Succinct Encoding of Binary Tree

A succinct encoding of Binary Tree takes close to minimum possible space. The number of structurally different binary trees on n nodes is n'th Catalan number. For large n, this is about 4^n; thus we need at least about $\log_2 4^n = 2n$ bits to encode it. A succinct binary tree therefore would occupy 2n+o(n) bits.

One simple representation which meets this bound is to visit the nodes of the tree in preorder, outputting "1" for an internal node and "0" for a leaf. If the tree contains data, we can simply simultaneously store it in a consecutive array in preorder.

Below is algorithm for encoding:

```
function EncodeSuccinct(node n, bitstring structure, array data) {
    if n = nil then
        append 0 to structure;
    else
        append 1 to structure;
        append n.data to data;
        EncodeSuccinct(n.left, structure, data);
        EncodeSuccinct(n.right, structure, data);
}
```

And below is algorithm for decoding

```
function DecodeSuccinct(bitstring structure, array data) {
    remove first bit of structure and put it in b
    if b = 1 then
        create a new node n
        remove first element of data and put it in n.data
        n.left = DecodeSuccinct(structure, data)
        n.right = DecodeSuccinct(structure, data)
        return n
    else
        return nil
}
```

Source: https://en.wikipedia.org/wiki/Binary_tree#Succinct_encodings

Example:

```
Input:
        10
     /      \
   20       30
  /  \        \
 40   50      70

Data Array (Contains preorder traversal)
10 20 40 50 30 70

Structure Array
1 1 1 0 0 1 0 0 1 0 1 0 0
1 indicates data and 0 indicates NULL
```

Below is C++ implementation of above algorithms.

C++

```
// C++ program to demonstrate Succinct Tree Encoding and decoding
#include<bits/stdc++.h>
using namespace std;

// A Binary Tree Node
// A Binary Tree Node
struct Node
{
    int key;
    struct Node* left, *right;
};

// Utility function to create new Node
Node *newNode(int key)
{
    Node *temp = new Node;
    temp->key  = key;
    temp->left  = temp->right = NULL;
    return (temp);
}

// This function fills lists 'struc' and 'data'.  'struc' list
// stores structure information. 'data' list stores tree data
void EncodeSuccinct(Node *root, list<bool> &struc, list<int> &data)
{
    // If root is NULL, put 0 in structure array and return
    if (root == NULL)
    {
        struc.push_back(0);
        return;
    }

    // Else place 1 in structure array, key in 'data' array
    // and recur for left and right children
    struc.push_back(1);
    data.push_back(root->key);
    EncodeSuccinct(root->left, struc, data);
    EncodeSuccinct(root->right, struc, data);
}

// Constructs tree from 'struc' and 'data'
Node *DecodeSuccinct(list<bool> &struc, list<int> &data)
{
    if (struc.size() <= 0)
        return NULL;

    // Remove one item from from structure list
    bool b = struc.front();
    struc.pop_front();

    // If removed bit is 1,
    if (b == 1)
    {
         // remove an item from data list
        int key = data.front();
        data.pop_front();

        // Create a tree node with the removed data
        Node *root = newNode(key);

        // And recur to create left and right subtrees
        root->left = DecodeSuccinct(struc, data);
        root->right = DecodeSuccinct(struc, data);
        return root;
    }

    return NULL;
}

// A utility function to print tree
void preorder(Node* root)
{
    if (root)
    {
        cout << "key: "<< root->key;
        if (root->left)
            cout << " | left child: " << root->left->key;
        if (root->right)
```

```
            cout << " | right child: " << root->right->key;
        cout << endl;
        preorder(root->left);
        preorder(root->right);
    }
}

// Driver program
int main()
{
    // Let us construct the Tree shown in the above figure
    Node *root         = newNode(10);
    root->left         = newNode(20);
    root->right        = newNode(30);
    root->left->left   = newNode(40);
    root->left->right  = newNode(50);
    root->right->right = newNode(70);

    cout << "Given Tree\n";
    preorder(root);
    list<bool> struc;
    list<int>  data;
    EncodeSuccinct(root, struc, data);

    cout << "\nEncoded Tree\n";
    cout << "Structure List\n";
    list<bool>::iterator si; // Structure iterator
    for (si = struc.begin(); si != struc.end(); ++si)
        cout << *si << " ";

    cout << "\nData List\n";
    list<int>::iterator di; // Data iIterator
    for (di = data.begin(); di != data.end(); ++di)
        cout << *di << " ";

    Node *newroot = DecodeSuccinct(struc, data);

    cout << "\n\nPreorder traversal of decoded tree\n";
    preorder(newroot);

    return 0;
}
```

Python

```
# Python program to demonstrate Succient Tree Encoding and Decoding

# Node structure
class Node:
    # Utility function to create new Node
    def __init__(self , key):
        self.key = key
        self.left = None
        self.right = None

def EncodeSuccint(root , struc , data):

    # If root is None , put 0 in structure array and return
    if root is None :
        struc.append(0)
        return

    # Else place 1 in structure array, key in 'data' array
    # and recur for left and right children
    struc.append(1)
    data.append(root.key)
    EncodeSuccint(root.left , struc , data)
    EncodeSuccint(root.right , struc ,data)
```

```
# Constructs tree from 'struc' and 'data'
def DecodeSuccinct(struc , data):
    if(len(struc) <= 0):
        return None

    # Remove one item from structure list
    b = struc[0]
    struc.pop(0)

    # If removed bit is 1
    if b == 1:
        key = data[0]
        data.pop(0)

        #Create a tree node with removed data
        root = Node(key)

        #And recur to create left and right subtrees
        root.left = DecodeSuccinct(struc , data);
        root.right = DecodeSuccinct(struc , data);
        return root

    return None


def preorder(root):
    if root is not None:
        print "key: %d" %(root.key),

        if root.left is not None:
            print "| left child: %d" %(root.left.key),
        if root.right is not None:
            print "| right child %d" %(root.right.key),
        print ""
        preorder(root.left)
        preorder(root.right)

# Driver Program
root = Node(10)
root.left = Node(20)
root.right = Node(30)
root.left.left = Node(40)
root.left.right = Node(50)
root.right.right = Node(70)

print "Given Tree"
preorder(root)
struc = []
data = []
EncodeSuccint(root , struc , data)

print "\nEncoded Tree"
print "Structure List"

for i in struc:
    print i ,

print "\nDataList"
for value in data:
    print value,

newroot = DecodeSuccinct(struc , data)

print "\n\nPreorder Traversal of decoded tree"
preorder(newroot)

# This code is contributed by Nikhil Kumar Singh(nickzuck_007)
```

Output:

```
Given Tree
key: 10 | left child: 20 | right child: 30
key: 20 | left child: 40 | right child: 50
key: 40
key: 50
key: 30 | right child: 70
key: 70

Encoded Tree
Structure List
1 1 1 0 0 1 0 0 1 0 1 0 0
Data List
10 20 40 50 30 70

Preorder traversal of decoded tree
key: 10 | left child: 20 | right child: 30
key: 20 | left child: 40 | right child: 50
key: 40
key: 50
key: 30 | right child: 70
key: 70
```