COSC 3360 - 24967 - Fundamentals of Operating Systems

Navigation

- Dashboard
  - Site home
  - Site pages
  - My courses
    - COSC3360SP2023-01
      - Competencies
      - Grades
      - General
      - EXAM 1
      - EXAM 2
      - EXAM 3
      - PROGRAMMING ASSIGNMENTS
        - **Programming Assignment 1**
          - Description
          - Submission view
        - Programming Assignment 2
        - Programming Assignment 3
      - PRACTICE EXAM 2

Description | Submission view

Available from: Tuesday, 17 January 2023, 12:00 AM
Due date: Wednesday, 22 February 2023, 11:59 PM
Requested files: main.cpp, huffmanTree.h, huffmanTree.cpp (Download)
Type of work: Individual work

This programming assignment closes at 11:58:59 PM on 02/22/2023.

Similarity Threshold: 95%

Problem:

For this assignment, you will create a multithreaded Huffman decompressor (https://en.wikipedia.org/wiki/Huffman_coding).

To generate the Huffman tree, you must execute the following steps:

1. Read the alphabet information from an input file.
   - Each line of the input file contains information (character and frequency) about a symbol from the alphabet. The input file format is as follows (each line as:
     - A char representing the symbol.
     - An integer representing the frequency of the symbol.
2. Arrange the symbols based on their frequencies. If two or more symbols have the same frequency, they will be sorted based on their ASCII value.
3. Execute the Huffman algorithm to generate the tree. To guarantee that your solution generates the expected output from the test cases, every time you generate an internal node of the tree, you must insert this node into the queue of nodes as the lowest node based on its frequency. Additionally, you must label the edge to the left child as 0 and the edge to the right child as 1.

After generating the Huffman Tree, your solution reads a compressed file with the following format:

1. A string representing the binary code of the symbol.
2. A list of n integers (where n is the frequency of the symbol) representing the positions where the symbol appears in the message.

The compressed file has m lines (where m is the number of symbols in the alphabet).

Your solution must create m POSIX threads to decompress the file and print the original message.

Process:

Given the following input file with the information about the alphabet:

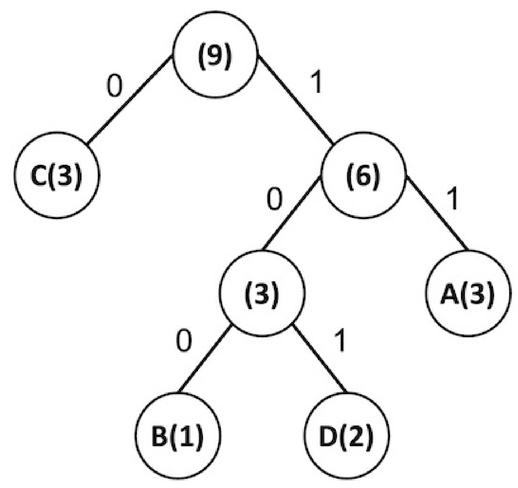
```
A 3
C 3
B 1
D 2
```

And the following compressed file:

```
11 1 3 5
0 0 2 4
101 6 8
100 7
```

Your solution must execute the following steps:

- Read from STDIN the filename (string) of the input file.
- Read the alphabet information from the input file.
- Generate the Huffman Tree. Given the input file above, the generated Huffman Tree is:



- Print the Huffman codes from the generated tree. The symbols information (leaf nodes) must be printed from left to right

  Given the previous Huffman tree, the expected output is:

```
Symbol: C, Frequency: 3, Code: 0
Symbol: B, Frequency: 1, Code: 100
Symbol: D, Frequency: 2, Code: 101
Symbol: A, Frequency: 3, Code: 11
```

- Read from STDIN the filename (string) of the compressed file.
- Read the information from the compressed file.
- Create n POSIX threads (where n is the number of symbols to decompress). Each child thread executes the following tasks:

  - Receives the Huffman tree and the information about the symbol to decompress (binary code and list of positions) from the main thread.

  - Uses the Huffman tree to determine the character from the original message based on the binary code.

  - Stores the decompressed character (as many times as needed based on the list of positions) on a memory location accessible by the main thread.

- Print the original message. For the compressed file above, the expected original message is:

```
Original message: CACACADBD
```

NOTES:

- Not using multiple POSIX threads to implement your solution will translate into a penalty of 100%.
- A penalty of 100% will be applied to solutions that do not compile.
- The huffmanTree.h and huffmanTree.cpp files are optional (if you want to implement an OOP solution).
- You must use the output statement format based on the example above.
- You can safely assume that the input will always be valid.

Requested files

main.cpp

```
1 // Write your program here
```

huffmanTree.h

```
1 // Write the definition of the huffmanTree class here.
```

huffmanTree.cpp

```
1 // Write the implementation of the member functions of the huffmanTree class here.
```

VPL