CSE 100 Advanced Data Structures Fall 2018

Homework 4

Instructors: *Leo Porter (Sec. A)*
& *Debashis Sahoo (Sec. B)*

Due on: Thursday Nov. 15th *(40 points)*

Name: Shihtan Chan PID: A15677346

Instructions

1. Answer each problem in the boxes provided. Any writing outside of the boxes *will NOT be graded*. Do not turn in responses recorded on separate sheets.
2. Handwritten or typed responses are accepted. In either case, make sure all answers are in the appropriate boxes.
3. All responses *must* be neat and legible. Illegible answers will result in zero points.
4. Make sure to *scan in portrait mode* and to *select the corresponding pages* on Gradescope for each question.
5. You may use code from any of the class resources, including Stepik. You may not use code from other sources.

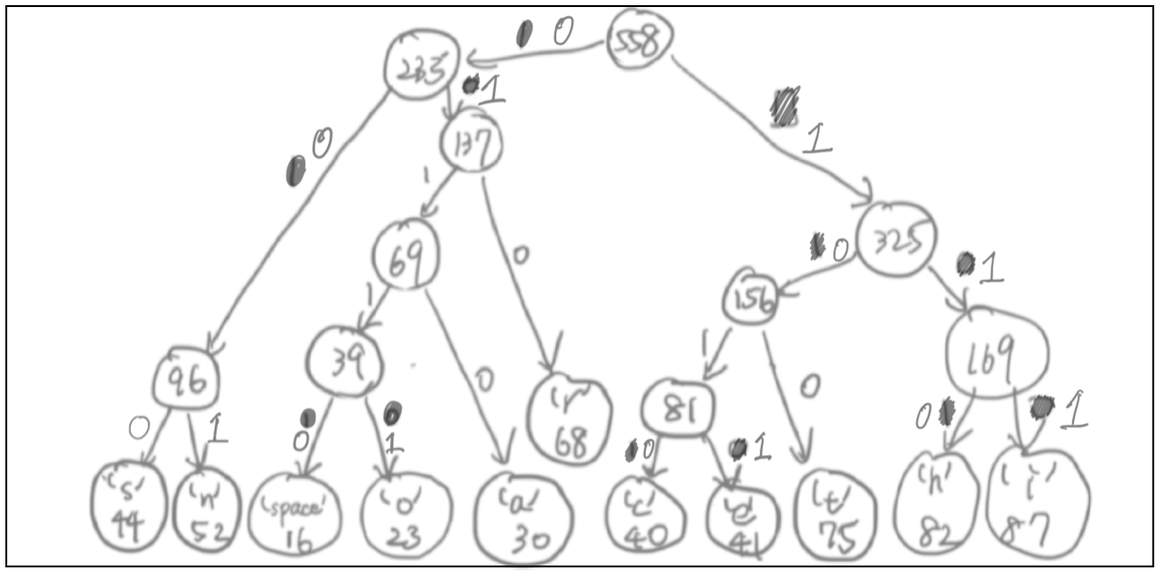
1. (15 points **- Correctness**) Huffman Encoding

   (a) (5 points **- Correctness**) Create a Huffman Tree on the following distribution of characters:

   {'a': 30, 'c': 40, 'e': 41, 'h': 82, 'i': 87, 'n': 52, 'o': 23, 'r': 68, 's': 44, 't': 75, ' ':16}

   (the last character is a space, you can free to call it 'space' or ' ' in you tree).

   At each split, clearly indicate which side gets '0' and which gets '1'. At each split, you should assign '0' to the subtree with the lower total weight and '1' to the other subtree. If both subtrees have the same weight, you may assign '0' and '1' arbitrarily.



   (b) (5 points **- Correctness**) Using **your** above tree, encode the sentence "this is a secret".
   Using **your** above tree, decode the sequence '00101111011101111000111011100001110000101111100'. (If there are trailing bits at the end of your decoding, explicitly state what the trailing part of the sequence is.)

encode this is a secret:
100 110 111 000 01110 111 000 01110 0110 01110 000 1011 1010 010 1011 100
decode the sequence:
no it is not

(c) (5 points **- Correctness**)

What is the runtime of building a huffman encoding?

Once you already have an encoding, what is the runtime of encoding a document?

Once you already have an encoding, what is the runtime of decoding a document?

Define the variables you think you need. If you feel you need (reasonable) clarifying assumptions, state them next to your answer. (Different answers will be allowed depending on your assumptions, but incorrect answers will not be given credit).

If the elements (alphabets) are sorted by frequency, the runtime of building a Huffman encoding is o(n). n is the number of elements (alphabets)
If the elements (alphabets) are not sorted, the runtime of building a Huffman is o(nlogn)
n is the number of elements (alphabets) Since we need to sort it by appearance frequency first and keep finding the smallest node.
encoding is o(nlogk). n is the number of elements (alphabets) to be encoded. k is the number of elements (alphabets) used to construct the Huffman tree.
Runtime of encoding a document with n alphabets is o(nlogn) when the probability of alphabets' appearance in document is same as frequency while building the Huffman tree.
Runtime of decoding a document with n bits is o(n) since we can find next bit along the tree path in o(1) time (left or right node).

2. (15 points **- Correctness**) Suppose you've been given the following code:

```
class AVLNode {
 public:
  AVLNode * left, right, parent;
  int value;
  int height; // root node stores tree height; a single node tree has height 1
};

//Rotate left @ top.
//E.g. top->parent should remain an ancestor of all of its descendants
//and top's location in the tree should be replaced by top->right
void rotateLeft(AVLNode * top);

void rotateRight(AVLNode * top) { /* Implementation not shown */ }
```

(a) (10 points **- Correctness**) Implement rotateLeft in either psuedocode or C++ (not English):

```
void rotateLeft(AVLNode* top){
if(top->parent){
   if(top->parent->left==top){
      top->parent->left=top->right;
   }
   else{
      top->parent->right=top->right;
   }
   top->right->parent=top->parent;
}
else{
   top->right->parent=nullptr;
}
top->parent=top->right;
if(top->right->left){
   AVLNode* tmp=top->right->left;
   top->right->left=top;
   tmp->parent=top;
   top->right=tmp;
}
else{
   top->right->left=top;
   top->right=nullptr;
}
int left=top->left->height;
int right=0;
if(top->right)right=top->right->height;
top->height=max(left,right)+1;
left=top->height;
right=0;
if(top->right->right)right=top->right->right->height;
top->right->height=max(left,right)+1;
}
```

(b) (5 points **- Correctness**) Given the following method signature:

```
//Double rotate @ top
//so that top->parent remains an ancestor of all of its descendants
//and so that this method would be appropriate to call for
//balancing a tree consisting of three nodes:
// {top, top->left, top->left->right}
//(though this should work when there are more nodes in the tree)
void doubleRotate_LeftKink(AVLNode * top);

void doubleRotate_RightKink(AVLNode * top) { /* Implementation not shown */ }
```

Implement `doubleRotate_LeftKink` in either psuedocode or C++ (not English). You may assume the setup on the previous page, and that rotateLeft and rotateRight have been implemented:

```
void doubleRotate_LeftKink(AVLNode* top){
    rotateLeft(top->left);
    rotateRight(top);
}
```

3. (5 points - **Completeness**) Implement insert for an AVL tree in either psuedocode or C++ (not English). You may assume the code from problem (2).

```
//returns false if data is already in the tree, otherwise inserts
//data into the tree and returns true.
bool insert(AVLNode * root, int data);
```

```
bool insert(AVLNode* root, int data){
    if(!root){root=new AVLNode;root->value=data;root->height=1;return true;}
    bool ans;
    if(root->value==data)return false;
    if(root->value<data){
        if(!(root->left)){
            root->left=new AVLNode; root->left->value=data; root->left->height=1;
            root->left->parent=root;
            ans=true;
        }
        else ans=insert(root->left,data);
    }
    else{
        if(!(root->right)){
            root->right=new AVLNode; root->right->value=data; root->right->height=1;
            root->right->parent=root;
            ans=true;
        }
        else ans=insert(root->right,data);
    }
    if(!ans)return false;
    int left=0;
    int right=0;
    if(root->left)left=root->left->height;
    if(root->right)right=root->right->height;
    if(left-right>1){
        if(root->left->value > data)rotateRight(root);
        else  doubleRotate_LeftKink(root);
    }
    else if(right-left>1){
        if(root->right->value < data)rotateRight(root);
        else doubleRotate_RightKink(root);
    }
    return true;
}
```

4. (1 point - **Completeness**) Why, historically, are the colors in RB trees Red and Black?

Historically, the color "red" was chosen because it was the best-looking color produced by the color laser printer available to the authors. Another response from Guibas states that it was because of the red and black pens available to them to draw the trees.

5. (5 points - **Completeness**) Implement insert for an RB tree in either psuedocode or C++ (not English). You may assume the code below. Assume that the rotation methods correctly change all the pointers, but do not change the colors. You may use both this and the next page for your answer.

```
class BSTNode {
  public:
   BSTNode * left, right, parent;
   int value;
   bool color; //false is red, true is black
 };

 void rotateLeft(BSTNode * top) { /* Implementation not shown */ }
 void rotateRight(BSTNode * top) { /* Implementation not shown */ }
 void doubleRotate_LeftKink(BSTNode * top) { /* Implementation not shown */ }
 void doubleRotate_RightKink(BSTNode * top) { /* Implementation not shown */ }

 bool insert(BSTNode * root, int data);
```

```
bool BSTInsert(BSTNode *root,BSTNode *cur)
{if(!root)root=cur;
   if(root->value==cur->value)return false;
   if (cur->value < root->value)
   {
      if(!root->left){
         root->left  = cur;
         root->left->parent = root;
         return true;
      }
      return BSTInsert(root->left, cur);
   }
   if (cur->value > root->value)
   {
      if(!root->right){
         root->right = cur;
         root->right->parent = root;
         return true;
      }
      return BSTInsert(root->right, cur);
   }
}
bool insert(BSTNode * root, int data)
{
   BSTNode *tmp = new BSTNode;
   tmp->value=data;
   tmp->color=false;
   //first do binary search tree insert
   bool success = BSTInsert(root, tmp);
   if(!success)return false;
   //then  fix violations
   fixViolation(root, tmp);
   return true;}
```

```
void fixViolation(BSTNode * root, BSTNode * cur)
{
   BSTNode *parent = NULL;
   BSTNode *grandparent = NULL;
   while ((cur!=root)&&(!cur->color)&&(!cur->parent->color))
   {
      parent = cur->parent;
      grandparent=cur->parent->parent;
      if (parent == grandparent->left)
      {
         BSTNode *uncle = grandparent->right;
         if (uncle&&!uncle->color)
         {
            grandparent->color = false;
            parent->color = true;
            uncle->color = true;
            cur = grandparent;
         }
         else
         {
            if (cur == parent->right)
            {
               rotateLeft(parent);
               cur = parent;
               parent = cur->parent;
            }
            rotateRight(grandparent);
            bool tmp=true;
            tmp=parent->color;
            parent->color=grandparent->color;
            grandparent->color=tmp;
            cur = parent;
         }
      }
      else
      {
         BSTNode *uncle = grandparent->left;
         if ((uncle) && (!uncle->color))
         {
            grandparent->color = false;
            parent->color = true;
            uncle->color = true;
            cur = grandparent;
         }
         else
         {
            if (cur == parent->left)
            {
               rotateRight(parent);
               cur = parent;
               parent = cur->parent;
            }
            rotateLeft(grandparent);
            bool tmp=true;
            tmp=parent->color;
            parent->color=grandparent->color;
            grandparent->color=tmp;
            cur = parent;
         }
      }
   }
   root->color = true;
}
```