Homework #5
CSCI 4041: Algorithms and Data Structures

Last revision April 15, 2020

This homework is about Huffman's algorithm and about graph traversal. It is worth **30 points**, and it will be due on Canvas at **11:55 PM** on **April 24, 2020.**

1. (15 points.) A possible use for Huffman's algorithm is to compress text files so they take up less disk space. For example, suppose we have a file containing the text of Leo Tolstoy's huge novel *War and Peace*, whose size is over three megabytes. If we use Huffman's algorithm to compress it, then we might get a file whose size is less than one megabyte. However, if we do that, then we need a way to *expand* the compressed file, so we can recover the original text of *War and Peace.*

Write a pseudocode procedure called HUFFMAN-EXPAND that does this. It takes two parameters, *f* and *r.* The parameter *f* is a file that was compressed using Huffman's algorithm. The parameter *r* points to the root of the Huffman tree that was used to compress *f.* Your procedure HUFFMAN-EXPAND must read the file *f*, use the tree *r* to expand it one character at a time, and print the resulting characters.

Here's what you must know to write HUFFMAN-EXPAND. The file *f* contains *bits.* Each bit is either `0` or `1`. The procedure READ-BIT(*f*) reads the next bit from *f* and returns it. For example, suppose that the first three bits in *f* are `101`. Then the first time READ-BIT(*f*) is called, it returns `1`. The second time it is called, it returns `0`. The third time, it returns `1`, etc. If READ-BIT(*f*) reaches the end of *f*, so there are no more bits to be read, then it returns `-1`.

Suppose that *p* points to a node in the Huffman tree *r.* The procedure IS-EXTERNAL(*p*) returns TRUE if *p* is an external node, and FALSE otherwise. If *p* is an external node, then *p.char* points to the character in *p.* The procedure PRINT-CHAR(*c*) prints a character *c.* The procedure IS-INTERNAL(*p*) returns TRUE if *p* is an internal node, and FALSE otherwise. If *p* is an internal node, then *p.left* points to the root of *p*'s left subtree, and *p.right* points to the root of *p*'s right subtree. You might need only one of IS-EXTERNAL and IS-INTERNAL to write HUFFMAN-EXPAND.

My version of HUFFMAN-EXPAND has only ten lines of pseudocode! If you find yourself writing many fewer lines than this, or many more lines than this, then you do not understand this question, and you should ask for help.

2. (15 points.) In the lectures, the procedure GRAPH-BREADTH-FIRST takes a graph *G* and a source vertex *s* as its parameters. Starting from *s,* it visits *G*'s vertexes in breadth-first order. (The misnamed procedure BFS on page 595 of Cormen is very similar.) Write a new version of this procedure, called MARKY-GRAPH-BREADTH-FIRST, that does the same thing, but is different in two respects.

1. MARKY-GRAPH-BREADTH-FIRST must not use *color* attributes in any way. Instead of *v.color*, each vertex *v* must have an attribute *v.mark,* whose value is either TRUE or FALSE. If *v.mark* is FALSE, then *v* has never been visited. If *v.mark* is TRUE, then *v* has been visited, and will never be visited again. MARKY-GRAPH-BREADTH-FIRST must use *v.mark* instead of *v.color* to keep from being trapped in cycles.

2. MARKY-GRAPH-BREADTH-FIRST must not use *v.d* or *v*.π in any way.

GRAPH-BREADTH-FIRST calls a procedure VISIT each time it visits a vertex. MARKY-GRAPH-BREADTH-FIRST must do this also. However, do not write a definition for the procedure VISIT! We don't know what it does—it serves only to show when a vertex is visited. If you think you must write VISIT, then you do not understand this question, and you should ask for help.

My version of MARKY-GRAPH-BREADTH-FIRST has only twelve lines of pseudocode! If you find yourself writing many fewer lines than this, or many more lines than this, then you also do not understand this question, and you should ask for help.