Assignment 3

Meet Desai (202311031)

20.3-4) What happens if you call VEB-TREE-INSERT with an element that is already in the vEB tree? What happens if you call VEB-TREE-DELETE with an element that is not in the vEB tree? Explain why the procedures exhibit the behavior that they do. Show how to modify vEB trees and their operations so that we can check in constant time whether an element is present.

Handling Existing and Non-existing Elements in vEB Tree Operations:

1. **Inserting an Existing Element into a vEB Tree:** In the scenario where `VEB-TREE-INSERT` is applied to an element that already resides within the vEB tree, the procedure will recognize the element's presence and maintain the tree's structure intact. Since vEB trees are designed to encode element presence within their arrangement, attempting to insert an element already present does not alter the structure, preserving its original state.
2. **Deleting a Non-existing Element from a vEB Tree:** Should `VEB-TREE-DELETE` be employed to remove an element not currently present in the vEB tree, the procedure will identify the absence of the specified element and leave the tree's configuration unaltered. As vEB trees signify element presence through their structure, trying to delete an element that isn't encoded within the structure results in no modifications being made.

Enabling Constant-Time Element Presence Verification:

To achieve instantaneous confirmation of an element's presence in a vEB tree, a supplementary data structure can be implemented alongside the tree itself. This auxiliary structure guarantees constant-time element presence checks but comes with increased space complexity.

1. **Hash Table Approach:** One option is to employ a hash table where elements serve as keys, and their presence status in the vEB tree acts as the associated value. Utilizing this hash table, one can swiftly validate an element's presence by examining its status in the hash table.
2. **Bitmap Approach:** Alternatively, a bitmap can be utilized, where each bit corresponds to an element's presence or absence in the vEB tree. Upon inserting an element, the corresponding bit is set to 1; when deleting, it is set to 0. Directly accessing the bit grants immediate knowledge of an element's presence

20.3-5) Suppose that instead of p" u clusters, each with universe size p# u, we constructed vEB trees to have u1=k clusters, each with universe size u11=k, where k>1 is a constant. If we were to modify the operations appropriately, what would be their running times? For the purpose of analysis, assume that u1=k and u11=k are always integers.

If you call `VEB-TREE-INSERT` with an element that is already in the vEB tree, the operation will proceed without making any changes to the tree. This is because vEB trees are designed to store unique elements, and they do not allow duplicates. Since the element is already present, the procedure will recognize that and won't perform any modifications.

If you call `VEB-TREE-DELETE` with an element that is not in the vEB tree, the operation will simply terminate without making any changes. This is because the procedure needs to locate the element in the tree to delete it, and since the element is not present, there's nothing to delete.

To modify vEB trees and their operations to allow constant-time checking whether an element is present, you can use an additional data structure like a hash table or a bitmap. When inserting elements into the vEB tree, you can also insert them into the hash table or set the corresponding bit in the bitmap. This way, you can check if an element is present in constant time by looking it up in the hash table or checking the bit in the bitmap.

Keep in mind that introducing this additional data structure will slightly increase the overhead for insertion and deletion operations, but it will provide the constant-time element presence checking you're looking for