index that suggests where an entry can be found or inserted.

In many situations, hash tables turn out to be on average more efficient than search trees or any other table lookup structure. For this reason, they are widely used in many kinds of computer software, particularly for associative arrays, database indexing, caches, and sets.

TABLE I
TIME COMPLEXITY

ALGORITHM	AVERAGE	WORST CASE
Space	O(*n*)	O(*n*)
Search	O(1)	O(*n*)
Insert	O(1)	O(*n*)
Delete	O(1)	O(*n*)

1) COLLISIONS :

Hash collisions are practically unavoidable when hashing a random subset of a large set of possible keys. For example, if 2,450 keys are hashed into a million buckets, even with a perfectly

uniform random distribution, according to the **birthday paradox** there is approximately a 95% chance of at least two of the keys being hashed to the same slot.

Fig 3: Probability of collision among hash table entries represented on a graph.

Therefore, almost all hash table implementations have some collision resolution strategy to handle such events. Some methods are Linear Probing, Quadratic Probing, Double Hashing, Separate Chaining, Open Addressing. All these methods require that the keys (or pointers to them) be stored in the table, together with the associated values.

2) COLLISION RESOLUTION :

In this project, since there are 2 slots for each branch, only two indexes are used and the students are hashed to the corresponding index, and hence collisions are bound to occur.

Collisions are avoided by using linked list data structures by the methods of open hashing/ separate chaining. Separate chaining is one of the most commonly used collision resolution techniques. It is usually implemented using linked lists. In separate chaining, each element of the hash table is a linked list. To store an element in the hash table you must insert it into a specific linked list. If there is any collision (i.e. two different elements have same hash value) then store both the elements in the same linked list.

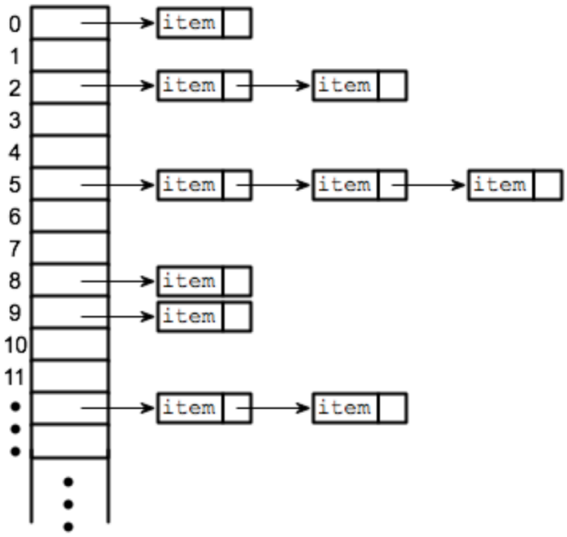

Fig 4: Collision resolution technique using separate chaining, with linked lists implementation

C. QUEUES

Queues are data structures that follow the **first in first out (FIFO)** i.e. The first element that is added to the queue is the first one to be removed. It is a linear data structure and an abstract data type. Elements are always added to the back and removed from the front. Think of it as a line of people waiting for a bus. The person who is at the beginning of the line is the first one to enter the bus.

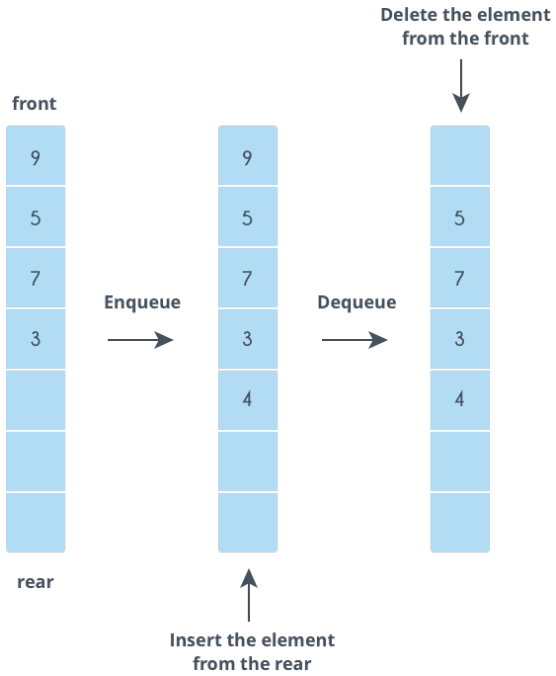

Fig 5: Representation of Queue Data Structure. Operations "enqueue" and "dequeue" are shown.

IV. Algorithms

A. *Sorting Data*

This project deals with Hostel Room Allotment to students on basis of their ranks obtained. Hence it is necessary to sort the data of students according to their NCGPA to get a sorted rank list.

There are many sorting algorithms available but to handle a huge and a large data set, an optimized and efficient algorithm is required which reduces the time complexity. The time complexity of an algorithm is the total time taken by the algorithm for its execution. The lesser the time-complexity, the faster is the execution.

Merge Sort is often preferred for sorting a linked list. The slow random-access performance of a linked list makes some other algorithms (such as quicksort) perform poorly, and others (such as heapsort) completely impossible.

TABLE III

TIME COMPLEXITY OF MERGE SORT

WORST-CASE	O(*n logn*)
BEST-CASE	O(*n logn*) typical, O(*n*) natural variant
AVERAGE	O(*n logn*)
WORST-CASE SPACE COMPLEXITY	O(*n*) total with O(*n*) auxiliary, O(*1*) auxiliary with linked lists.

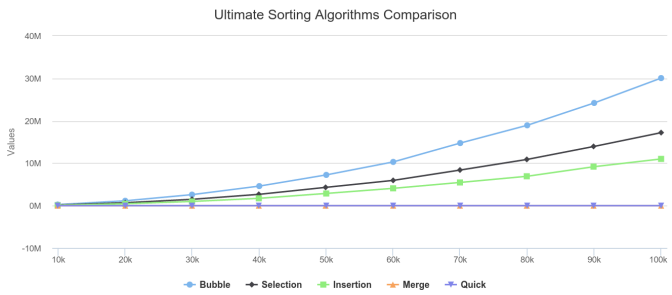

Fig 6: Graphical Representation of the sorting algorithms against different set of values and their performance

PSEUDOCODE :

```
struct node
{
        int data;
        struct node* next;
};

void Merge_Sort(struct node **headref)
{
        struct node* head = *headref;
        struct node *a;
        struct node *b;
        if(head==NULL || head->next==NULL)
                return;

        /*Split the linked list into two
        equal sublists.*/
        FrontBackSplit(head, &a, &b);

        //Sort the two halves
        Merge_Sort(&a);
        Merge_Sort(&b);

        /*Merge the sorted first half and
        second half, recursively and update
        the head pointer*/
        Merge(a,b);
}

struct node* Merge(struct node* a,
                                struct node* b)
{
        struct node* result=NULL;
        if(a==NULL)
                return b;
        if(b==NULL)
                return a;
        if(a->data <= b->data)
        {
                result=a;
                result->next=Merge(a->next,b);
        }
        else
        {
                result=b;
                result->next=Merge(a,b->next);
        }
        return result;
}
```

V. Problem Description

A Comparative Analysis between the Existing Model and the Proposed Model

- The project is based on the allotment of hostel rooms based on the NCGPA of the students of different branches.
- Separate hash tables have been used to represent different branches and each part

of the hash table is a slot (morning/evening).

- According to the different branches offered as B.TECH programmes, and user input of slot students are placed in the corresponding hash tables. There is no hash function used, instead the input of slot(morning/ evening) is taken from the user. Collisions are managed by open hashing/separate chaining using linked lists.
- Once the student records have been entered, then the next step is to calculate the NCGPA of different students. For this first the CGPA of branch topper and class topper was found, and then the NCGPA was calculated.
- Then, students of different slots within a hash table are merged and sorted them based on their NCGPA.
- All the records in the different hash tables were merged and sorted to obtain the final list based on their NCGPA.
- After this, the students are arranged in a queue on the basis of their NCGPA rank.
- Unlike the existing model, in this project students who are not attending the counselling are already marked absent and those who will attend the counselling are present in the queue.
- After final list is created, they are arranged in a queue which follows FIFO concept, the first element (student) coming out of the queue will select the room and the element is deleted from the queue.

VI. Conclusion

The idea of this project is to create a hassle – free and a user-friendly environment for the hostel allotment of the student based on their grades and thereby ranks obtained. The data structures used in this project and algorithms implemented are done taking into consideration the large dataset of a student database in a university so that they execute in minimal time and have lesser time-complexity for faster execution. The efficient execution of the process in taken into account while creating the program.

Acknowledgment

On accomplishment of this project, I feel privileged and grateful in taking this opportunity to express my gratitude to many people who have contributed to the completion of this project.

First of all, I would like express my deepest gratitude to **Prof. Seetha R** for her motivating mentorship during the project and valuable guidance throughout.

I thank my Parents, all my friends and classmates for their unparalleled support.

Above all, to the Almighty, the author of knowledge and wisdom for his support.

References

[1] Thomas H. Cormen, *Introduction to Algorithms*, 3rd ed., Charles E. Leiserson, Ed. The MIT Press, 1989.

[2] J. Breckling, Ed., *Data Structures and Algorithms Made Easy*, Narasimha Karumanchi, 2011.

[3] Ching-Kuang Shene, "A comaparative study of Linked Lists sorting Algorithms," *ReasearchGate.*, Michigan Technological University, Aug. 1997.

[4] (2002) The IEEE website. [Online]. Available: http://www.ieee.org/

[5] GeeksForGeeks : https://www.geeksforgeeks.org/

[6] Programiz - Learn to code for free : https://www.programiz.com/

Video Link

https://youtu.be/Ew6JAtvvED4