**LAB # 05**

**Implementing Hashing techniques and using HashTable ADT**

**Object**

Implementing Hashing techniques with collision resolution.

**Theory**

Hashing technique is used for ordering and accessing elements in a list in a relatively constant amount of time by manipulating the key to identify its location in the list.

Functions use to range a key values which is transformed into a range of array index values is called hash function and the particular index value is called key in a hash table.

***hkey*** = hfunc(***key)*** is a hash function for integer keys The integer ***hkey*** is called the hash value of ***key***

Different types of hash functions are used for the mapping of keys into tables.

* + 1. Division Method
    2. Mid-square Method
    3. Folding Method

A hash table is a data structure that offers very fast insertion and searching. No matter how many data items there are, insertion and searching (and sometimes deletion) can take close to constant time.

**Collision:**

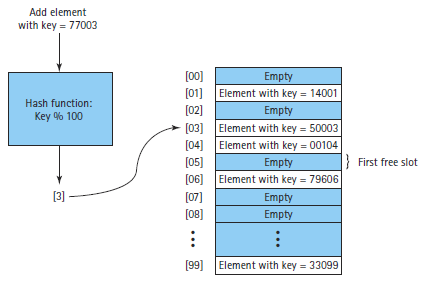
The condition when resulting two or more keys produce the same hash location.

A good hash function minimizes collision by spreading the elements uniformly throughout the array. We can minimize collision but it’s extremely difficult to avoid them completely. There are many collision resolution algorithm.

1. Linear Probing
2. Quadratic Probing
3. Double Hashing.

**Linear Probing:**

A simple approach to resolving collisions is to store the colliding element in the next available space.



|  |  |
| --- | --- |
| **Class Name:** **DataItem** |  |
| **Member Variables** | **Responsibilities** |
| private int iData;  // data item (key) | public int getKey()  //return iData |
| **Constructors** |  |
| public DataItem(int ii)  iData = ii |  |

|  |  |
| --- | --- |
| **Class Name: HashTable** |  |
| **Member Variables** | **Responsibilities** |
| private DataItem[] hashArray; // array holds hash table | public void displayTable() |
| private int arraySize | public int hashFunc(int key) |
| **Constructors** | public void insert(DataItem item)  // insert a DataItem |
| HashTable(int size) // constructor | public DataItem delete(int key) |
|  | public int find(int key) // find item with key |
|  | public int hashFunc2(int key) // if double hashing used |

**Insert Algorithm**:

insert(int key, DataItem item)//insertitem at location

Precond : assumes table not full

1. int hashVal = hashFunc1(key); // hash the key
2. int stepSize = 1 // for linear prob

// until empty cell or -1

1. repeat while hashArray[hashVal] ≠ null && hashArray[hashVal].getKey() ≠ -1)
2. hashVal += stepSize; // add the step
3. hashVal %= arraySize; // for wraparound

// end of loop

1. hashArray[hashVal] = item; // insert item
2. return

**Find Algorithm:**

find(int key) // find item with key

// (assumes table not full)

1. int hashVal = hashFunc1(key); // hash the key
2. int stepSize = 1; // get step size
3. Repeat while hashArray[hashVal] != null) // until empty cell

// is correct hashVal?

1. If hashArray[hashVal].getKey() == key then
2. return hashVal; // yes, return index of table

else

1. hashVal += stepSize; // add the step
2. hashVal %= arraySize; // for wraparound

//end of loop

return null; // can’t find item

**Delete Algorithm:**

delete(int key) // delete a DataItem

1. loc:= find(key)
2. If loc=-1 then return null
3. Else
4. DataItem temp = hashArray[loc]; // save item
5. hashArray[loc] = nonItem; // delete item
6. return temp; // return item

//end of loop

// end delete()

**Linear Probing**

linear\_probing\_insert(K)  
if (table is full) error  
probe = h(K)  
while (table[probe] occupied)  
probe = (probe + 1) mod M  
table[probe] = K

**Hash ADT in Java**

|  |  |
| --- | --- |
| **Class Name: Hashtable** | **Super Class: Java.util Package** |
| **Responsibilities** |  |
| Void clear()  Clears this hashtable so that it contains no keys. | Object get(Object key)  Returns the value to which the specified key is mapped, or null if this map contains no mapping for the key |
| boolean isEmpty()  Tests if this hashtable maps no keys to values. | Object put(K key, V value)  Maps the specified key to the specified value in this hashtable. |
| boolean contains(Object value)  Tests if some key maps into the specified value in this hashtable | Int size()  Returns the number of keys in this hashtable. |
| boolean containsKey(Object key)  Tests if the specified object is a key in this hashtable. |  |
| boolean containsValue(Object value)  Returns true if this hashtable maps one or more keys to this value |  |

**Task**

* Write a class to maintain a hash function to implement a digit-folding approach in the hash function. Your program should work for any array size and any key length. Use linear probing.
* Enter data of a cricket team 11 players which is supposed to be a hash table value and insert runs of each player as a data, find out key treat’s Rank# of a player.

For example:

Runs are 30 mod by 11 which is index no 8; rank#8 is a rank of a team member.

(Use HashTable ADT class)