|  |  |  |  |
| --- | --- | --- | --- |
| **Course Name:** | **Data Structures Laboratory (using C/C++)** | **Semester:** | **III** |
| **Date of Performance:** | **17 / 10 / 2023** | **Batch No:** | **A - 3** |
| **Faculty Name:** | **Prof. Om Goswami** | **Roll No:** | **16014022050** |
| **Faculty Sign & Date:** |  | **Grade/Marks:** | **\_\_\_ / 25** |

**Experiment No: 9**

**Title: Hash Tables**

|  |
| --- |
| **Aim and Objective of the Experiment:** |
| Implementation of hash tables and its basic operations in VLAB. |

|  |
| --- |
| **COs to be achieved:** |
| **CO 3:** Explain concepts of advanced data structures like set, map dictionary.  **CO 4:** Demonstrate sorting and searching methods. |

|  |
| --- |
| **Books/Journals/Websites referred:** |
| 1. <https://ds1-iiith.vlabs.ac.in/exp/hash-tables/hash-tables/hash-tables-operations.html> 2. <https://ds1-iiith.vlabs.ac.in/exp/hash-tables/hash-tables/ht_practice.html> |

|  |
| --- |
| **Tools required:** |
| **DEV C/C++ compiler/ Code blocks C compiler** |

|  |
| --- |
| **Theory:** |
| Hash Table is a data structure which stores data in an associative manner. In a hash table, data is stored in an array format, where each data value has its own unique index value. Access of data becomes very fast if we know the index of the desired data.  Thus, it becomes a data structure in which insertion and search operations are very fast irrespective of the size of the data. Hash Table uses an array as a storage medium and uses hash technique to generate an index where an element is to be inserted or is to be located from.  **VLAB Link for experiment:**  Link: <https://ds1-iiith.vlabs.ac.in/exp/hash-tables/hash-tables/hash-tables-operations.html>  Practice: <https://ds1-iiith.vlabs.ac.in/exp/hash-tables/hash-tables/ht_practice.html> |

|  |
| --- |
| **Implementation Details:** |
| 1. **Enlist all the Steps followed and various options explored.** 2. **Explain your program logic and methods used.** 3. **Explain the Importance of the approach followed by you.** |

|  |
| --- |
| **Simulation Screenshots:** |
| * **Insert:**          * **Search:**        * **Remove:**      * **Quiz:** |

|  |
| --- |
| **Output:** |
|  |

|  |
| --- |
| **Conclusion:** |
| Through this experiment on hash tables, we gained practical insights into the efficient storage and retrieval of key-value pairs, highlighting the significance of hash functions in minimizing collision occurrences for optimal performance. |

|  |
| --- |
| **Post Lab:** |
| **Write an algorithm for performing basic primary operations on a hash table - search, insert, delete an element in a hash table (any one).**   1. **Seach Operation:**  * Compute the hash value for the given key. * Check if the element exists at the computed hash index. * Return the element if found; otherwise, return an indication that the element is not present.   // Search operation in the hash table  struct Node\* search(struct HashTable\* hashTable, int key) {      int index = hashFunction(key, hashTable->size);      struct Node\* currentNode = hashTable->array[index];      while (currentNode != NULL) {          if (currentNode->key == key) {              return currentNode; // Found the element          }          currentNode = currentNode->next; // Move to the next node      }      return NULL; // Element not found  }   1. **Insert Operation:**  * Compute the hash value for the given key. * Check if the element already exists at the computed hash index. * If it does, update the value of the existing element. * If it doesn't, insert a new element with the key and value at the computed hash index.   // Insert operation in the hash table  void insert(struct HashTable\* hashTable, int key, int value) {      int index = hashFunction(key, hashTable->size);      struct Node\* newNode = (struct Node\*)malloc(sizeof(struct Node));      newNode->key = key;      newNode->value = value;      newNode->next = hashTable->array[index];      hashTable->array[index] = newNode;  } |

|  |
| --- |
| **Signature of faculty in-charge with Date:** |