|  |
| --- |
| TAD HashTable <K,V> |
| HashTable<T> = {Table = (NodeHash< K, V>1, NodeHash< K, V>2 …. NodeHash< K, V>n ) n > 0 } ∧ NodeHash< K, V>.getKey= getHash(Key) ∧  Table[Key] = NodeHash<K, V>, n = Number Of buckets.(Size) |
| {Inv.: - getHash(Hash Function) = |Key % n|  If NodeHash<K,V>.HashValue = NodeHash<K,V>n.HashValue,  Then NodeHash<K,V>.setNext = NodeHash<K,V>n  Number of buckets = Constant.  Load Factor = (NodeHash<K, V> in table / n) ≈ 1  }. |
| Primitive Operations:   * Put: Key x Value → Table. * Get: Key → NodeHash<K, V >. * ContainsKey: Key → Boolean. * Remove: Key → Table * GetHash: Key → Key. * SetSize: Number of Buckets → Table |

|  |
| --- |
| Put(Key, Value)  “This method is used to insert a key-value pair into the data structure. It checks if the provided key is not null and then proceeds to insert the key-value pair into the data structure based on the key's hash value. If the specified index in the Table array is empty, a new node containing the key and value is created and placed at that index. If the index is not empty, it iterates through the linked list at that index and adds the new node at the end of the list.”  {pre: NodeHash.Key ≠ Null}  {post: HashNode<K, V> = Table[Key],} |

|  |
| --- |
| Get(Key)  “This method is used to retrieve the value associated with a given key from the data structure. It first checks if the provided key is not null and then calculates the hash value for the key. If there is an entry at the index determined by the hash value, it iterates through the linked list at that index to find the node with the matching key.”  {pre: NodeHash.Key ≠ Null}  {post: If Key is found, then Return NodeHash<K,V>,  If not, then Null  } |

|  |
| --- |
| ContainsKey(Key)  “This method is used to determine whether a specific key exists in the data structure. It computes the hash of the provided key and checks whether an entry exists at the index determined by the hash. If the index is empty (that is, there is no entry at that position), this method returns false, indicating that the key does not exist. If an entry exists at that index, the method will traverse the linked list at that index to find the node with the matching key.”  {pre: NodeHash.Key ≠ Null }  {post: returns True if Table[Key] ≠ Null,  If not, then returns False.  } |

|  |
| --- |
| Remove(Key)  “This method is used to remove a key-value pair from the data structure based on the provided key. It first calculates the hash value for the given key and checks if there is an entry at the index determined by the hash value. If the index is not empty, the method iterates through the linked list at that index to locate the node with the matching key.”  {pre: NodeHash.Key ≠ Null}  {post : If Table[Key] ≠ Null, Then Table[Key] = Null,  Else, Tabla[n] = Tabla[n]  } |

|  |
| --- |
| GetHash(Key)  “This method is used to calculate a hash code for a given key, which is used to determine the index where the key-value pair should be stored in the data structure.”  {pre: Number of buckets(n) > 0}  {post: Returns Int Key} |

|  |
| --- |
| SetSize(n)  “This method is used to set the size of the data structure.”  {pre: Number of buckets(n) > 0}  {post: new Tabla[n]} |