Section Solutions #6

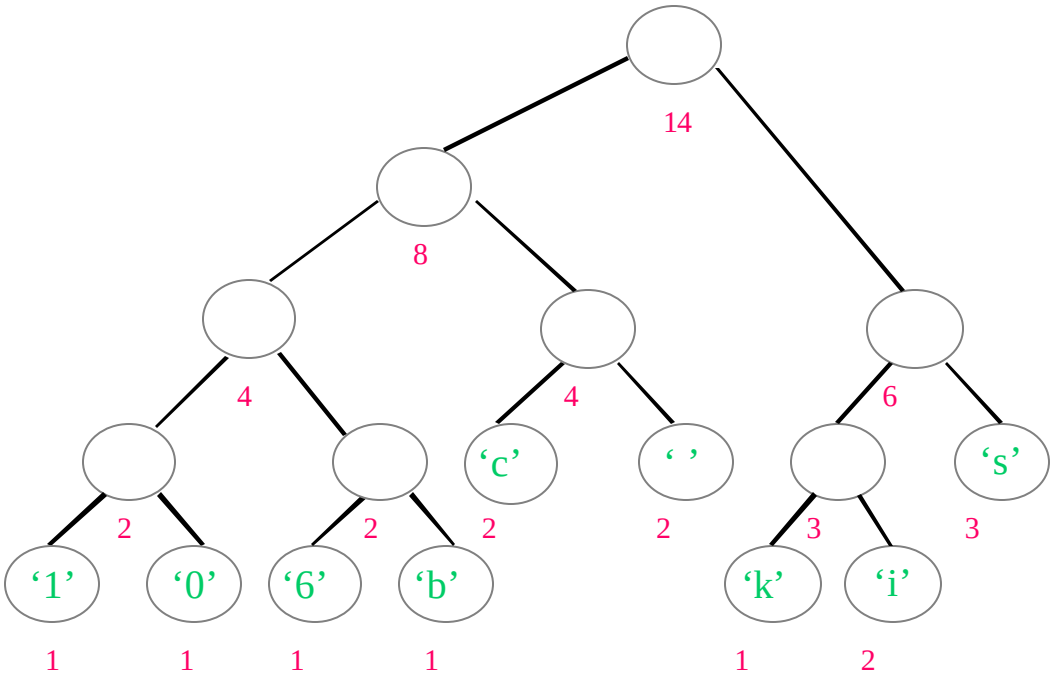
Problem 1: Trimming Leaves

```
/* A node is considered a leaf if its two children
 * both NULL */


static bool IsLeaf(node *theNode)
{
   return((theNode->left == NULL) && (theNode->right == NULL));
}


/* A modification of the DeleteNode function used in binary tree
 * delete.  This function will only delete when called on a leaf.
 * Otherwise the tree is untouched. */


static void DeleteLeaf(node **tptr)
{
   node *t;

   t = *tptr;
   /* Only delete if we're sure the node is a leaf */
   if (IsLeaf(t)) {
      *tptr = NULL;
      free(t);
   }
}


/* The basic strategy is to recursively trim the trees.  Either the
 * current node itself is a leaf, and we need to Delete it from the
 * tree, or we need to recursively trim its left and right subtrees. */


void TrimLeaves(node **tree)
{
   if (*tree != NULL)
   {
      if (IsLeaf(*tree))
         DeleteLeaf(tree);
      else
      {
         TrimLeaves(&(*tree)->left);
         TrimLeaves(&(*tree)->right);
      }
   }
}
```

Problem 2:

```
static void Encode(FILE *input, FILE *output)
{
   int curChar, prevChar;
   int count;

   prevChar = fgetc(input);

   while (prevChar != EOF){

     count = 1;
     while(TRUE){
       curChar = fgetc(input);
       if (curChar!=prevChar) break;
       count++;
     }

     if (count > 1){
       fprintf(output, "%d%c", count, prevChar);
     } else {
       fputc(prevChar, output);
     }

     prevChar = curChar;
   }
}


static void Decode(FILE *input, FILE *output)
{
   int curChar;
   int count, i;

   while (TRUE){
     count = 0;
     curChar = fgetc(input);

     // EOF here is allowed
     if (curChar == EOF) return;

     while(isdigit(curChar)){
       // this is what StringToInteger does
       count = (count * 10) + (curChar - '0');
       curChar = fgetc(input);

       // EOF in here is an error
       if (curChar == EOF) Error("Unexpected end of file.");
     }

     // having a count of zero means there was no number before the
     // character -- so make sure we print the one character.
     if (count == 0) count = 1;

     for (i=0; i<count; i++){
       fputc(curChar, output);
     }
   }
}
```

Problem 3:



Problem 4:

a) Run-length encoding would be better when there are long runs of similar characters while huffman encoding excels in files where the distribution of characters is unbalanced (i.e., tons of a's and b's with a few c's and d's). Huffman encoding is good for the aforementioned case and bad for cases where there is an even distribution of all 256 ascii characters (i.e., a files with one of each). That would just encode to 8 bits per character, not shrinking the file at all. In addition, it would have to write out the encoding tree, making the file actually larger.

b) If you try to encode a file that has already been compressed using huffman encoding, your file will most likely grow, since a file that has been encoded using this algorithm has a pretty even distribution of 0's and 1's, and thus would act like the case where there is an even distribution of all the characters.

c) There is no real right answer to this one. Discuss amongst yourself. Ask your section leader. =)