NAME : - Jinggui Tan & Siddhant Barua
CWID :- 10427381 & 10439929

CS 590 Assignment 5

- We wish to implement a dictionary by using direct addressing on a huge array. At the start, the array entries may contain garbage, and initializing the entire array is impractical because of its size. Describe a scheme for implementing a direct-address dictionary on a huge array. Each stored object should use O(1) space; the operations SEARCH, INSERT, and DELETE should take O(1) time each; and the initialization of the data structure should take O(1) time. (Hint:Use an additional stack, whose size is the number of keys actually stored in the dictionary,to help determine whether a given entry in the huge array is valid or not.)

Answer :

Additional stack implemented by an array used to store the index of key => kStack.
Where size is the number of keys existed. Increase size after each insertion and decrease size after each deletion.
Another stack which is also implemented by an array to store value => vStack.
You can also store the key-value pair together if you wish. That doesn't matter.
Huge array => table.

Initialize :

```
size = 0;
```

Insert :

```
if(search(key) == null) //if not exist
      Input key,value;
      size ++;
      index <- Size;
      kStack[index] <- key;
      vStack[index] <- value;
      table[key] <- index;
fi
```

NAME : - Jinggui Tan & Siddhant Barua
CWID :- 10427381 & 10439929

Search:

```
Input key;
index = table[key];
If 0 < index <= size  and kStack[index] == key
      return vStack[index];
Else
      Return null;
fi
```

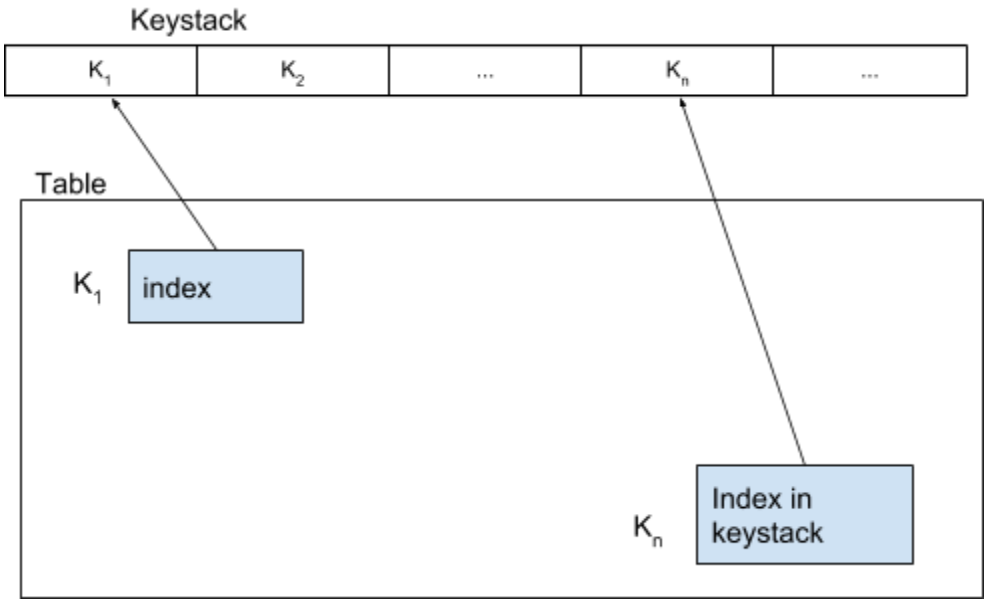
Delete:

```
input key;
If search(key) != null //exists
      index<- table[Key]; // index of key to be deleted

      kStack[index] <- kStack[size]; // swap last key and deleted key
      vStack[index] <- vStack[size]; // swap last key and deleted key

      table[kStack[size]]<-index;    // swap the position in table
                                     // between index of last key and
                                     // index of key to be deleted
      table[key] <- 0; // set the position of the key to be deleted to 0
      size --;
fi
```

The above operations (Initialize, Insert, Search and Delete) can all be finished in constant time O(1)

NAME : - Jinggui Tan & Siddhant Barua
CWID :- 10427381 & 10439929

Insert and search

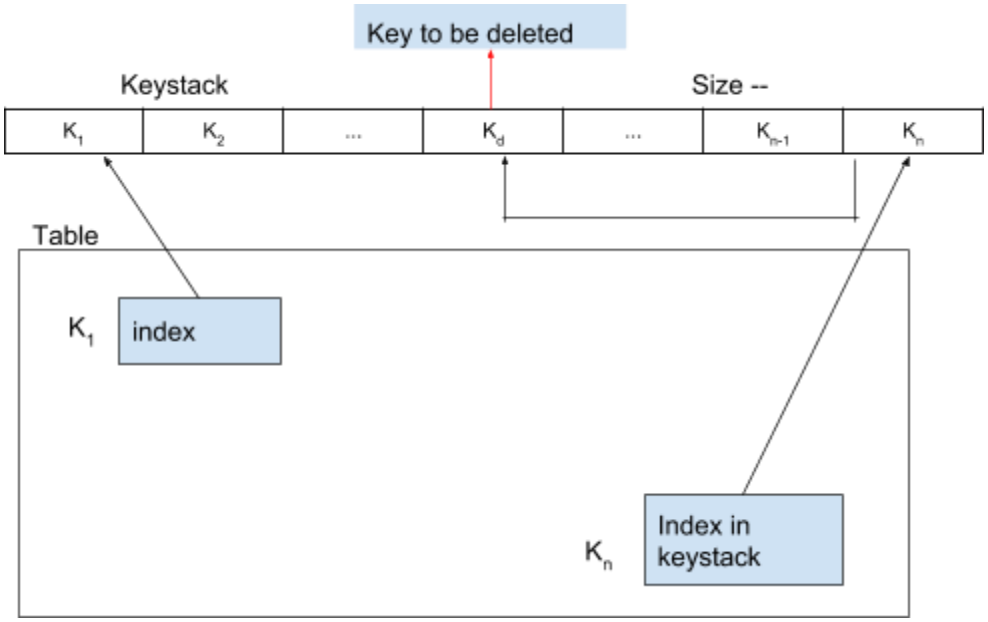


The index must be less than or equal to size and also be greater than 0;

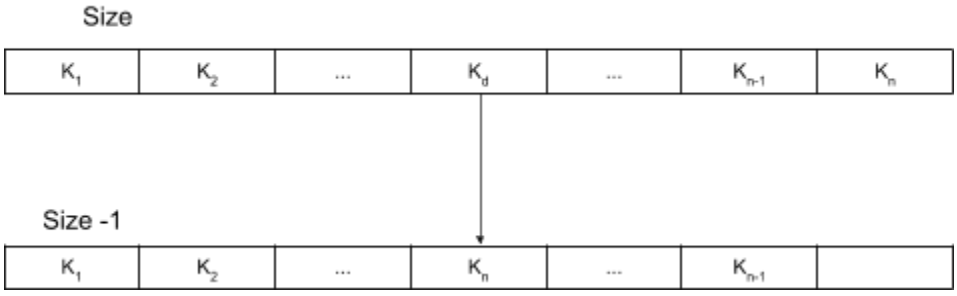
If Table[key] > size then it's invalid
If Table[key] $\leq$ size then it may be dirty data hence we check if Kstack [table[key]] == key;

NAME : - Jinggui Tan & Siddhant Barua
CWID :- 10427381 & 10439929

Delete

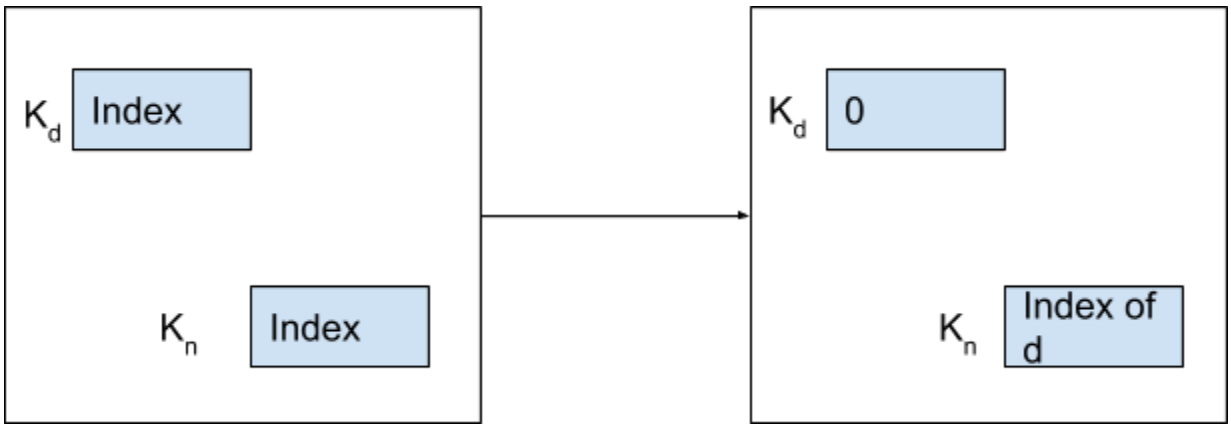


When we delete an item, we can't just delete that key from the stack, because it's a stack (implemented using an array). If the item is in the middle we delete it, then we need to remove each item after that item , this leads to time complexity of O(n).
Insted, we replace it, replace it with the last item/key.
Switch Key_{size} and $Key_{to\ be\ deleted}$ then do size -- ;



NAME : - Jinggui Tan & Siddhant Barua
CWID :- 10427381 & 10439929

Everything in the table should be replaced in the same way



NAME : - Jinggui Tan & Siddhant Barua
CWID :- 10427381 & 10439929

- Consider a hash table of size m = 1000 and a corresponding hash function:

  $h(k) = \lfloor m(kA \bmod 1) \rfloor, \; A = \frac{\sqrt{5}-1}{2}$

  Compute the locations to which the keys 61, 62, 63, 64, and 65 are mapped.

➢ We are given the hash table size of m = 1000,
We then consider the value of A = $\frac{\sqrt{5}-1}{2}$ = 0.61803398875
We are given the Keys to be as follows k = 61, 62, 63, 64 and 65.

- For $k = 61$,

  $h(k) = [m(kA \; mod \; 1)], \; where \; A = 0.61803398875.$

  $h(k) = [1000 * (61 * 0.61803398875 \; mod \; 1)]$

  $h(k) = [1000 * (37.7000733137 \; mod \; 1)]$

  $h(k) = [1000 * 0.7000733137]$

  $h(k) = 700.0733137 \; or \; 700 \; (approx) \; hence \; h(61) = 700.$

  The location of k = 61 will be at 700.

- For $k = 62$,

  $h(k) = [m(kA \; mod \; 1)], \; A = 0.61803398875.$

  $h(k) = [1000 * (62 * 0.61803398875 \; mod \; 1)]$

  $h(k) = [1000 * (38.3181073025 \; mod \; 1)]$

  $h(k) = [1000 * 0.3181073025]$

  $h(k) = 318.1073025 \; or \; 318 \; (approx) \; or \; h(62) = 318.$

  The location of k = 62 will be at 318.

NAME : - Jinggui Tan & Siddhant Barua
CWID :- 10427381 & 10439929

- For $k = 63$,

  $h(k) = [m(kA \bmod 1)]$, $A = 0.61803398875.$

  $h(k) = [\,1000 * (\,63 * 0.61803398875 \bmod 1)]$

  $h(k) = [\,1000 * (\,38.93614129125 \bmod 1)]$

  $h(k) = [\,1000 * 0.93614129125\,]$

  $h(k) = 936.14129125$ *or* 936 (*approx*) *or* $h(63) = 936$.

  The location of k = 63 will be at 936.

- For $k = 64$,

  $h(k) = [m(kA \bmod 1)]$, $A = 0.61803398875.$

  $h(k) = [\,1000 * (\,64 * 0.61803398875 \bmod 1)]$

  $h(k) = [\,1000 * (\,39.55417528 \bmod 1)]$

  $h(k) = [\,1000 * 0.55417528\,]$

  $h(k) = 554.17528$ *or* 554 (*approx*) *or* $h(64) = 554$.

  The location of k = 64 will be at 554.

- For $k = 65$,

  $h(k) = [m(kA \bmod 1)]$, $A = 0.61803398875..$

  $h(k) = [\,1000 * (\,65 * 0.61803398875 \bmod 1)]$

  $h(k) = [\,1000 * (\,40.17220926875 \bmod 1)]$

  $h(k) = [\,1000 * 0.17220926875\,]$

  $h(k) = 172.20926875$ *or* 172 (*approx*) *or* $h(64) = 172$.

NAME : - Jinggui Tan & Siddhant Barua
CWID :- 10427381 & 10439929

The location of k = 64 will be at 172.