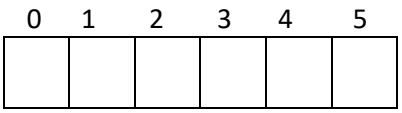
HW 5

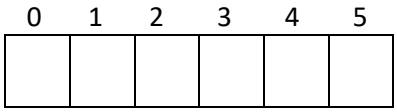
1. We wish to implement a dictionary by using direct addressing on a huge array. At the start, the array entries may contain garbage, and initializing the entire array is impractical because of its size. Describe a scheme for implementing a direct-address dictionary on a huge array. Each stored object 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.)

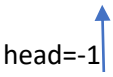
Solution:
Let us consider an additional stack S and the array be X.
Initially both the array and the stack are empty.
INITIALIZE:
Since the stack is empty, let there be a variable h(the head) which points to -1.

Array X:



Stack S:





Thus initialization takes constant time O (1).

Let us now try to insert an element with key "a" into the dictionary.

INSERT: Before we try to insert a particular key into the dictionary, we should first check whether it is already present in the dictionary. We do this by using the SEARCH operation which is explained in the next section.
If it is present we should replace the key, otherwise we need to insert the key at the respective index.
Now the insertion of the key in the respective index should follow the following condition:
i)Increment the head to point to the next position.
ie, h->h+1

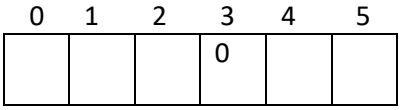
ii)Store the value of the variable h at the "key' th " index in array X.
ie ,X[a] = h

iii)Store the value of the key at the "h'th" index in the stack array S.

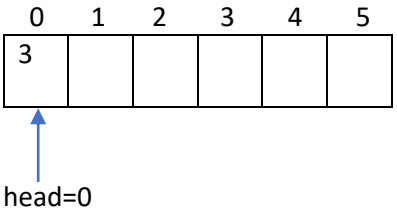
ie,S[h] = a

Let's consider a key a=3. As per the above steps we have,
h=0 (h-> -1+1)
X[3] = 0
S[0] = 3

Array X:

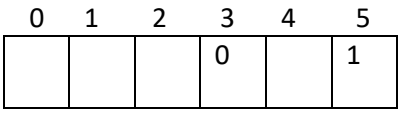


Stack S:

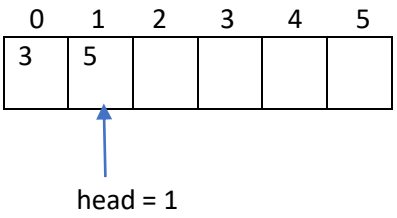


Let's now consider the next key, suppose a = 5;
h=1 (h-> 0+1)
X[5] = 1
S[1] = 5

Array X:



Stack S:



We can insert the other keys in the dictionary in a similar way.

Thus it can be seen that the insertion operation takes O(1) running time as well as O(1) space.

SEARCH:
Suppose we now want to search the key 'a' in the array.
While searching the following conditions should be satisfied:

$0 < X[a] < h$
$S[X[a]] = a$
If the above conditions are true return S[X[a]],
Else return "key not found"

Let's continue with the example that we have considered above. Suppose we want to search the key a=5 that we have inserted.

$0 < X[5] < 1$
$S[X[5]] = 5$ ie, $S[1]=5$
which stands true.

Therefore the key is found.
Return the key S[X[5]]

For searching any element the above steps would always be performed ,this implies that the search operation takes O(1) running time.

DELETE:
Suppose we want to delete an element with key "a".
But before the delete operation ,we need to check if the key is present in the array X(dictionary).
If the key is not found (ie. record is not present in the dictionary) ,return a suitable message like "No record found to delete!"
If key is found , delete the key.

When we delete the key in the array X, the position in the stack array S that used to point to this deleted key in X will be left empty leaving an empty space .

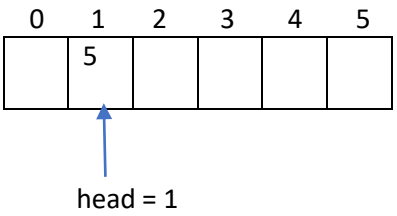
Perform the following steps to fill in the empty space :
i)Copy the element at the position in the stack to which the head points and store it at the location in the stack where there is an empty space.
$S[X[a]] = S[h]$

Suppose in our example ,we have deleted key a=3,

Array X:

0	1	2	3	4	5
					1

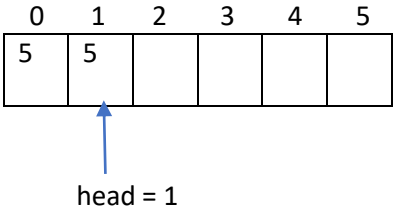
Stack S:



This left a empty space at S[0]
Now ,
S[h] = S[1] = 5
S[X[a]] = S[X[3]] = S[0]

As per our condition ,assign
S[0] = 5

Stack S:



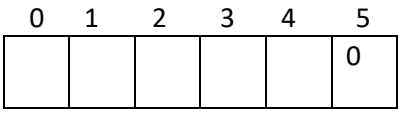
Thus the gap gets filled.
But after we assign the value of head to the empty space ,the original value at the head and its corresponding entry in array X needs to be handled.
This is to be done as follows:

X[S[h]] = X[a]
S[h] = NULL
X[a] = NULL
h-> h-1

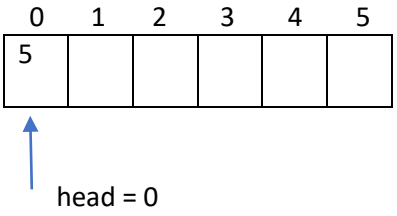
In our example,
X[a] = X[3] = 0
X[S[h]] = X[S[1]] = X[5]

Therefore , X[5] = 0
S[1] = NULL
X[3] = NULL
h = 0 (h-> 1-1=0)

Array X:



Stack S:



Thus it can be seen that the deletion operation takes O(1) running time as we perform the above mentioned steps for deletion of any given element.

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

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

Solution:

h(k)=[m(kA mod 1)], A=[sqrt(5)-1]/2 =0.618

A=0.6180339887, m=1000

For key k= 61 :

h(k)=[m(kA mod 1)]

h(k)=1000x[(61x0. 6180339887)mod 1]

h(k)=1000x(37.700 mod1)

h(k)=1000x0.700

h(k)=700

For key k= 62:

h(k)=[m(kA mod 1)]

h(k)=1000x[(62x0. 6180339887)mod 1]

h(k)=1000x(38.318 mod1)

h(k)=1000x0.318

h(k)=318

For key k= 63:

h(k)=[m(kA mod 1)]

h(k)=1000x[(63x0. 6180339887)mod 1]

h(k)=1000x(38.936mod1)

h(k)=1000x0.936

h(k)=936

For key k= 64:

h(k)=[m(kA mod 1)]

h(k)=1000x[(64x0. 6180339887)mod 1]

h(k)=1000x(39.554 mod1)

h(k)=1000x0.554

h(k)=554

For key k= 65:

h(k)=[m(kA mod 1)]

h(k)=1000x[(65x0. 6180339887)mod 1]

h(k)=1000x(40.172 mod1)

h(k)=1000x0.172

h(k)=172