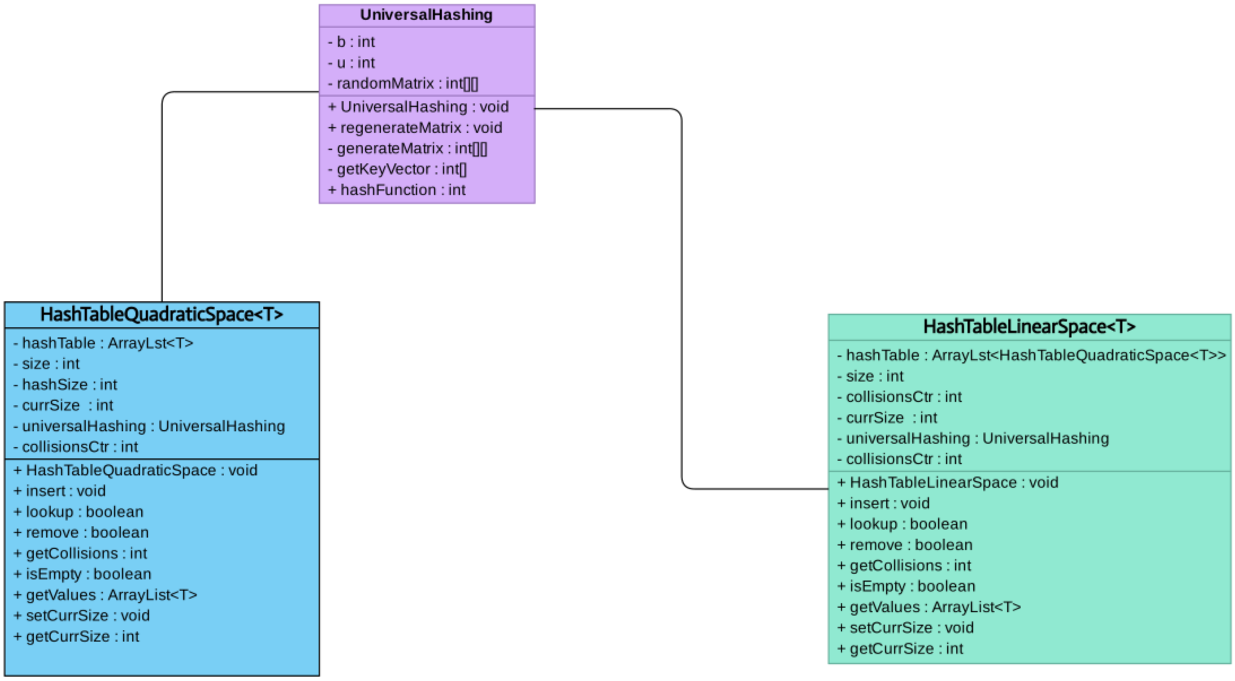
# Perfect Hashing and Universal Hashing

Student 1 : Ahmed Adel Abu Def
ID 1:  19015264

Student 2 : Muhammad Ibrahim Elkotb
ID 2 : 19016258

## Contents

# UML Diagram


UniversalHashing
- b : int
- u : int
- randomMatrix : int[][]
+ UniversalHashing : void
+ regenerateMatrix : void
- generateMatrix : int[][]
- getKeyVector : int[]
+ hashFunction : int
HashTableQuadraticSpace<T>
- hashTable : ArrayLst<T>
- size : int
- hashSize : int
- currSize  : int
- universalHashing : UniversalHashing
- collisionsCtr : int
+ HashTableQuadraticSpace : void
+ insert : void
+ lookup : boolean
+ remove : boolean
+ getCollisions : int
+ isEmpty : boolean
+ getValues : ArrayList<T>
+ setCurrSize : void
+ getCurrSize : int
HashTableLinearSpace<T>
- hashTable : ArrayLst<HashTableQuadraticSpace<T>>
- size : int
- collisionsCtr : int
- currSize  : int
- universalHashing : UniversalHashing
- collisionsCtr : int
+ HashTableLinearSpace : void
+ insert : void
+ lookup : boolean
+ remove : boolean
+ getCollisions : int
+ isEmpty : boolean
+ getValues : ArrayList<T>
+ setCurrSize : void
+ getCurrSize : int

# Universal Hashing

A probability distribution H over hash functions from U to {1, ..., M} is universal if for all x = y in U, we have

$$P\ r[h(x) = h(y)] \leq 1/M \qquad (1)$$

## Theorem

If H is universal, then for any set S ⊂ U, for any x ∈ U (that we might want to insert or lookup), for a random h taken from H, the expected number of collisions between x and other elements in S is at most N/M.

## Method

- Matrix method is used to construct hash Functions.
- If The table size is M and is power of 2, so an index is b-bits long with M = $2^b$
- h(x) = h * x (matrix multiplication with mod 2 addition)
- h will be a random matrix with values ranging from 0 to 1

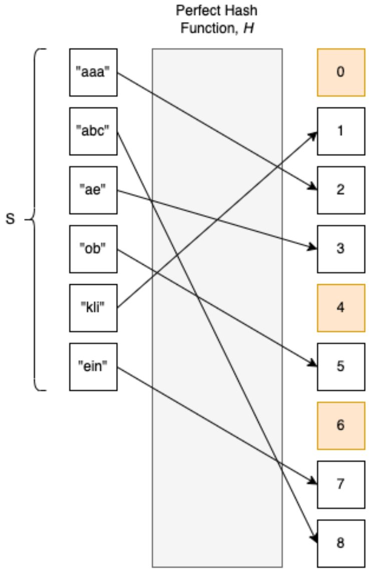
$$\begin{array}{cccc} h & x & & h(x) \\ \begin{bmatrix}1&0&0&0\\0&1&1&1\\1&1&1&0\end{bmatrix} & \begin{bmatrix}1\\0\\1\\0\end{bmatrix} & = & \begin{bmatrix}1\\1\\0\end{bmatrix}\end{array}$$

- **We can show that for x = y, P r[h(x) = h(y)] = 1/M = 1/2^b**

### Implementation

- generateMatrix returns 2D int array after generating random entries.
- getKeyVector returns 1D Vector that is a binary representation of x.
- hashFunction returns a key using randomly generated h Matrix.

# Perfect Hashing



A **perfect hash function** *h* for a set *S* is a hash function that maps distinct elements in *S* to a set of *m* integers, with no collisions. In mathematical terms, it is an injective function.

Disadvantages of perfect hash functions are that *S* needs to be known for the construction of the perfect hash function.

## O($N^2$) Space Solution

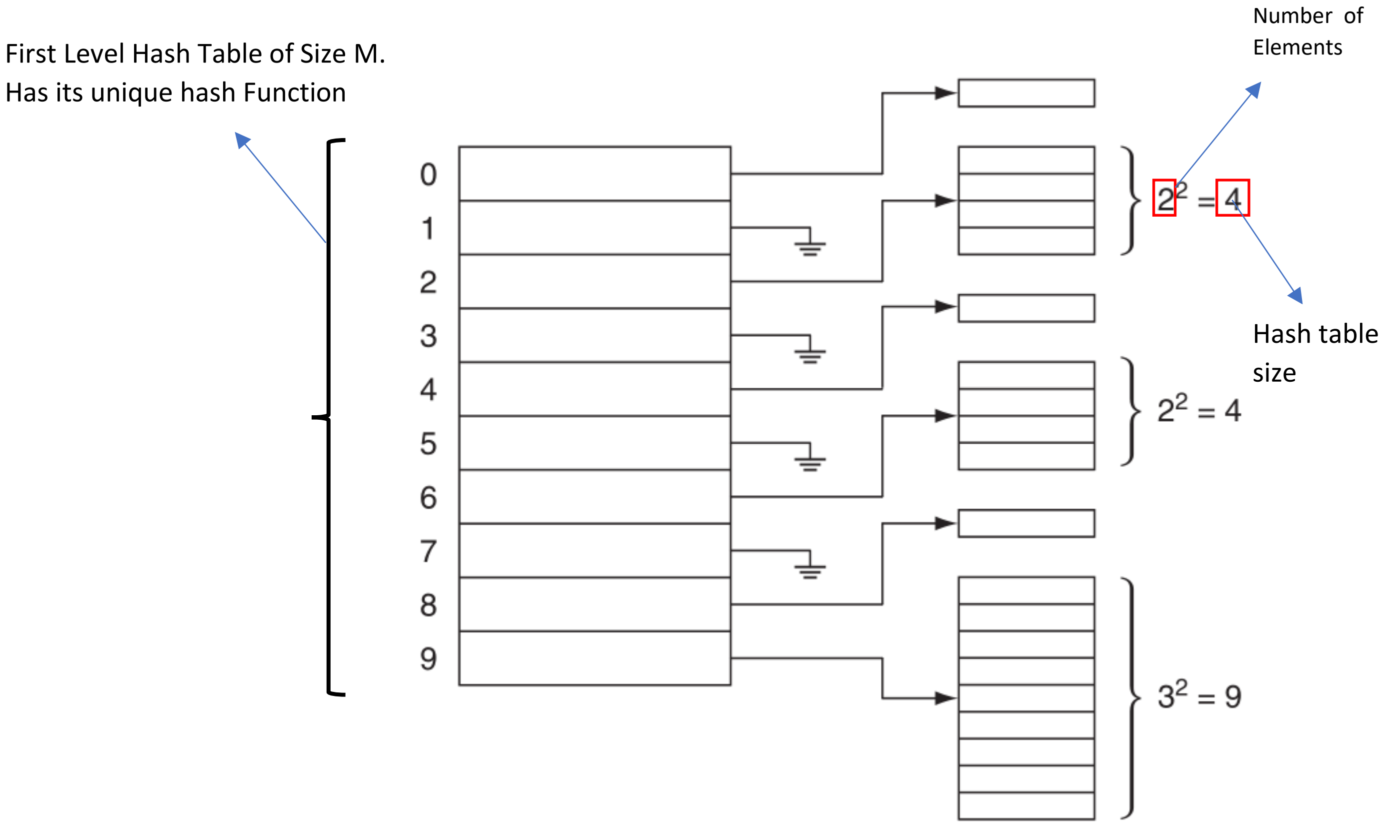
- If there are N elements to be inserted, A Hash table of size $N^2$ is made.
- Universal Hashing is used to generate a hash function.
- This hash function is used to hash every element.
- If a Collision occurs then we regenerate a Matrix to make a new Hash Function independent from the previous ones and rehash every element into table using the new hash function.
- A $N^2$ hash table is made to ensure that the probability that a Collision occurs is ½ and that total amount of Collisions is at most 2.
- Having a Table size of $N^2$ is impractical and a new solution is necessary.

### Implementation

- Insert function that takes the element as a parameter and checks for an empty bin using a hash function.
- Lookup takes an element as a parameter returns Boolean Value True if the given element exists in the hash table, False otherwise.
- Remove takes an element as a parameter returns Boolean value Ture if the element exists in the hash table and is removed successfully, False otherwise.

# O(N) Space Solution

- A cleverer approach to take is to make 2 level hash Table where every entry in the hash table is a hash table of Quadratic Space Solution in which its size grows as the number of Collisions on this entry grows.
- The First level Hash Table uses a hash function generated using Universal Hashing scheme.
- The First level tries to hash every element into empty bins.
- If a Collision is detected, then a new Hash Table inside the bin is constructed with a unique hash Function then it tries to hash every element hashed using the First level hash function into the bin (where collision occurred) using the new second level hash Function.

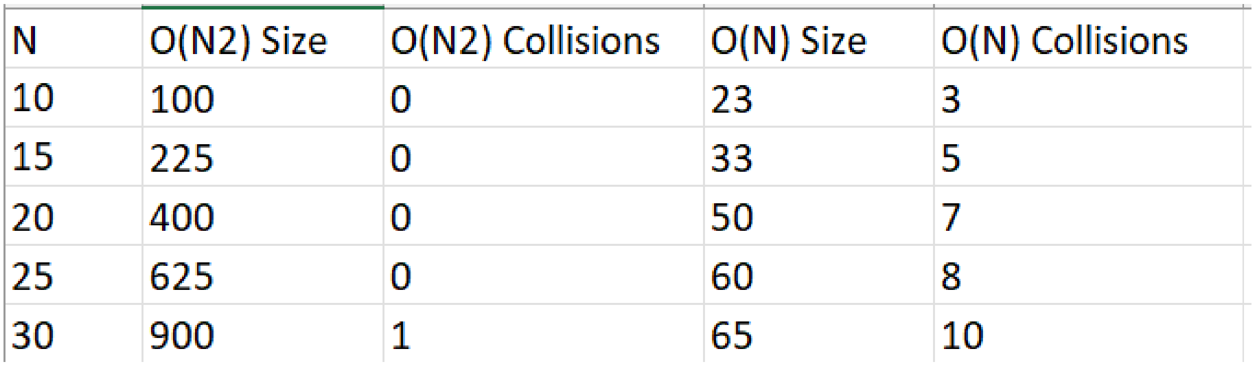


- The initial size of each entry is 1, if an entry has N elements in it then its size should be $N^2$ (from the $O(N^2)$ Solution).
- Each Entry is a Hash Table of Size Squared to number of elements inside it.
- Each Entry Hash Table has a unique hash function that is generated randomly using Universal Hashing Scheme and generated every time a Collision Occurs into this entry.

### Implementation

- Insert function that takes the element as a parameter and checks for an empty bin using a hash function.
- Lookup takes an element as a parameter returns Boolean Value True if the given element exists in the hash table, False otherwise.
- Remove takes an element as a parameter returns Boolean value Ture if the element exists in the hash table and is removed successfully, False otherwise.

## Space Complexity Analysis



| N | O(N2) Size | O(N2) Collisions | O(N) Size | O(N) Collisions |
|---|---|---|---|---|
| 10 | 100 | 0 | 23 | 3 |
| 15 | 225 | 0 | 33 | 5 |
| 20 | 400 | 0 | 50 | 7 |
| 25 | 625 | 0 | 60 | 8 |
| 30 | 900 | 1 | 65 | 10 |