1 To implement a dictionary by using direct addressing on a huge array

To implement the dictionary using direct addressing on huge array, we can use an additional stack. Let H to be the given huge array and S to be the array which is stack. To achieve direct addressing, we will be storing elements in stack S and corresponding pointers (keys) into the array H.

Suppose n = number of keys to be stored in the dictionary = stack size
Let's assume that keys are non-negative integers.

1.1 Initialization

First create an object type stack of size n. Here type of stack depends on what kind of elements one wants to store in the dictionary. This can be done by creating a top pointer and assigning it to -1. Top points to the last element added to the stack. top=-1 implies dictionary being empty. Hence, it can be used for proof checking after init(n) operation to ensure that empty dictionary has been created.

1.1.1 Pseudocode (INIT(n))

(1) Create stack of size n (S[n])
(2) S.top = -1

Analysis:(Assumption: Huge array H is already given)
Since initialization of stack takes constant time, overall time for initialization operation = O(1).

1.2 Search

Suppose we want to search an element in the dictionary with key = k. If valid key is found, then this function will return an element with key=k otherwise it will return -1 on unsuccessful search.
To implement search, two conditions should hold true:
i) given key should exists in the stack
ii) the corresponding location should be between 0 and stack size(n).

1.2.1 Pseudocode (SEARCH(H, S, k))

```
(1) If (H[k] >= 0 and H[k] <= S.top and S[H[k]] == k)
(2)      return S[H[k]]
(3) else
(4)      return -1
```

Analysis: If..else statement takes constant time since simple comparisons are done. Hence total time for search operation = O(1).

1.3 Insert

This function will add an element in the dictionary whose key is k. If an element with key=k already exists, then it will just replace it, otherwise it will push k into the stack S. Since the stack size = n the overflow case is omitted.

1.3.1 Pseudocode (INSERT(H, S, k))

```
(1) temp = SEARCH(H, S, k)
(2) if(temp exists)
(3)     S[H[temp]] = k
(4) else
(5)     S.top = S.top + 1
(6)     H[k] = S.top
(7)     S[S.top] = k
```

Analysis: As mentioned earlier search operation takes O(1) time which is constant. All other statements are just assignments hence they will take constant time too. So total time for insert operation = O(1)

1.4 Delete

This function will delete an element with the key k from the dictionary. First it will do the search to check if element exists in the dictionary. If element exists, we are removing that element by replacing it with top element of stack.

1.4.1 Pseudocode (DELETE(H, S, k))

```
(1) loc = SEARCH(H, S, k)
(2) if(loc exists)
(3)     S[H[loc]] = S[S.top]
(4)     H[S[S.top]] = H[k]
(5)     S[S.top] = null
(6)     H[k] = null
(7)     S.top = S.top - 1
(8) else
(9)     return ("key does not exist in the dictionary")
```

Analysis: Again search operation will take O(1) time which is constant and all other statements are just assignments hence they will take constant time too. Thus, total time for delete operation = O(1)

Hence, we can implement a direct-address dictionary on a huge array using an additional stack, which we can be used to determine whether a given entry in the huge array is valid or not and this all can be done in constant O(1) time.

2 Compute the locations for given keys

For k=61:

$$\begin{aligned} h(k) &= \lfloor m(kA \ mod \ 1) \rfloor && (1)\\ &= \lfloor 1000((61 * 0.61803398875) \ mod \ 1) \rfloor && (2)\\ &= \lfloor 1000((37.7000733138) \ mod \ 1) \rfloor && (3)\\ &= \lfloor 1000(0.7000733138) \rfloor && (4)\\ &= \lfloor 700.0733138 \rfloor && (5)\\ &= 700 && (6) \end{aligned}$$

For k=62:

$$\begin{aligned} h(k) &= \lfloor m(kA \ mod \ 1) \rfloor && (7)\\ &= \lfloor 1000((62 * 0.61803398875) \ mod \ 1) \rfloor && (8)\\ &= \lfloor 1000((38.3181073025) \ mod \ 1) \rfloor && (9)\\ &= \lfloor 1000(0.3181073025) \rfloor && (10)\\ &= \lfloor 318.1073025 \rfloor && (11)\\ &= 318 && (12) \end{aligned}$$

For k=63:

$$\begin{aligned} h(k) &= \lfloor m(kA \ mod \ 1) \rfloor && (13)\\ &= \lfloor 1000((63 * 0.61803398875) \ mod \ 1) \rfloor && (14)\\ &= \lfloor 1000((38.9361412913) \ mod \ 1) \rfloor && (15)\\ &= \lfloor 1000(0.9361412913) \rfloor && (16)\\ &= \lfloor 936.1412913 \rfloor && (17)\\ &= 936 && (18) \end{aligned}$$

For k=64:

$$
\begin{aligned}
h(k) &= \lfloor m(kA \ mod \ 1) \rfloor && (19)\\
&= \lfloor 1000((64 * 0.61803398875) \ mod \ 1) \rfloor && (20)\\
&= \lfloor 1000((39.55417528) \ mod \ 1) \rfloor && (21)\\
&= \lfloor 1000(0.55417528) \rfloor && (22)\\
&= \lfloor 554.17528 \rfloor && (23)\\
&= 554 && (24)
\end{aligned}
$$

For k=65:

$$
\begin{aligned}
h(k) &= \lfloor m(kA \ mod \ 1) \rfloor && (25)\\
&= \lfloor 1000((65 * 0.61803398875) \ mod \ 1) \rfloor && (26)\\
&= \lfloor 1000((40.1722092688) \ mod \ 1) \rfloor && (27)\\
&= \lfloor 1000(0.1722092688) \rfloor && (28)\\
&= \lfloor 172.2092688 \rfloor && (29)\\
&= 172 && (30)
\end{aligned}
$$