Contact: David Zhang – dzhang [at] cs [dot] toronto [dot] edu

1. The VeB queue works with giant arrays; the time cost of initializing them to all zeros would be prohibitive. Devise a way to avoid initializing large arrays. More specifically, develop a data structure that holds n items according to an index $i \in \{1, \ldots, n\}$ and supports the following operations in $O(1)$ time (worst case; i.e. not amortized, not expected) per operation:

   Init Initializes the data structure (assuming that the necessary space has been allocated) to empty.

   Set(i, x) places item x at index i in the data structure.

   Get(i) returns the item at index i, or "empty" if nothing is there.

   Your data structure should use $O(n)$ space and should work **regardless** of what garbage values are stored in that space at the beginning of execution. You can assume n fits in one machine word.
   Hint: Use extra space to remember which entries of the array have been initialized. But remember: the extra space also starts out with garbage entries![1]

If we desire $O(1)$ worst-case operations, then we cannot use lazy or randomized data structures. We will use arrays to map indices to keys. The big question is: Is the key at index i garbage? This is impossible to answer if using a single array. We can try using two arrays with a $count$ variable corresponding to the number of keys that are currently present.

Have one array store all the keys in an 'active' subarray from indices $1 \ldots count$. Then, store the position of each key at index i in the second array. When we query the data structure, if index i in the second array returns a position $j \leq count$, then the first array at position j is a valid key. Unfortunately, we cannot verify that j is not garbage. Therefore, use a third array.

Call the array mapping input indices to the 'active' block array, A. Call the array maintaining the 'active' block B, and call the array that actually contains the keys C.

- On initialization, allocate space for A, B and C.
- On inserts, assign $C(i) = x$, $count = count + 1$, $B(count) = i$, $A(i) = count$.
- On queries, first check that $A(i) \leq count$ to verify that $B(A(i))$ is a valid entry. Further, $B(A(i)) = i$ verifies that the data structure indeed stores the key corresponding to index i. Hence, $C(i)$ is guaranteed to be non-garbage. If $A(i) > count$ or $B(A(i)) \neq i$, then the data structure does not store anything at index i.

All operations run in $O(1)$ worst-case time and the space requirement is $O(n)$.

[1]This question was taken from Karger's 6.854 Advanced Algorithms Homework:
https://nb.mit.edu/nb_viewer.html?id=4c730237d23143602d1ff49aab3f0d41